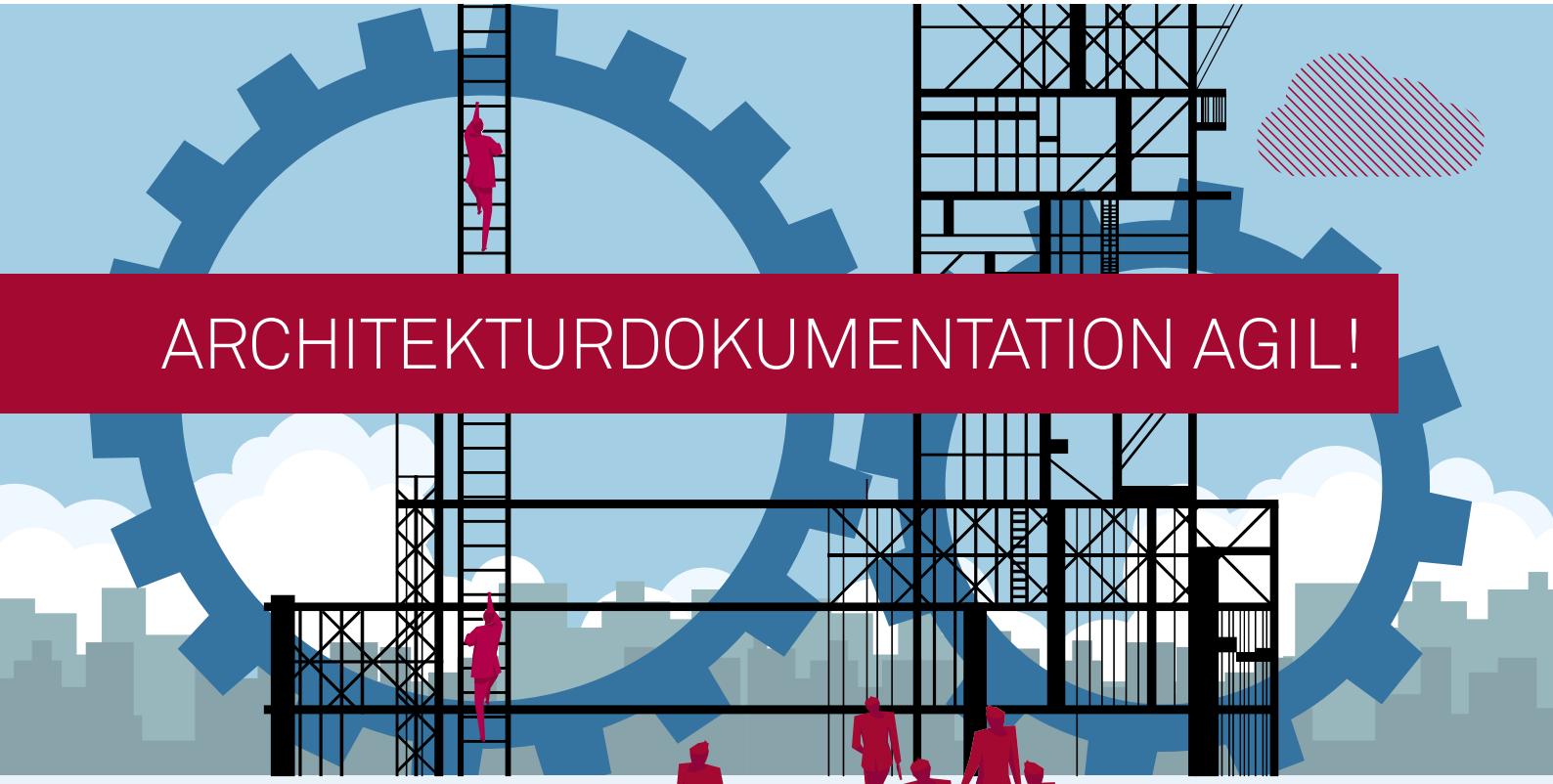

ARCHITEKTURDOKUMENTATION AGIL!

