FOLIO

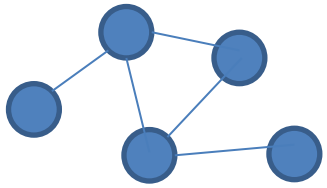
Technische Aspekte

Julian Ladisch, Verbundzentrale des GBV (VZG)

FOLIO-Informationstag
Universitätsbibliothek Stuttgart, 18.10.2017

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

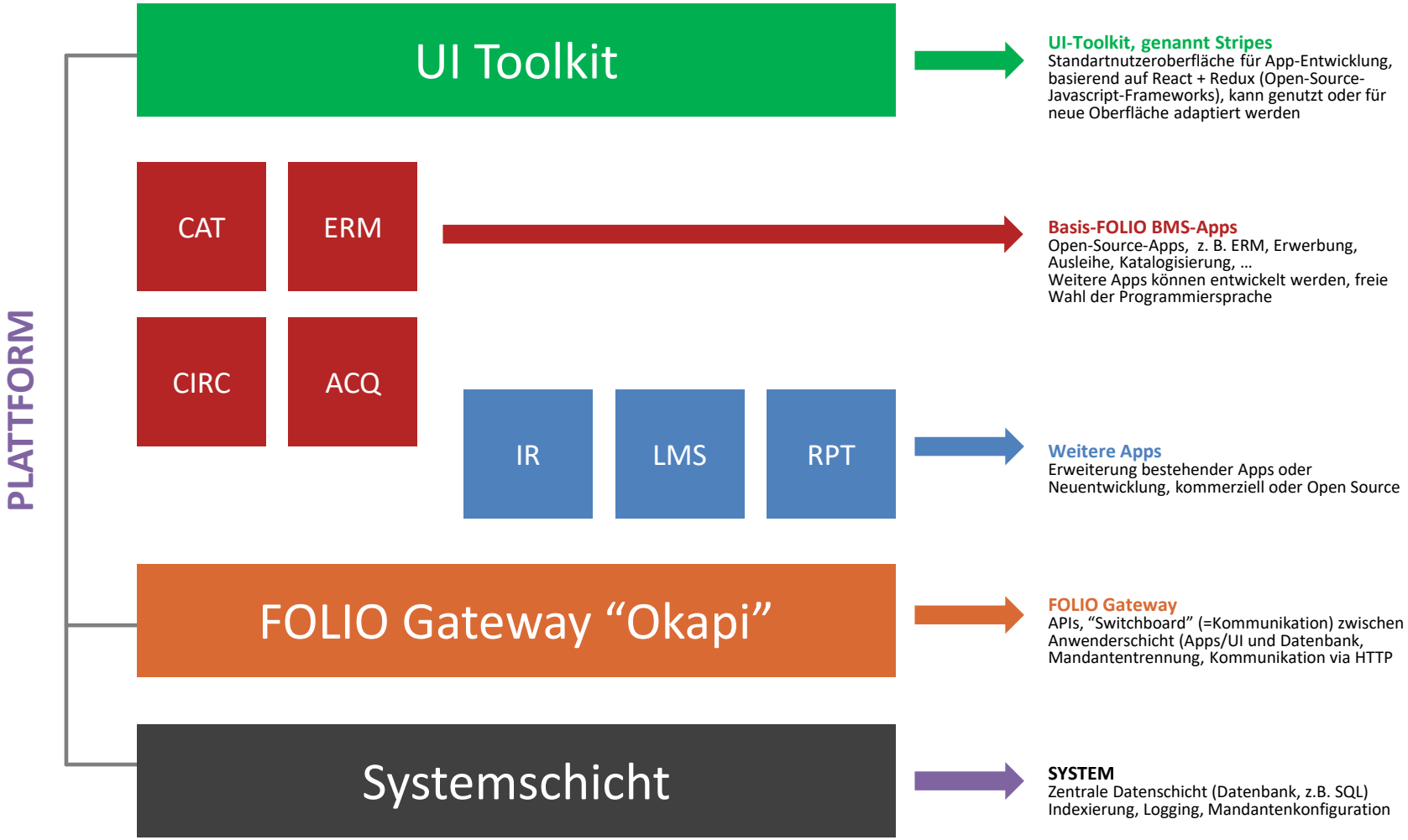
- Unterstützung verschiedener Support-Modelle
  - cloud-basiert, Hosting, lokal
  - kommerziell, Verbund, selber
- Mandantenfähig
- Flexibel erweiterbar, modular
- „Plug and Play“-Applikation
- Basierend auf heutigen Anforderungen mit Ausrichtung auf zukünftige Bedürfnisse

