embarc
Software Consulting GmbH

BenutzMichRichtig.jar

Bewusster Umgang mit 3rd Party Libraries

HARM GNOYKE, EMBARC

Berlin Expert Days

17.09.2015

BenutzMichRichtig.jar

Bewusster Umgang mit 3rd Party Libraries

Der Vortrag zeigt den Umgang mit Fremdbibliotheken (3rd Party Libraries) und die Fallstricke, die beim Einsatz lauern. Häufig wird in Projekten nur der kurzfristige Nutzen beim Einsatz einer Library betrachtet: Die betreffende Funktionalität braucht nicht selbst entwickelt und gepflegt werden. Das freut natürlich den Projektleiter: Das so freigewordene Budget kann anderweitig verplant werden.

Doch hinter solch einem leichtfertigen Einsatz verbergen sich auch Kosten und Risiken, die die Weiterentwicklung des Projektes bremsen, technologischen Fortschritt behindern und im Extremfall das Projekt in eine Sackgasse bringen können.
Diese Session beleuchtet den nachhaltigen Umgang mit 3rd Party Libraries in Projekten von verschiedenen Seiten. Es gibt Orientierung bei der Auswahl der Bibliotheken und beschreibt verschiedene Strategien zum anschließenden Einsatz. Wie geht man mit unterschiedlichen Releasezyklen um? Wie mit unterschiedlichen Lizenzen? Wie vermeidet man Risiken und minimiert die Kosten bei Fehlentscheidungen?
Tipps für den Umgang mit Herstellern (ob Open Source oder kommerziell) gespeist aus Erfahrungen in Projekten mit bis zu 70 Entwicklern runden den Vortrag ab und geben den Zuhörern damit Sicherheit bei den nächsten Überlegungen zu 3rd Party Libraries.

Harm Gnoyke

- Softwareentwickler + -architekt bei embarc in Hamburg
- Vorher Hamburg Süd, sd&m (heute Capgemini)

Schwerpunkte:

- Softwaredesign
- Softwarearchitektur (Bewertung, Methodik)
- Java Technologien

Harm.Gnoyke@embarc.de

@HarmGnoyke

xing.to/HarmG

embarc Software Consulting GmbH

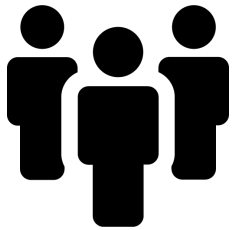
Was sind 3rd Party Libraries?

In Software-Projekten gibt es mindestens zwei Parteien:

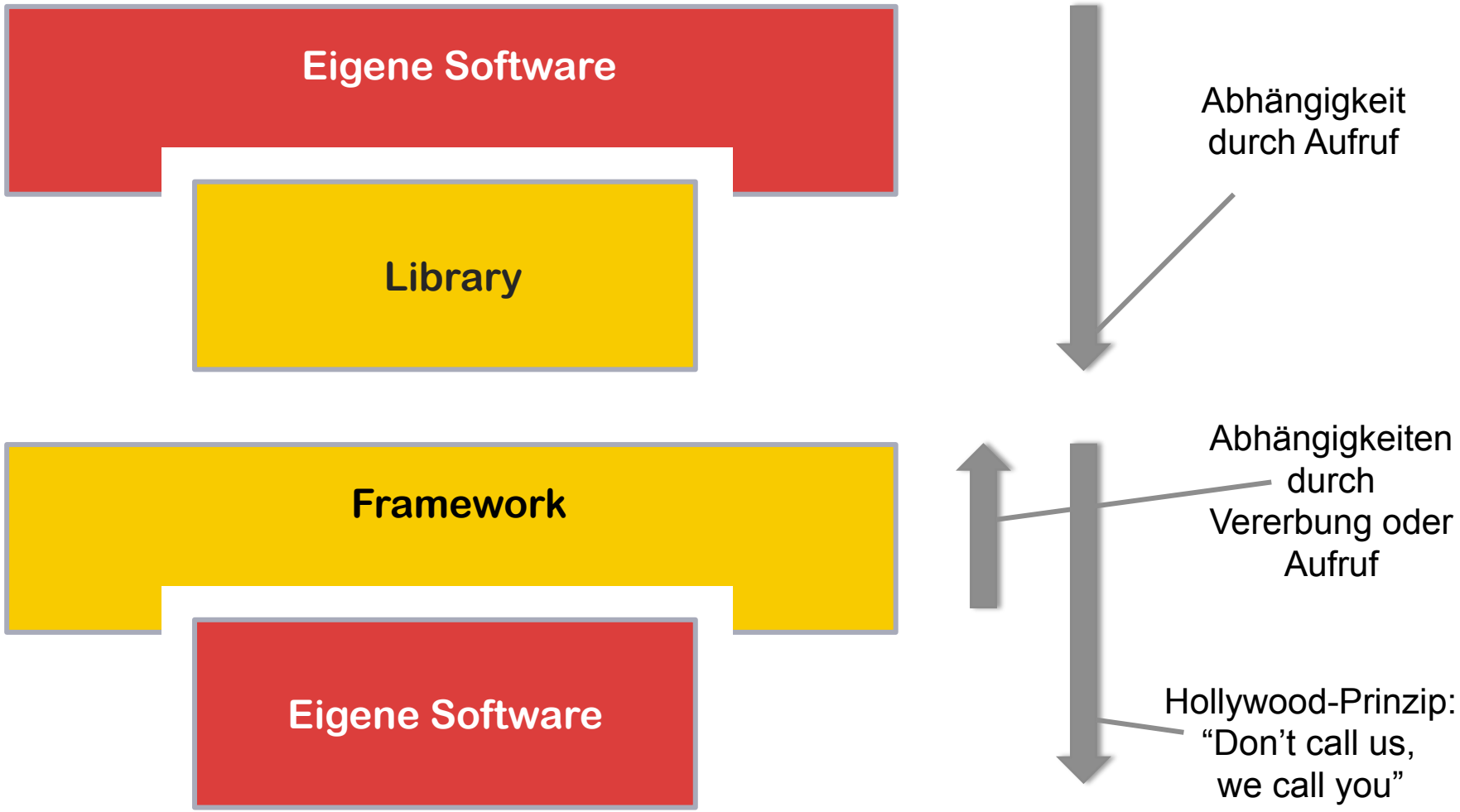
- Auftraggeber
- Auftragnehmer

Teile einer Software, die von einer “projektfremden” Partei bereitgestellt werden, kommen von einer 3rd Party:

- Produkthersteller
- Open Source
- *Anderes Projektteam aus dem eigenen Unternehmen*



Abgrenzung Library vs. Framework



Agenda

1 **Not Invented Here**

2 Wie entscheide ich mich?

3 Rund um die Entscheidung

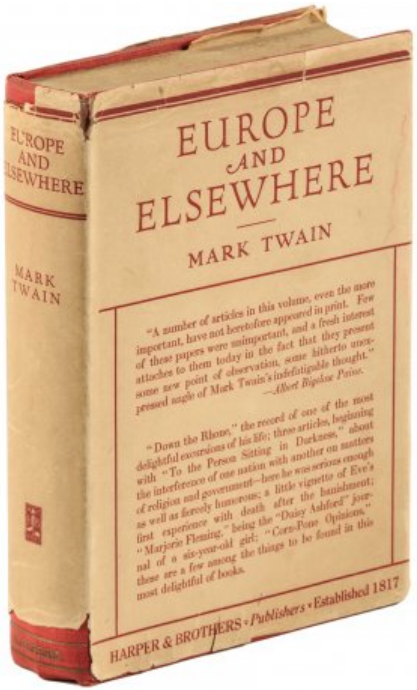
4 Im laufenden Projekt

5 Fazit und Ausblick

1

Not Invented Here

"The slowness of one section of the world about adopting the valuable ideas of another section of it is a curious thing and unaccountable. This form of stupidity is confined to no community, to no nation; it is universal. The fact is the human race is not only slow about borrowing valuable ideas - it sometimes persists in not borrowing them at all."



➔ Some National Stupidities, Mark Twain; Europe and Elsewhere, New York 1923

Die offensichtlichen Argumente

Make ⟷ **Buy**

Selbst bauen

- Mehr Zeit nötig
- Weniger Abhängigkeiten
- Know How direkt im Projekt

3rd Party Library einbinden

- Zeit sparen
- Verantwortung verlagern
- Komplexität reduzieren

Langfristige Auswirkungen auf die Qualität

Agenda

1 Not Invented Here

2 **Wie entscheide ich mich?**

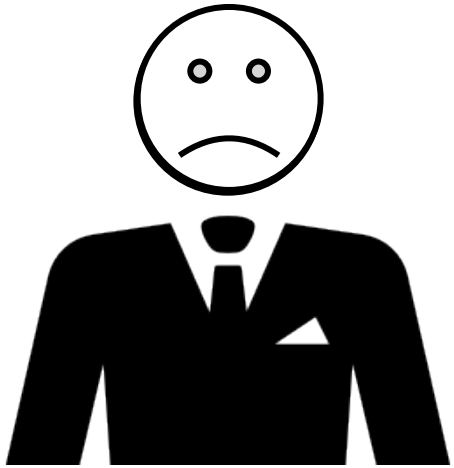
3 Rund um die Entscheidung

4 Im laufenden Projekt

5 Fazit und Ausblick

2

Motivationsfaktor Zeit reicht nicht



Einbinden einer Library ist toll:

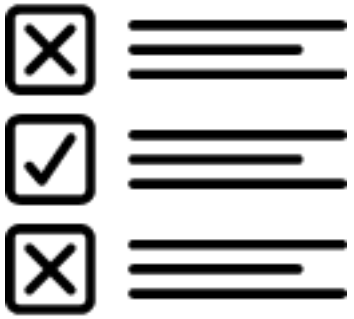
- Man bekommt Funktionalität geschenkt!
- Man spart Entwicklungszeit für andere Features!

Doch es gibt auch nachgelagerte Kosten:

- Fehlerbehebung und Updates der genutzten Bibliothek
- Bei kommerziellen Produkten Wartungsverträge (Gefahr des Vendor Lock-In)
- Auswirkungen auf Qualität der eigenen Software die wichtigsten Merkmale nach ISO 25001: Wartbarkeit, Performance, Sicherheit, Zuverlässigkeit, Kompatibilität

Was sind sinnvolle Kriterien?

- Einsatzzweck
- Aktualität
- Änderbarkeit
- Verständlichkeit
- Aufwand
- Verbreitung
- Interoperabilität
- Kapselungsmöglichkeit
- Tests
- Branching
- Wichtigkeit für Projekt
- ...



Kriterien für unterschiedliche Kontexte anpassen!