Viele Unternehmen und Behörden befinden sich in der Phase der agilen Transition. Dabei werden gewohnte Dinge und Abläufe buchstäblich auf den Kopf gestellt. Denn in einer agilen Welt mit selbstverantwortlichen Teams wird kein Top-down-, sondern ein Bottom-up-Ansatz gelebt – auch in der Softwarearchitektur. Waren die Softwarearchitekten noch in den 1990er-/2000er-Jahren die Herrscher über Monolithen, so werden sie in Zeiten von Microservices und agilen Teams immer mehr zu Beratern ihrer Teams. Doch wie klappt dieser Change bezüglich Wissen und Verantwortung bei Architekten in der agilen Transition?

| von KLAUS FRANZ

EIN ERFOLGREICHER LÖSUNGSANSATZ ANHAND EINES SZENARIOS

Als technischer Ausgangspunkt dient ein monolithisches System, das durch ein Großprojekt nach und nach in eine Microservice-Landschaft migriert wird. Sozialer Ausgangspunkt ist ein Projekt mit 100 Mitarbeiterinnen und Mitarbeitern sowie ein Architekturteam, das die Zügel fest in der Hand hält. Die Mitarbeiterfluktuation ist hoch, das Wissen in den Teams über ihr betreutes Modul entsprechend schlecht ausgeprägt. Der Lösungsansatz besteht darin, das Wissen durch die entsprechende Architekturdokumentation in die Teams zu bringen und durch Übergang der Dokumentation auch die Verantwortlichkeit der Teams für die erstellten Software-Artefakte zu stärken.

DOKUMENTATIONSAUFBAU

Die Herausforderung: Aus dem grobgranularen Architekturhandbuch, das den Monolithen beschreibt, muss ein verteiltes Architekturhandbuch erstellt werden.

Der Grundgedanke: Die Architektur der Software kann auch auf die Dokumentation angewendet werden.

Die Lösung: Sie besteht in einem auf einem Open-Source-Projekt basierenden Template zur Entwicklung, Dokumentation und Kommunikation von Softwarearchitekturen namens arc42.

Ein Grundprinzip von arc42 besteht darin, die Beschreibung eines Gesamtsystems aus den Beschreibungen der Teilsysteme zusammenzusetzen. Die Teilsysteme werden nicht vollständig dokumentiert, sondern nur die für das Teilsystem relevanten Kapitel beschrieben.

Allerdings handelt es sich im beschriebenen Szenario nicht um ein Gesamtsystem mit exklusiven Teilsystemen, sondern um ein System, das aus der Orchestrierung von Microservices besteht. Das heißt, die Microservices arbeiten nicht exklusiv für ein Gesamtsystem, sondern bedienen auch andere Systeme. Um die Dokumentation für die Teams im Rahmen zu halten, wurden für alle zu bedienenden Gesamtsysteme Referenzarchitekturen dokumentiert, auf die die Architekturdokumentationen der Microservices verweisen können.

TOOLING

Es gibt viele Tools für die Dokumentation von Architekturen in Unternehmen – und in der Regel kommen bei einer Architekturdokumentation auch mehrere Tools zum Einsatz. Doch der Fokus des Teams soll nicht darauf liegen, mehrere Architekturdokumentationswerkzeuge zu beherrschen, sondern darauf, Software zu entwickeln. Außerdem muss die Dokumentation von unterschiedlichen Personen erstellt beziehungsweise bearbeitet und versioniert werden können. Daher eignet sich das Konzept von „Documentation as Code" an dieser Stelle sehr gut.

DOKUMENTERSTELLUNG

Das Konzept sieht vor, Architekturdokumentation genau wie Code zu behandeln und somit zur Codeerstellung und -pflege auch die gleichen Tools zu verwenden. Die Dokumentation liegt im gleichen Versionskontrollsystem beziehungsweise ist selbst Teil des Codes.