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



Technologien

Moderner Softwarestack
aus bewährten Komponenten

Frontend (= im Browser)

- JavaScript
- 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 http://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/blob/master/README.md

vert.x

- Bibliothek für Java
- Ermöglicht einfache Nebenläufigkeit
  - Umgeht Probleme paralleler Programmierung
  - Asynchrone Kommunikation
  - Reaktive Programmierung
- http://vertx.io/

RAML

- RAML = RESTful API Modeling Language
- Schnittstellenbeschreibung der Module
- Daraus wird automatisch erzeugt:
  - Schnittstellendokumentation, siehe http://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

Datenbank

- PostgreSQL
  - Index Data hat 2016 Beispielimplementierung mit MongoDB und PostgreSQL durchgeführt
- Wahl fiel auf PostgreSQL. Grund: unterstützt
  - 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.

Mandantentrennung

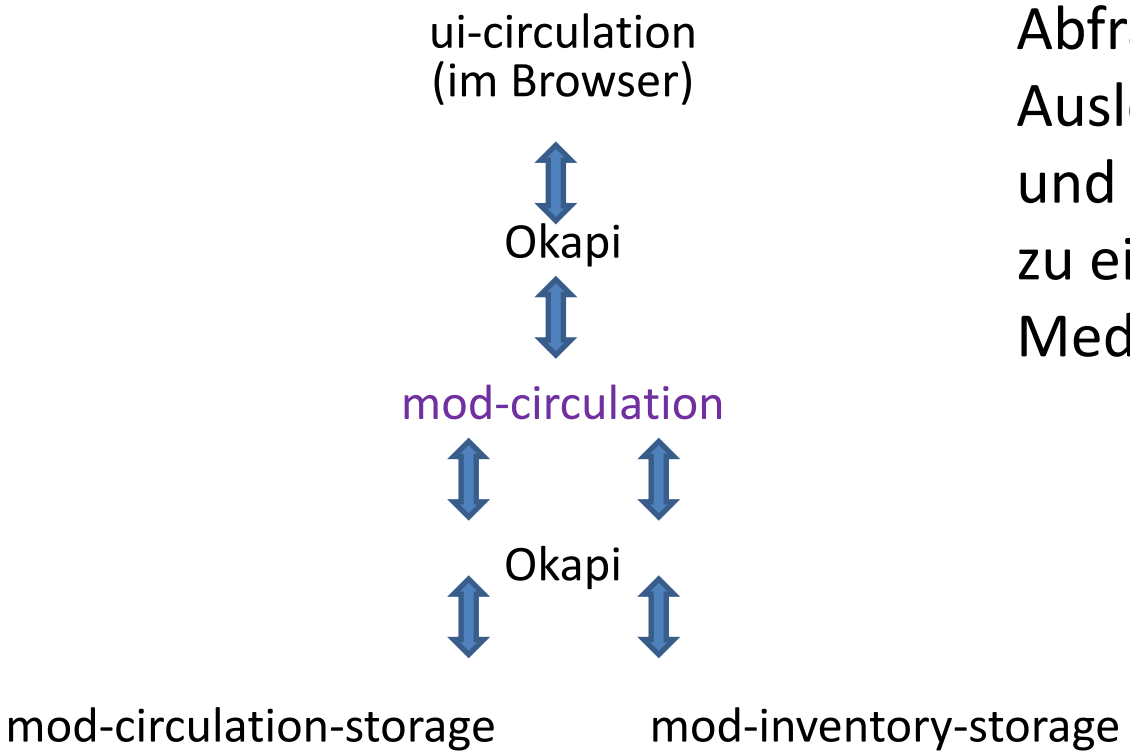
- Mandant = völlig unabhängige Institution (Institutsbibliothek kein Mandant, sondern hat granulare hierarchische Zugriffsrechte)
- Je Mandant eine eigene logische Datenbank (PostgreSQL „Schema“) mit eigenem Datenbanknutzer (PostgreSQL „Role“)
- Datenbankverbindung mit Datenbanknutzer, PostgreSQL garantiert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

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.

