www.folio-bib.org

Bibliothekssystem Reloaded: Die Architektur unter FOLIO

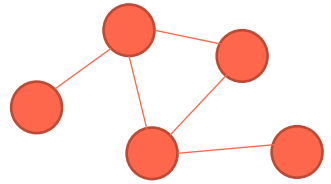
Julian Ladisch, Verbundzentrale des GBV

22. GBV-Verbundkonferenz, Kiel

30. August 2018

Technisches Konzept

- Offene Plattform: Library Service Platform (LSP)
- Plattform stellt Infrastruktur für funktionale Module bereit
- Funktionale Module → eigenständige Programme
  - Können unabhängig voneinander entwickelt werden
  - Können einzeln ausgewählt und installiert werden
  - Kommunikation über Schnittstellen
- Design orientiert sich an Microservice-Idee



Technisches Konzept

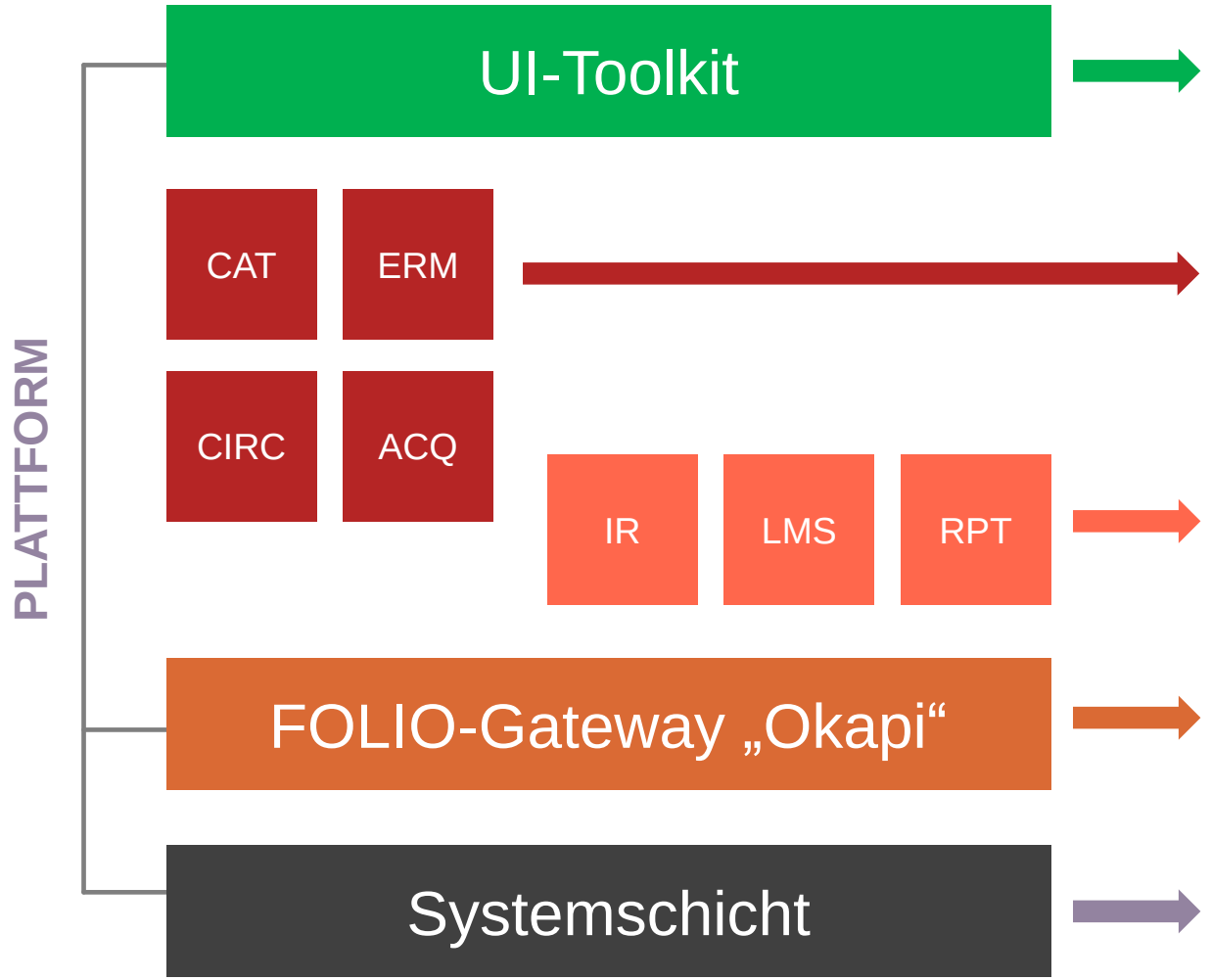
- Unterstützung verschiedener Support-Modelle
  - cloud-basiert, Hosting, lokal
  - kommerziell, Verbund, selber
- Mandantenfähig
- Flexibel erweiterbar, modular
- „Plug and Play“-Applikation
- Basierend auf heutigen Anforderungen mit Ausrichtung auf zukünftige Bedürfnisse

