

Leichtgewichtige Software- Reviews mit LASR

Stefan Toth, embarc

Basel One, 2026



Stefan Toth

CEO, Berater für Agilität
Softwarearchitektur



Stefan.Toth@embarc.de

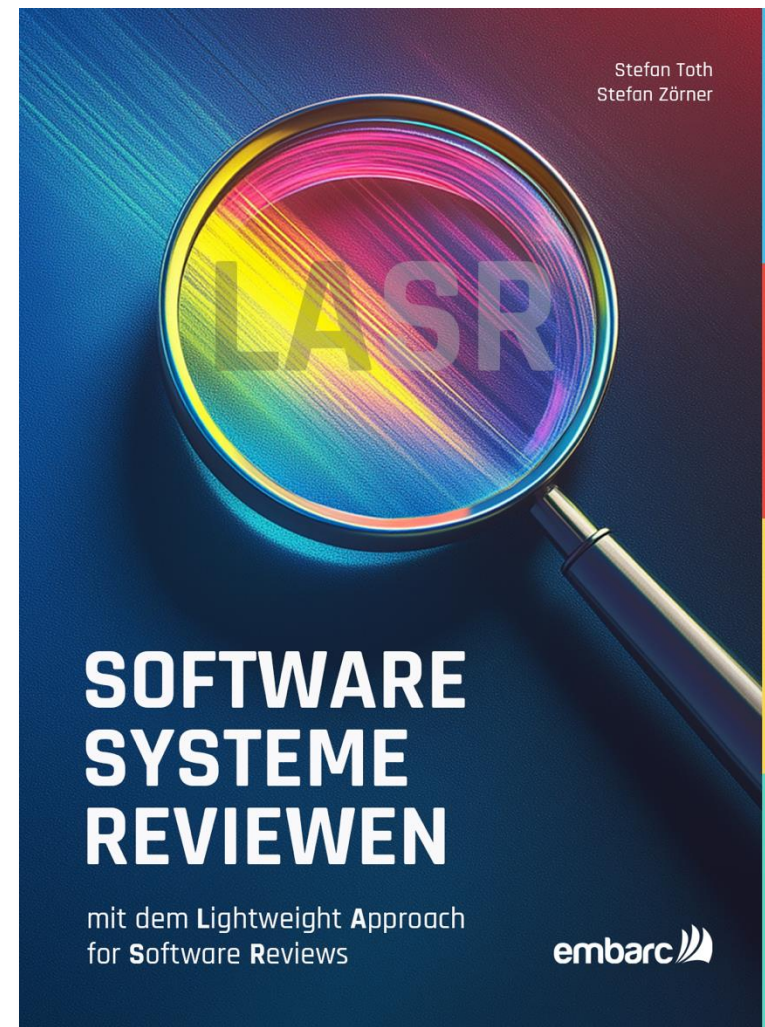


linkedin.com/in/sto-embarc



www.embarc.de







Agenda

Leichtgewichtige
Software-Reviews
mit LASR

01. Warum leichtgewichtig bewerten?

02. Verstehe, was Dich speziell macht

03. Durchleuchte die Architektur

04. Erhöhe die Konfidenz (bei Bedarf)

05. Weitere Informationen

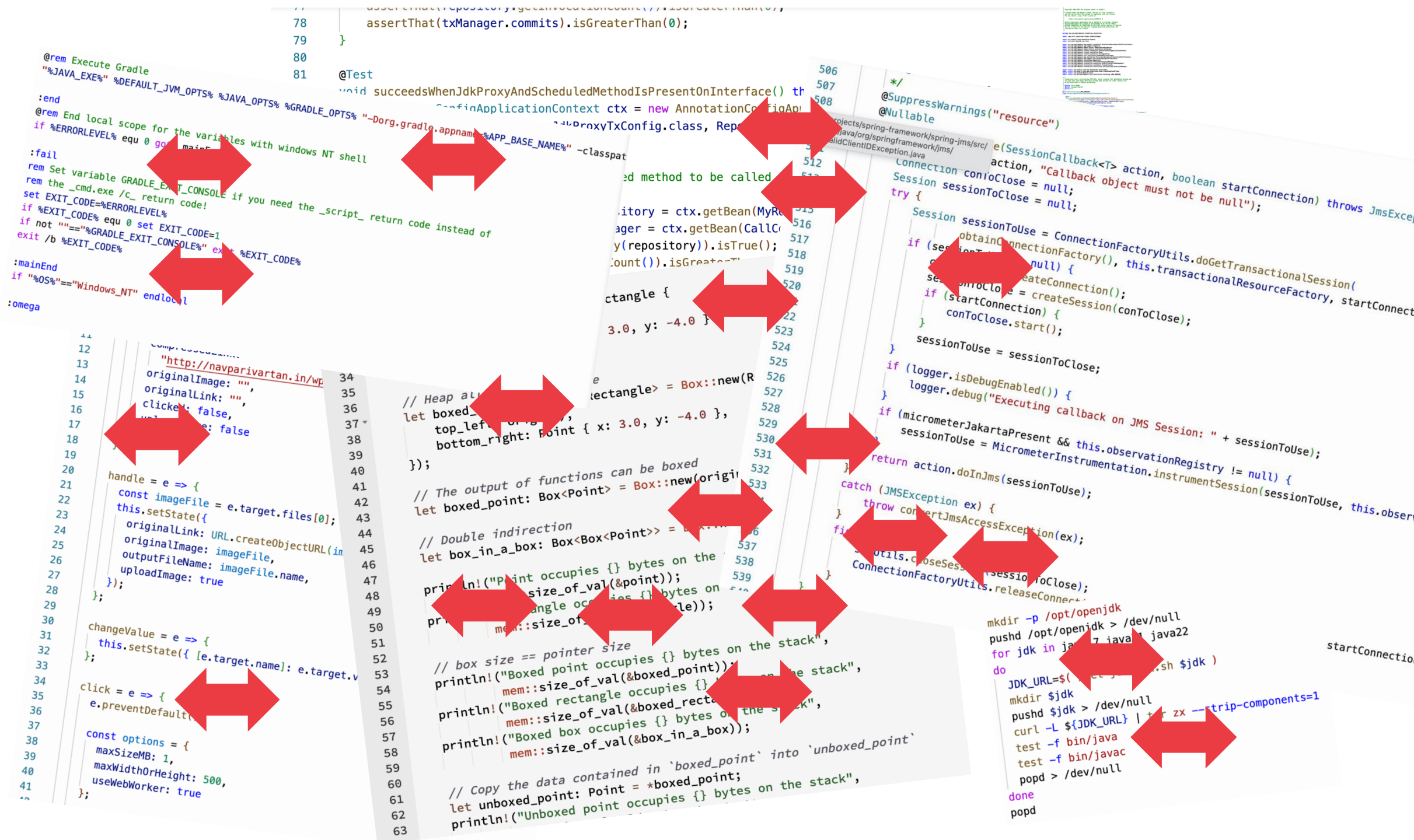




Warum Reviews?

Kostet etwas und schafft keinen direkten Wert...





```

78     assertThat(txManager.commits).isGreaterThan(0);
79 }
80
81 @Test
    void succeedsWhenJdkProxyAndScheduledMethodIsPresentOnInterface() throws Exception {
        // inApplicationContext ctx = new AnnotationConfigAp
        // proxyTxConfig.class, RepoConfigB
    }

```

```

@rem Execute Gradle
"%JAVA_EXE%" %DEFAULT_JVM_OPTS% %JAVA_OPTS% %GRADLE_OPTS% "-Dorg.gradle.appname=%APP_BASE_NAME%" -classpath
:cmd
@rem End local scope for the variables with windows NT shell
if %ERRORLEVEL% equ 0 goto mainEnd
:fail
rem Set variable GRADLE_EXIT_CONSOLE if you need the _script_ return code instead of
rem the _cmd.exe /c_ return code!
set EXIT_CODE=%ERRORLEVEL%
if %EXIT_CODE% equ 0 set EXIT_CODE=1
if not "" "%GRADLE_EXIT_CONSOLE%" exit %EXIT_CODE%
exit /b %EXIT_CODE%
:mainEnd
if "%OS%"=="Windows_NT" endlocal
:omega

```

```

12
13
14     "http://navparivartan.in/wp
15     originalImage: "",
16     originalLink: "",
17     clicked: false,
18     uploadImage: false
19 };
20
21 handle = e => {
22     const imageFile = e.target.files[0];
23     this.setState({
24         originalLink: URL.createObjectURL(imageFile),
25         originalImage: imageFile,
26         outputFileName: imageFile.name,
27         uploadImage: true
28     });
29 };
30
31 changeValue = e => {
32     this.setState({ [e.target.name]: e.target.v
33 });
34
35 click = e => {
36     e.preventDefault();
37
38     const options = {
39         maxSizeMB: 1,
40         maxWidthOrHeight: 500,
41         useWebWorker: true
42     };

```

```

34
35 // Heap allocation
36 let boxed_rectangle = {
37     top_left: origin(),
38     bottom_right: Point { x: 3.0, y: -4.0 },
39 };
40
41 // The output of functions can be boxed
42 let boxed_point: Box<Point> = Box::new(origin());
43
44 // Double indirection
45 let box_in_a_box: Box<Box<Point>> = Box::new(boxed_point);
46
47 println!("Point occupies {} bytes on the stack", mem::size_of_val(&point));
48 println!("Rectangle occupies {} bytes on the stack", mem::size_of_val(&rectangle));
49
50 // box size == pointer size
51 println!("Boxed point occupies {} bytes on the stack", mem::size_of_val(&boxed_point));
52 println!("Boxed rectangle occupies {} bytes on the stack", mem::size_of_val(&boxed_rectangle));
53 println!("Boxed box occupies {} bytes on the stack", mem::size_of_val(&box_in_a_box));
54
55 // Copy the data contained in 'boxed_point' into 'unboxed_point'
56 let unboxed_point: Point = *boxed_point;
57 println!("Unboxed point occupies {} bytes on the stack",

```

```

506
507
508 */
@SuppressWarnings("resource")
@Nullable
e(SessionCallback<T> action, boolean startConnection) throws JmsException {
    connection conToClose = null;
    Session sessionToClose = null;
    try {
        Session sessionToUse = ConnectionFactoryUtils.doGetTransactionalSession(
            obtainConnectionFactory(), this.transactionalResourceFactory, startConnection);
        if (sessionToUse == null) {
            conToClose = createConnection();
            sessionToClose = createSession(conToClose);
            if (startConnection) {
                conToClose.start();
            }
            sessionToUse = sessionToClose;
        }
        if (logger.isDebugEnabled()) {
            logger.debug("Executing callback on JMS Session: " + sessionToUse);
        }
        if (micrometerJakartaPresent && this.observationRegistry != null) {
            sessionToUse = MicrometerInstrumentation.instrumentSession(sessionToUse, this.observationRegistry);
        }
        return action.doInJms(sessionToUse);
    } catch (JMSException ex) {
        throw convertJmsAccessException(ex);
    } finally {
        JmsUtils.closeSession(sessionToClose);
        ConnectionFactoryUtils.releaseConnection(sessionToClose);
    }
}

```

```

mkdir -p /opt/openjdk
pushd /opt/openjdk > /dev/null
for jdk in java17 java21 java22
do
    JDK_URL=$( /get-jdk-url.sh $jdk )
    mkdir $jdk
    pushd $jdk > /dev/null
    curl -L $JDK_URL | tar zx --strip-components=1
    test -f bin/java
    test -f bin/javac
    popd > /dev/null
done
popd

```

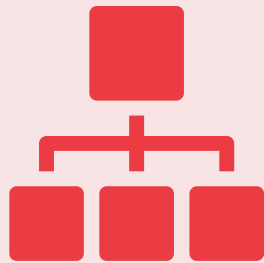


„Accidental Architecture“

Architektur-Themen

Zerlegung

Welcher Architekturstil?
Wie zerfällt die Anwendung?
Teilsysteme, Module,
Komponentenbildung,
Abhängigkeiten ...



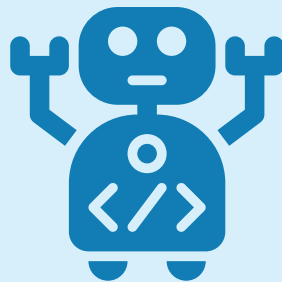
Zielumgebung

Wo läuft die Software?
Beim Endanwender, im
eigenen Rechenzentrum,
Cloud, Verteilung,
Virtualisierung ...



Technologie-Stack

Was setzen wir ein?
Programmiersprache(n)
Bibliotheken, Frameworks,
Middleware,
Querschnittsthemen

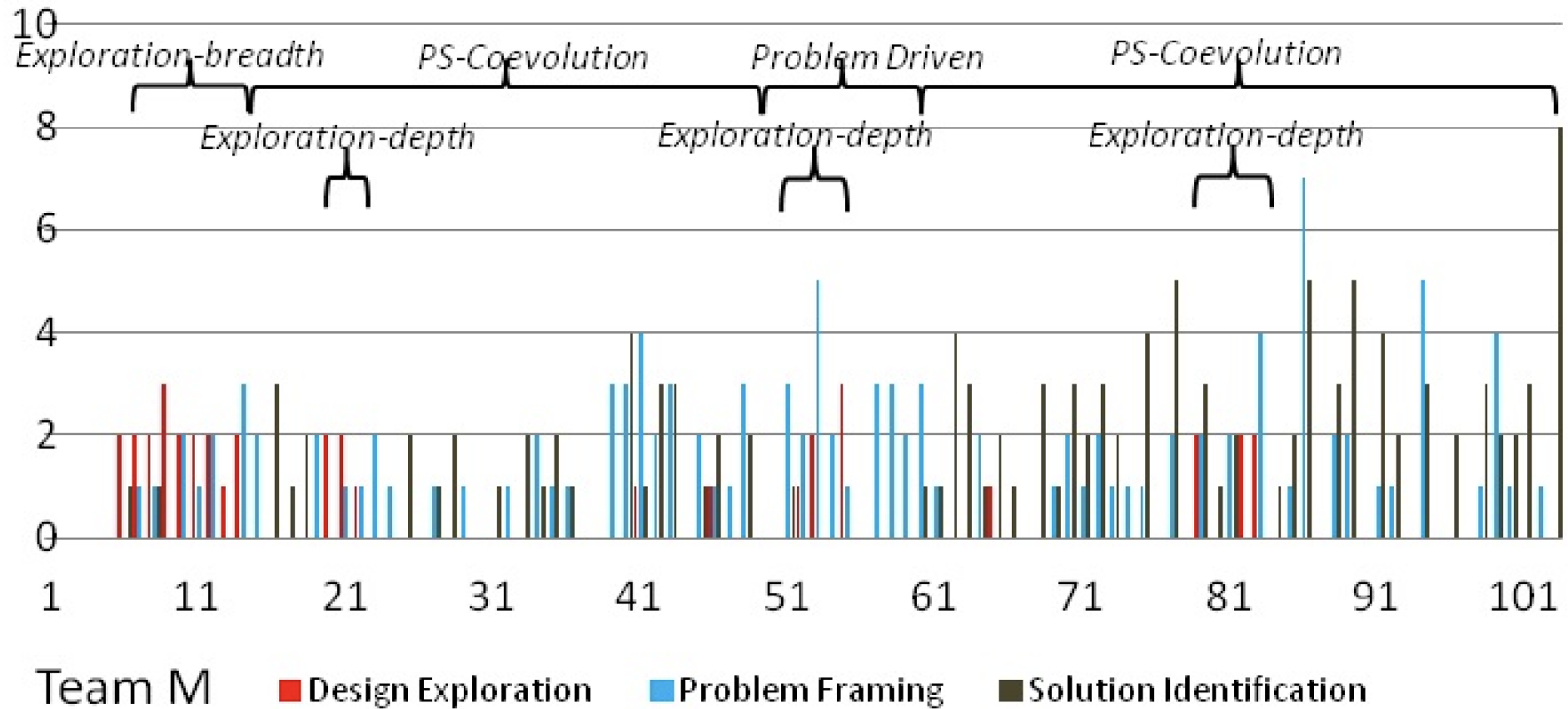


Vorgehen

Wie arbeiten wir?
Planen, Entwickeln, Testen,
Bauen, Dokumentieren,
Ausliefern, Nachjustieren, ...