Intermodulkommunikation

Beispiel Ausleih-App mit drei Ausleih-Modulen:



Abfrage von Ausleihdaten und Titeldaten zu einem Medium.

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

Dockermodule

```
$ vagrant init folio/stable
$ vagrant up
$ vagrant ssh
$ docker ps
IMAGE                                          PORTS
folioorg/mod-circulation:v2.1.0                0.0.0.0:9140->9801/tcp
folioorg/mod-circulation-storage:v1.0.3        0.0.0.0:9139->8081/tcp
folioorg/mod-inventory:v4.4.0                  0.0.0.0:9138->9403/tcp
folioorg/mod-inventory-storage:v4.2.0          0.0.0.0:9137->8081/tcp
folioorg/mod-users-bl:v1.0.2                   0.0.0.0:9136->8081/tcp
folioorg/mod-users:v12.0.0                     0.0.0.0:9135->8081/tcp
folioci/mod-configuration:0.2.0-SNAPSHOT       0.0.0.0:9134->8081/tcp
folioorg/mod-login:v3.0.0                      0.0.0.0:9133->8081/tcp
folioorg/mod-permissions:v4.0.1                0.0.0.0:9132->8081/tcp
folioorg/mod-authtoken:v0.4.0                  0.0.0.0:9131->8081/tcp
stripes                                        0.0.0.0:3000->80/tcp
```

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

Business-Logic-Modul, 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 9131-9140.

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)

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

Vagrant und Docker

- Derzeit startet jedes Storage-Modul eine eigene PostgreSQL-Instanz
  - ➢ Nützlich für die Softwareentwicklung.
- Jedes Storage-Modul kann aber per Parameter an eine externe PostgreSQL-Installation angebunden werden.
  - ➢ Ermöglicht Hochverfügbarkeit und Replikation durch PostgreSQL-Cluster.

Vagrant und Docker

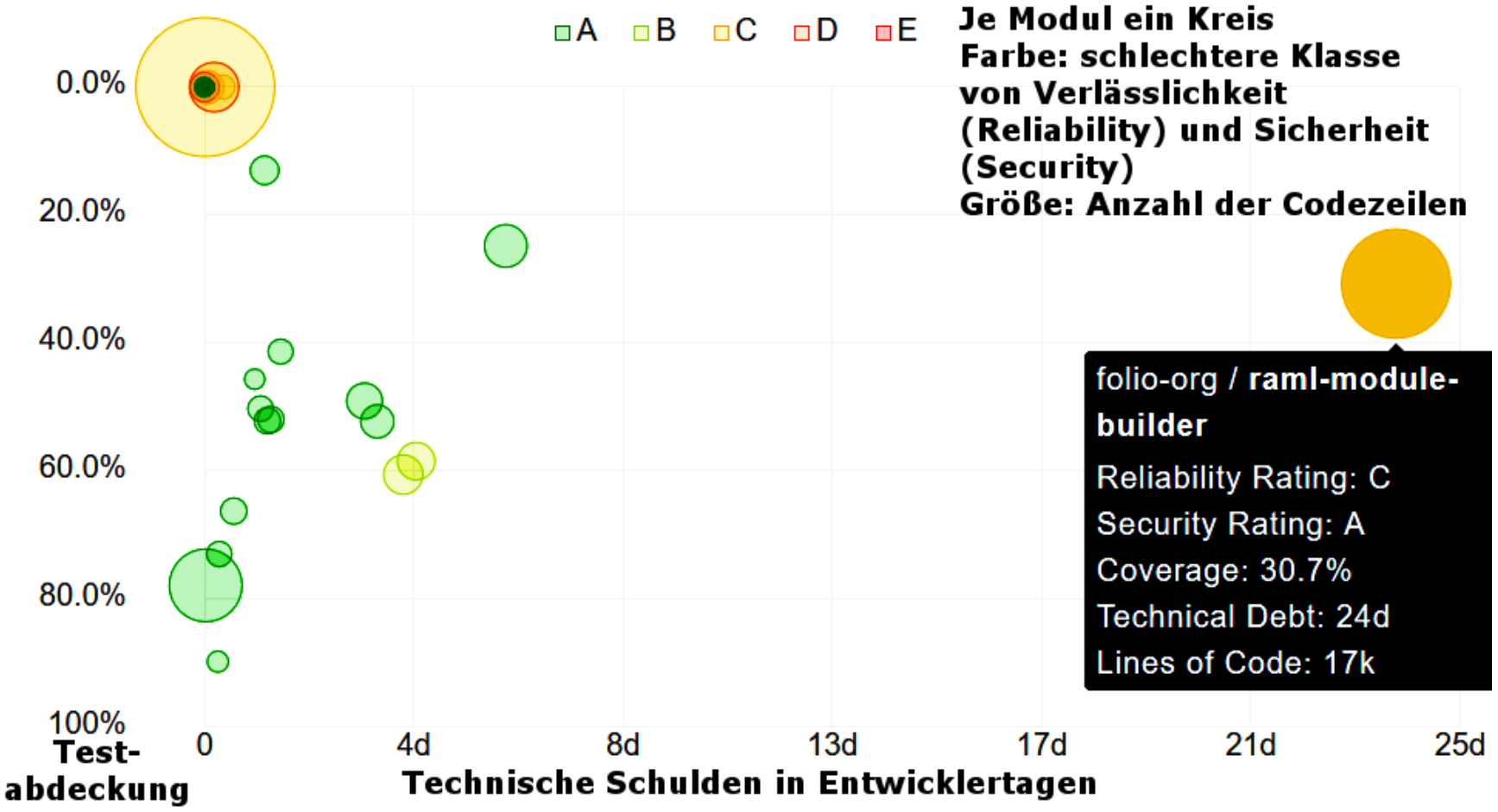
- Komplettes FOLIO-System als Vagrantbox
  - https://app.vagrantup.com/folio
  - https://github.com/folio-org/folio-ansible
- Einzelne Module als Dockercontainer
  - https://hub.docker.com/u/folioorg/
  - https://hub.docker.com/u/folioci/