Agenda

1 Not Invented Here

2 Wie entscheide ich mich?

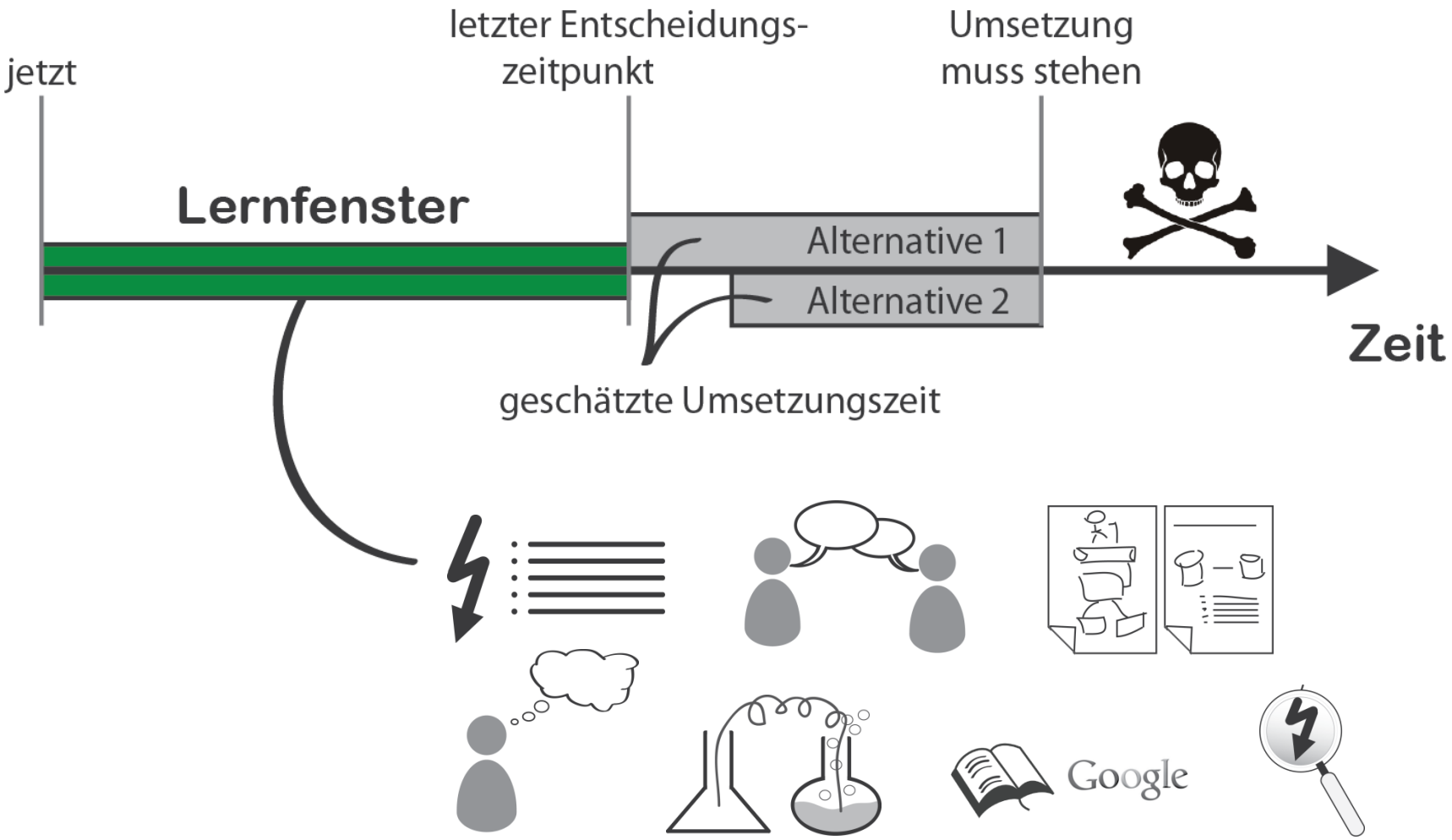
3 **Rund um die Entscheidung**

4 Im laufenden Projekt

5 Fazit und Ausblick

3

Entscheidung hinauszögern



Die Gretchenfrage

Kann ich die benötigte Funktionalität vielleicht doch selber bauen?

Darüber entscheiden:

- Umfang
- Komplexität
- Zeitplan

Orientierung bekomme ich eventuell von existierenden Bibliotheken

Qualität einer Library einschätzen

Community

- Wie viele Entwickler arbeiten (aktiv) mit?
- Wie schnell werden Fragen z.B. auf Mailing Listen beantwortet?

Verbreitung

- Wird es von anderen Projekten eingesetzt?
- Gibt es Erfahrungsberichte von anderen Projekten / Bücher / Blogposts darüber?

Release Zyklus

- Wie häufig gibt es Releases/Patches?
- Passt das zum eigenen Release Zyklus?

Qualität einer Library einschätzen

Test

- Gibt es für die Library ausreichend Unit Tests?
- Wird der eigene Anwendungsfall gut getestet?

Aktualität

- Wann wurde das letzte Release veröffentlicht?

Integration

- Bringt die Library noch andere Deployment-Artefakte mit? (z.B. DB-Tabellen)
- Wie passt es zu anderen Lösungen in der Anwendung?

Einsatzzweck im Überblick behalten

Übersicht

Library	Zweck	Besonderheit	etc.

Übersicht über genutzte 3rd Party Libraries

- Hilft den Zweck einer Library festzuhalten
- Erspart Redundanzen
- Kann kontinuierlich als Nachschlagewerk genutzt werden

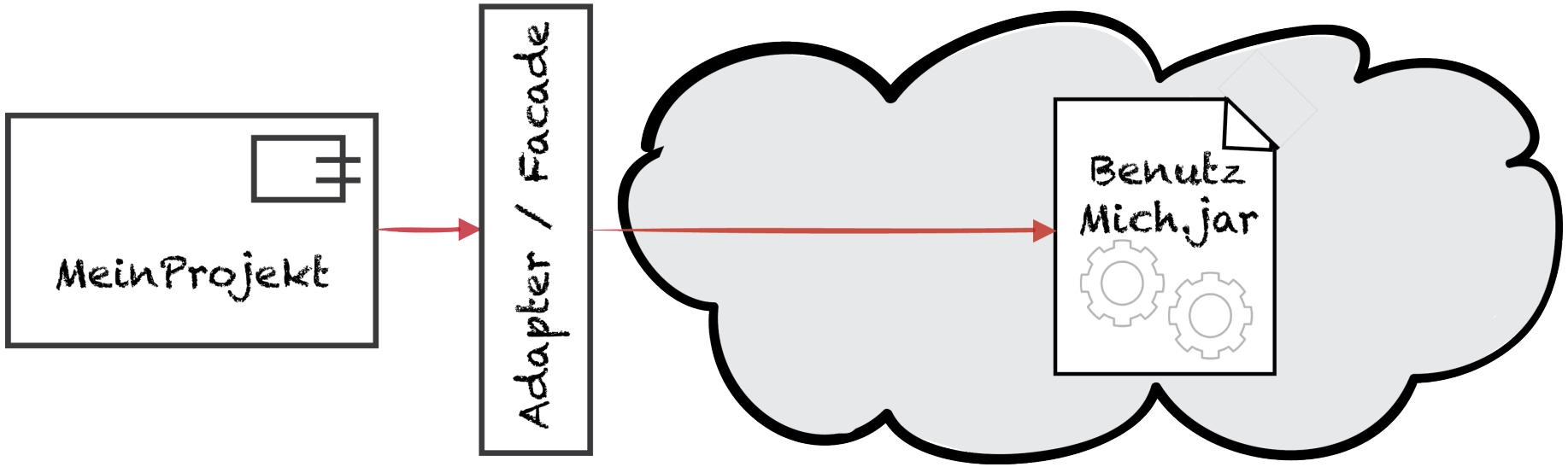
Festgehaltene Attribute können variieren

- Versionsnummern sind besser in der Versionskontrolle aufgehoben

Adapter und Façade

Adapter: Verbindung zweier bestehender Interfaces

Façade: Vereinfachtes Interface



In der Folge wird Wrapper als Oberbegriff für beide Muster gewählt.