Plattformdesign

„Durchgängig APIs“

- Das bedeutet, dass
  - ➢jeder Entwickler mit jeder Schicht in der Plattform interagieren kann, und
  - ➢keine Komponente zu groß ist, um sie zu ersetzen.

Technologien



UI-Toolkit, genannt Stripes
Standardnutzeroberfläche für App-Entwicklung, basierend auf React + Redux (Open-Source-Javascript-Frameworks), kann genutzt oder für neue Oberfläche adaptiert werden

Basis-FOLIO BMS-Apps
Open-Source-Apps, z. B. ERM, Erwerbung, Ausleihe, Katalogisierung, …
Weitere Apps können entwickelt werden, freie Wahl der Programmiersprache

Weitere Apps
Erweiterung bestehender Apps oder Neuentwicklung, kommerziell oder Open Source

FOLIO Gateway
APIs, “Switchboard” (=Kommunikation) zwischen Anwenderschicht (Apps/UI) und Datenbank, Mandantentrennung, Kommunikation via HTTP

SYSTEM
Zentrale Datenschicht (Datenbank, z.B. SQL)
Indexierung, Logging, Mandantenkonfiguration

Technologien

Moderner Softwarestack aus bewährten Komponenten

Frontend (= im Browser)

- JavaScript (ECMAScript 6)
- React/Redux

Backend (= auf Server)

- Java 8
- Vert.x (asynchrone Kommunikation)
- RAML
- PostgreSQL
  - JSONB (NoSQL) und
  - relationales SQL

React und Redux

- Sind Open-Source-JavaScript-Webframeworks für Single-Page-Applications (SPAs) = Einseitenwebanwendungen
- React bietet ein Grundgerüst für die Ausgabe von User-Interface-Komponenten in HTML
- Redux ist ein Datencontainer, vereinfacht Lesen vom und Schreiben zum Backend

- https://reactjs.org/ und https://redux.js.org/

Stripes

- Javascript-Programmbibliothek für Frontendmodule
- Basiert auf React + Redux
- Zugeschnitten auf Okapi
  - Kommunikation via Okapi zu Backendmodulen
  - Granulare Nutzerrechte
  - Locale (Sprache, Datumsformat, …)
  - Hotkeys (Tastaturabkürzungen)
  - Logging via Okapi
- https://github.com/folio-org/stripes-core/#readme

vert.x

- Bibliothek für Java
- Ermöglicht einfache Nebenläufigkeit
- Umgeht viele Probleme paralleler Programmierung
- Asynchrone Kommunikation
  - Vert.x verpackt synchrone HTTP-REST-Anfragen in eine asynchrone Schnittstelle
- Reaktive Programmierung
- Entwurfsmuster „Reactor“