Softwareentwicklung

http://dev.folio.org/

- Einstiegsseite für Entwickler
- Zugang zum Code
- Richtlinien
- Dokumentation
- Tutorial/Curriculum
- Komponenten und Mitarbeiter

Codeanalyse der Module



https://sonarcloud.io/organizations/folio-org/projects?view=visualizations&visualization=risk

Codeanalyse

- Automatische statische Codeanalyse
- SonarQube auf Sonarcloud.io
- Seit einem Monat in Betrieb, seitdem teils erhebliche Codeverbesserungen
- Ziel: Mindestens 80 % Codeabdeckung durch Modultests, weil das wirtschaftlich ist
- Tests der Frontendmodule vorhanden, aber noch nicht auf Sonarcloud, deshalb 0 %.

Softwareentwicklung

Team	Derzeitiger Fokus
Kernteams	User Management, Local Inventory, Resource Access, FOLIO Platform
Qulto	Single-Sign-On (Shibboleth), User Import, Kalender
Austin Team	eHoldings
Stacks	Acquisitions
UNAM	Fees and Fines (within Resource Access)

Kernteams – App-Entwickler

Index Data und verbundene Unternehmen

- Mike Taylor
- Kurt Nordstrom
- Jason Skomorowski
- Niels Erik (NE) Gilvad Nielsen
- Marc Johnson

OLE-Community

- Clint Bellanger – Auburn University
- Mark Stacy – University of Colorado
- Matt Connolly – Cornell
- Zak Burke – Cornell
- Michal Kuklis – Cornell
- Frances B. Webb – Cornell
- Julian Ladisch – GBV
- Jeremy Huff – Texas A&M
- William Welling – Texas A&M

Kern-Teams – Plattform und Infrastruktur

Backend/Kern

- Adam Dickmeiss
- Shale
- Heikki Levanto
- Mike Taylor
- Kurt Nordstrom
- Marc Johnson

Weitere

- John Coburn – UI components
- Charles Ledvina – Integration + Test
- John Malconian – Dev ops
- Wayne Schneider – Dev ops
- Jakub Skoczen – Leiter der Entwicklung und Product Owner der Plattform

App-Entwickler

Qulto Team

- Katalin Lovagné Szűcs – Product Owner
- Robert Sass
- Zoltán Erdős

Austin Team

- Charles Lowell – Tech Lead
- Jeffrey Cherewaty – UX Lead
- Joe LaSala
- Wil Wilsman
- Elrick Ryan