Selbst Top-Down Design ist unübersichtlich



Aus: "Decision Making in Software Architecture", Hans van Vlieta, Antony Tang, 2015

Wir handeln oft intuitiv



Anekdoten-
getrieben

nach Bauchgefühl

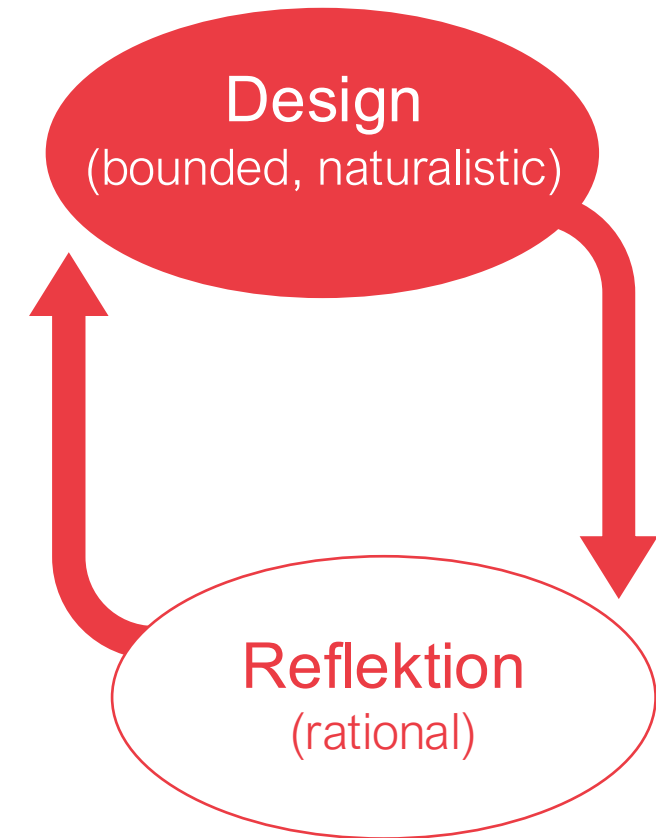
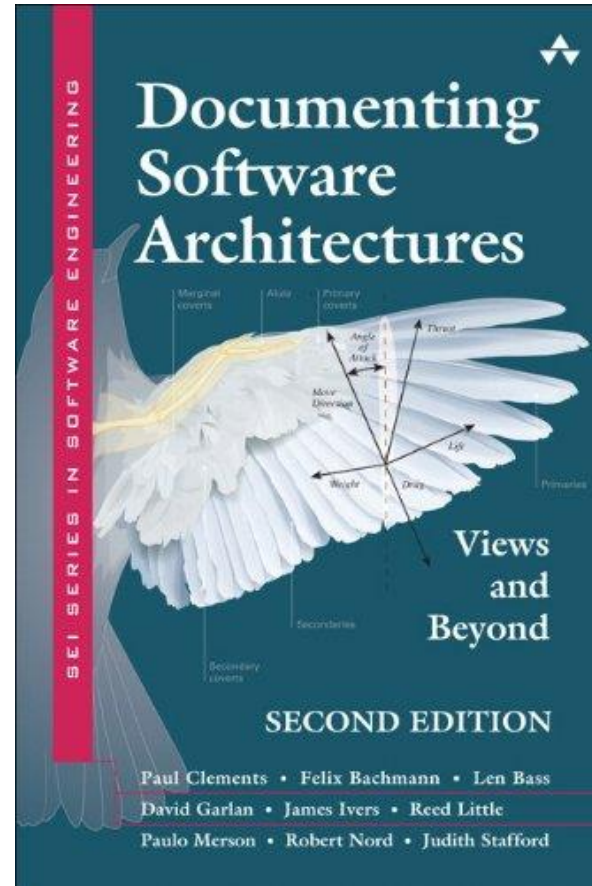
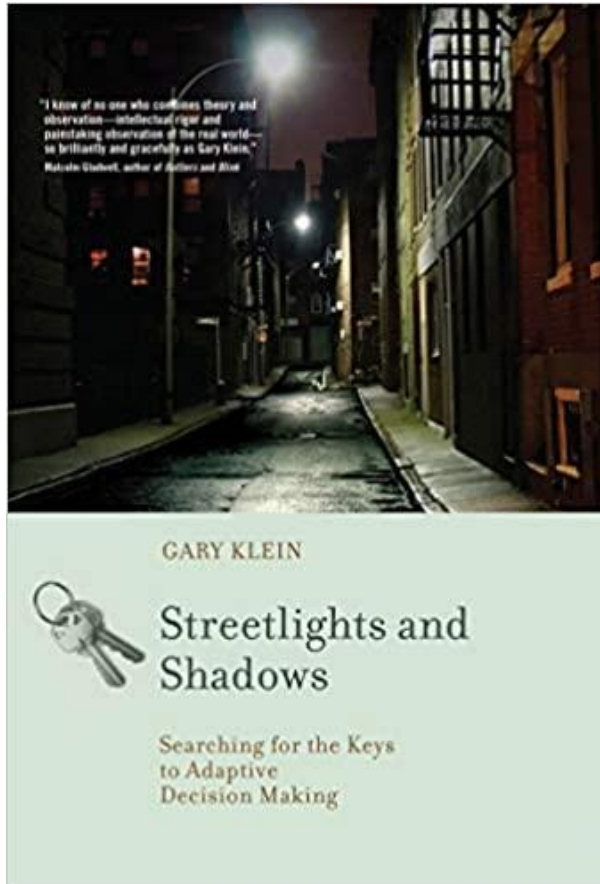
Konsens-
gesteuert

emotional

moralgetrieben

erfahrungsbasiert

Rationalität im nachhinein hilft





1998 1999

2000

2001 2002

ATAM – Architecture Tradeoff Analysis Method

„Konzeptioneller Fluss“ von ATAM



„Konzeptioneller Fluss“ von ATAM

I'm familiar with architecture evaluation methods such as the technical Architecture Tradeoff Analysis Method (ATAM) and more business-oriented Cost Benefit Analysis Method (CBAM). However, these methods are fairly large scale: they prescribe several brainstorming sessions, development of a host of scenarios describing tradeoffs, etc. While useful for internal projects or desktop applications that are

through the same 9 ATAM steps, but with less formality than what is described in the SEI's ATAM reference. Other ATAM reports that MITRE has participated in during the past have shown the ATAM to be a very "heavyweight" approach; the assessment of this project by necessity of the resource limitations and schedule demands had to be more of a "lightweight" assessment.

definitions from stakeholders and reviewing architectural evaluation methods. The lightweight factors are acquired by studying the five most commonly used lightweight methods and the Architecture-based Tradeoff Analysis Method (ATAM), the most well-known heavyweight method. Subsequently, the research addresses these features and

NON-RISKS



2024 2025

2026

2027 2028

Bewertungsmethoden

SAAM

Software Architecture Analysis Method

ATAM

Architecture Tradeoff Analysis Method

CBAM

Cost-Benefit Analysis Method

TARA

Tiny Architecture Review Approach

PBAR

Pattern Based Architecture Review

ARID

Architecture Review for Intermediate Designs

DCAR

Decision Centric Architecture Review

DASE

Decision and Scenario based arch. evaluation

Pre-Mortem

Risiko-Brainstorming and Mitigation

LASR

Lightweight Approach for Software Reviews

Stark moderierte Bewertungsmethoden

SAAM

Software Architecture Analysis Method

ATAM

Architecture Tradeoff Analysis Method

CBAM

Cost-Benefit Analysis Method

TARA

Tiny Architecture Review Approach

PBAR

Pattern Based Architecture Review

ARID

Architecture Review for Intermediate Designs

DCAR

Decision Centric Architecture Review

DASE

Decision and Scenario based arch. evaluation

Pre-Mortem

Risiko-Brainstorming and Mitigation

LASR

Lightweight Approach for Software Reviews

Individuelle Bewertungsmethoden

SAAM

Software Architecture Analysis Method

ATAM

Architecture Tradeoff Analysis Method

CBAM

Cost-Benefit Analysis Method

TARA

Tiny Architecture Review Approach

PBAR

Pattern Based Architecture Review

ARID

Architecture Review for Intermediate Designs

DCAR

Decision Centric Architecture Review

DASE

Decision and Scenario based arch. evaluation

Pre-Mortem

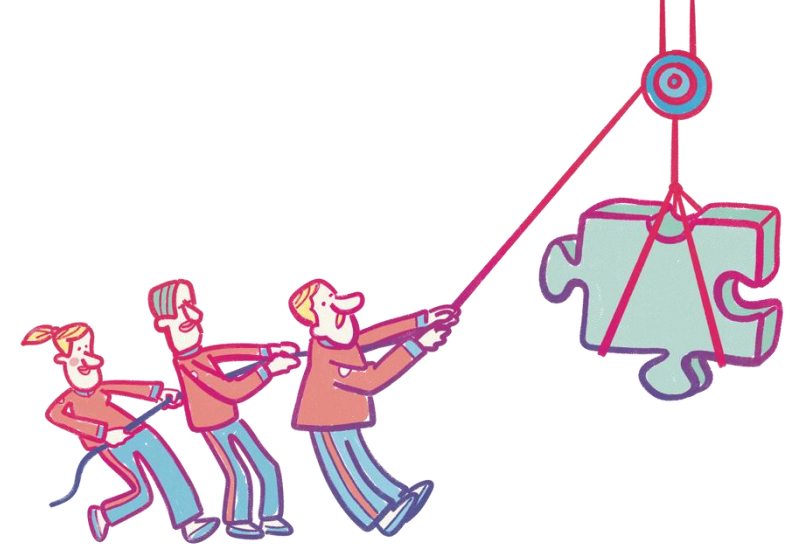
Risiko-Brainstorming and Mitigation

LASR

Lightweight Approach for Software Reviews

Kollaborative Methoden

Fokus: Gemeinsames Verstehen



- **Alle Teilnehmenden** sind **aktiv**
- **Implizites Wissen** ist **sichtbar**
- Schafft **gemeinsames Verständnis**
- **Widersprüche** sind **direkt sichtbar**
- **Ownership** ist **stark**
- **Methoden** müssen **stärker ausgearbeitet** sein
- **Konflikte** sind **unmoderiert**



Besonders wertvoll bei komplexen, cross-funktionalen Fragestellungen

Kollaborative Methoden...



DDD & Domain Modeling

- Event Storming
- Domain Storytelling
- Example Mapping
- Context Mapping



Agil & Produktfokus

- Retrospektiven
- Backlog Refinement
- Three Amigos
- Value Stream Mapping



Engineering / Dev

- Pair / Mob Programming
- Brown Bag Sessions
- Threat Modeling
- Hackathon



UX & Discovery

- Design Thinking Workshops
- Design Sprints
- Customer-Journey Mapping
- Co-Design Workshops

Kollaborative Bewertungsmethoden

SAAM

Software Architecture Analysis Method

ATAM

Architecture Tradeoff Analysis Method

CBAM

Cost-Benefit Analysis Method

TARA

Tiny Architecture Review Approach

PBAR

Pattern Based Architecture Review

ARID

Architecture Review for Intermediate Designs

DCAR

Decision Centric Architecture Review

DASE

Decision and Scenario based arch. evaluation

Pre-Mortem

Risiko-Brainstorming and Mitigation

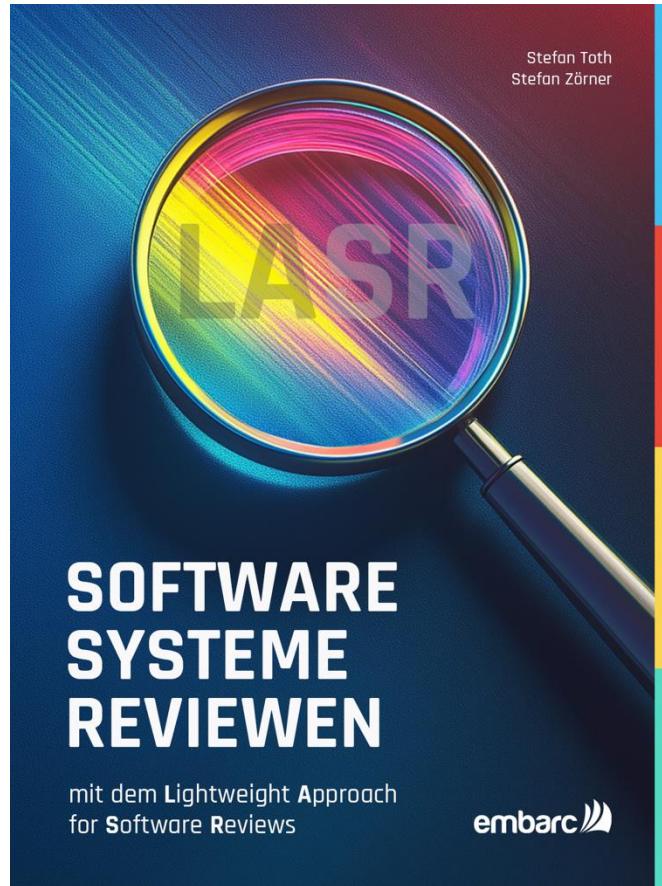
LASR

Lightweight Approach for Software Reviews

„LASR“



Erfahrungswissen



Software-Systeme reviewen mit dem Lightweight Approach for Software Reviews - LASR

Autoren: Stefan Toth, Stefan Zörner

Sprache: Deutsch, Englisch

Formate: EPUB, PDF, Print (DE)

→ leanpub.com/software-systeme-reviewen/

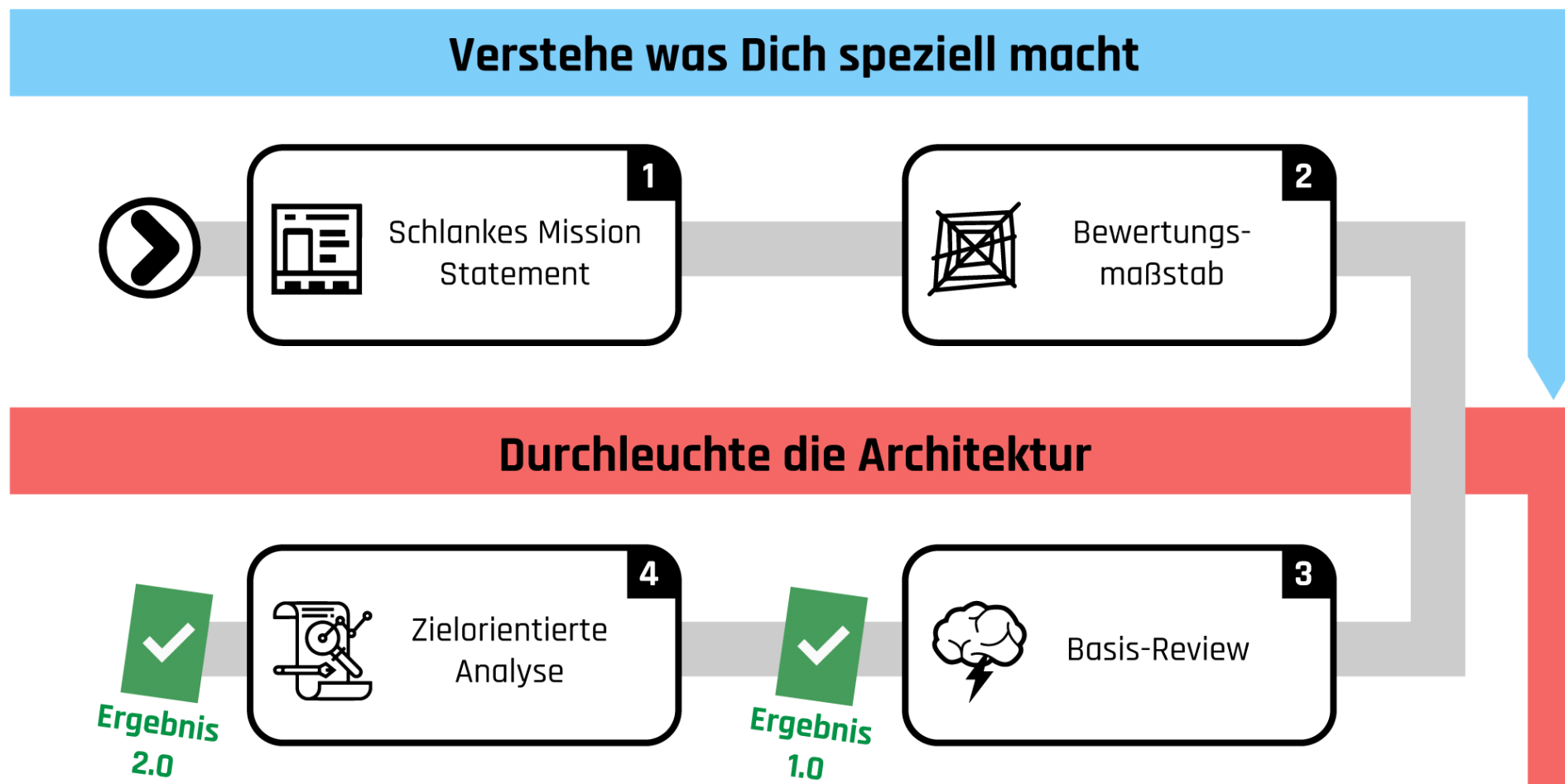
→ <https://www.amazon.de/Softwaresysteme-Reviewen-Lightweight-Approach-Software/dp/B0HM871M2F>

Markante Merkmale von LASR

- Schlankere Methode als ATAM, trotzdem **zielorientiert**
- Mit dem **eigenen Team** und potentiell **alleine durchführbar**
- Liefert **schnell** ein **erstes Ergebnis**, z.B. an einem Nachmittag
- **Spinnennetzgraphik** zur Erkenntnisverdichtung und -kommunikation
- optionale Aktivitäten zur schrittweisen **Konfidenzerhöhung bei Bedarf**
- **Unterstützungsmaterial** für Bewertungs-maßstab, Risikofindung und Ergebniserarbeitung



LASR Kernreview





Verstehe, was Dich speziell macht

01. Warum leichtgewichtig bewerten?

02. Verstehe, was Dich speziell macht

03. Durchleuchte die Architektur

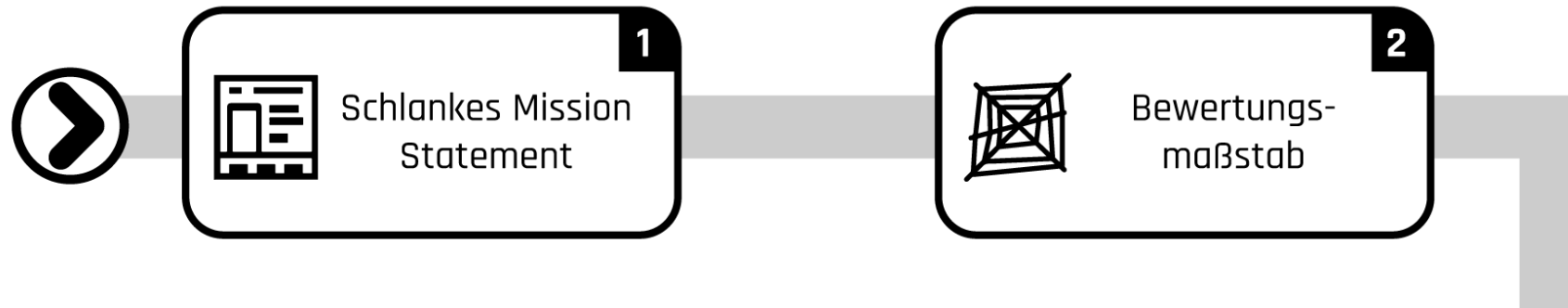
04. Erhöhe die Konfidenz (bei Bedarf)

05. Weitere Informationen



Verstehe, was Dich speziell macht

Verstehe was Dich speziell macht

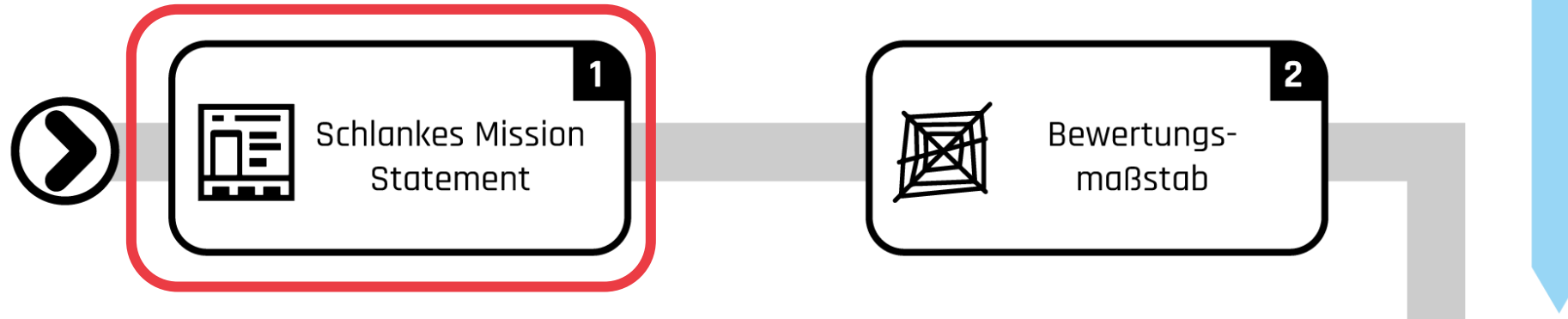


Was ist zu tun?