Wann lohnt sich das? Wann nicht?

Änderungen in der Library isolieren von der eigenen Software

In den meisten Fällen kann die Reaktion auf eine Änderung der Library im Wrapper erfolgen und schlägt nicht durch

Gut bei starker Verbreitung von Library-Aufrufen in der Software

Komplexe Library, die der Großteil der Entwickler nicht lernen soll

Nur wenige Entwickler arbeiten am Wrapper

Vereinfachtes Interface für den Rest

Standard APIs oder weit verbreitete Libraries und Frameworks

Wrapper verbirgt die Standard API

Kenntnisse der Libraries sind sowieso da

Frameworks <> Libraries, Wrapper sehr aufwändig oder unmöglich

Dokumenterzeugung mit OpenOffice

OpenOffice lässt sich aus Java mit der UNO API steuern.

Auszug aus einem Code Beispiel zum Einbinden einer Grafik:

```
// Querying for the interface XMultiServiceFactory on the XTextDocument
XMultiServiceFactory xMSFDoc = (XMultiServiceFactory)
     UnoRuntime.queryInterface(XMultiServiceFactory.class, xTextDoc);

// Creating the service GraphicObject
Object oGraphic = xMSFDoc.createInstance(
     "com.sun.star.text.TextGraphicObject");

// Querying for the interface XTextContent on the GraphicObject
XTextContent xTextContent = (XTextContent) UnoRuntime.queryInterface(
     XTextContent.class, oGraphic);
// Mehr Code...
```

https://wiki.openoffice.org/wiki/API/Samples/Java/Writer/GraphicsInserter

Eine Erleichterung

NOA (Nice Office Access, Achtung: alt!)

```
Map<String, String> replacements = new HashMap<String, String>();
replacements.put("SearchItem1", "ReplaceItem1");
replacements.put("SearchItem2", "ReplaceItem2");

for (String searchString : replacements.keySet()) {
    // replace all keywords by their respective keyValues
    // noa does not seem to support replacements(XReplaceable), so lets initiate a
    // search and replace the occurences individually
    SearchDescriptor searchDescriptor = new SearchDescriptor(searchString);
    searchDescriptor.setIsCaseSensitive(true);

    ISearchResult searchResult = this.parentWizard.getDocument().getSearchService().
                                        findAll(searchDescriptor);

    if(!searchResult.isEmpty()) {
          //...and now select the result
          List<ITextRange> textRanges = Arrays.asList(searchResult.getTextRanges());
          for (ITextRange textRange : textRanges) {
                    textRange.setText(replacements.get(searchString));
          }
    }
}
```

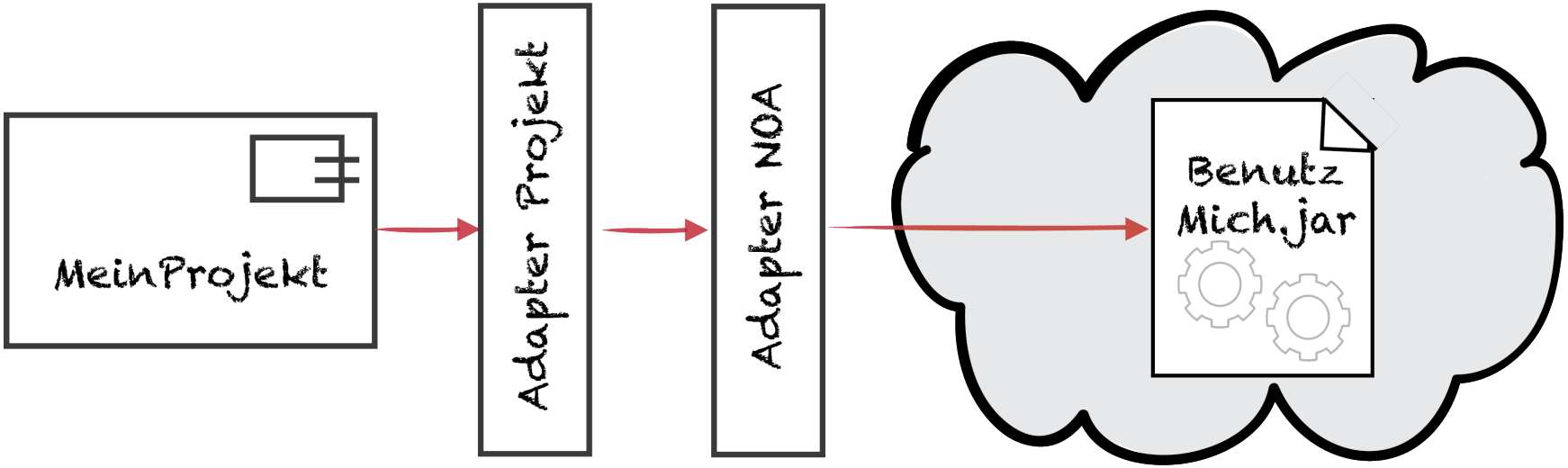
ITextRange ist ein Interface aus NOA, XTextRange wäre das UNO-Pendant.