- http://vertx.io/

RAML

- RAML = RESTful API Modeling Language
- Schnittstellenbeschreibung der Module
- Daraus wird automatisch erzeugt:
  - Schnittstellendokumentation, siehe https://dev.folio.org/doc/api/
  - Java-Code (Interfaces)
  - Validierung, die Okapi beim Schnittstellenaufruf durchführt:
    - Erforderliche Benutzerrechte vorhanden?
    - Datenformat korrekt?
- https://github.com/folio-org/raml-module-builder

Datenbankauswahl

- PostgreSQL
  - 2016 Tests auch mit MongoDB
- Wahl fiel auf PostgreSQL. Grund: unterstützt gleichzeitig (!)
  - sowohl relationales SQL-Datenbankmodell
  - als auch dokumentenbasiertes NoSQL-Datenbankmodell
- NoSQL = Not-only-SQL, in diesem Fall dokumentenbasiert (JSON-Dokumente)
- PostgreSQL kann JSON-Dokumente als JSONB verarbeiten, also in einem effizienten binären Format, bei dem das JSON-Dokument in seine Bestandteile zerlegt und dadurch indexierbar gemacht wird.

JSON

- JSON = JavaScript Object Notation
- In PostgreSQL speichert FOLIO die meisten Daten als JSONB.
- Datenaustauschformat der meisten FOLIO-APIs ist JSON.
- Vert.x bietet umfangreiche JSON-Unterstützung,
  in vert.x ist JSON das übliche Austauschformat.
- JSON ist ein sehr verbreitetes Datenaustauschformat für asynchrone Browser-Server-Kommunikation
  - auch für nicht-JavaScript-Programmiersprachen wie Java in FOLIO.

Mandantentrennung

- Mandant = völlig unabhängige Institution (Institutsbibliothek kein Mandant, sondern hat granulare hierarchische Zugriffsrechte)
- Je Mandant eine eigene logische Datenbank (PostgreSQL „Schema“) mit eigenem Datenbanknutzer (PostgreSQL „Role“)
- Datenbankverbindung mit Datenbanknutzer, PostgreSQL garantiert Mandantentrennung
- Tests in Okapi und RMB belegen effektive Mandantentrennung

Datenbankbetrieb

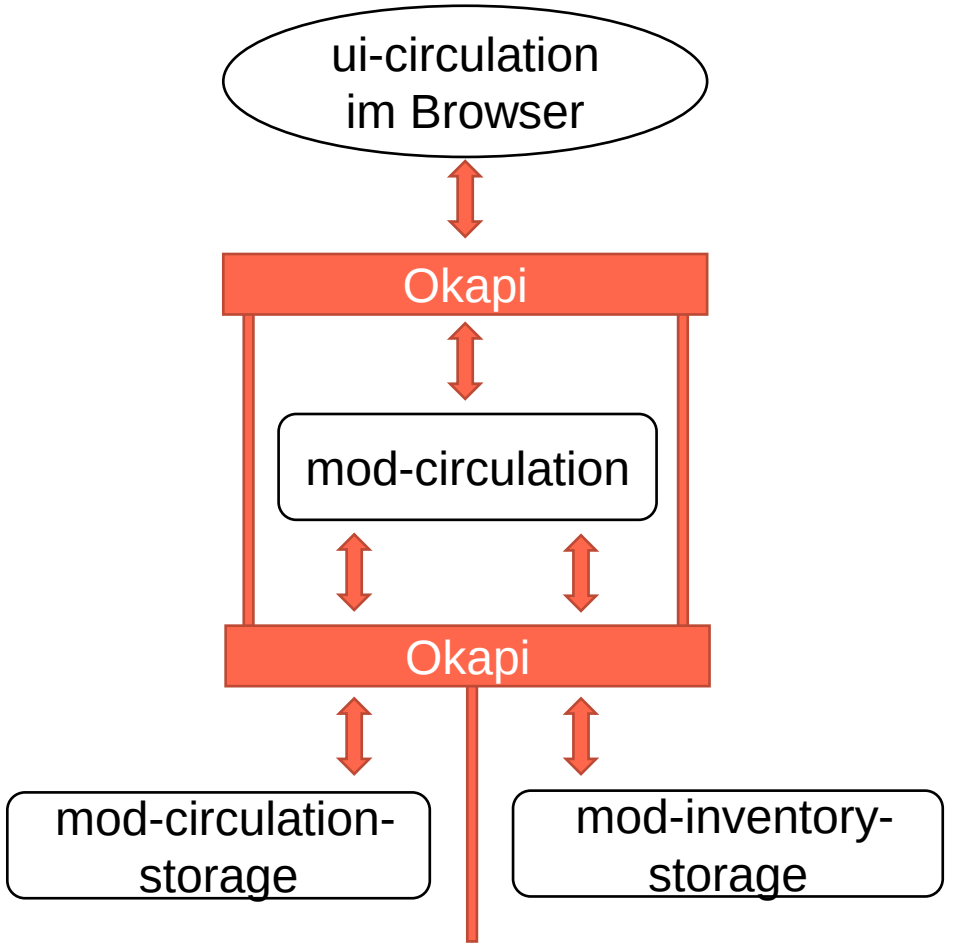
- Jedes Storage-Modul kann eine eigene PostgreSQL-Instanz starten.
- Nützlich für die Softwareentwicklung.
- Per Parameter kann gemeinsame externe PostgreSQL-Installation angebunden werden.
- Genutzt bei Demo- und Test-Installationen.
- Ermöglicht Hochverfügbarkeit und Replikation durch PostgreSQL-Cluster.