Als Format bietet sich dabei asciidoc[1] an, eine weitverbreitete, leichtgewichtige, schnell zu erlernende Markdown-Language. asciidoc-Dateien sind Plaintext-Dateien und können somit leicht in einem Versionskontrollsystem wie zum Beispiel Git gehandhabt werden. Gleichzeitig kann man aus asciidoc Dokumente in vielen weiterverarbeitbaren Formaten erstellen, wie beispielsweise MS Word, PDF, epub, docbook, Confluence-Seiten. Das bereits beschriebene arc42-Projekt bietet bereits ein arc42-Template im asciidoc-Format an.

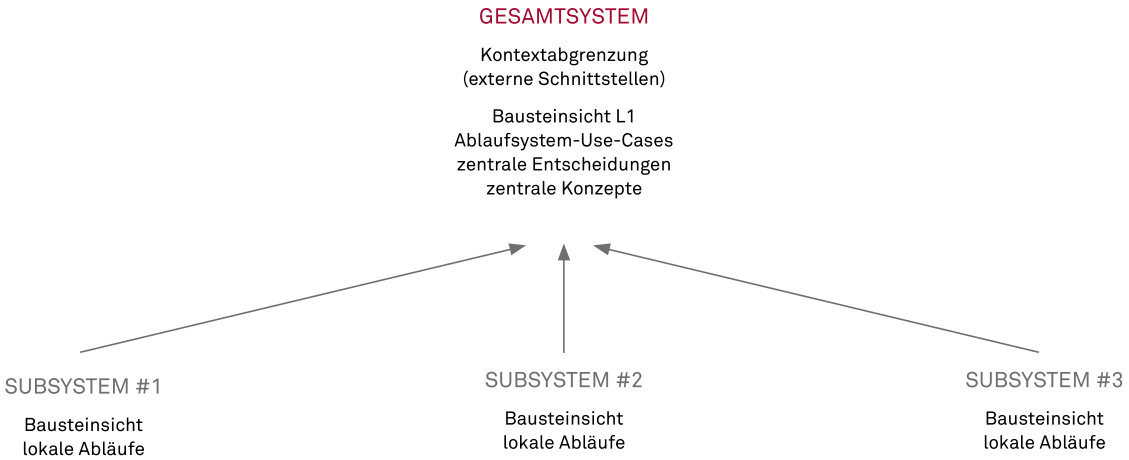

Abbildung 1: Aufbau der Dokumentation in verteilten Systemen (arc42-Empfehlung)

Schaubilder, die bei der Architekturdokumentation unerlässlich sind, können zum Beispiel mit PlantUML, einer sehr mächtigen und auch schnell zu erlernenden Plaintext-Notation für UML-Diagramme erstellt werden.

Ein Vorteil beim Einsatz von Plaintext-Notationen ist das automatische Erkennen von Unterschieden in Dokumentationen und Schaubildern.

QUALITÄTSSICHERUNG

Durch die Verwendung des Konzepts von „Documentation as Code“ kann auch der Prozess zur Qualitätssicherung anders gestaltet werden. Durch eine entsprechende Erweiterung der Versionsverwaltung (zum Beispiel bitbucket[2] für Git) kann bereits im Checkin-Prozess eine Qualitätssicherung durch andere Teammitglieder durchgeführt werden – mit dem positiven Nebeneffekt, dass das Team auf dem gleichen Wissensstand gehalten wird.

DOKUMENTERZEUGUNG

Mit der docToolchain[3] wird ein Konzept und Tool zur Verfügung gestellt, das aus den Plaintext-Dateien andere Formate generieren und auch andere Quellen für die Generierung der Architekturdokumentation einbinden kann (siehe Abbildung 5).

Ein Einsatzszenario ist beispielsweise, beim Bau der Software ein PDF zu erstellen, wohingegen beim Deployment der Software ein unternehmensweit zugänglicher Confluence-Bereich aktualisiert wird. So kann jeder im Unternehmen sehen, wie die aktuelle Software in Produktion aufgebaut ist.

SICHERSTELLUNG DER AKTUALITÄT DER DOKUMENTATION

Spricht man von „Erosion von Software“, hat man in der Regel die Dokumentation als den Bestandteil im Blick, der am schnellsten veraltet. Hingegen kann die Aktualität der Dokumentation im beschriebenen Szenario organisatorisch und/oder technisch sichergestellt werden.