Doppelt hält besser

Benötigte Standardfunktionalitäten für das Projekt:

- Gültigkeitsprüfung eines Templates gegen eine Liste von Platzhaltern
- Ersetzen von Platzhaltern mit Nutzdaten
- Erzeugung von PDFs zum Versand an Kunden

werden von einem Wrapper bereitgestellt, der noch vor NOA hängt:



Im Dilemma

Beispielanforderung: PDF-Generierung

		
Apache FOP	• Bessere Layout-Möglichkeiten • Weitere Output-Formate	• Formatting Objects als neue Technologie • Hoher Speicherbedarf
iText	• Schnelle Verarbeitung (auch für große Dokumente) • Dokumenterzeugung im Java Code	• Schönes Layout aufwändiger • Lizenz nötig für neue Version (>5.0) oder Nutzung von neuem Fork https://github.com/ymasory/iText-4.2.0/network

Die Ausnahmebehandlung

…man nimmt mehrere Libraries.
Aber das ist mit Zusatzkosten verbunden!

	+	−
Kombination aus beiden	• Library kann passend zum Szenario gewählt werden • Für Aufrufer transparent hinter einem Interface zu verbergen	2 Technologien zu • Lernen, • Warten, • Aktualisieren

Nicht nervös werden…

… wenn sich nach der Entscheidung Alternativen auftun.

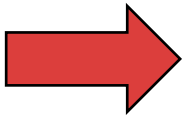
Zum Beispiel durch

- Neue Versionen
- Neue Bibliotheken

Welchen Mehrwert hätte die Alternative gegenüber der getroffenen Entscheidung?

Wie viel Aufwand ist schon verwendet?

Was würde die Alternative kosten?



Auf Veränderung der Entscheidung vorbereiten!

Agenda

1 Not Invented Here

2 Wie entscheide ich mich?

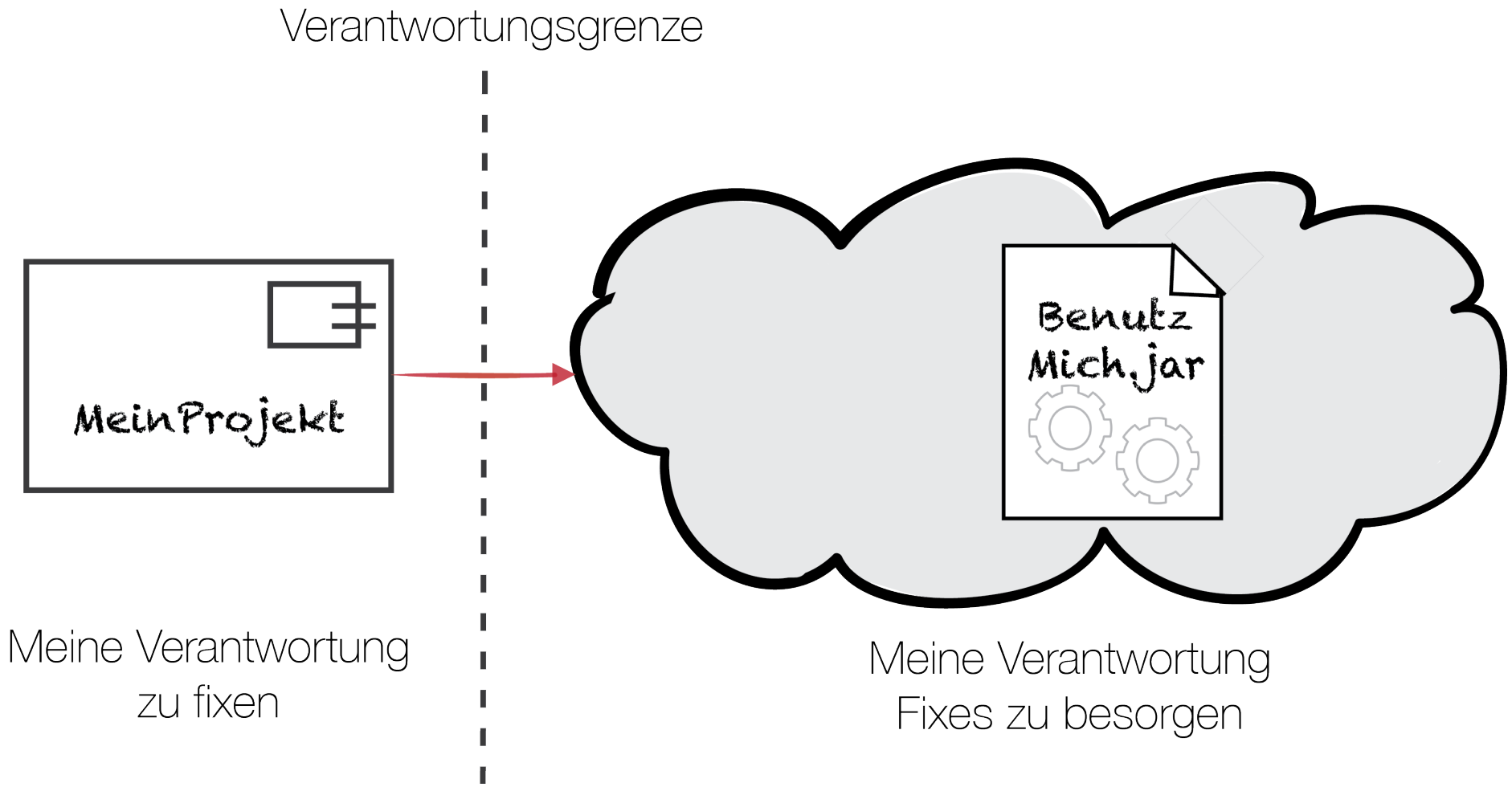
3 Rund um die Entscheidung

4 **Im laufenden Projekt**

5 Fazit und Ausblick

4

Verantwortung für Fixes verschoben



Wie besorge ich mir Fixes? (1/2)

Kommerzielle Software

- Kontakt zum Hersteller besorgen
- SLAs beachten
- Isoliertes Beispiel vorbereiten
  - Showcase nutzen
  - Eigene Anwendung strippen

Ab hier warten…

Kritikalität angemessen melden!