Intermodulkommunikation

Beispiel Ausleih-App
mit drei Ausleih-Modulen:

Zu einem Medium kombiniert mod-circulation die Ausleihdaten (mod-circulation-storage) und Titeldaten (mod-inventory-storage).

*-storage = Datenbank-abstraktionsschicht



Okapi

- Okapi macht Zugriffsprüfung
- Darf Mandant das Modul überhaupt nutzen?
- Hat Nutzer nötige Zugriffsrechte?
- Validierung der übergebenen Parameter
- Weiterleitung an Modul
- Falls mehrere Modul-Versionen vorhanden:
  - Auswahl der für Mandanten aktivierten Version

Module

- Module kommunizieren ausschließlich über Schnittstellen
- Unabhängigkeit
- Leicht wartbar, leicht austauschbar
- Lizenz je Modul unabhängig wählbar:
  - proprietär
  - virale Lizenz wie GPL oder AGPL
  - freizügige Lizenz wie Apache oder MIT
- Programmiersprache, -bibliothek unabhängig wählbar (Kernmodule aber einheitlich)

App- und Modularchitektur

- Appabgrenzung nach inhaltlicher Zusammengehörigkeit
- Nur wenig Datenaustausch zwischen Apps
- Beispiel: Ausleihe und Rückgabe sind eine App
  - Nur auf Benutzungsoberfläche zwei Menüpunkte
- Nicht: Nanoservices mit Minimodulen
- Je App ein Entwicklerteam
- Je App je ein Modul für Benutzungsoberfläche, Geschäftslogik und Datenhaltung (Datenbankansteuerung)