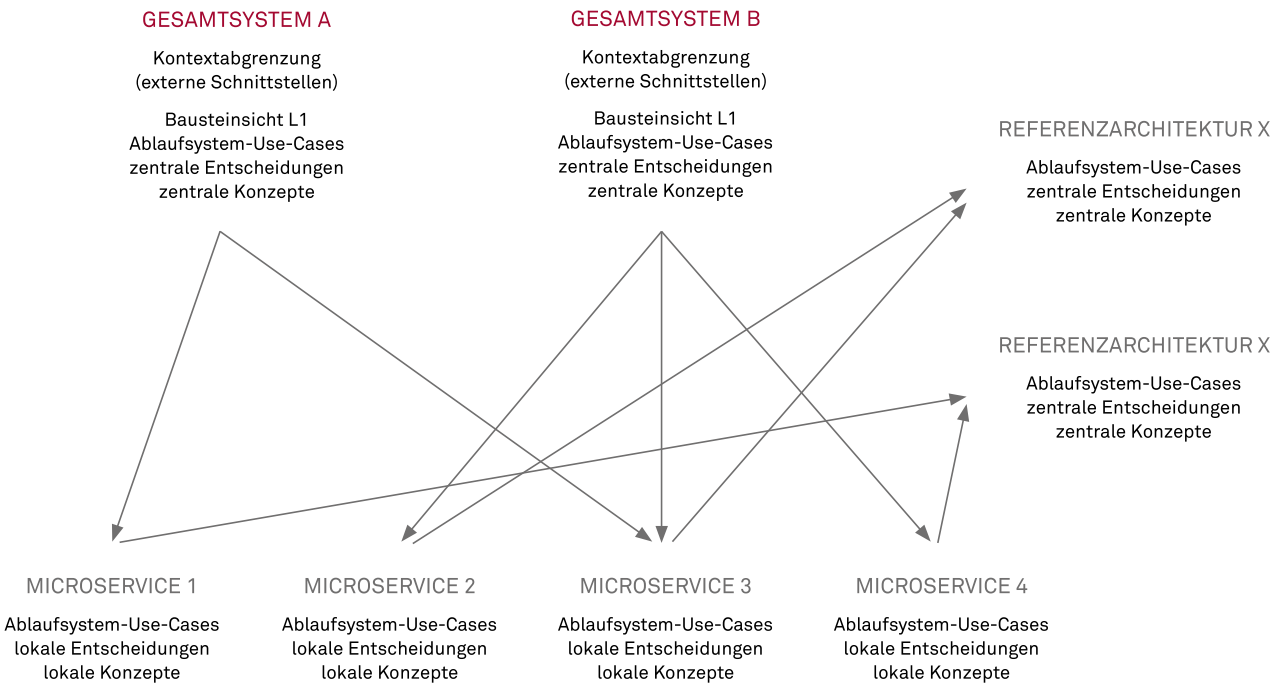

Abbildung 2: Aufbau der Architekturdokumentation in einer heterogenen Microservices-Landschaft

```
.Sequenzdiagramm REST-Service Nutzung
[plantuml, file="trackerschreib.jpg"]
--
@startuml
actor "REST-Nutzer" as nutzer
participant "Tracker" as tracker
database "Tracker-DB" as db

nutzer → tracker : Schreibe Eintrag
nutzer ++
tracker++

tracker →db : Schreibe Eintrag
db++
db→tracker : Eintrag geschrieben
db--

tracker→nutzer : Eintrag geschrieben
tracker--
nutzer--
@enduml
--
```