- Die Aufgabenstellung für das System zu einem prägnanten Mission Statement verdichten
- Die Top 3-5 Ziele des Systems identifizieren
- Jeweiliges Ziel-Level festlegen

Verstehe, was Dich speziell macht

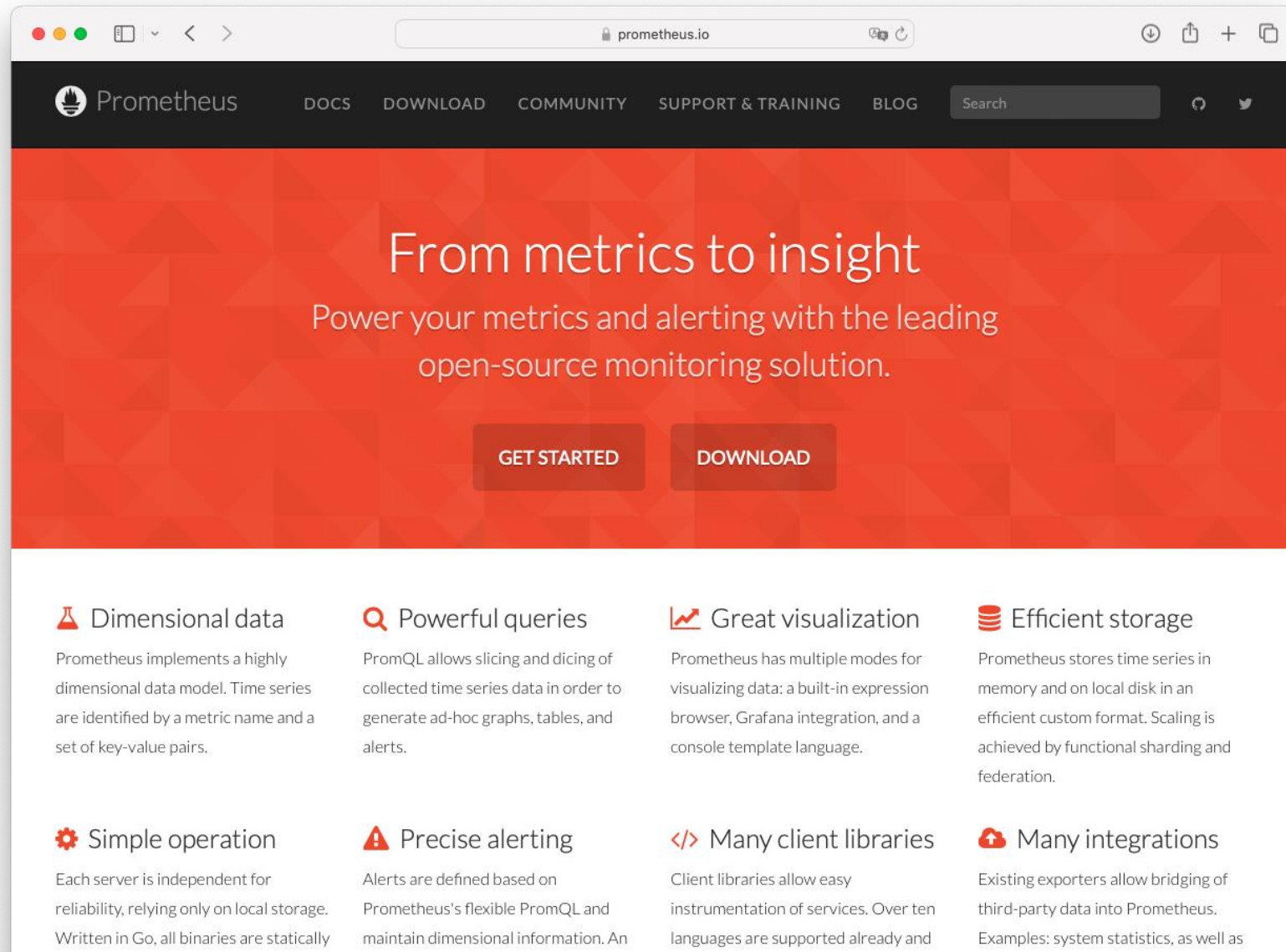
Verstehe was Dich speziell macht



Was ist zu tun?

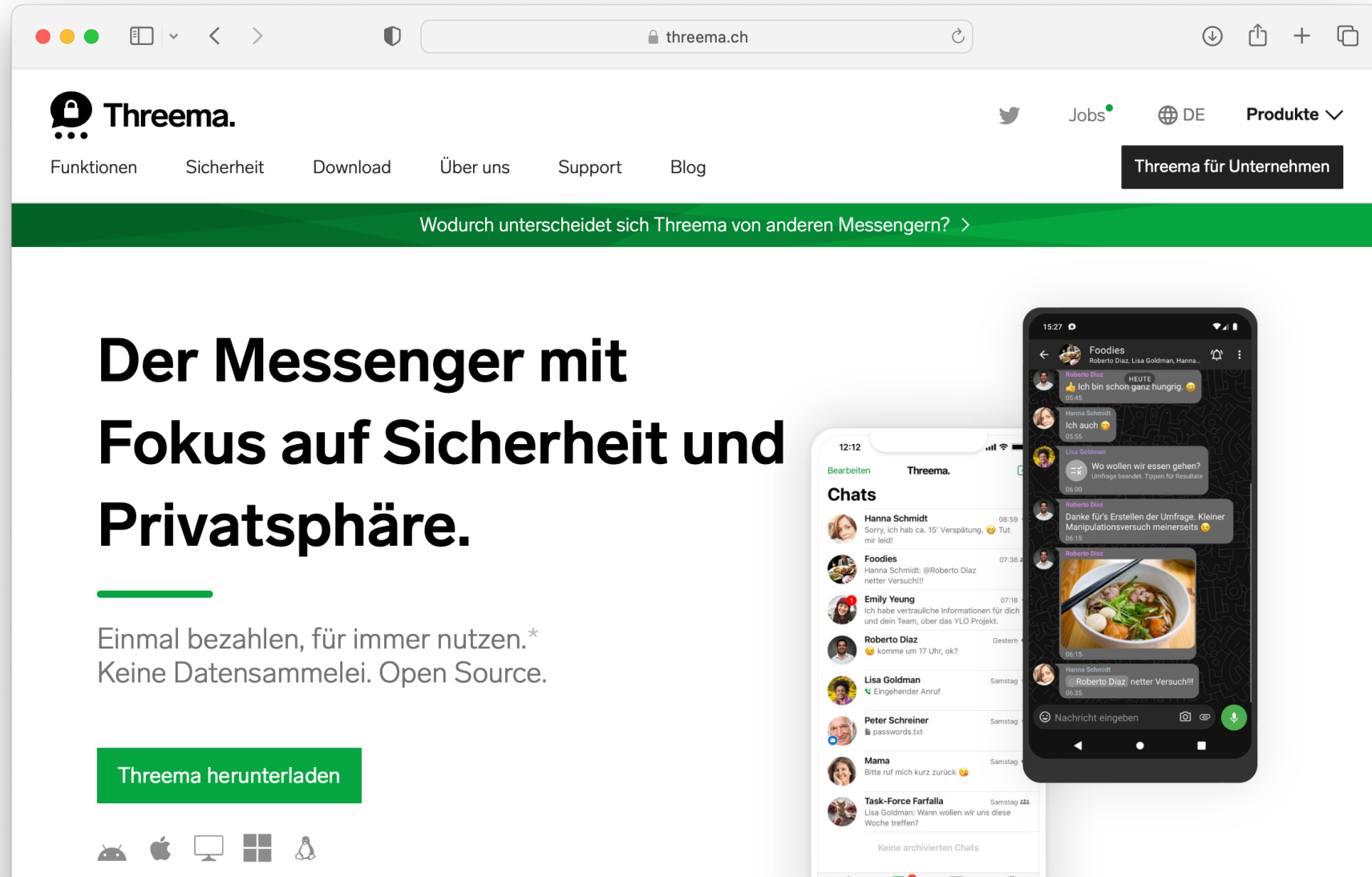
- Die Aufgabenstellung für das System zu einem prägnanten Mission Statement verdichten
- Die Top 3-5 Ziele des Systems identifizieren
- Jeweiliges Ziel-Level festlegen

Beispiel: Landing Page von Prometheus



→ prometheus.io

Weiteres Beispiel für eine Startseite



Erarbeiten im Workshop

LASR Schritt 1: Schlankes Mission Statement

Was würde prominent auf der Landing Page der zu betrachtenden Softwarelösung stehen?

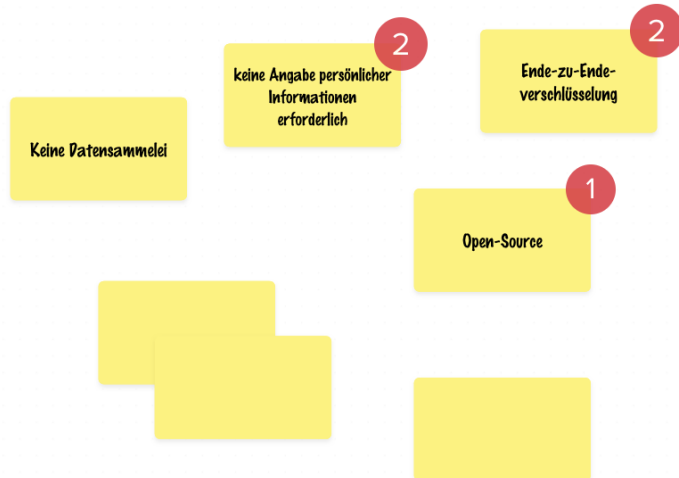


1 Brainstorming

Leitfragen:

Was sind zentrale Features oder Eigenschaften unserer Software?

Was macht unsere Software besonders? Was besonders gut?
Welche Ansprüche haben wichtige Beteiligte an die Softwarelösung? ("Claims")



2 Clustern und Ausdünnen

Voting. Welche der gefundenen Punkte sollten Eurer Meinung nach auf jeden Fall auftauchen? Ziel: ca. 7 +/- 2.



← Beispiel im digitalen Whiteboard (Mural)

Quelle: S. Toth, S. Zörner:
„Software-Systeme
reviewen“, Leanpub 2023

Qualitätsmerkmale (Begriffe nach ISO 25010:2023)



Funktionale Eignung (Functional Suitability)

Sind die berechneten Ergebnisse genau genug / exakt, ist die Funktionalität angemessen? ...



Effizienz (Performance Efficiency)

Antwortet die Software schnell, hat sie einen hohen Durchsatz, einen geringen Ressourcenverbrauch? ...



Kompatibilität (Compatibility)

Ist die Software konform zu Standards, arbeitet sie gut mit anderen zusammen? ...



Benutzbarkeit (Interaction Capability)

Ist die Software intuitiv zu bedienen, wiedererkennbar, leicht zu erlernen, attraktiv? ...



Zuverlässigkeit (Reliability)

Ist das System verfügbar, tolerant gegenüber Fehlern, nach Abstürzen schnell wieder hergestellt? ...



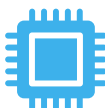
Sicherheit (Security)

Ist das System sicher vor Angriffen? Sind Daten und Funktion vor unberechtigtem Zugriff geschützt? ...



Wartbarkeit (Maintainability)

Ist die Software leicht zu ändern, erweitern, testen, verstehen? Lassen sich Teile wiederverwenden? ...



Übertragbarkeit (Flexibility)

Ist die Software leicht auf andere Situationen oder Zielumgebungen (z.B. anderes OS) übertragbar? ...

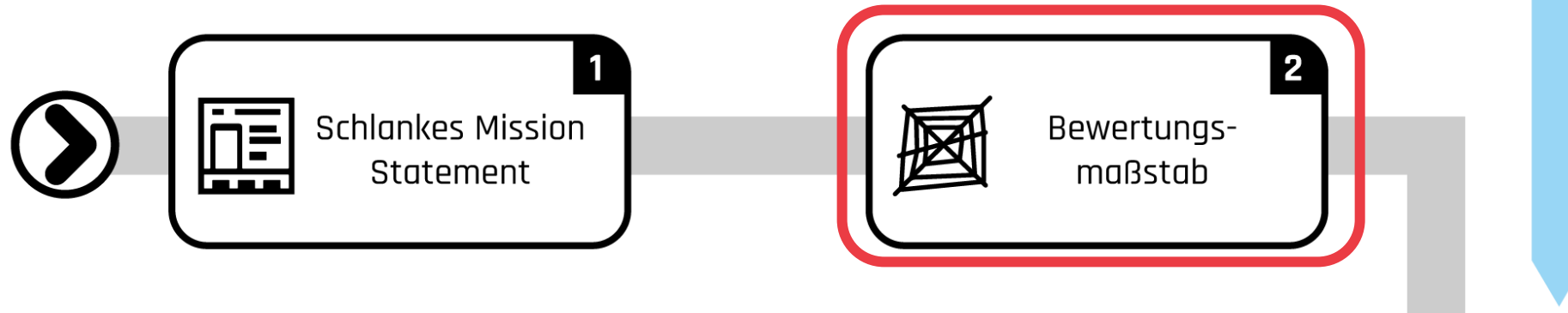


Betriebssicherheit (Safety)

Sind Personen, Tiere, Sachen und Umwelt vor Schäden durch die Software geschützt? ...

Verstehe, was Dich speziell macht

Verstehe was Dich speziell macht



Was ist zu tun?

- Die Aufgabenstellung für das System zu einem prägnanten Mission Statement verdichten
- **Die Top 3-5 Ziele des Systems identifizieren**
- **Jeweiliges Ziel-Level festlegen**

Die Qualitätsziele von Threema



Sicherheit

Kommunikations- und Nutzerdaten sind geschützt.



Benutzbarkeit

Einfach zu verwenden.



Zuverlässigkeit

Zuverlässig und effizient im Betrieb.



Kompatibilität

Interoperabel über alle Plattformen hinweg.



Wartbarkeit

Leicht erweiterbar und anpassbar.



Top-5-Challenger (Unterstützungsmaterial)

Material:

- 14 Karten mit Qualitätsmerkmalen



Vorbereitung:

- Mischt die Karten in einem verdeckten Stapel.
- Deckt 5 Karten zufällig auf und legt sie nebeneinander. (das ist Eure Start Top-5)
- Legt die übrigen 9 in einer 3 x 3 Matrix für alle gut sichtbar offen aus.

Top-5-Challenger, Vorbereitung (Beispiel)

Auditierbarkeit

Ist die Software aus einer juristischen, finanziellen oder Sicherheitsperspektive gut zu überprüfen?

Nachvollziehbarkeit
Nachweisbarkeit
Zurechenbarkeit

Skalierbarkeit

Kann die Software auf Lastschwankungen und Wachstum etwa in den Mengengeräten angemessen reagieren?

Deployment-Flexibilität
Elastizität
Wachstums-effizienz

Kosteneffizienz

Ist der Betrieb der Software wirtschaftlich optimiert und lässt sich ihr Einsatz gut planen?

Sparsamkeit
Schätzbarkeit
Kostentransparenz

Funktionale Eignung

Sind die berechneten Ergebnisse genau genug / exakt, ist die Funktionalität angemessen?

Angemessenheit
Korrektheit
Vollständigkeit

Kompatibilität

Ist die Software konform zu Standards, arbeitet sie gut mit anderen zusammen?

Koexistenz
Interoperabilität
Versionierungsfähigkeit

Start Top-5

Performanz

Antwortet die Software schnell, hat sie einen hohen Durchsatz, einen geringen Ressourcenverbrauch?

Zeitverhalten
Verbrauchsverhalten
Kapazität

Nachhaltigkeit

Ist die Software umweltverträglich? Sind Produktionsprobleme leicht auffindbar und behaltbar?

Energieeffizienz
Emissionsarmut
Langlebigkeit

Betreibbarkeit

Lässt sich die Software gut betreiben? Sind Produktionsprobleme leicht auffindbar und behaltbar?

Beobachtbarkeit
Aktualisierbarkeit
Reaktionseffizienz

Benutzbarkeit

Ist die Software intuitiv zu bedienen, wiederkennbar, leicht zu erlernen, strukturiert?

Bedienbarkeit
Erlernbarkeit
Barrierefreiheit

Portierbarkeit

Ist die Software leicht auf andere Zielumgebungen übertragbar?

Überfragbarkeit
Installierbarkeit
Austauschbarkeit

Safety

Sind Personen, Tiere, Sachen oder Umwelt vor Schäden durch die Software geschützt?

Betriebssicherheit
Verifizierbarkeit
Regulatorische Compliance

Sicherheit

Ist das System sicher vor Angriffen? Sind Daten und Funktionen vor unberechtigtem Zugriff geschützt?

Vertraulichkeit
Authentizität
Datenintegrität

Wartbarkeit

Ist die Software leicht zu ändern, erweitern, testen, verstehen? Lassen sich Teile wiederverwenden?

Analysierbarkeit
Änderbarkeit
Erweiterbarkeit

Zuverlässigkeit

Ist das System verfügbar, tolerant gegenüber Fehlern, nach Abstürzen schnell wieder hergestellt?

Verfügbarkeit
Wiederherstellbarkeit
Fehlertoleranz

Kandidaten

- Team-Mitglied schlägt Ziel vor, das in Top-5 fehlt.
- Die Gruppe diskutiert darüber.
- Der Challenger verdrängt eine Karte oder landet selbst auf dem Ablagestapel.

Nach 2 Runden, Beispiel (Beispiel)

Performanz ISO-Std

Antwortet die Software schnell, hat sie einen hohen Durchsatz, einen geringen Ressourcenverbrauch?

Zeitverhalten
Verbrauchsverhalten
Kapazität 02

Skalierbarkeit

Kann die Software auf Lastschwankungen und Wachstum etwa in den Mengengeräten angemessen reagieren?

Deployment-Flexibilität
Elastizität
Wachstums-effizienz 03

Kosteneffizienz

Ist der Betrieb der Software wirtschaftlich optimiert und lässt sich ihr Einsatz gut planen?