- Nicht über- oder untertreiben
- So wie man es sich von den Benutzern der eigenen Software wünscht

Wie besorge ich mir Fixes? (2/2)

Open Source Software

- Kontakt zur Community suchen (Bug-Tracker, Mailing List, etc)
- SLAs gibt es (meistens) nicht ;-)
- Isoliertes Beispiel vorbereiten
  - Showcase nutzen
  - Eigene Anwendung strippen

Ab hier warten…

oder selbst aktiv werden

- Pull Request schon vorbereiten!

Kein Fix in Sicht

An Workarounds arbeiten:

- Gibt es schon ähnliche gemeldete Bugs?
  - Workarounds dafür prüfen und evtl. Patches nutzen
- Gibt es auch funktionale Workarounds?
  - Die eigene Funktionalität überprüfen

Sollte ein Workaround gefunden sein, wappnet man sich für das Ankommen des „richtigen“ Fixes

- Workaround dokumentieren
  - Empfehlung: Direkt im Source Code markieren inkl. Bug-ID
  - Im Backlog (oder auch der Library-Übersicht) festhalten, dass Nacharbeiten nötig sind

Was tun, wenn der Fix kommt?

Möglichst schnell einbauen, sowie es der eigene *Releasezyklus* zulässt.

Kriterien zur Beurteilung:

- Kritikalität des Fixes für die eigene Anwendung
- Testbarkeit des Fixes
  - im Entwicklertest
  - im Benutzertest
- Isolierte Änderung vs. Release mit mehreren Änderungen
  - Beeinflusst die beiden vorherigen Punkte

Open Source…

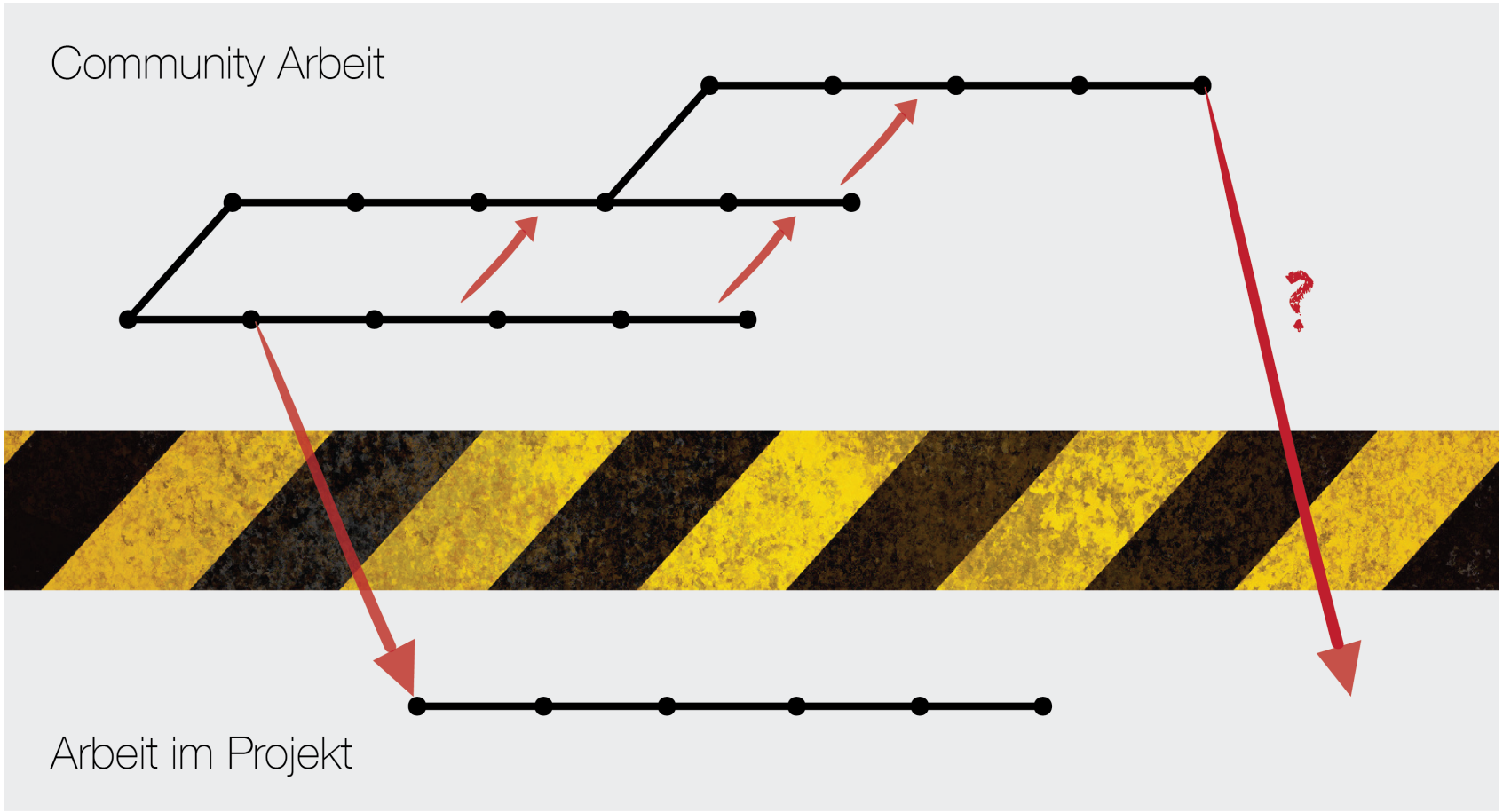
…kann man auch selbst schnell fixen.