Dockermodule

```
$ vagrant init folio/snapshot
$ vagrant up
$ vagrant ssh
$ docker ps
IMAGE                                                PORTS
stripes                                              0.0.0.0:3000->80/tcp
folioci/mod-inventory:7.0.1-SNAPSHOT.71              0.0.0.0:9137->9403/tcp
folioci/mod-inventory-storage:7.2.2-SNAPSHOT.74      0.0.0.0:9132->8081/tcp
folioci/mod-circulation:7.2.0-SNAPSHOT.97            0.0.0.0:9134->9801/tcp
folioci/mod-circulation-storage:4.3.0-SNAPSHOT.62    0.0.0.0:9131->8081/tcp
folioci/mod-users-bl:2.2.1-SNAPSHOT.9                0.0.0.0:9145->8081/tcp
folioci/mod-users:14.4.1-SNAPSHOT.13                 0.0.0.0:9133->8081/tcp
folioci/mod-codex-ekb:0.0.5-SNAPSHOT.57              0.0.0.0:9147->8081/tcp
folioci/mod-codex-inventory:1.0.3-SNAPSHOT.29        0.0.0.0:9146->8081/tcp
folioci/mod-login:4.0.1-SNAPSHOT.9                   0.0.0.0:9144->8081/tcp
folioci/mod-permissions:5.0.1-SNAPSHOT.10            0.0.0.0:9143->8081/tcp
folioci/mod-codex-mux:2.1.3-SNAPSHOT.43              0.0.0.0:9142->8081/tcp
folioci/mod-notes:2.0.2-SNAPSHOT.46                  0.0.0.0:9141->8081/tcp
folioci/mod-notify:1.1.6-SNAPSHOT.37                 0.0.0.0:9140->8081/tcp
folioci/mod-login-saml:1.0.2-SNAPSHOT.16             0.0.0.0:9139->8081/tcp
folioci/mod-authtoken:1.2.0-SNAPSHOT.15              0.0.0.0:9138->8081/tcp
folioci/mod-kb-ebsco:0.1.1-SNAPSHOT.18               0.0.0.0:9136->8081/tcp
folioci/mod-configuration:4.0.1-SNAPSHOT.25          0.0.0.0:9135->8081/tcp
```

stripes = GUI aller Apps (Single Page Application, SPA)

Farbbedeutung bei folioci/*:
Geschäftslogikmodul, z.B. für Kombination von Daten verschiedener Module
Systemmodul, das direkt auf PostgreSQL-Datenbank zugreift

Okapi nimmt Anfragen vom Benutzerbrowser und den Modulen auf Port 9130 entgegen und leitet sie weiter an 9131-9147.

Vagrant und Docker

- Ein Dockercontainer je Modul
  - ggf. je Version eines Moduls
- Viele Mandanten nutzen denselben Container
- Dockeridee: Je Dockercontainer ein (Betriebssystem-)Prozess
- Okapi koordiniert Start, Stopp und Interprozesskommunikation
- Vagrant: Zusammenstellung von passenden Okapi- und Modulversionen
- Ausblick: Überlegungen der SysOps-SIG zu Produktions-Deployment-Umgebung mit Kubernetes, Rancher und Minikube

Vagrant und Docker

- Komplettes FOLIO-System als Vagrantbox
  - https://app.vagrantup.com/folio
  - https://github.com/folio-org/folio-ansible
- Einzelne Module als Dockercontainer
  - https://hub.docker.com/u/folioorg/
  - https://hub.docker.com/u/folioci/