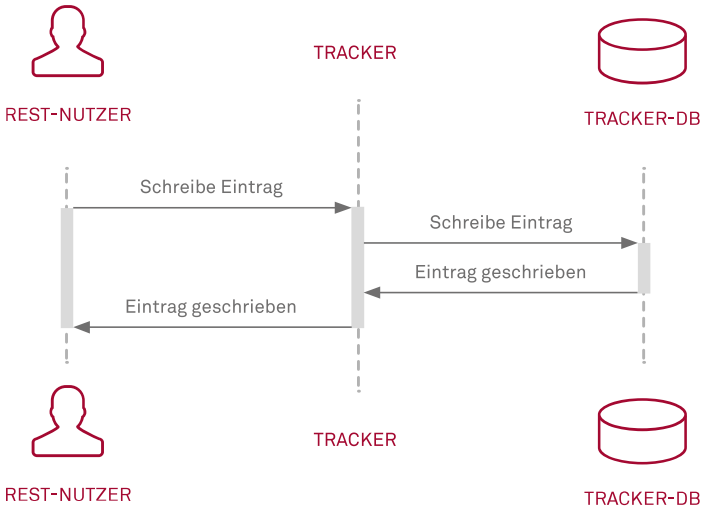

Abbildung 4: Sequenzdiagramm in PlantUML-Notation

Dabei ist die organisatorische Variante, die Dokumentation in die Definition-of-Done aufzunehmen, und die technische Variante, ein QS-Werkzeug, wie zum Beispiel JQAssistant, beim Erzeugen des Codes (und somit der Architekturdokumentation) einzubinden. JQAssistant analysiert den Code statisch und ermittelt mithilfe vorher festgelegter Regeln Abweichungen zu definierten Architekturstandards.[4] Diese Abweichungen können auf eine veraltete Dokumentation hinweisen.

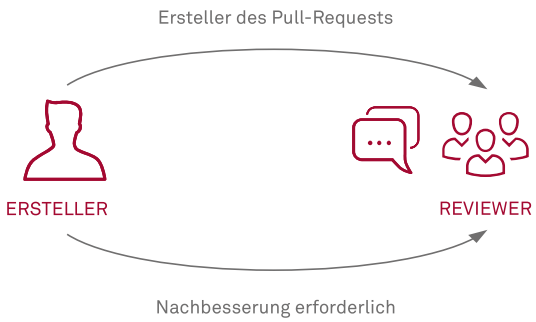

Abbildung 5: Prozess zur Qualitätssicherung

FAZIT

Der Einsatz des Konzepts „Documentation as Code“ rückt die Dokumentation in den Fokus der täglichen Arbeit des Teams. Es stehen bereits viele Tools zur Verfügung, die Projekte dabei unterstützen, mit Dokumentation umzugehen, und bei geringem Aufwand einen erheblichen Mehrwert schaffen. Durch „Documentation as Code“ auf Basis eines Versionskontrollsystems gehören Fragen wie „Welcher Dokumentationsstand passt zu welchem Softwarestand?“ und „Wer hat was geändert?“ der Vergangenheit an. Das Team als selbstverantwortliche Einheit kann sich über den Review-Prozess zu einem frühen Zeitpunkt Feedback von erfahrenen Architekten holen. Der Review-Prozess kann aber auch zur Wissensverbreitung genutzt werden.

AUSBLICK

„Documentation as Code“ bietet viele Möglichkeiten, um mit Dokumentation in Software-Projekten umzugehen. Man könnte diesen Bottom-up-Ansatz auch für das EAM nutzen und aus der Dokumentation entsprechende EAM-Artefakte generieren. Auch die Automatisierung von Dokumentation ist durch die Verwendung von Plaintext-Dateien sehr viel einfacher möglich. Darüber hinaus könnte der JQAssistant nicht nur zur Kontrolle der Aktualität der Architekturdokumentation, sondern auch zur Generierung von Teilen der Dokumentation eingesetzt werden. Das hier beschriebene Tool-Set ist außerdem nicht auf Architekturdokumentation beschränkt, sondern kann auch für andere Dokumente verwendet werden, wie zum Beispiel Betriebshandbücher, Release-Notes etc. Oder auch für ein Anwenderhandbuch mit automatisch aktualisierten Screenshots. ●

1 https://de.wikipedia.org/wiki/AsciiDoc
2 https://bitbucket.org/
3 https://doctoolchain.github.io/docToolchain/
4 Siehe .public Ausgabe 02-2018