Sparsamkeit
Schätzbarkeit
Kostentransparenz 05

Funktionale Eignung ISO-Std

Sind die berechneten Ergebnisse genau genug / exakt, ist die Funktionalität angemessen?

Angemessenheit
Korrektheit
Vollständigkeit 01

Sicherheit ISO-Std

Ist das System sicher vor Angriffen? Sind Daten und Funktion vor unberechtigtem Zugriff geschützt?

Vertraulichkeit
Authentizität
Datenintegrität 12

Aktuelle Top-5

Kompetenz

Ist die Software konform zu Standards, welche sie gut mit anderen zusammen?

Kompetenz
Interoperabilität
Verwekempfängigkeit

Auditorbarkeit

Ist die Software konform zu Standards, welche sie gut mit anderen zusammen?

Auditorbarkeit
Interoperabilität
Verwekempfängigkeit

Ablagestapel

Nachhaltigkeit

Ist die Software umweltverträglich? Sind Produktionsprobleme leicht auffindbar und behaltbar?

Energieeffizienz
Emissionsarmut
Langlebigkeit 04

Betriebbarkeit

Löst sich die Software gut betreiben? Sind Produktionsprobleme leicht auffindbar und behaltbar?

Beobachtbarkeit
Aktualisierbarkeit
Reaktionseffizienz 05

Benutzbarkeit ISO-Std

Ist die Software intuitiv zu bedienen, wiedererkennbar, leicht zu erlernen, strukturiert?

Bedienbarkeit
Erlernbarkeit
Barrierefreiheit 09

Portierbarkeit ISO-Std

Ist die Software leicht auf andere Zielumgebungen übertragbar?

Überfragbarkeit
Installierbarkeit
Austauschbarkeit 10

Safety ISO-Std

Sind Personen, Tiere, Sachen oder Umwelt vor Schäden durch die Software geschützt?

Betriebssicherheit
Verifizierbarkeit
Regulatorische Compliance 11

Wartbarkeit ISO-Std

Ist die Software leicht zu ändern, erweitern, testen, verstehen? Lassen sich Teile wiederverwenden?

Analysierbarkeit
Änderbarkeit
Erweiterbarkeit 13

Zuverlässigkeit ISO-Std

Ist das System verfügbar, tolerant gegenüber Fehlern, nach Abstürzen schnell wieder hergestellt?

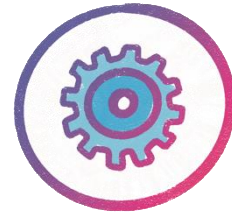
Verfügbarkeit
Wiederherstellbarkeit
Fehlertoleranz 14

Kandidaten

Übung.

Qualitätsziele ermitteln

Einigt euch in eurer Gruppe auf ein Software-System. Ermittelt mit Hilfe von Top-5-Challenger die aus eurer Sicht wichtigsten Qualitätsziele.



Geforderte Ergebnisse

- Ein **Softwaresystem**, auf das ihr euch in der Gruppe geeinigt habt.
- Die Top 3 bis maximal 5 wichtigsten **Qualitätsziele** für dieses System
- **Bonus:** Konkretisierung und Diskussion, wie „fordernd“ die Ziele sind



Tipps & Hinweise

- Neben den Spielkarten erhaltet ihr auch auch detaillierte „Arbeitsanweisungen“ auf einem DIN-A4-Blatt.



Details



Material pro Gruppe

- 14 Karten mit Qualitätsmerkmalen
- DIN-A4-Zettel mit „Arbeitsanweisungen“



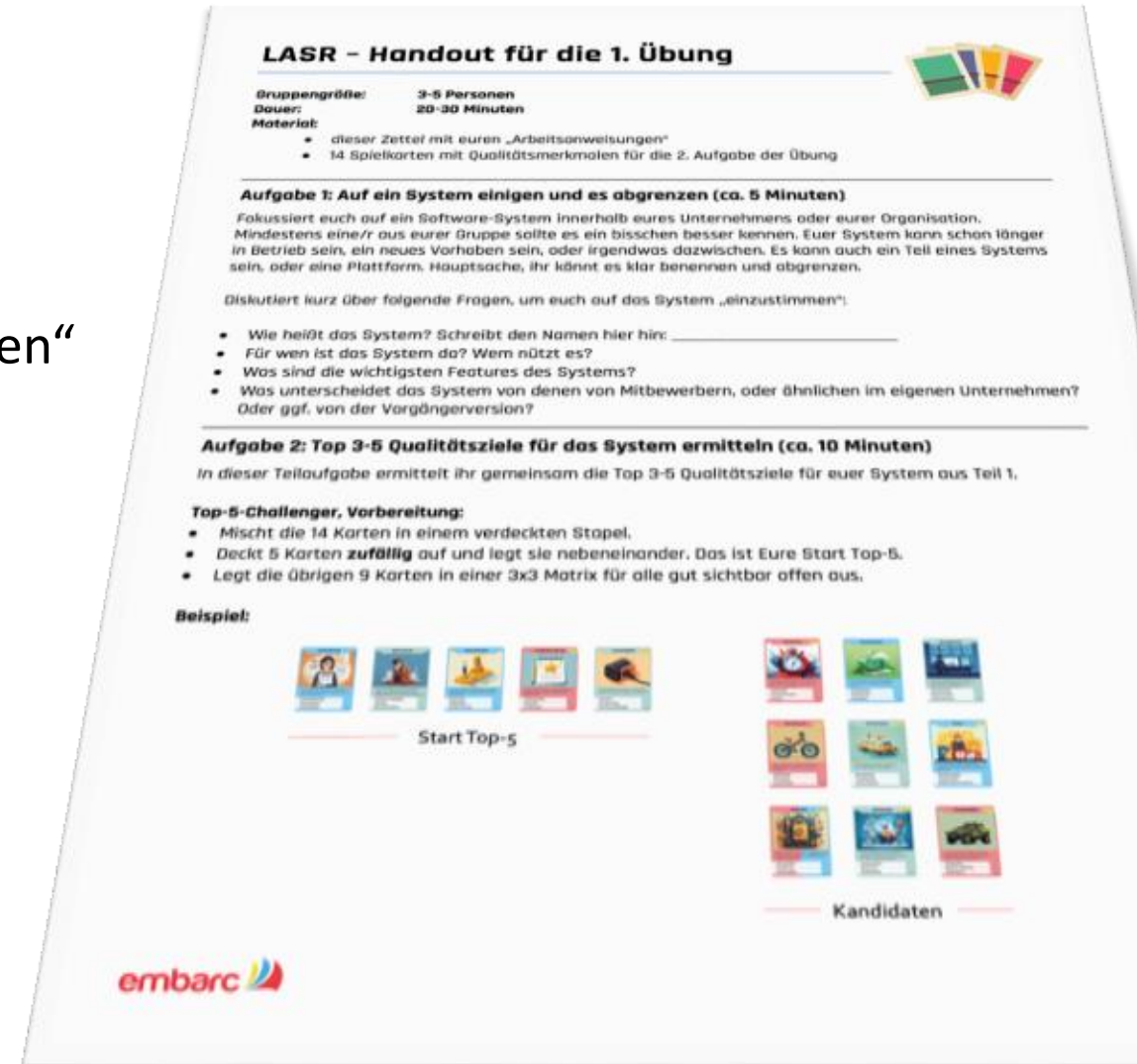
Gruppengröße

- 3-5 Personen
- Ideal ist 4

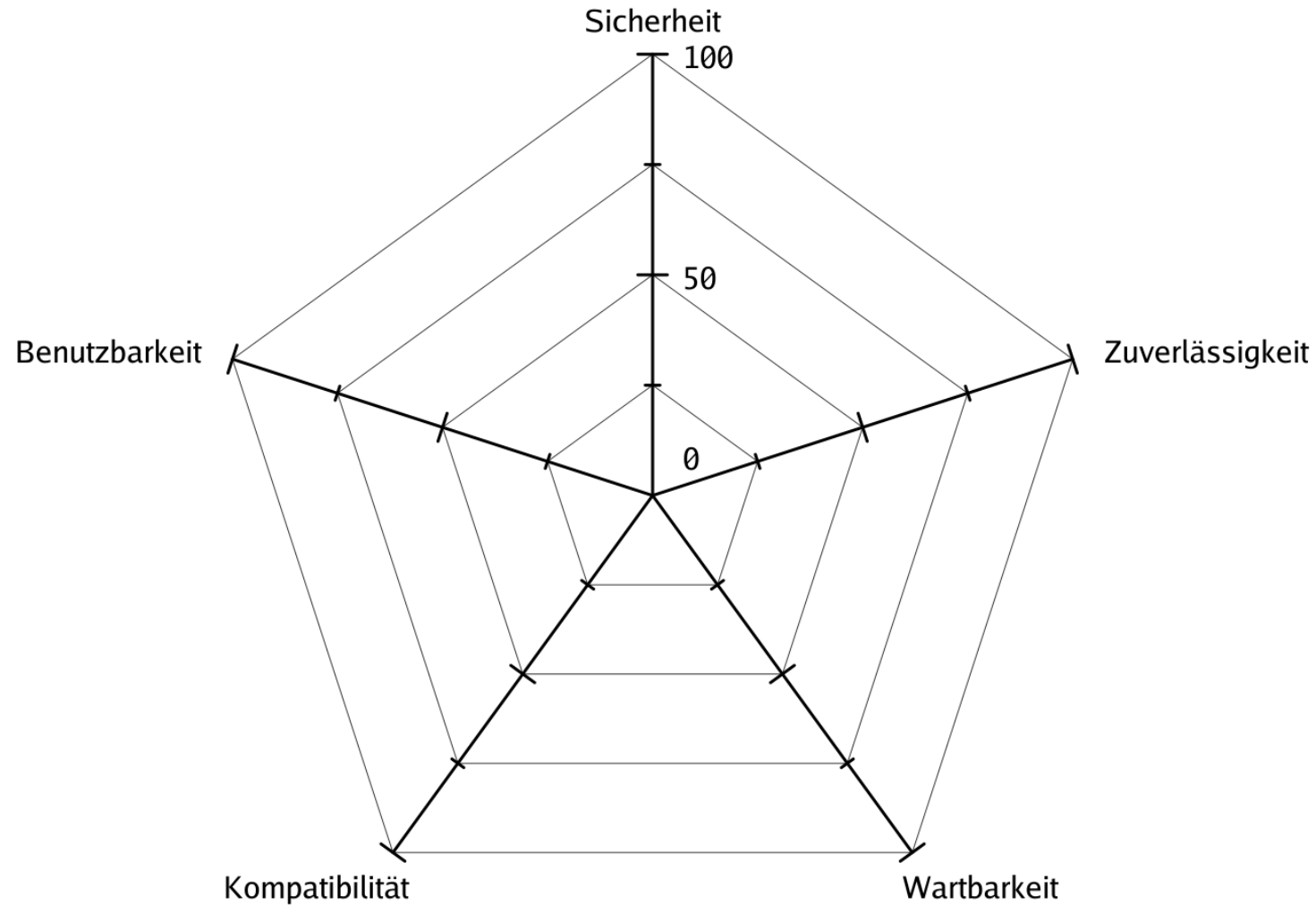


Zeitlicher Rahmen

- 15-20 Minuten

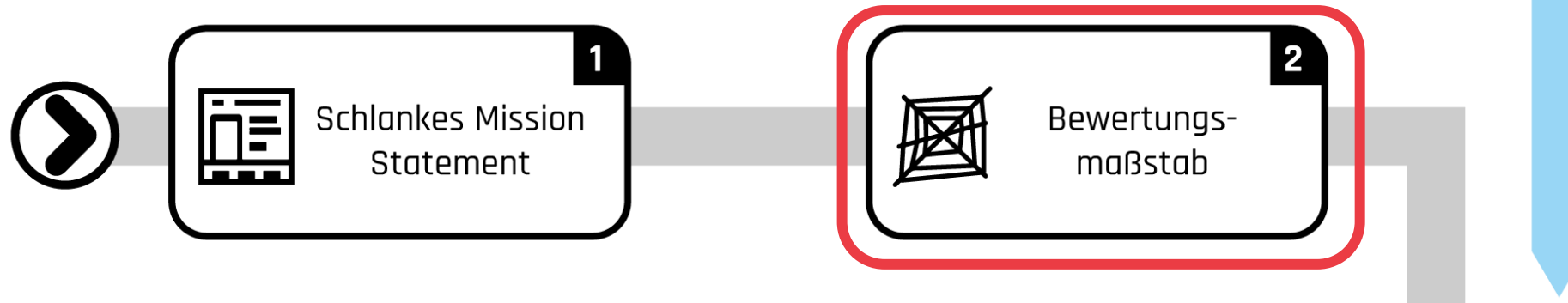


Ergebnis nach diesem Schritt (Beispiel)



Verstehe, was Dich speziell macht

Verstehe was Dich speziell macht

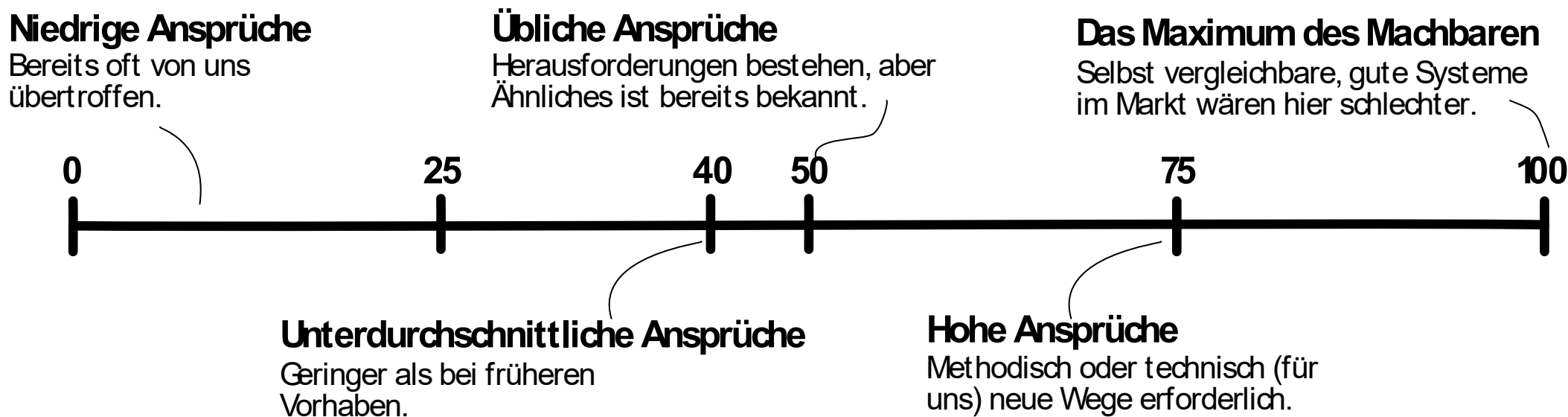


Was ist zu tun?

- Die Aufgabenstellung für das System zu einem prägnanten Mission Statement verdichten
- **Die Top 3-5 Ziele des Systems identifizieren**
- **Jeweiliges Ziel-Level festlegen**

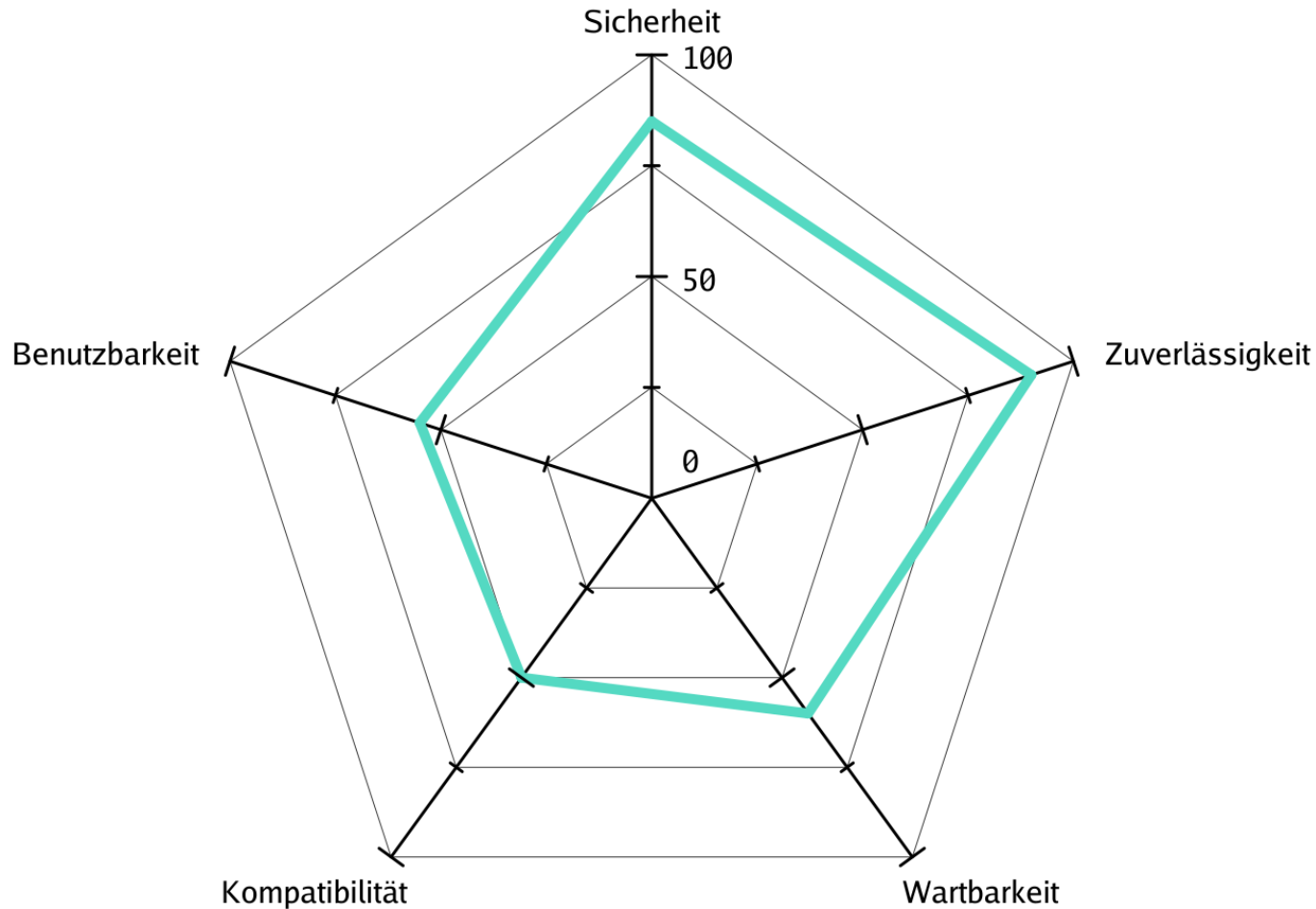
Zielwerte bestimmen

Der Zielwert zwischen 0 und 100 beschreibt jeweils, wie hoch die Erwartungen in dem Bereich sind. Im Vergleich zu anderen Zielen der Top-3-5 und anderen Vorhaben aus dem Kontext des Unternehmens / der Organisation.



