Sonderdruck aus OBJEKTspektrum 06/2017

Visual Analytics für Wissensmanagement

Praxisbericht aus der Bundesagentur für Arbeit

Wissen entwickeln, teilen und sichern ist in vielen Organisationen eine große Herausforderung. Um das eigene Wissen besser zu verwalten, hat das Design&Implementierungsteam (D&I-Team) der IT-Anwendung ALLEGRO der Bundesagentur für Arbeit ein neues, auf einer modernen Architektur basierendes System für das Wissensmanagement konzipiert: Es visualisiert das Know-how und umfasst Frühwarnkomponenten, um Engpässe von Wissensträgern oder eine bevorstehende Kluft zwischen Bedarf und Vorhandensein von Wissen rechtzeitig zu erkennen.

Die Bundesagentur für Arbeit (BA) entwickelt und betreibt seit über sieben Jahren ein geschäftskritisches Softwaresystem, über das täglich hunderttausend Transaktionen laufen. Es gewährleistet die sichere Abwicklung von über 25 Milliarden Euro pro Jahr. Das System ALLEGRO [Wiki] umfasst mehr als 740.000 Codezeilen, das Entwicklungsteam besteht aus nicht weniger als 70 Entwicklern.

Eine wesentliche Herausforderung in diesem Projekt (man sagt bei der BA IT-Verfahren) ist, dass Experten zwar hochspezialisierte Kenntnisse zu den Implementierungsdetails des zweiten Sozialgesetzbuches (SGB II) aufgebaut haben, sie ihr Wissen jedoch nur mit hohem Aufwand weitergeben können. Deshalb hat das IT-Verfahren ALLEGRO der BA ein neues Wissensmanagementsystem konzipiert, das sich auf die Visualisierung des Wissens konzentriert und Frühwarnkomponenten, etwa über immer größer werdende Wissensinseln, Engpässe von Wissensträgern sowie schwach oder gar nicht abgedeckte Wissensgebiete, umfasst.

Dieses Pilotsystem setzt auf Big-Data-Visualisierung, visuelle Analytik sowie Mustererkennung. Letztere hat vor allem gefährdeten Code im Sinne des Wissensmanagements im Blick, aber auch die Wissensisolation, die Bedrohung durch Wissensverlust sowie bereits verlorenes Wissen.

Die Wissensmanagementanforderungen einer großen Behörde sind außergewöhnlich hoch, da die Flüsse von externem Fachwissen volatil und die Wissensströme wenig stabil sind. Das Pilotsystem kann die Veränderung von Wissen über große Zeiträume analysieren und visualisieren. Im Mittelpunkt stehen hier das Volumen, der Durchfluss und die Nutzung des Wissens.

Die Herausforderung

Hochkritische und wissensintensive Projekte, in deren Kontext auch externe Mitarbeiter Wissen aufbauen und entscheidende Wissensflüsse von außen kommen, haben einen besonders hohen Anspruch an das Wissensmanagement. Nur so können die Verantwortlichen die Volatilität der Wissensflüsse in den Griff bekommen. Dabei setzen sich die Teams in der Regel aus einem Mix von Experten mit Spezialwissen zusammen, die nicht ad hoc ersetzbar sind. Zudem sind die Anforderungen an das Wissensmanagement insbesondere in der Anlauf- und Endphase von Projekten besonders hoch, denn: Das Wissen von Spezialisten, die das Projekt verlassen, muss sicher in explizites Wissen transformiert oder auf andere künftige Wissensträger übertragen werden. Genauso ist es notwendig, dass neue Wissensarbeiter im Projekt einen schnellen, direkten und vernetzten Zugang zum benötigten Wissen bekommen.

Ein typischer Dialog, den sicher viele Leser aus der eigenen Arbeitspraxis kennen, startet mit der Frage: „Wer kennt sich denn mit der Komponente A44 aus?“ Nicht selten bekommt man dann zu hören: „Das war Klaus Meyer. Er hat sie nicht nur konzipiert, sondern auch selbst umgesetzt.“ Auf die Frage, wie man ihn denn erreichen könne, erfährt der Frager dann: „Er ist leider nicht mehr im Projekt! Und außer ihm war kein anderer Kollege maßgeblich eingebunden.“