Auf was man dabei achten sollte:

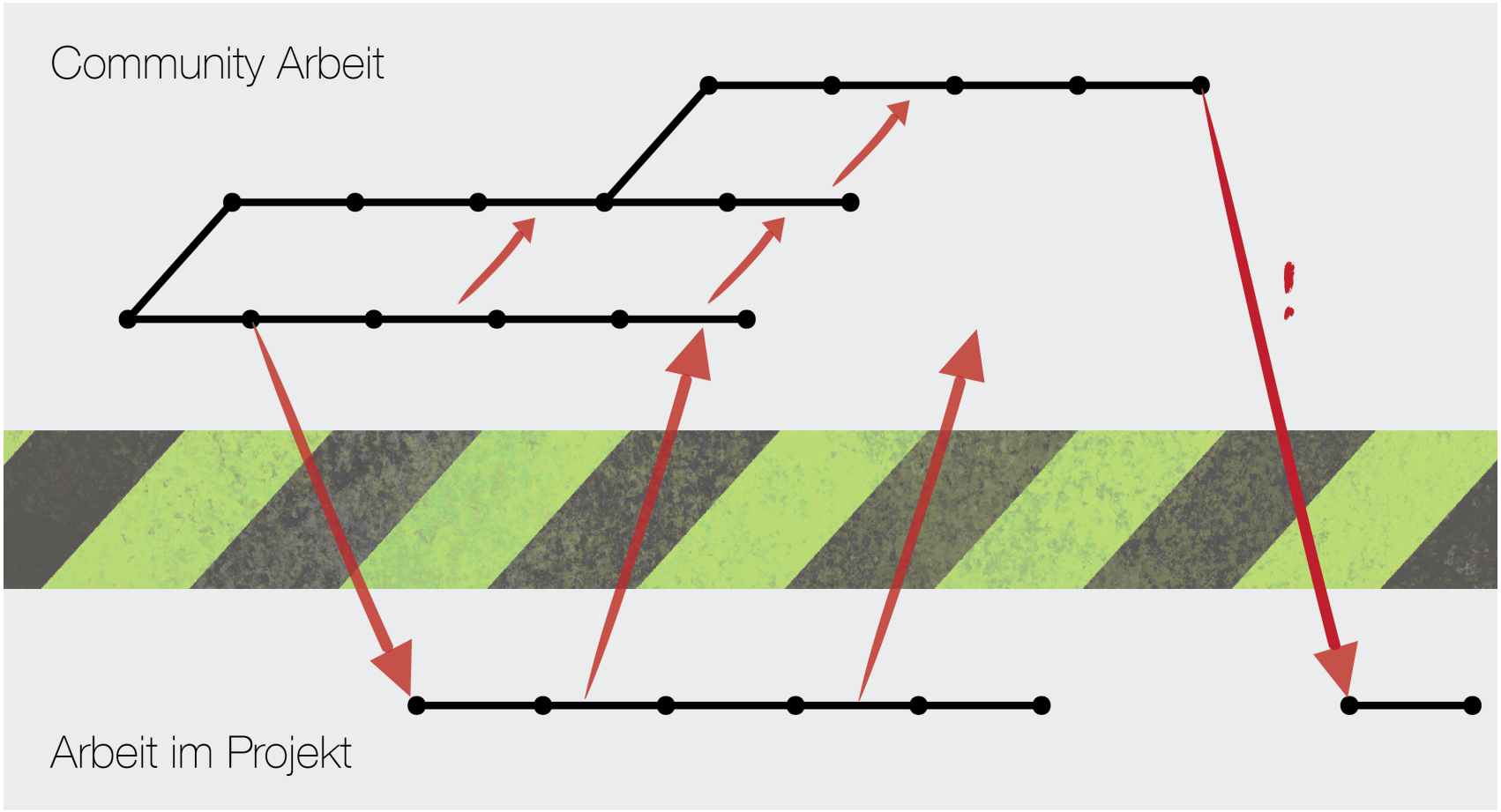
- Eigene Fixes immer zurückfließen lassen
- Fremde Fixes kontinuierlich einbauen

Beides sollte möglichst zeitnah geschehen

Sackgasse



Nicht in der Sackgasse landen



Strategien bei der Aktualisierung

- “Everybody freeze” – Niemals aktualisieren
  - unrealistisch, weil sich immer etwas ändert in einer Software

- „Always update“- Neue Versionen der Library sofort übernehmen
  - das andere Extrem, weil ich nicht immer fremdbestimmte Änderungen machen kann

Die Wahrheit liegt also irgendwo in der Mitte (und ist unterschiedlich je nach Library und Projekt)

- Risiken beim Update minimieren durch
  - Frühzeitigen Einbau (Vermeiden in später Phase)
  - Dedizierte Tests, z.B. Consumer Driven Contracts
  - Einsatz von Wrappern

Versionsnummern helfen

Semantic Versioning schlägt folgende Änderungen der Versionsnummer vor:

MAJOR	MINOR	PATCH
Inkompatibel	Neue oder leicht veränderte Funktionalität	Bugfixes

Viele Projekte im Open Source Bereich halten sich schon an dieses Schema.

Auch nutzbar für nur unternehmensintern veröffentlichte Libraries oder fachliche Komponenten

Agenda

1. Not Invented Here
2. Wie entscheide ich mich?
3. Rund um die Entscheidung
4. Im laufenden Projekt
5. **Fazit und Ausblick**

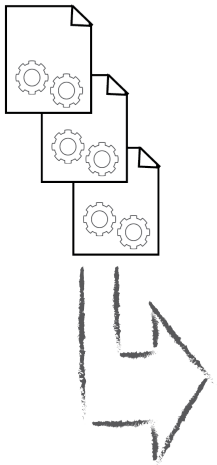
5

Der bewusste Umgang

Übersicht

Library	Zweck	Besonderheit	etc.

Optionen

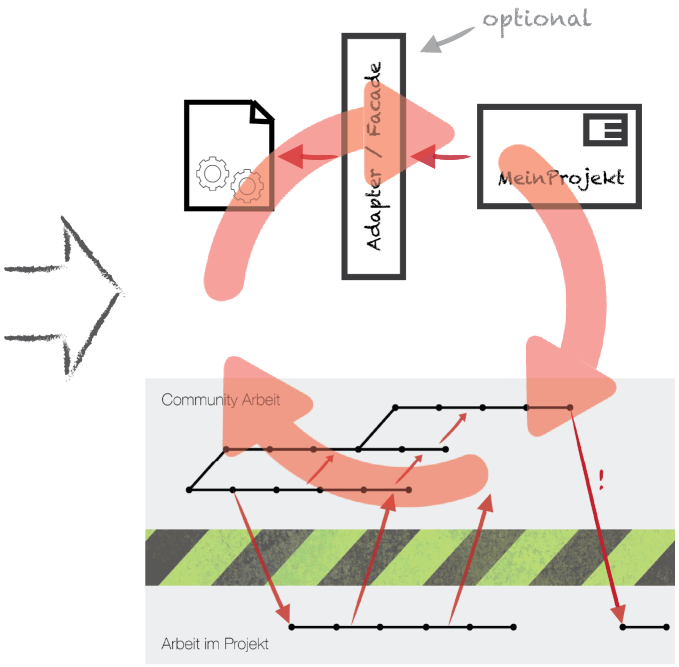


Kriterien

+/−

- Aktualität
- Änderbarkeit
- Verständlichkeit
- Verbreitung
- Interoperabilität
- Kapselung
- Tests
- Verbreitung
- etc.