Hinweis: Es handelt sich hier um eine Einschätzung, nichts Messbares.

Bewertungsmaßstab einzeichnen



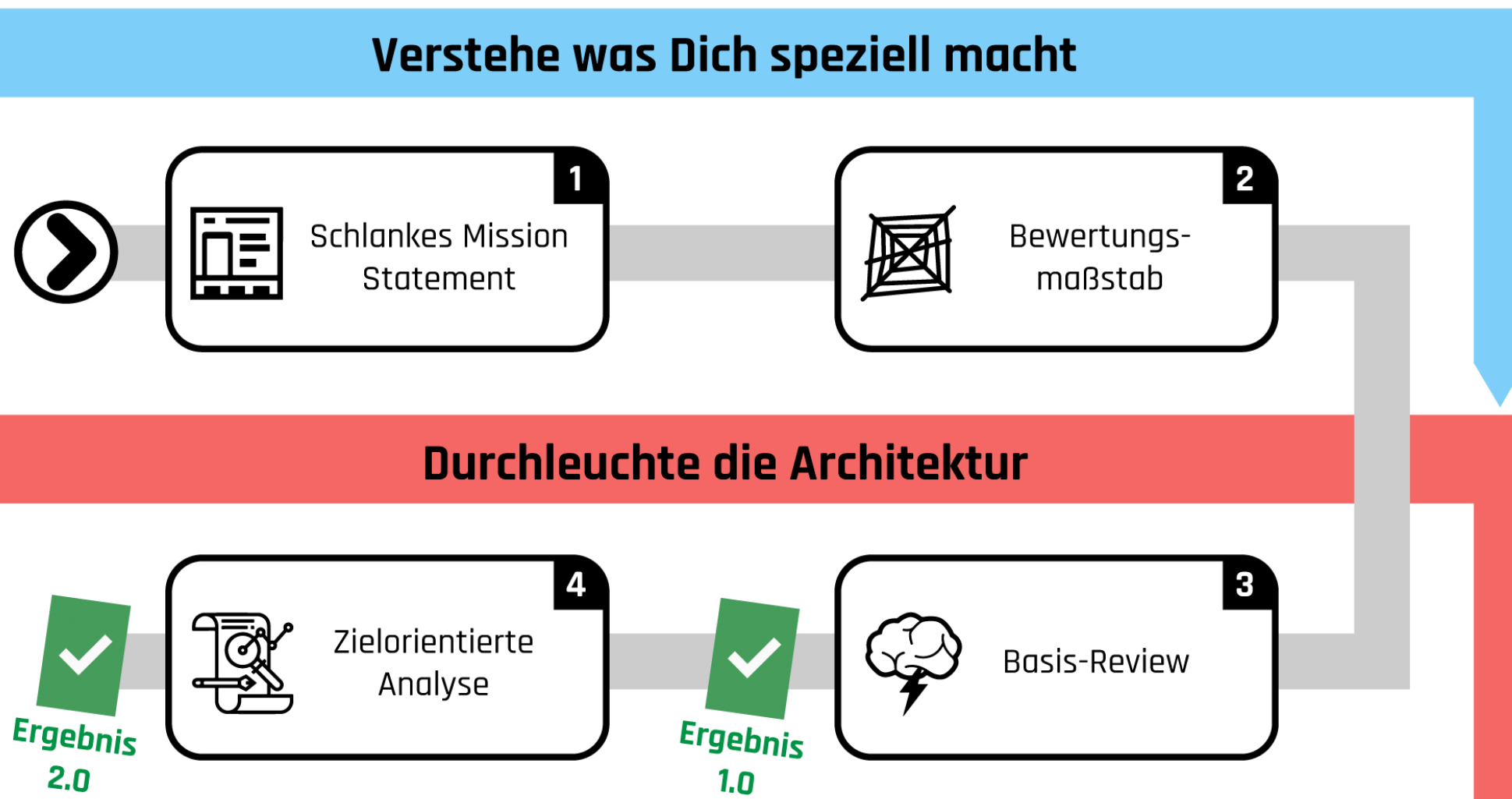
LASR-Ergebnisdiagramm

Die Top-3-5 Qualitätsziele bilden die Achsen des Diagramms.

Die Zielwerte spannen darauf eine grüne Linie auf.

← Beispiel

Mit Schritt 2 steht der Ergebnisrahmen.





Durchleuchte die Architektur

01. Warum leichtgewichtig bewerten?

02. Verstehe, was Dich speziell macht

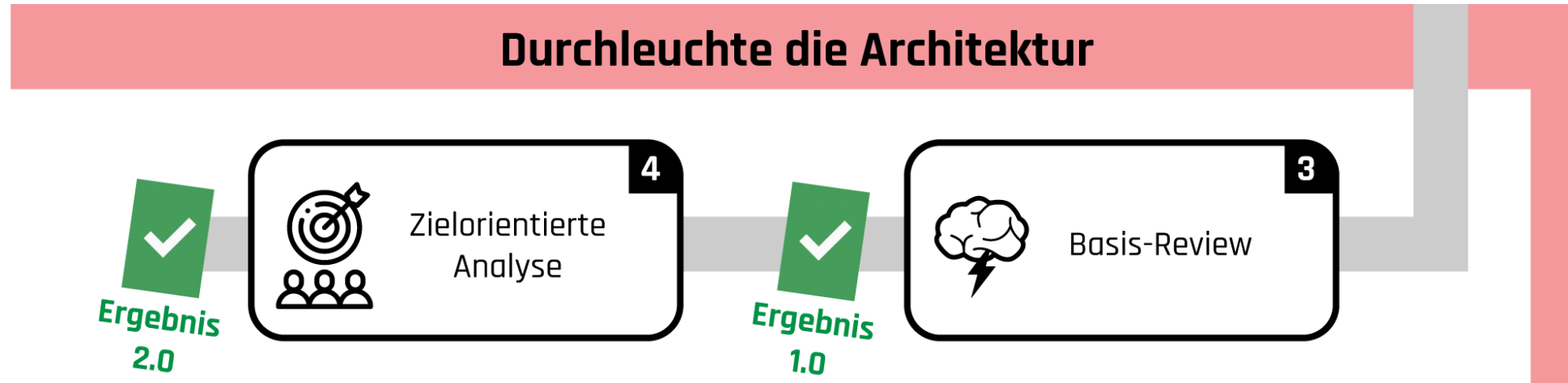
03. Durchleuchte die Architektur

04. Erhöhe die Konfidenz (bei Bedarf)

05. Weitere Informationen



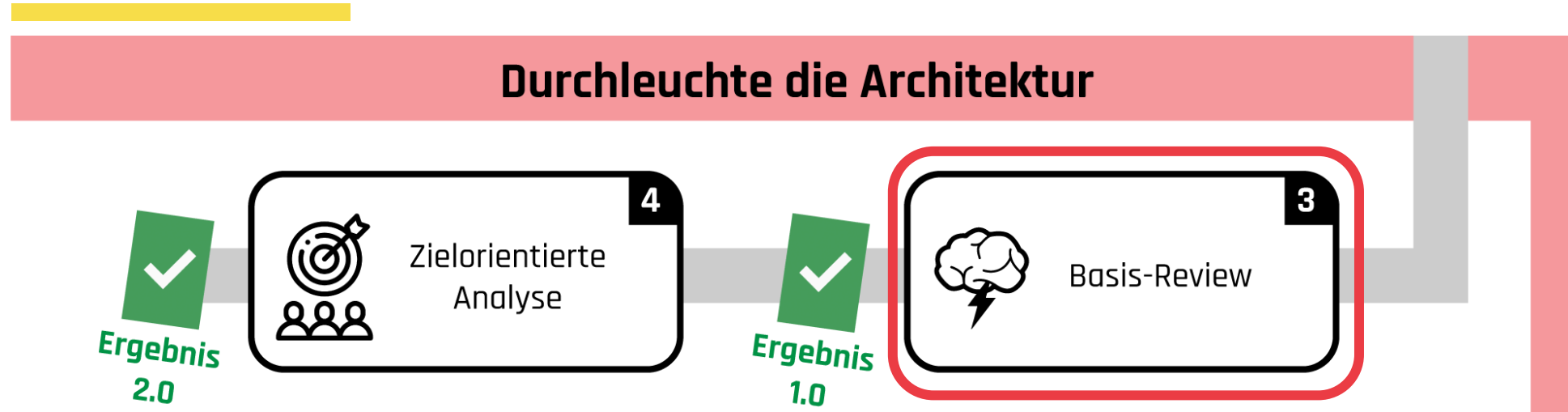
Durchleuchte die Architektur



Was ist zu tun?

- Die größten Risiken des Systems identifizieren
- Die aus den Risiken resultierende Abweichung zu den Zielen quantifizieren („Lücken“)
- Zielorientierte Analysen durchführen, um das Review-Ergebnis bei Bedarf zu schärfen

Durchleuchte die Architektur



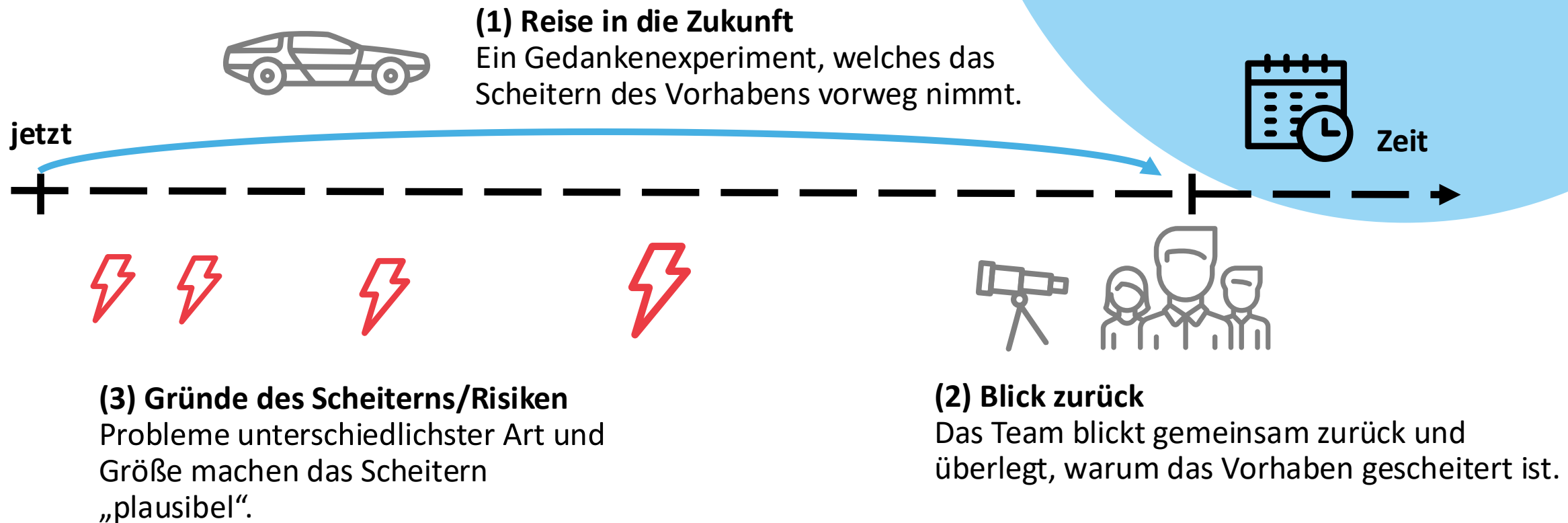
Was ist zu tun?

- Die größten Risiken des Systems identifizieren
- Die aus den Risiken resultierende Abweichung zu den Zielen quantifizieren („Lücken“)
- Zielorientierte Analysen durchführen, um das Review-Ergebnis bei Bedarf zu schärfen

Grundidee: Pre Mortem



Was ist ein Pre-Mortem?



- Pre-Mortems werfen einen kritischen Blick auf Vorhaben. Und zwar aus einer hypothetischen Zukunft, in der das Vorhaben gescheitert ist.
- Aus dieser Perspektive ergeben sich „Gründe des Scheiterns“, die aus heutiger Sicht mögliche Risiken darstellen.

Die 8 Risiko-Kategorien von LASR



Softwarelösung 1

- Unpassende Technologien
- Komplexe Lösungen
- Unausgereifte Fremdlösungen
- Behindernde Frameworks / Libraries
- Strukturprobleme
- ...



Kompetenz und Erfahrung 2

- Fehlendes oder isoliertes Wissen
- Zu kleine Lernfenster
- Fehlendes Prozessverständnis
- Tool-Probleme
- Schwere Schätzbarkeit
- ...



Zielsetzungen und Erwartungen 3

- Unrealistische Ziele
- Überzogene Erwartungen
- Knappe Deadlines
- Instabile Anforderungen
- Fehlender Kundenkontakt
- ...



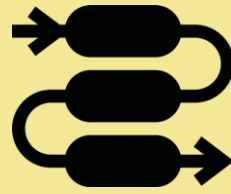
Fremdsysteme und Plattformen 4

- Instabile Plattformen
- Fehleranfällige Fremdsysteme
- Unpassende Buy/ Take Wahl
- Lizenzschwierigkeiten
- Vendor Lock-In
- Integrationsprobleme
- ...



Altsysteme und Atlanten 5

- Legacy-Behinderungen
- Innovationsstau
- Zerbrechliche Lösungen
- Fehlende Tests
- Wenig Dokumentation
- Fehlendes Verständnis
- ...



Organisation und Prozesse 6

- Geschwollene Prozesse
- Behindernde Rollenmodelle
- Organisationsgrenzen
- Politik
- Engernde Standards
- Isolierte Entscheidungskompetenzen
- ...



Betrieb und Deployment 7

- Blockierende Prozesse
- Geringe Automatisierung
- Wenig Feedback
- Fehlende Betriebs- / Ausfallkonzepte
- Tool-Probleme
- ...



Weiche Faktoren 8

- Uneinigkeiten
- Konflikte
- Fehlende Disziplin
- Kommunikationsbarrieren
- Rollenthematiken
- Unpassende Kultur
- ...

Typische Risikothemen als Ideengeber



Zu hohe Komplexität der Lösung

Hat die Domäne eine sehr hohe Komplexität? Gibt es unüberlegte, schnelle Lösungen oder fehlende Abstraktionen? Komplexität gefährdet den Überblick bzw. Wartbarkeit, Korrektheit, Sicherheit, ...

**Softwarelösung**1.1



Unpassende Lösungs-Strukturierung

Folgt die Softwarestruktur der Domäne? Sind andere technische oder organisatorische Einflüsse (Conway...) gut aufgegriffen? Falls nicht könnte Wartbarkeit, Zuverlässigkeit etc. leiden.

**Softwarelösung**1.2



Inadäquates Daten-Handling

Ist die Ablage, der Transport und das Mapping von Daten konzeptionell und technisch passend gelöst? Sind Daten ausreichend schnell und rechtssicher, im richtigen Format an den richtigen Stellen?

**Softwarelösung**1.3



Problematische Konzepte & Technologien

Passen die eingesetzten Muster und Konzepte zu den Zielen? Sind sie konsistent angewendet? Sind die verwendeten Technologien und Frameworks passend und etabliert?

**Softwarelösung**1.4



Softwarelösung

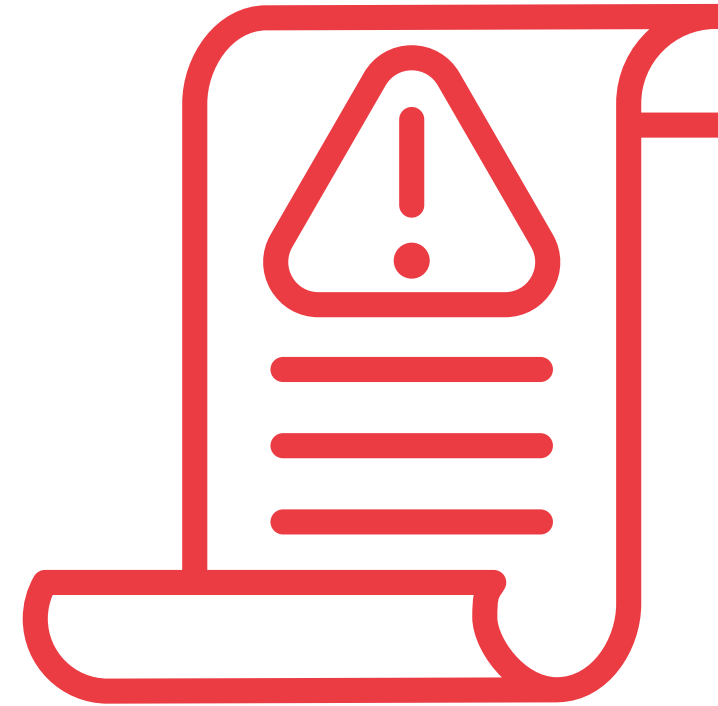
1

- Unpassende Technologien
- Komplexe Lösungen
- Unausgereifte Fremdlösungen
- Behindernde Frameworks / ...
- Strukturprobleme
- ...

Brainstorming-Unterstützung: Das LASR-Kartenset hält insgesamt 32 konkrete Risikokarten als Ideengeber parat, 4 in jeder der 8 Kategorien.

Disclaimer

- Die folgenden **Beispiel-Risiken** sind **realistisch** für Dinge, die Team-Mitglieder im Rahmen eines Risiko-Workshops im Pre-Mortem aufwerfen.
- Bezogen auf unser Beispiel **Threema** hier sind sie **rein fiktiv**. Sie dienen der Illustration.



Beispielkarte



Zu hohe Ziele oder Erwartungen

Sind Qualitätsziele (z.B. Performanz) zu ambitioniert? Stehen hohe Einzelziele guter Gesamtqualität im Weg? Werden Randbedingungen verletzt oder unnötige technische Risiken eingegangen?