Bestimmtes Wissen liegt nur noch in Form von Artefakten vor, die lange nicht

Anforderung / Gestaltungseinfluss	Design-Entscheidung
Hohe Performanz und Skalierbarkeit	Verwendung einer spaltenorientierten NoSQL-Datenbank (Cassandra)
	Horizontale Skalierung via isolierte Container (Docker)
Hohe Portabilität, Robustheit und Wartbarkeit	Leichtgewichtiger Servletcontainer Jetty
	Einweggebrauchsprinzip via Embeddedness- und Containeransatz
	Embedded: Einbettung von Jetty und Cassandra in die Anwendung
	Container: Verwendung des De-facto-Standards Docker als offene, verbreitete und portable Containertechnologie
Online-/Serviceorientierung und Mobilfähigkeit	Mobilfähige Webanwendung auf Basis von AngularJS 4
	Einsatz von D3.JS für webbasierte Datenvisualisierung
	Einsatz von Jetty, einem JSR 315-Container: Die meisten JAX-RS-Implementierungen nutzen den Komfort des Servlet 3.0 API
	Jersey ist eine solche Implementierung, die JSR 339 umsetzt und somit JAX-RS 2.0 unterstützt. Sie wird ebenfalls verwendet

Tabelle 1: Anforderung, Gestaltungseinfluss und Design-Entscheidung bei der Entwicklung des Pilotsystems

verändert oder aktualisiert wurden. Das Wissen zu damit zusammenhängenden Entscheidungen ist damit verloren – wie etwa zum Design und zum Hintergrund für bestimmte Funktionsweisen. Und ein gesamthaftes Nachvollziehen bedeutet Aufwand und ist nicht immer von Erfolg gekrönt.

Die Entwicklung eines Wissensmanagement-Pilotsystems

Aus diesem Grund haben wir ein Pilotsystem konzipiert, das genau an diesem Punkt ansetzt. Mit Blick auf das Wissensmanagement soll es Antizipation, Prävention sowie eine ganzheitliche Sicht und Analytik erlauben. Das bedeutet konkret:

- *Antizipation:* Das Pilotsystem kann Wissensengpässe vorhersagen sowie isoliertes und künftig benötigtes Wissen erkennen.
- *Prävention:* Das System beugt kritischen Wissenslagen vor und hebt bei drohendem Wissensverlust (z. B. durch Beendigung des Projekteinsatzes von Spezialisten) betroffene Wissensbereiche und potenzielle künftige Wissensträger visuell hervor.
- *Ganzheitliche Sicht und Analytik:* Die historische Entwicklung von kritischem Wissen und dessen Entwicklungsgeschwindigkeit macht das Pilotsystem nachvollziehbar. Zudem zeigt es auf, welche Bereiche sich rasant und welche sich kaum entwickeln.

Bei diesen Analytik-orientierten Anforderungen galt es für uns, eine passende Architektur zu entwickeln, die den Ansprüchen nachhaltig standhält. Darüber hinaus muss sie mit großen Datenmengen umgehen und diese hochperformant verarbeiten sowie visualisieren können. An dieser Stelle kommen wir zu einer entscheidenden Frage: Wie soll die Architektur des Pilotsystems aussehen und worauf basieren die Design-Entscheidungen?
Drei wesentliche Faktoren haben die Design-Entscheidungen bei der BA stark geprägt:

- die nichtfunktionalen Anforderungen Performanz und Skalierbarkeit,
- das Prinzip der Zwölf-Faktoren-App [12FA] als Methode für die Konstruktion hochportabler, robuster und wartbarer Anwendungen und
- die Online-/Serviceorientierung als Bestandteil der digitalen Agenda der BA.