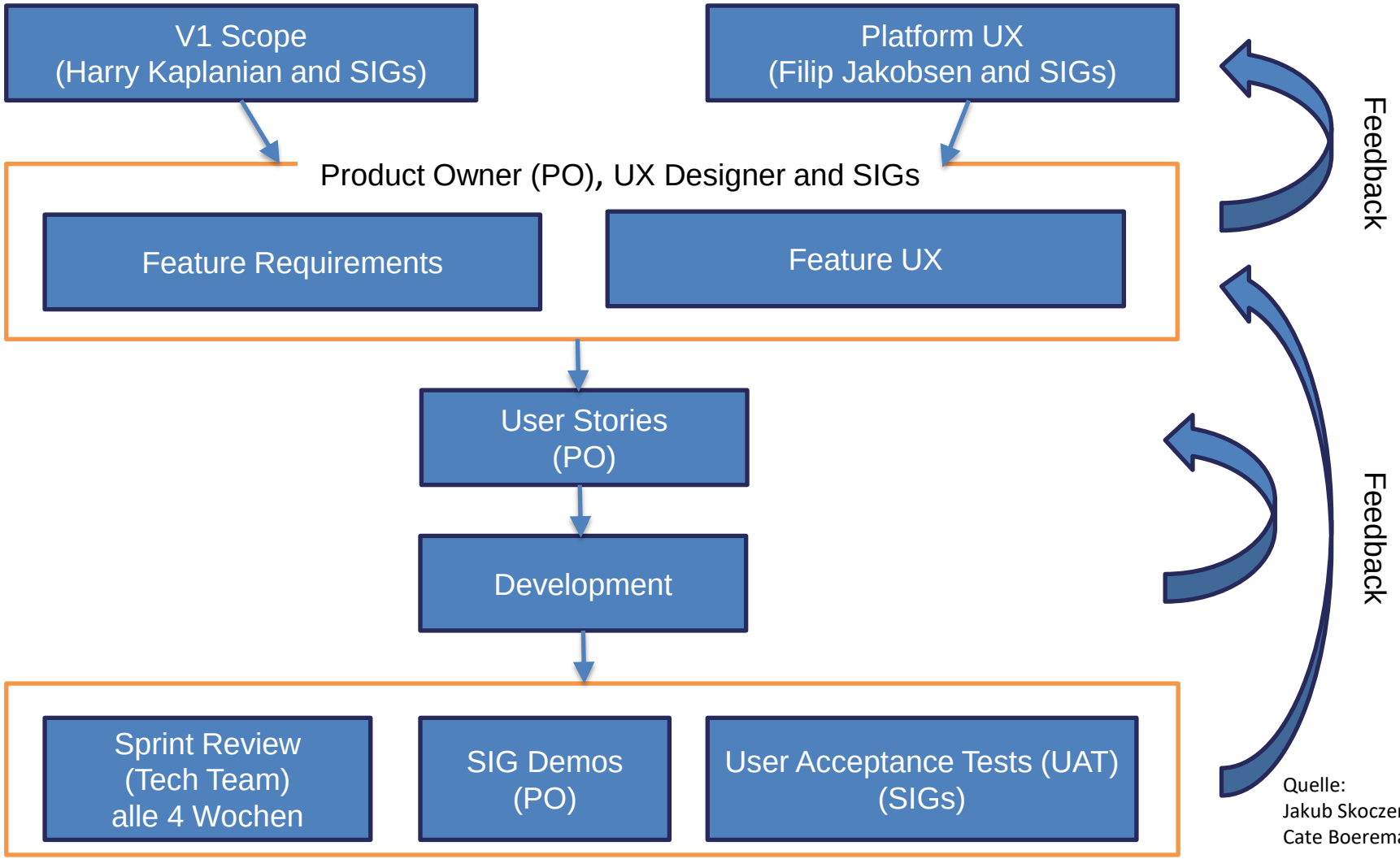
Stacks

- JD Benito – Senior Applications Specialist
- Kevin Horeck – UI Developer and UX Lead