Zielsetzung

Hypothetisches Risiko (1)



Zu hohe Ziele oder Erwartungen

Sind Qualitätsziele (z.B. Performanz) zu ambitioniert? Stehen hohe Einzelziele guter Gesamtqualität im Weg? Werden Randbedingungen verletzt oder unnötige technische Risiken eingegangen?



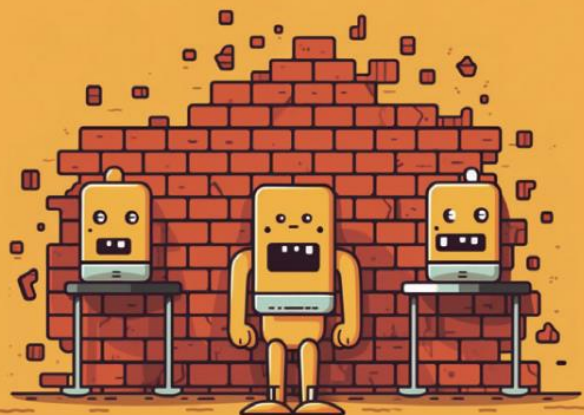
Zielsetzungen & Erwartungen

3.1

Aufhebung des Gruppen-Limits

„Wir haben die Begrenzung in Gruppen-Größen aufgehoben. Unsere User sind begeistert von den neuen Möglichkeiten. Später bricht der Chat-Server im Threema-Backend unter der massiven Last und der Menge zu speichernden Nachrichten zusammen.“

Hypothetisches Risiko (2)



Vendor-Lock-In oder Support-Probleme

Gibt es Abhängigkeiten von Lieferanten, deren Ziele oder Einsatzzwecke abweichen? Stehen Abkündigungen im Raum? Gibt es potentielle Probleme bei Support, Lizenzmodellen, Kosten, ...?



Fremdsysteme & Plattformen

4.4

Das Electron-Framework fällt weg

„Wir waren gezwungen das Electron-Framework in unserer Desktop-App durch eine Alternative / etwas Selbstgebautes zu ersetzen. Das zieht sich. Zwischenzeitlich haben wir den Download des Desktop-Clients von der Seite genommen, weil er auf aktuellen OS nicht mehr funktioniert.“

Hypothetisches Risiko (3)



© embarc GmbH

Einengende Standards oder Randbedingungen

Werden Architekturentscheidungen (oft) durch Standards, Budget- oder Zeitbeschränkungen erschwert? Gibt es einschneidende rechtliche Aspekte rund um Daten oder Internationalisierung?



Organisation & Prozesse

6.4

Private Kommunikation scannen

„Die europäische Gesetzgebung hat Messenger-Anbieter verpflichtet, sämtliche private Kommunikation und Dateien zu durchleuchten. Threema muss daraufhin die Lösung für private Nutzer komplett neu denken.“





Fehlendes Lösungsverständnis

Fehlen Wissen, Ansprechpartner und/oder Dokumentation für Teile des Systems?
Sitzten Ansprechpartner an Bottlenecks?
Sind zentrale Lösungen tw. wenig nachvollziehbar oder "historisch gewachsen"?

 **Altsysteme und Altlasten**

53

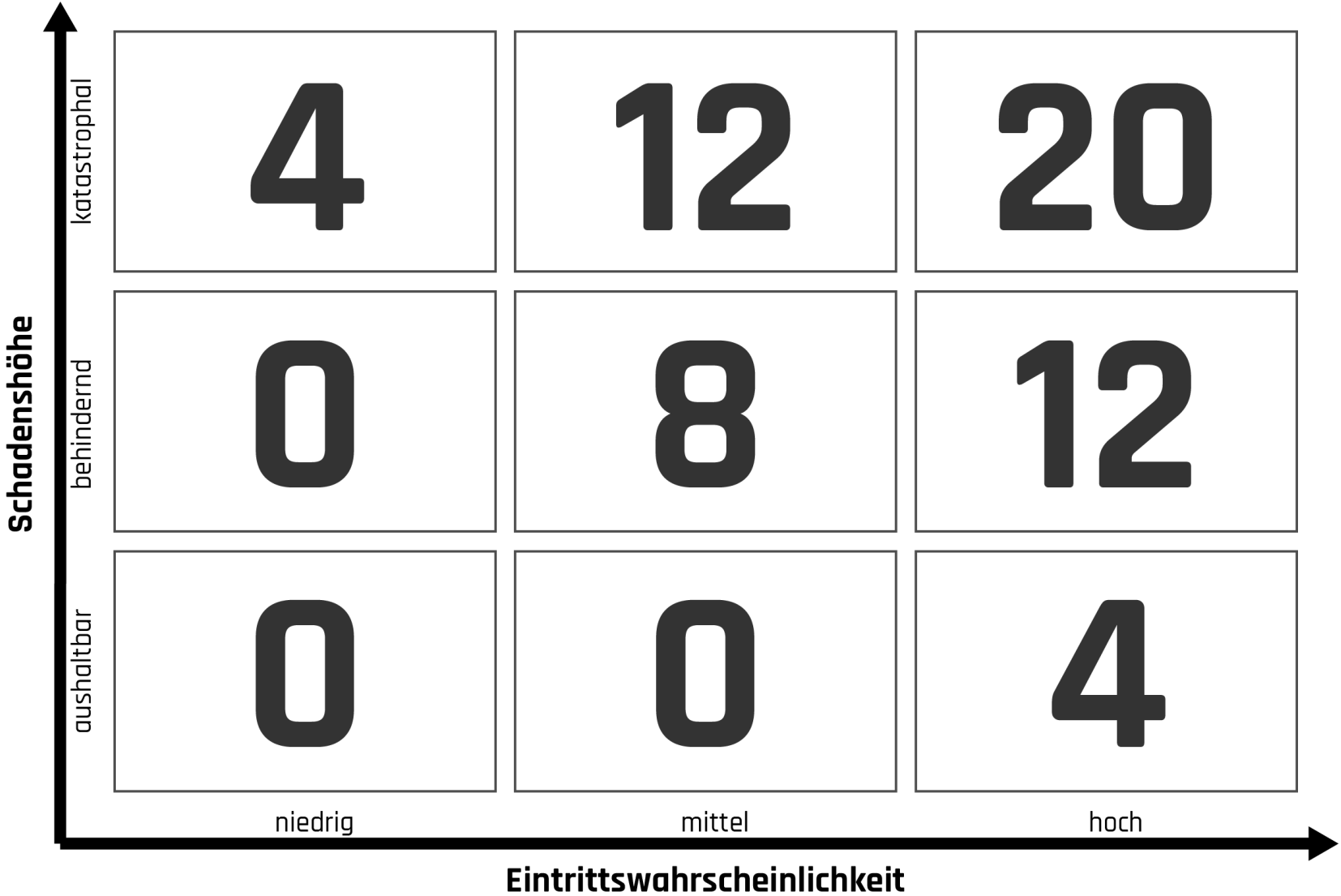
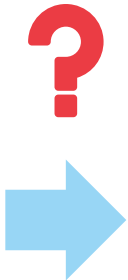
Risiken einordnen



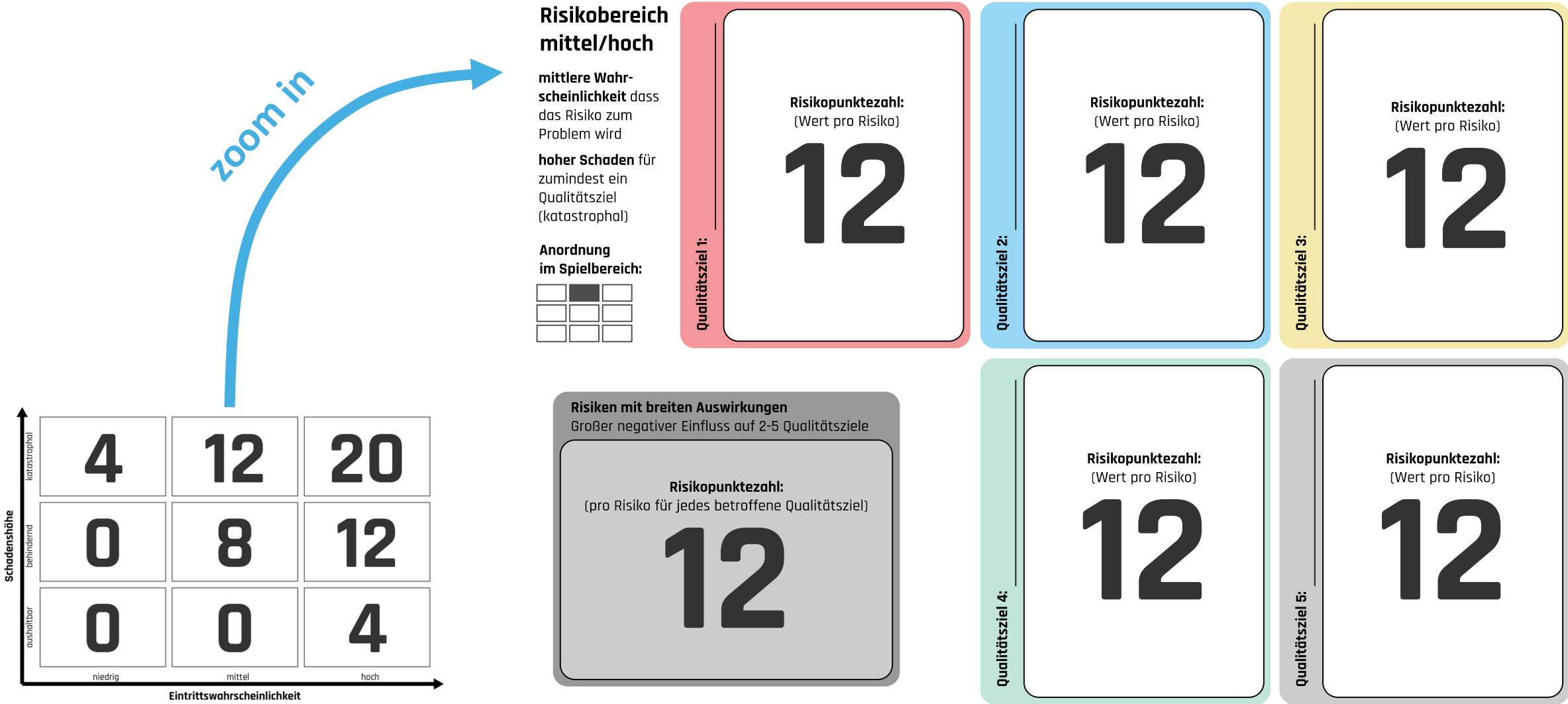
Falsche Konzept- / Musteranwendung

Auch anerkannte Muster und Konzepte können in manchen Kontexten nachteilig sein. Passen die Konzepte zu den Zielen? Sind sie konsistent angewendet? Falls nicht, leidet nicht nur die Verständlichkeit.

 Softwarelösung 1.1



Den betroffenen Zielen zuordnen



Einhalten
wahrscheinlich

Adap-
tations-
strategie kann
nicht mehr
ausgewendet werden

Parasiten trifft
den Bus

FB
steht nicht mehr
mit dem digitalen Bereich
"Kunden" haben die Auswahl
nicht mehr

kein Unterstützung von
Angular durch Google

Rechenkanalnutzung
kann nicht mehr umgesetzt
werden

Neue Betriebsplattform
wird eingeführt

Domänenwissen
müht sich
Laiemal arbeitet
an neuen Projekt
man da durch werden
technische Schulden
aufgebaut

Wendel der...
große Menge an
technischen Schulden
→ Versionsupdates
→ Entwickler

Risikobereich niedrig/hoch

Risikopunktzahl: (Wert pro Risiko) 4

Anordnung im Spielbereich:

Risikobereich mittel/hoch

Risikopunktzahl: (Wert pro Risiko) 12

Risikobereich hoch/hoch

Risikopunktzahl: (Wert pro Risiko) 20

Risikobereich niedrig/mittel

Risikopunktzahl: (Wert pro Risiko) 0

Risikobereich mittel/mittel

Risikopunktzahl: (Wert pro Risiko) 8

Risikobereich hoch/mittel

Risikopunktzahl: (Wert pro Risiko) 12

Risikobereich mittel/niedrig

Risikopunktzahl: (Wert pro Risiko) 0

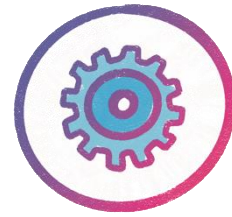
Risikobereich hoch/niedrig

Risikopunktzahl: (Wert pro Risiko) 4

Übung.

Risikothemen und Risiken ermitteln

Identifiziert für euer Softwaresystem aus der letzten Übung Risiko-Kategorien und konkrete Risiken mit Hilfe der LASR-Karten.



Geforderte Ergebnisse

- 3-4 **Themen**, in denen ihr in eurem Softwaresystem Risiken vermutet
- Konkrete **Risiken**, notiert auf Post-its
- **Einschätzung** der Risiken nach Eintritt und Schaden.



Tipps & Hinweise

- Ihr erhaltet neben den Spielkarten auch wieder detaillierte „Arbeitsanweisungen“ auf einem DIN-A4-Blatt.



Details



Material pro Gruppe

- 8 Karten mit Risiko-Kategorien
- 32 Karten mit Standard-Risiken
- Spielplan zum Zusammenbauen
- DIN-A4-Zettel mit „Arbeitsanweisungen“



Gruppengröße

- 3-5 Personen (gleiche Gruppen)
- Ideal ist 4



Zeitlicher Rahmen

- 30-40 Minuten

LASR - Handout für die 2. Übung



Gruppengröße: 3-5 Personen (Ideal: 4)
Dauer: 30-40 Minuten
Material:

- dieser Zettel mit euren „Arbeitsanweisungen“
- 8 Spielkarten mit Risiko-Kategorien (graue Rückseite) für die 1. Aufgabe der Übung
- 32 Spielkarten mit Standardrisiken für die 2. Aufgabe der Übung
- Post-its, Stift und Klebepunkte

Bei dieser Übung arbeitet ihr weiter mit eurem System aus der 1. Übung und den dort erhobenen Top 3-5 Qualitätszielen.

Tipp: Legt die betreffenden Qualitätsziel-Karten aus der 1. Übung gerne offen aus, um Euch daran zu erinnern. Eine/r aus eurer Gruppe sollte sie noch haben (und auch ein Foto mit dem Ergebnis).

Aufgabe 1: Wichtige Risiko-Kategorien ermitteln (ca. 10 Minuten)

In dieser Teilaufgabe ermittelt ihr gemeinsam die Themenbereiche, bei denen ihr Risiken für euer System vermutet.

Vorbereitung:

- Mischt die 8 Karten mit den Risiko-Kategorien (graue Rückseite) gut durch.
- Verteilt sie geheim und gleichmäßig in eurer Gruppe (so gut es geht, bei 4 Mitgliedern erhält jeder 2).
 - Sonderfall a) Bei 3 Leuten ist die Verteilung (2, 3, 3)
 - Sonderfall b) Bei 5 Leuten bilden 2 ein Team und ihr verteilt: 2, 2, 2, und 2
- Behaltet die Karten für euch geheim.

Ablauf („Draft“)

1. Schaut Euch die Kategorie-Karten auf eurer Hand an. Bei welchem Thema vermutet ihr am ehesten gravierende Risiken bezüglich der Erreichung der Top-Qualitätsziele eures Systems. Diese Karte behaltet ihr auf der Hand.
2. Die andere(n) Karte(n) gebt im Uhrzeiger weiter. Ihr habt im Anschluss eine alte und 1-2 neue Karten in der Hand (2 neue, falls eure Nachbarin 3 Karten hatte)
3. Wiederholt Schritt 1. Vielleicht wollt ihr die neue Karte (oder eine davon) auf der Hand behalten. Vielleicht bleibt ihr aber auch bei eurer Karte aus Schritt 1.
4. Die andere/n Karte/n gebt ihr wieder im Uhrzeigersinn weiter.
5. Wiederholt Schritt 1 ein letztes Mal.
6. Legt nun die Kategorie-Karte, die ihr zuletzt behalten habt, offen vor Euch ab.

Je nach Gruppengröße habt ihr nun 3-4 Kategorien identifiziert, in denen ihr vor allem Risiken in eurer Software-Lösung vermutet.

Beispielsergebnis:






embarc

Einheitlich
wahrscheinlich

Aura-kompart-
Schnittstelle kann
nicht mehr
angeteilt werden

Risikobereich
mittelhoch

unvollständiger
schwierig beschaffbar durch
den hohen
Prozentsatz von
hoher Schaden
auszuweisen
Quantität
von
Angebot
den Käufer

Parasit trifft
den Bus ☹️

#B spricht nicht mehr
mit den Augen, sondern
"Kunden" hören die Antwort
mit mehr.

Klein Lintfortsburg van
Angular door Google

Rechen kann umsetzung
kann nicht mehr umgesetzt
werden.

Neue Betriebsdatthom
wird eingeführt-

Domänenwissen
müht sich

Einengender Projekt

Zeit mussel arbeitet
an neuen Projekt
und dadurch werden
technische Schulden
aufgebaut.

Domänenwissen
nicht bekommen

Einengender Projekt.

Zeit muss arbeiten
an neuen Projekt
und dadurch werden
technische Schulden
aufgebaut.

Wiederholungsfragen

→

große Menge an
technischen Schulden
→ Wissensupdates
→ Entwickler

Risikopunkte
Risiko für jedes betro

Risikopunktezahl
(Wert pro Risiko)

Wiederholungsfragen

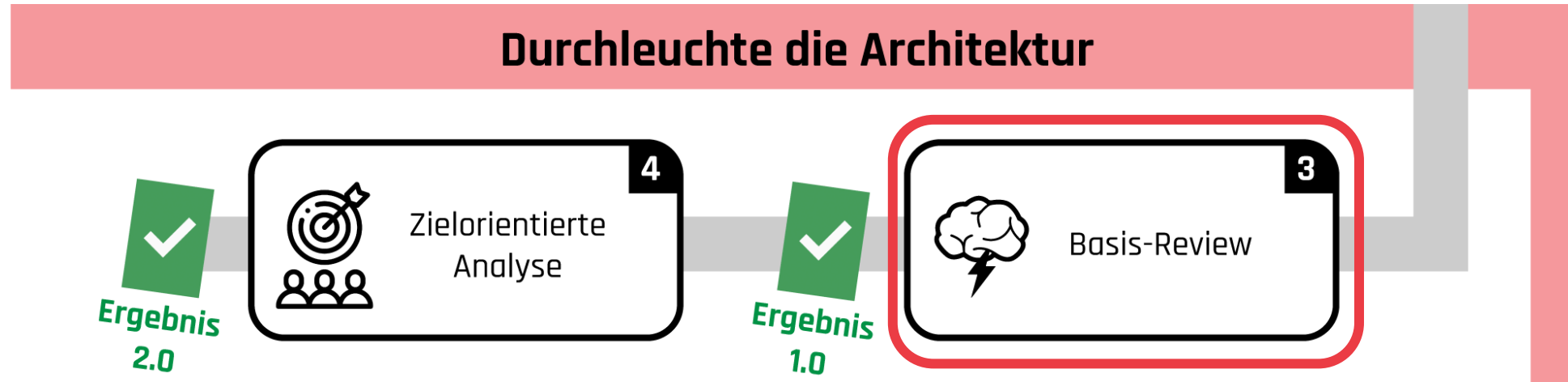
→

große Menge an
technischen Schulden
→ Wissensupdates
→ Entwickler

Risikopunkte
Risiko für jedes betro

Risikopunktezahl
(Wert pro Risiko)

Durchleuchte die Architektur



Was ist zu tun?