Arbeit im Projekt



Der bewusste Umgang

Behalte den Überblick über eingesetzte Libraries

Bei der Entscheidung:

Wäge Alternativen ab

Überlege Dir Kriterien für Deinen Kontext

Nur bei triftigen Gründen Entscheidungen umstoßen

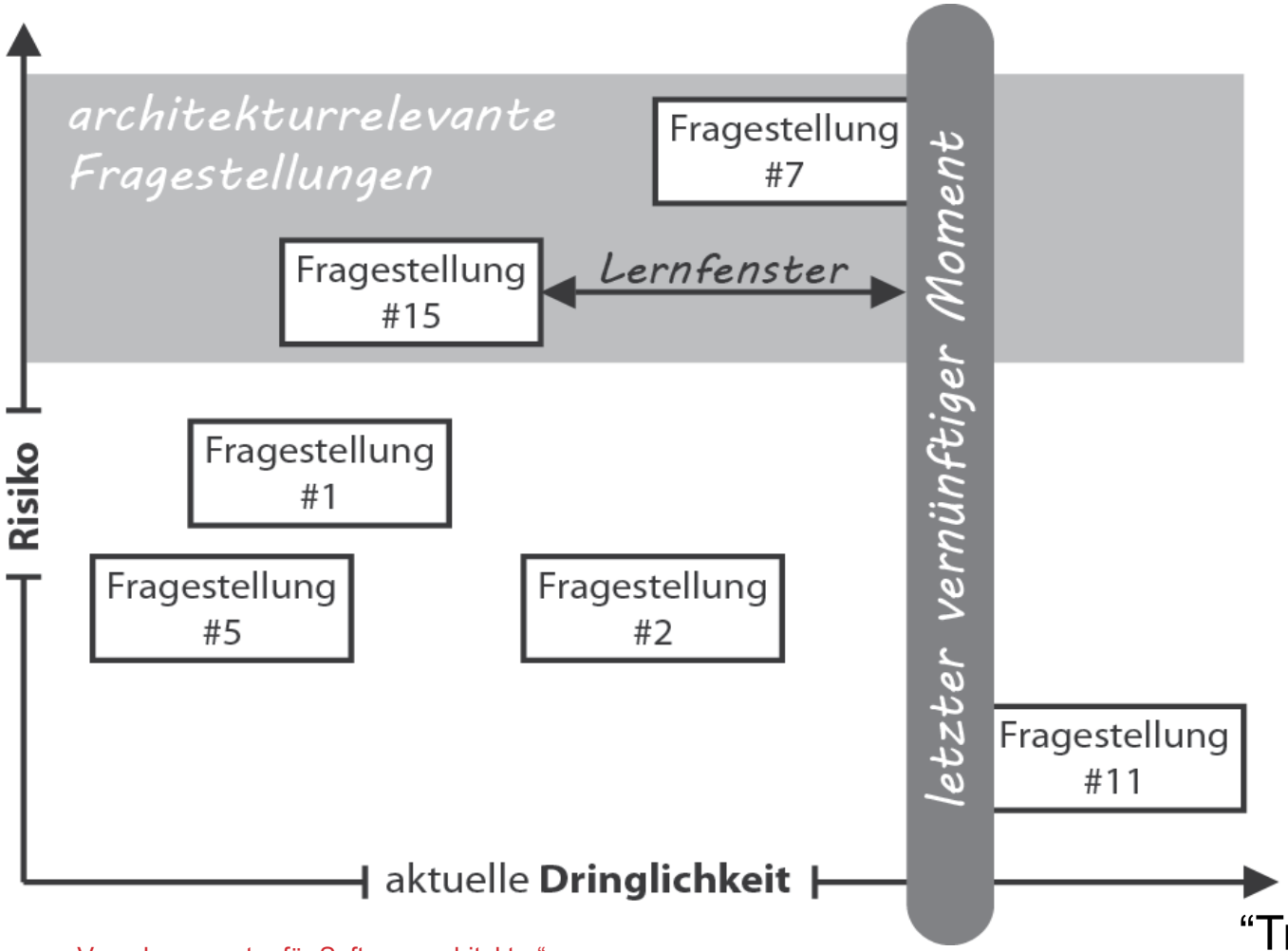
Im Projekt:

Überlege ob Wrapper sinnvoll sind

Baue Kommunikation mit der 3rd Party auf

Definiere eine Update Strategie

Entscheidung hinauszögern - LVM



aus „Vorgehensmuster für Softwarearchitektur“
von Stefan Toth, 2. Auflage 2015

“Triff Entscheidungen so spät wie möglich”

Ausstieg aus Not Invented Here

Groupthink fördert Not Invented Here

Man kann aber ausbrechen:

- Keine Präferenzen für eine Entscheidung aus dem Management vorgeben
- Alle Alternativen berücksichtigen
- Unabhängige Gruppen arbeiten am gleichen Problem
- Mit Meinungen von außerhalb der Gruppe abgleichen
- „Advocatus Diabolus“ als Rolle vergeben:
  - Nimmt eine Kontra-Perspektive ein
  - Fördert Diskussion der Gruppe

Vielen Dank.

Ich freue mich auf Eure Fragen!

 Harm.Gnoyke@embarc.de

 @HarmGnoyke

 xing.to/HarmG

embarc
Software Consulting GmbH

DOWNLOAD FOLIEN:
http://www.embarc.de/blog/