Produkt- und UX-Leiter

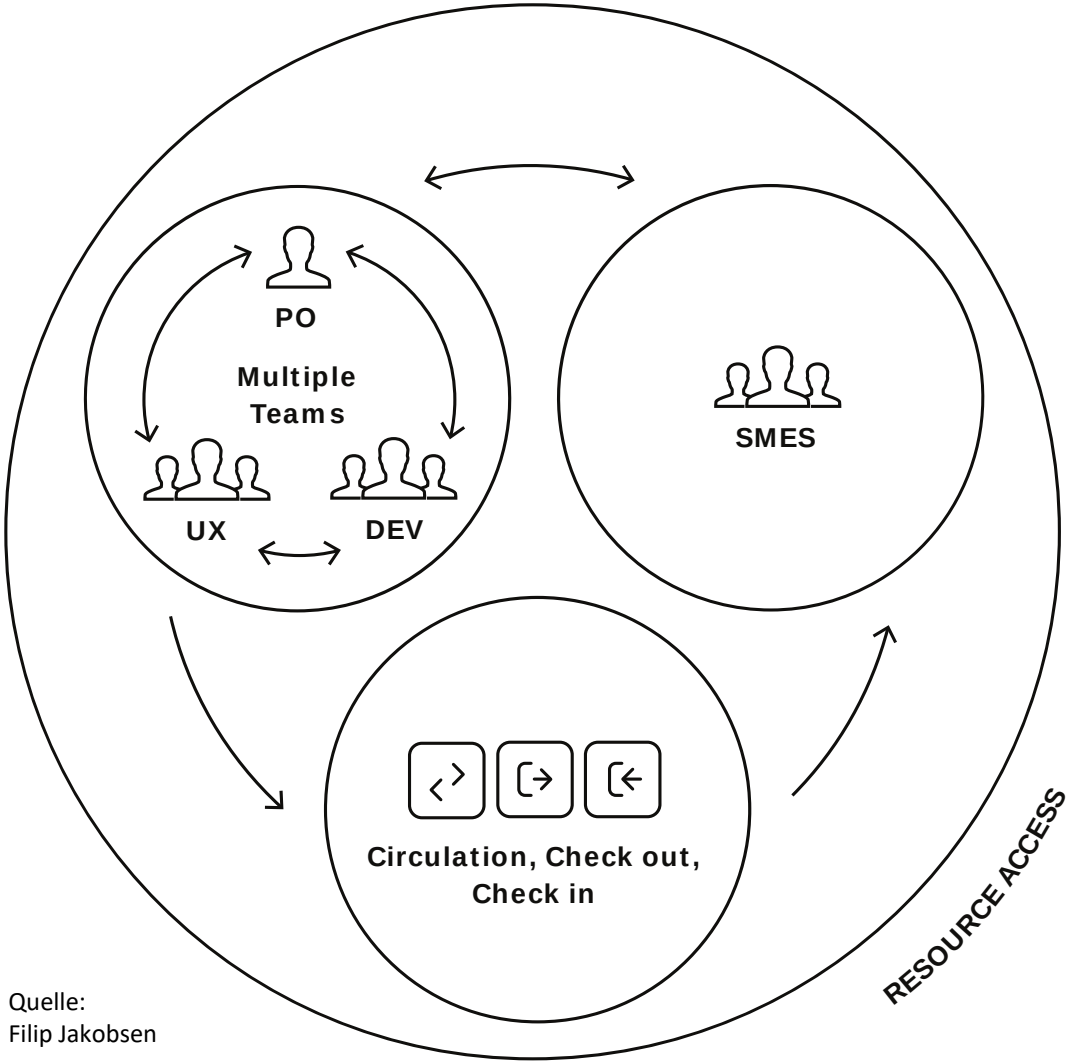
- Produkt
  - Cate Boerema – PO Lead and Core Team
  - Charlotte Whitt – Core Team
  - Holly Mistlebauer – UNAM and Core Team
  - Khalilah Gambrell – Austin Team
  - Katalin Szűcs – Qulto
  - Dennis Bridges – Stacks
  - Tania Fersenheim – Core Team
- UX (= User Experience)
  - Filip Jakobsen – UX Lead and Core Team
  - Kimie Kester – UNAM and Core Team
  - Darcy Branchini – Core Team
  - Kevin Horek – Stacks
  - Jeffrey Cherewaty – Austin Team