- Die größten Risiken des Systems identifizieren
- **Die aus den Risiken resultierende Abweichung zu den Zielen quantifizieren („Lücken“)**
- Zielorientierte Analysen durchführen, um das Review-Ergebnis bei Bedarf zu schärfen

Von Risiken zu „Lücken“

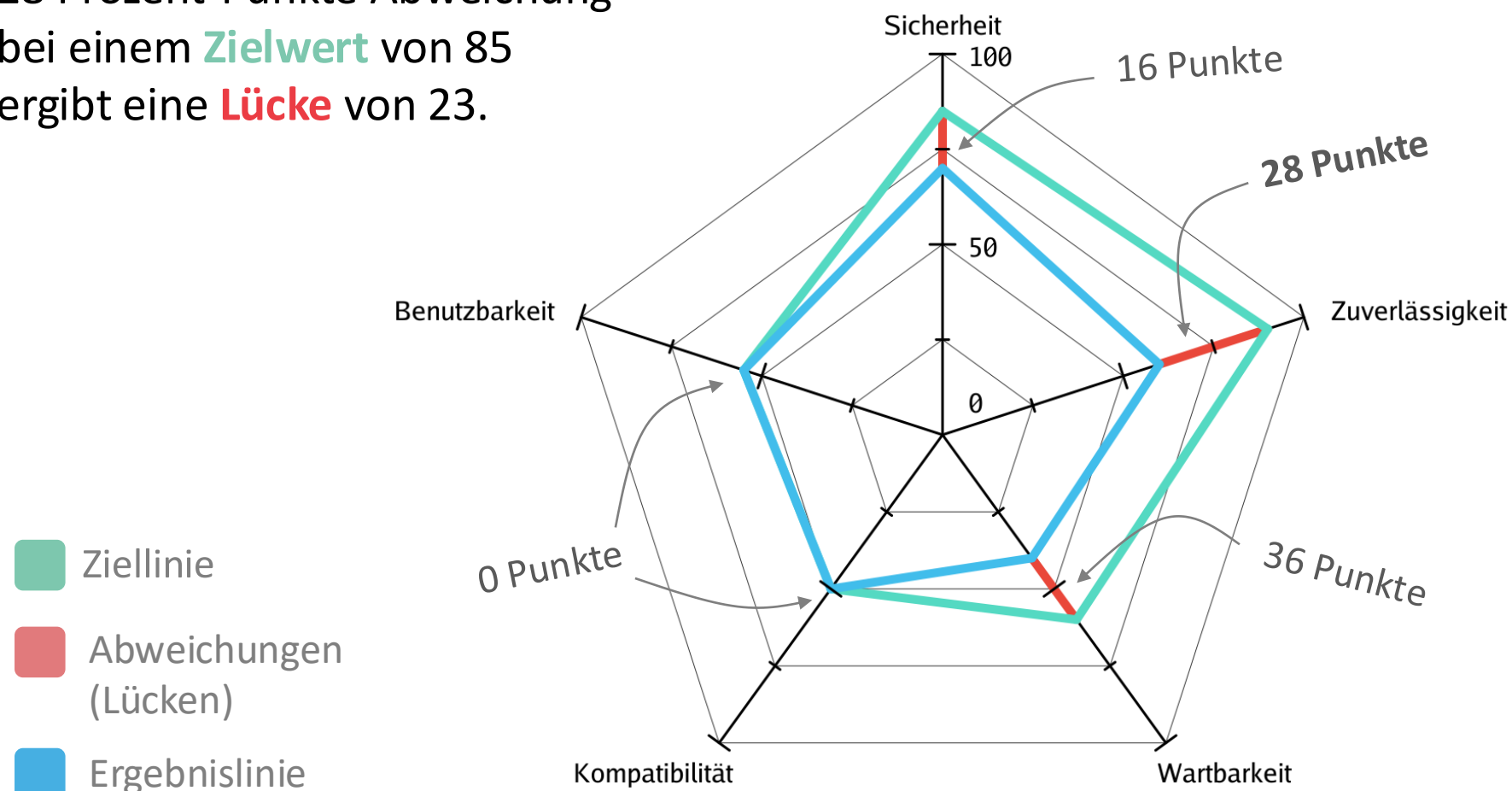


28% Abweichung vom Ziellevel auf der Qualitätszielachse

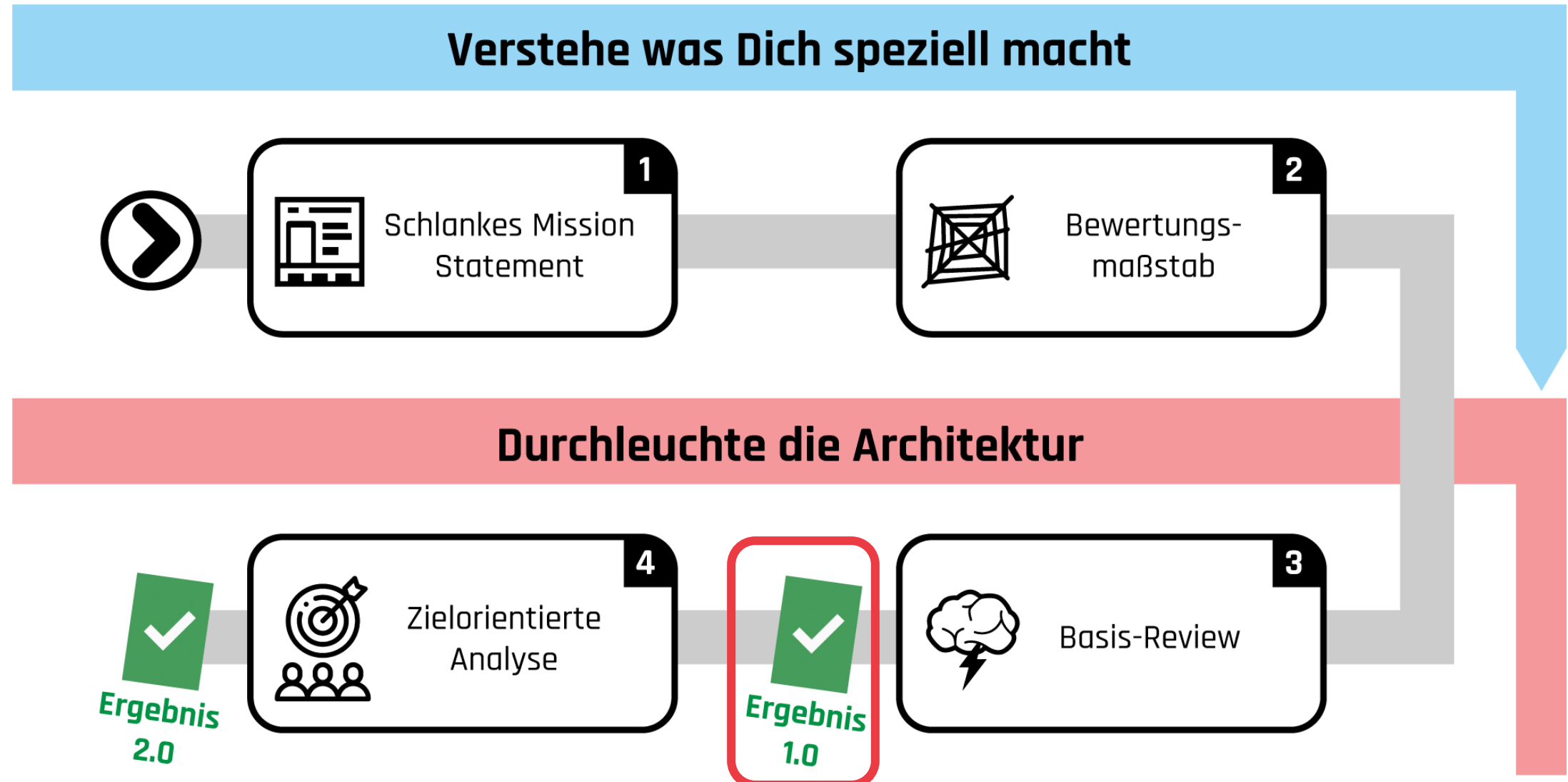
Abweichungen einzeichnen

Beispiel

28 Prozent-Punkte Abweichung
bei einem **Zielwert** von 85
ergibt eine **Lücke** von 23.



Basis-Review, 1. Ergebnis nach Schritt 3



Ergebnis 1.0

LASR Ergebnis 1.0

Symbolhaft dargestellte Elemente des Ergebnisses von LASR nach Schritt 3 (Basis-Review)

Mission Statement

[Ein kurzer Text (oder Slogan) zur gebauten Lösung, der die wichtigsten Merkmale und Eigenschaften der Software vermittelt und zentrale qualitative Anforderungen anklingen lässt - ggf. als Set von „Claims“ aus einem Brainstorming.]

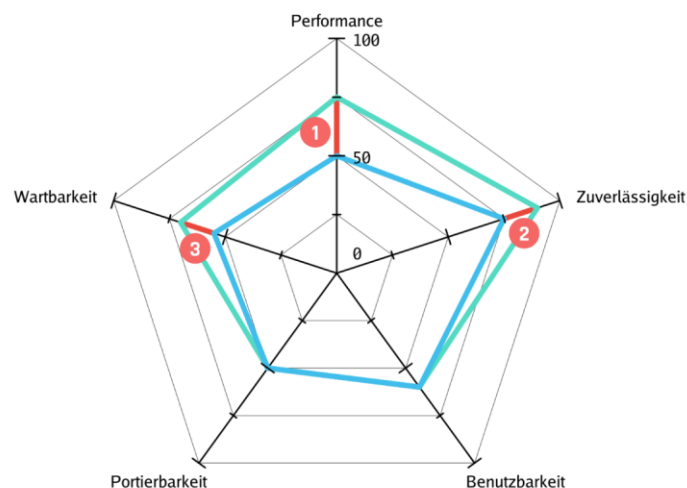
Claim

Claim

Claim

Claim

LASR Ergebnisdiagramm (Spinnennetzgrafik)



Lücken und Risiken

1 [Kurze Lücken-Beschreibung + ausgewählte Details]

Risiko
(2C)

Risiko
(3B)

Risiko
(2B)

Risiko
(1C)

2 [Kurze Lücken-Beschreibung + ausgewählte Details]

3 [Kurze Lücken-Beschreibung + ausgewählte Details]

Stärken und Schwächen

Ansatz (+)	[kurze Beschreibung]

Ansatz (-)	[kurze Beschreibung]

04.

Erhöhe die Konfidenz (bei Bedarf)

01. Warum leichtgewichtig bewerten?

02. Verstehe, was Dich speziell macht

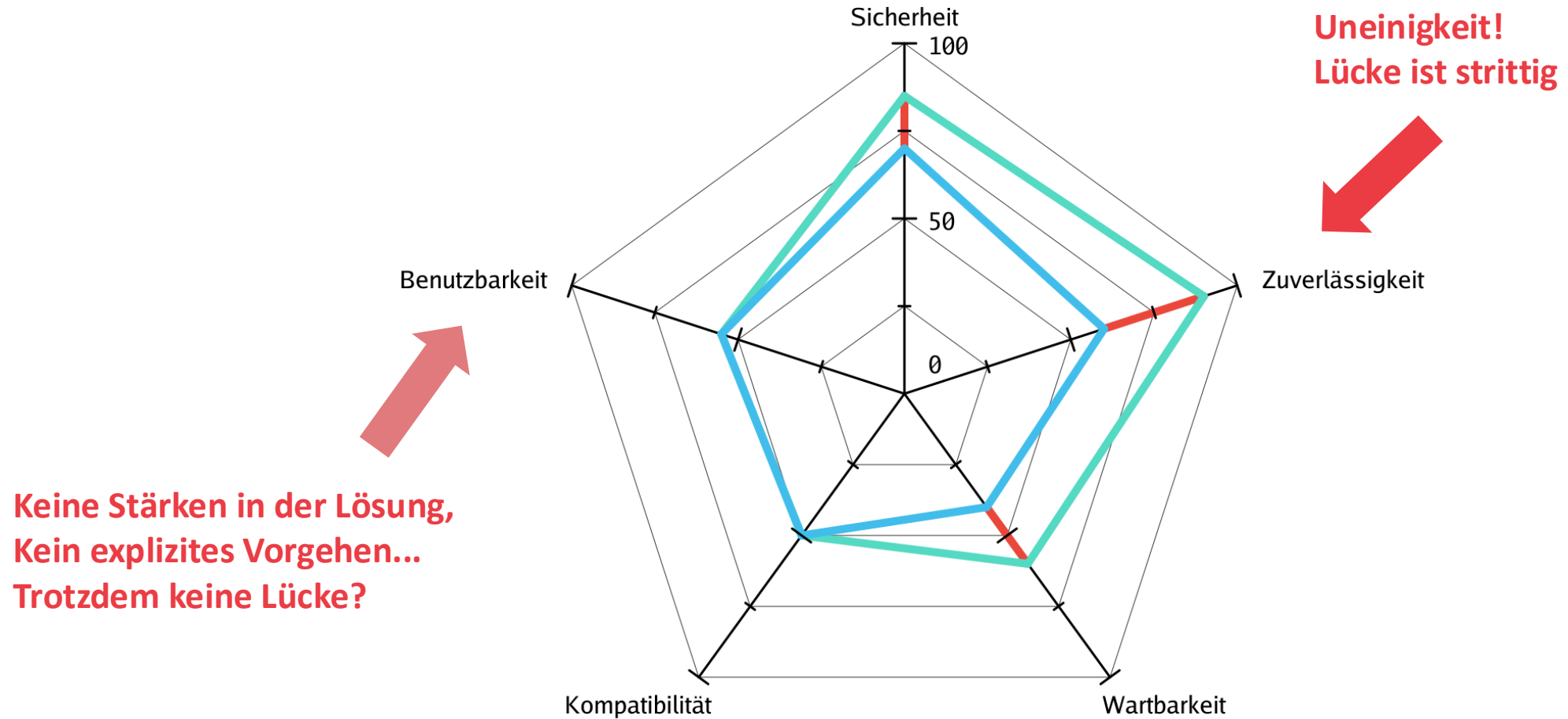
03. Durchleuchte die Architektur

04. Erhöhe die Konfidenz (bei Bedarf)

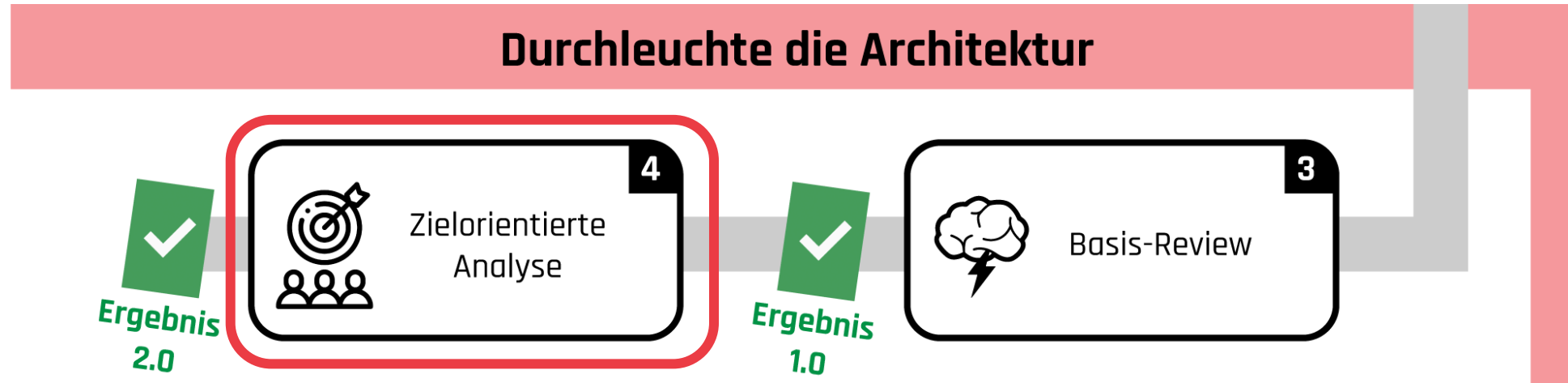
05. Weitere Informationen



Fokus: Zielorientierte Analyse (Beispiele)



Durchleuchte die Architektur



Was ist zu tun?

- Die größten Risiken des Systems identifizieren
- Die aus den Risiken resultierende Abweichung zu den Zielen quantifizieren („Lücken“)
- **Zielorientierte Analysen durchführen, um das Review-Ergebnis bei Bedarf zu schärfen**

Template

Vorlage, um die Ergebnisse der Analyse zu sammeln, zu verdichten und TODOs abzuleiten.

Beispiel →

Zielachse

Lücke (Schritt 3):

36

Lücke geschärft:

20

Zuverlässigkeit ISO-Std

Ist das System verfügbar, tolerant gegenüber Fehlern, nach Abstürzen schnell wieder hergestellt?

Verfügbarkeit
Wiederherstellbarkeit
Fehlertoleranz

Konkrete Risiken

Welche Risiken sind der Zielachse zugeordnet?

Materialdaten wären bei Netzausfall abgeschnitten. Nachgelagerte Probleme?

keine QA betroffen
✓ leichte Mitigation

Verbindungswitch bei Problemen momentan nicht automatisch

keine QA betroffen
✓ leichte Mitigation

SLAs zu Netzwerkverbindung ungeklärt

keine QA betroffen
✓ leichte Mitigation

Erkennung von Ausfall und Ursache ist manuell

keine QA betroffen
✓ leichte Mitigation

Risiko-Einschätzung

1C	2C	3C
1B	2B	3B
1A	2A	3A

Schadenshöhe (Y-Achse)
Eintrittswahrscheinlichkeit (X-Achse)

Qualitätsaussagen

Welche Aussagen illustrieren die Zielachse?