Die hohen Performanz- und Skalierbarkeitsanforderungen sind ohne Weiteres

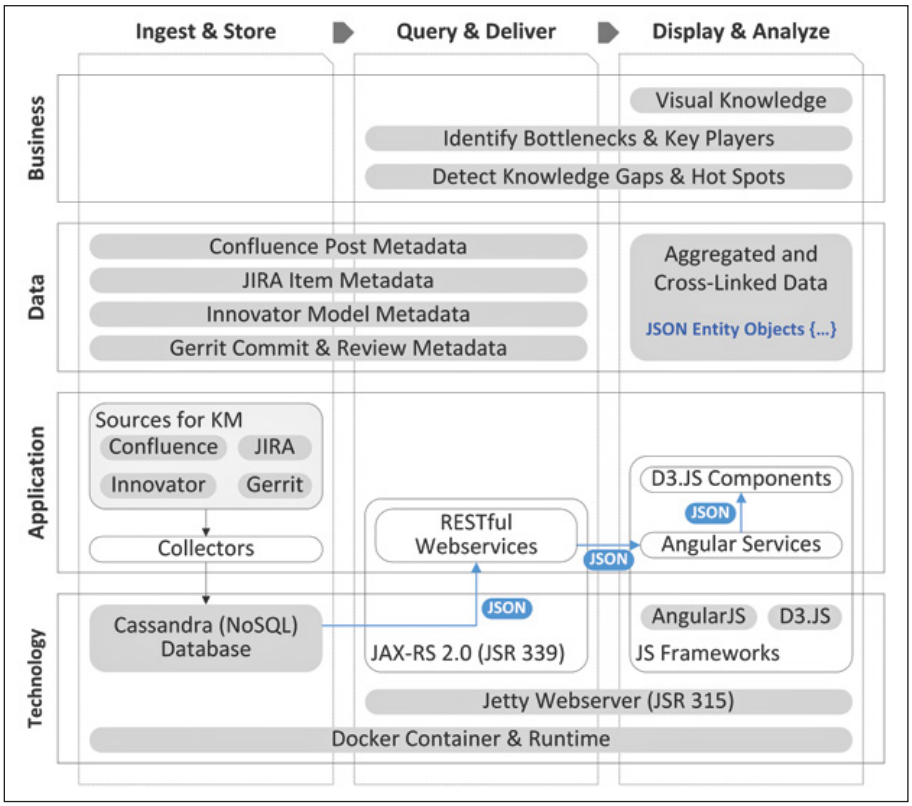


Abb. 1: Gesamtarchitektur des analytischen Wissensmanagementsystems

nachvollziehbar – sie gehen auf die schiere System- und Projektgröße zurück. Damit das Pilotsystem eine ganzheitliche Sicht und Analytik auf das gesamte im Projekt vorhandene und entstehende Wissen herstellen kann, muss es mit dem großen und wachsenden Volumen an Daten- und Informationen umgehen können. Nur so werden auch antizipative und präventive Wissensmanagementfähigkeiten verfügbar.
Der zweite Faktor steht mit dem ersten in Verbindung, da insbesondere die horizontale Skalierbarkeit auch das Prinzip der Zwölf-Faktoren-App schafft. Es erfüllt jedoch auch mit dem Teilprinzip „Einweggebrauch" die Robustheit, Wartbarkeit und Portabilität. Diese nichtfunktionalen Anforderungen sind für das Wissensmanagement-Pilotsystem relevant, weil die Portierbarkeit zwischen Ausführungsumgebungen dafür sorgt, dass es vom Betriebssystem unabhängig ist. So können die Entwickler beispielsweise auf Windows-Rechnern entwickeln, während die produktive Software auf Linux-Systemen läuft. In der Kritikalität ihrer Projekte begründet, sind bei der BA die Robustheit und Wartbarkeit als grundsätzliche Anforderungen in der Softwareentwicklung verankert.
Der dritte Faktor ist die in der digitalen Agenda der BA festgeschriebene Online- und Serviceorientierung. Damit verfolgt die BA das Ziel, möglichst webbasierte und geräteunabhängige Anwendungen bereitzustellen, auf welche die Nutzer leicht über Mobilgeräte zugreifen können. Damit werden einerseits alte Desktop-Rechner obsolet und andererseits die Arbeitsplätze von BA-Mitarbeitern mobil. Dieser Aspekt wird besonders dann wichtig, wenn die BA das pilotierte Wissensmanagementsystem bundesweit etablieren möchte. Dann kann zum Beispiel ein Kollege in der Zentrale einen Experten an anderen Standorten, sogenannten Liegenschaften, ausfindig machen.
Das Systemdesign des Pilotsystems greift diese nichtfunktionalen Anforderungen und architekturbezogenen Einflusstreiber auf und ist entsprechend gestaltet. **Tabelle 1** gibt einen Überblick über alle Design-Entscheidungen, die wir auf die genannten Anforderungen abgestimmt haben und durch die architekturbezogenen Treiber beeinflussten. Dies reflektiert die Architekturgrafik in **Abbildung 1**.