Prozess- und Teamstrukturen



Quelle:
Jakub Skoczen,
Cate Boerema

Prozess- und Teamstrukturen



Quelle:
Filip Jakobsen

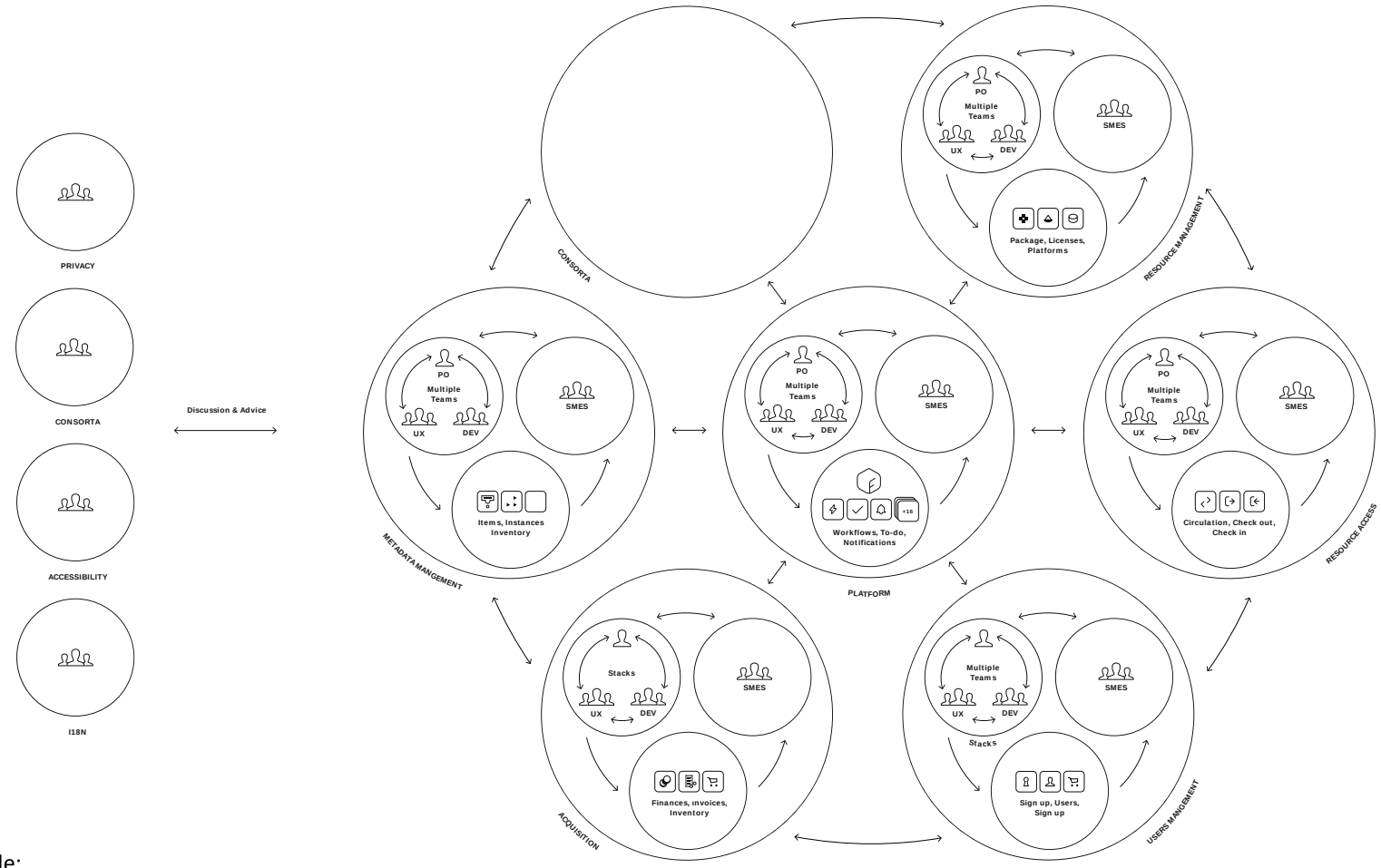
SMES = Subject Matter Experts aus den Bibliotheken

UX = User Experience Designer

DEV = Developer = Softwareentwickler

PO = Product Owner, Produktverantwortlicher und Leiter des Moduls, koordiniert SMES, UX und DEV

Prozess- und Teamstrukturen



Quelle:
Filip Jakobsen

Vielen Dank!

Julian Ladisch
julian.ladisch@gbv.de