! Mehrstündiger Netzausfall auf AP-Gebiete -> Notlauf trotzdem gewährleistet

H Risiko: H/M/L

! Die Verbindung zum zentralen Server ist unterbrochen. Die Anlage bleibt maximal 60 min unbenutzbar

L H/M/L

Es kommt zu einem Datenverlust. Es ist keine Nutzer-Interaktion erforderlich (Daten: erfasst, protokolliert und Retng.).

L H/M/L

! Zentrale QAs - Qualitätsaussagen

H Technisches Risiko dass QA erreicht wird: High / Medium / Low

Stärken der Lösung

Welche Lösungen bringen uns Zielen näher?
Welche Aspekte erleichtern Umgang mit Risiken?

Message-basierte Kommunikation (async, retry-fähig)

Produktion ohne aktive Verbindung möglich (Arbeitsvorrat bei liegt uns)

Versorgung der V-DB auch asynchron - Daten gehen nicht verloren

Verarbeitungs-komponente kann Jobs ohne Inhalt überbrücken

Schwächen der Lösung

Welche Lösungsaspekte wirken problematisch?
Welche Aspekte verschulden Risiken?

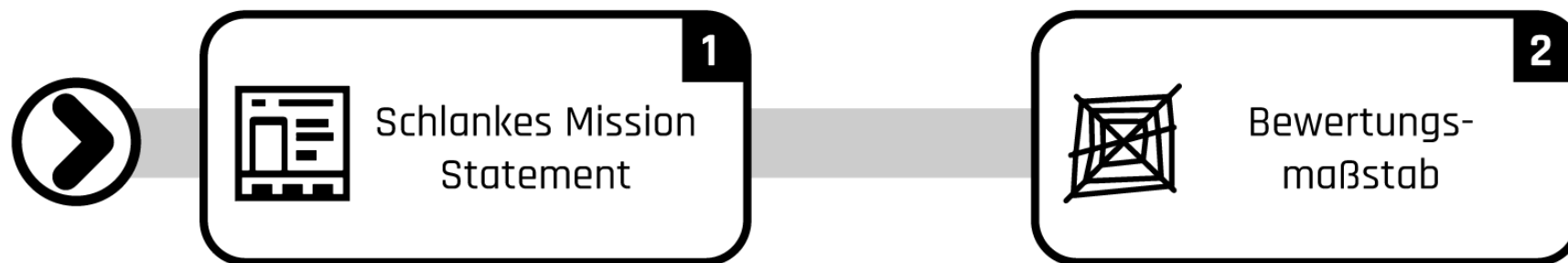
Fehlererkennung über Videos kostet viel Bandbreite

Noch keine Erkennung von Verbindungs-problemen

Datenintensive Verbindung über VPN

Tätigkeiten und Schritte im Kern-Review

Verstehe was Dich speziell macht



Durchleuchte die Architektur





Weitere Informationen

01. Warum leichtgewichtig bewerten?

02. Verstehe, was Dich speziell macht

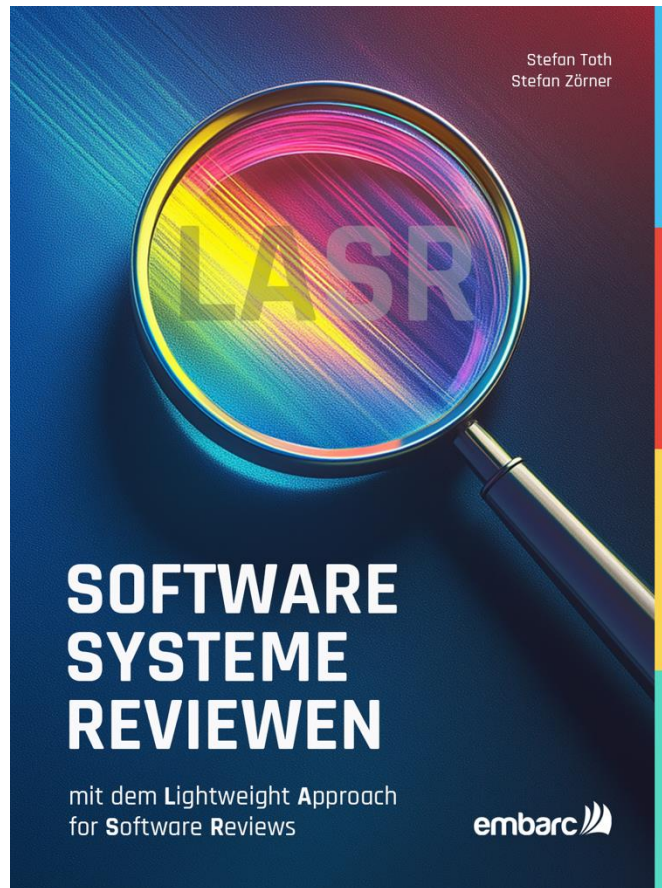
03. Durchleuchte die Architektur

04. Erhöhe die Konfidenz (bei Bedarf)

05. Weitere Informationen



Erfahrungswissen



Software-Systeme reviewen mit dem Lightweight Approach for Software Reviews - LASR

Autoren: Stefan Toth, Stefan Zörner

Sprache: Deutsch, Englisch

Formate: EPUB, PDF, Print (DE)

→ leanpub.com/software-systeme-reviewen/

→ <https://www.amazon.de/Softwaresysteme-Reviewen-Lightweight-Approach-Software/dp/B0HM871M2F>

IEEE Paper zu LASR

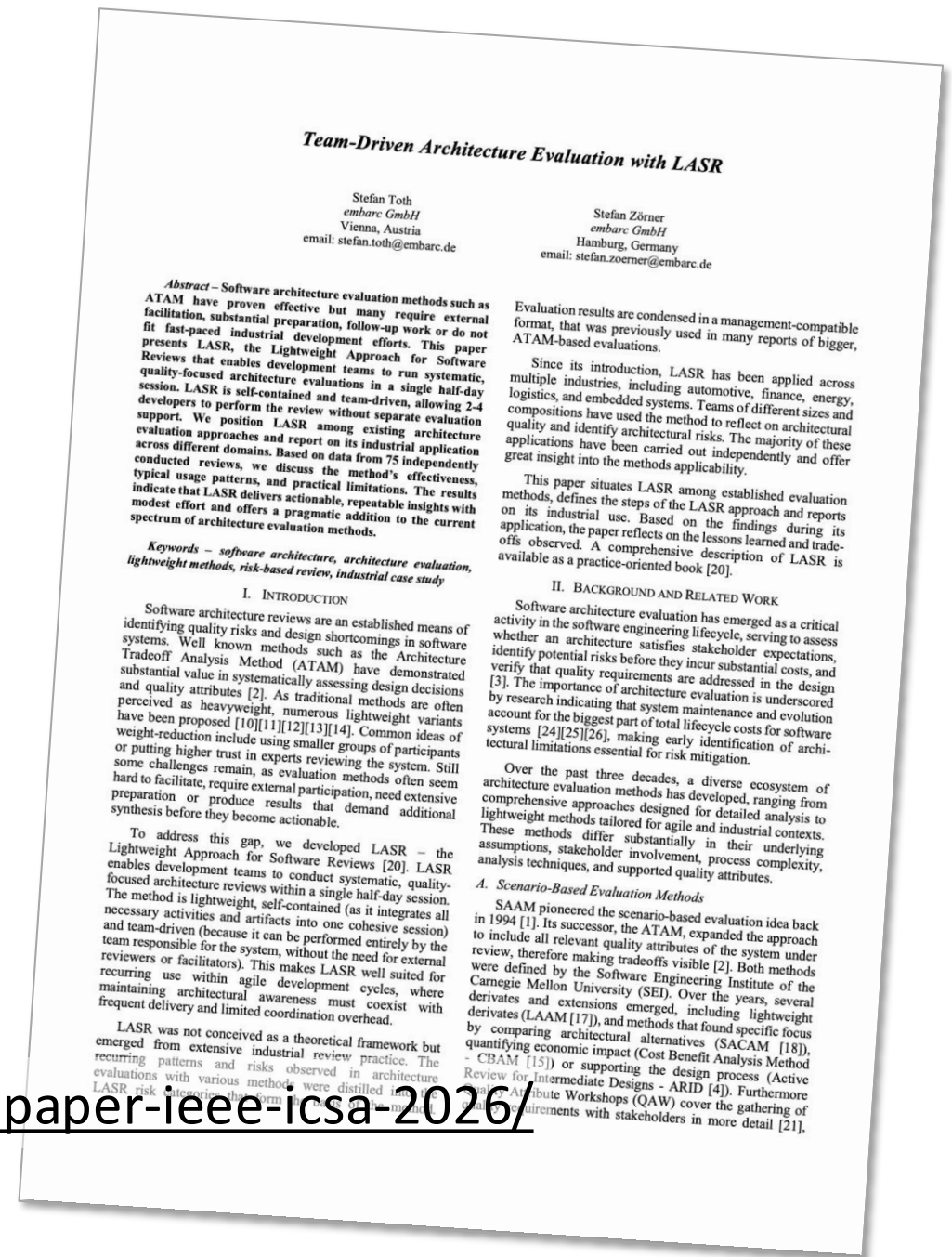
Team-Driven Architecture Evaluation with LASR

IEEE 23rd International Conference on Software Architecture, Amsterdam 2026

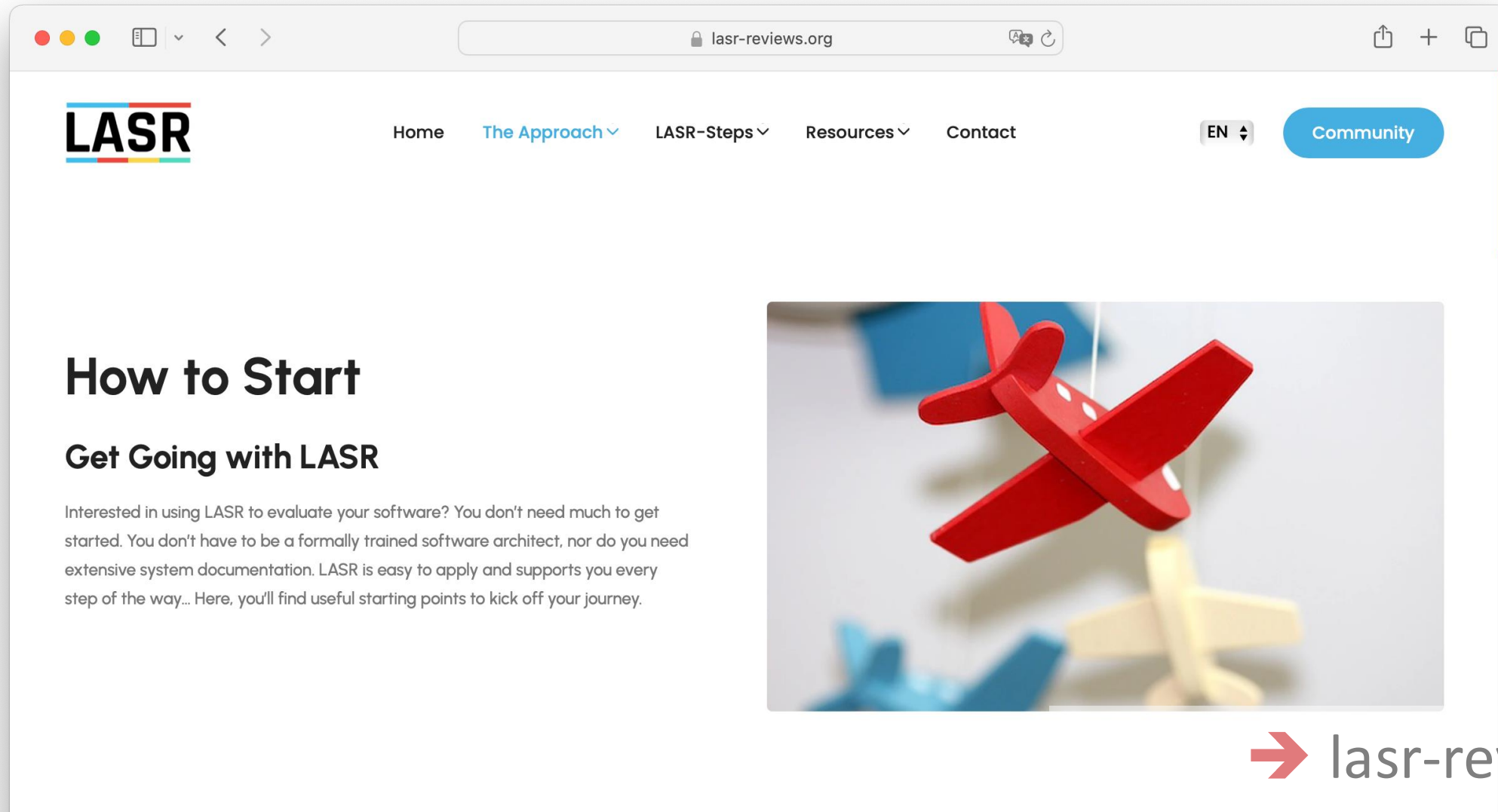
ICSA Distinguished Paper Award



➔ <https://www.lasr-reviews.org/research/lasr-paper-ieee-icsa-2026/>



„Offizielle“ LASR-Webseite (EN + DE)

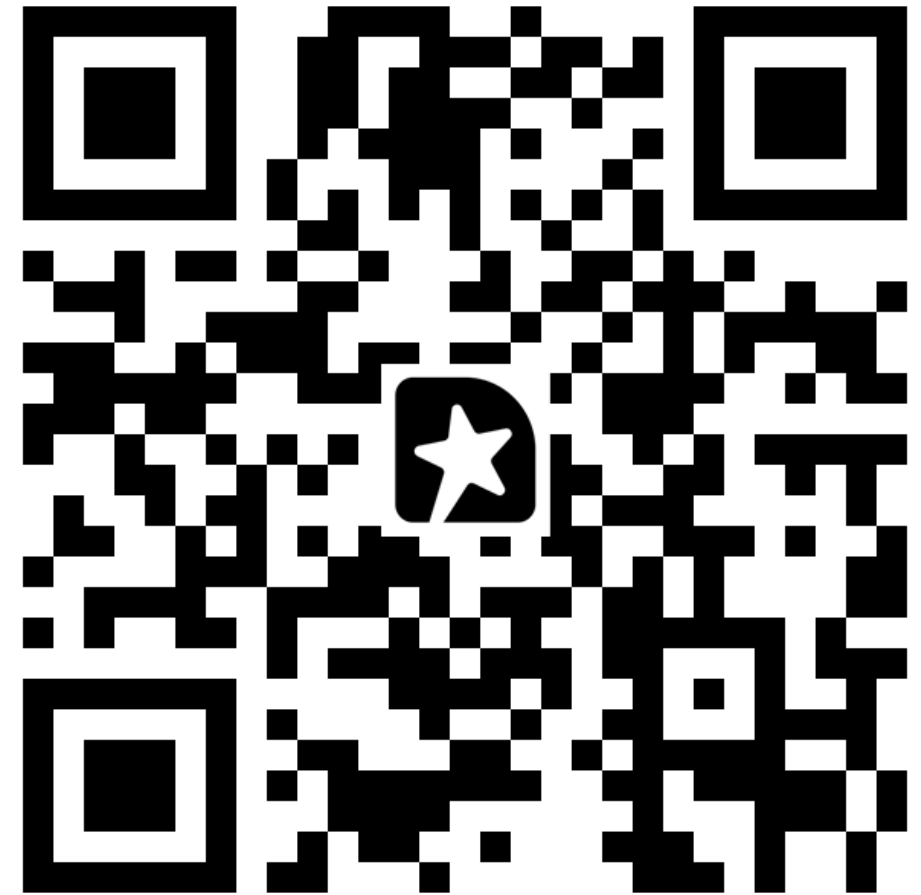
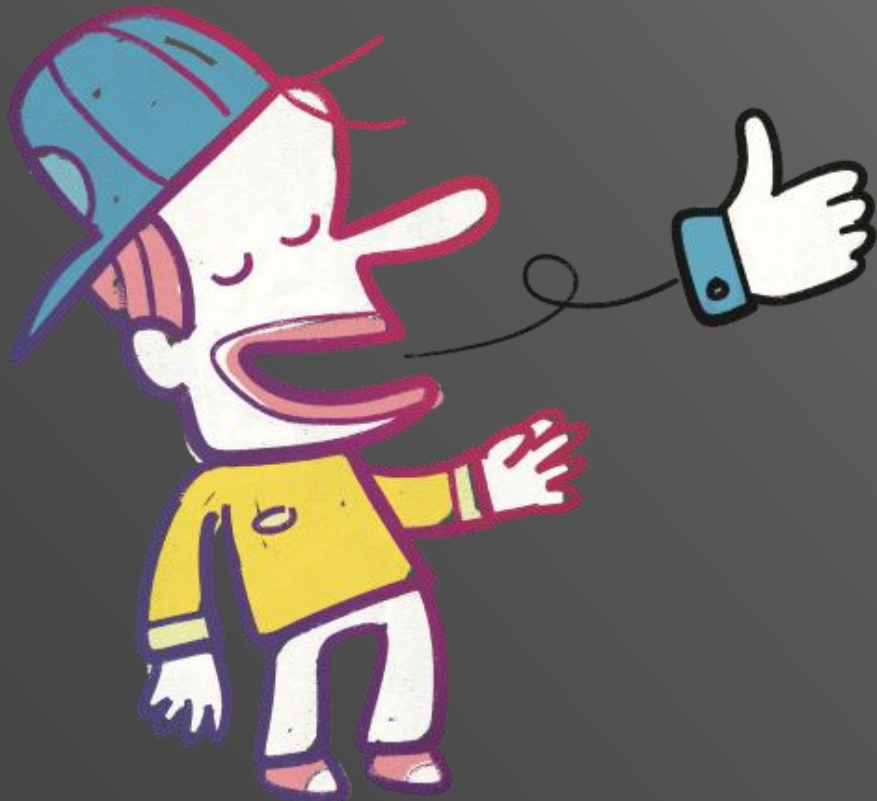


→ lasr-reviews.org

Findet mich auf LinkedIn...



Feedback & Fragen?



LASR - Softwaresysteme im Team reviewen



#BaselOne26

baselone.ch