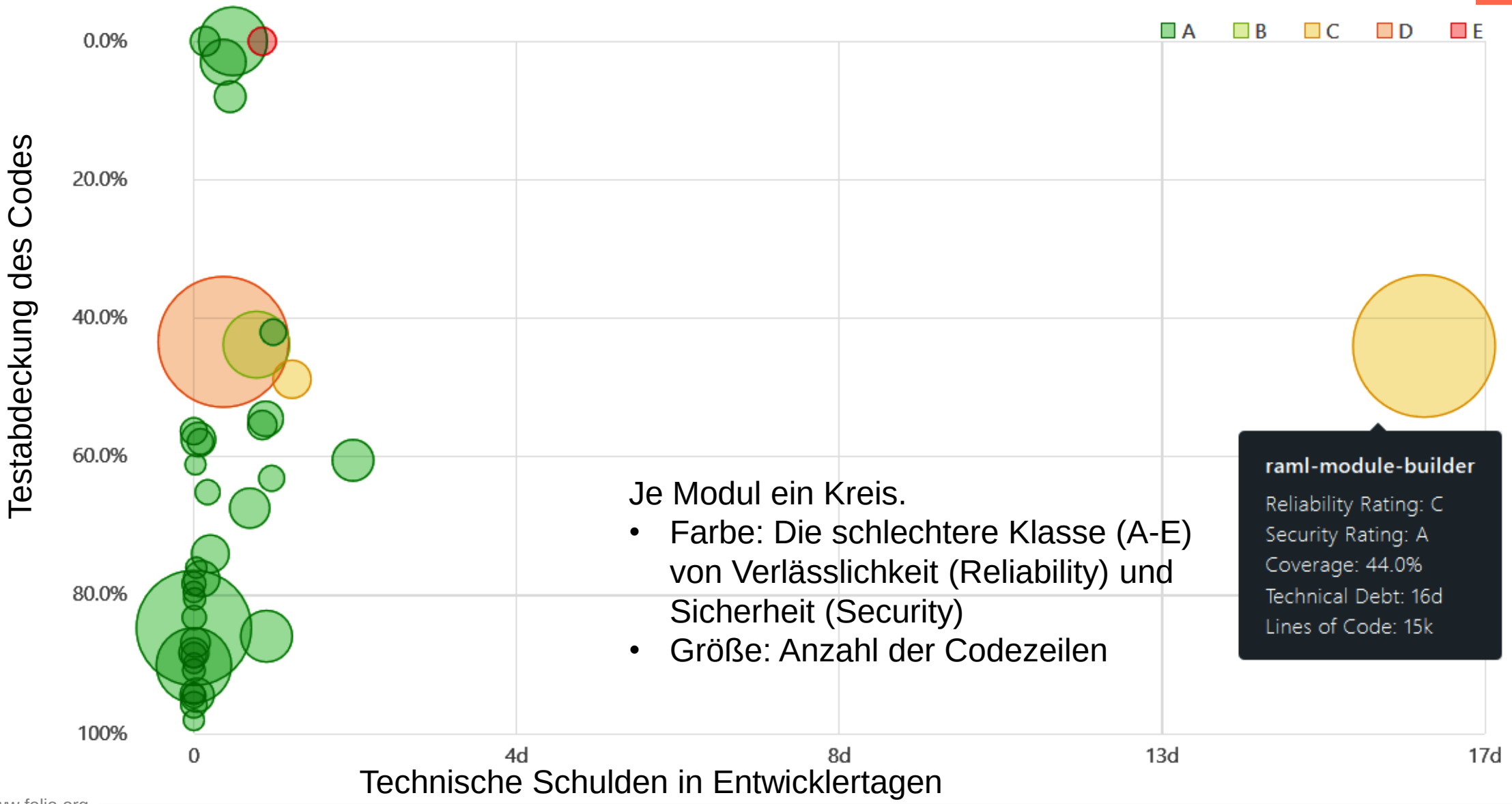
Technische Evaluation

- Die technische Basis der FOLIO-Plattform haben sowohl Vertreter der OLE-Community als auch EBSCO evaluiert.
- Ergebnis: Keine grundsätzlichen Bedenken, FOLIO ist auf dem richtigen Weg.
- Verbesserungsvorschläge/Hinweise wurden inzwischen größtenteils umgesetzt.

Technical Council

- Legt technische Grundsätze fest, z.B.
  - Architekturfragen
  - Standards für Code
  - Datenschutz
- Berät das Product Council
- Vermittelt zwischen Interessen
- Existiert seit Juni 2018

Codeanalyse der Backend-Module



Je Modul ein Kreis.

- Farbe: Die schlechtere Klasse (A-E) von Verlässlichkeit (Reliability) und Sicherheit (Security)
- Größe: Anzahl der Codezeilen

folio

future of libraries is open

Vielen Dank!

Julian Ladisch
julian.ladisch@gbv.de
www.folio-bib.org