Performanz und Skalierbarkeit mit NoSQL-Datenbanken

Die Datenbasis für das analytische Wissensmanagement, die das Pilotsystem bewerkstelligen soll, ist sehr groß. In den mehr als 740.000 Codezeilen steckt das auszuwertende Wissen. Letzteres haben Entwickler mit ca. 50 000 Commits an-

gereichert, also neuen Quelltext oder Dateien im Versionsverwaltungssystem. Und täglich kommen neue hinzu. Ein hiesiges Modell-Repository (MID Innovator) sowie JIRA-, Confluence- und Wiki-Seiten bilden weitere Datenquellen für die Wissensauswertung. Das bedeutet, dass die Datenquellen zusätzlich sehr heterogen sind.

Nun stellt sich folgende Frage: NoSQL ist schön und gut – aber warum braucht es eine spaltenorientierte Datenbank und warum Cassandra?

Im NoSQL-Umfeld existieren neben spaltenorientierten auch Graphendatenbanken, die zur Abbildung von Wissensstrukturen sogar naheliegender zu sein scheinen. Außerdem gibt es dokumentenorientierte Datenbanken und Schlüssel-Werte-Datenbanken (Key-Value Stores). Bei dieser Design-Entscheidung mussten wir zwischen der Eignung für Datenvolumen und -komplexität abwägen:

- *Schlüssel-Werte-Datenbanken* wie etwa Riak oder Redis bieten die einfachste Datenstruktur, eignen sich aber an erster Stelle für sehr große Datenvolumina.
- Danach folgen an zweiter Stelle *spaltenorientierte Datenbanken* wie Cassandra oder HBase, deren Datenstrukturen sich nicht auf Schlüssel-Werte-Paare beschränken und sich den relationalen Datenbanken annähern.
- *Dokumentenorientierte Datenbanken* à la Couchbase oder MongoDB bieten – was die Datenstruktur angeht – noch mehr Freiheitsgrade: Sie sind schemalos und verwalten Daten in XML- oder JSON-Dokumenten.
- Die komplexeste Datenstruktur bringen *Graphendatenbanken* wie beispielsweise Neo4J mit.

Im Gegenzug zur wachsenden Komplexität nimmt die Eignung von dokumentenorientierten und noch mehr von Graphendatenbanken für großvolumige Daten ab. Für unsere Zwecke ist der performante Umgang mit sehr vielen Daten wichtig und die Strukturierung durch spaltenorientierte Datenbanken sinnvoller als eine irreversible Festschreibung der kompletten Datenschicht des Wissensmanagementsystems durch eine Graphenorientierung. Dies würde auch die Gesamtarchitektur über alle Schichten hinweg bis zum Frontend maßgeblich mitbestimmen. Gewinnt die Graphenorientierung im Projekt künftig an Bedeutung, haben die Verantwortlichen mit der eher generisch orientierten Design-Entscheidung für Cassandra viele Gestaltungsmöglichkeiten, denn: Cassandra bildet bereits den Kern der Graphendatenbank DSE Graph von DataStax.

Für unsere Wissensmanagementzwecke benötigen wir Analysen wie die Summenbildung über einzelne Attribute. Hierzu gehört zum Beispiel die Anzahl von Klassen oder Commits in einem bestimmten, auf Package-Ebene definierbarem Wissensbereich. Für derartige Aggregate über viele Zeilen und nur einzelne beziehungsweise wenige Spalten sind spaltenorientierte Datenbanken nicht nur geeignet, sondern speziell konzipiert.

Eine weitere Eigenschaft, die für Cassandra spricht, ist der fortgeschrittene und teils mächtige JDBC-Treiber von DataStax für die reibungslose Verwendung in Java. Die Abfragesprache Cassandra Query Language (CQL) ähnelt SQL und ist als eine Art „SQL für Cassandra“ zu verstehen [Yara17].

Portabilität dank Embedded-Ansatz und Containertechnologie

Dadurch, dass wir den leichtgewichtigen Webserver Jetty und die Cassandra-Datenbank – beide eingebettet – mit der Containertechnologie Docker kombiniert haben, wird das gesamte Pilotsystem hochportabel. Auch der Build- beziehungsweise Installationsprozess wird hochautomatisiert.

Und an dieser Stelle kommt nun das Build-Management-Framework Maven ins Spiel, für das sehr ausgereifte Plug-ins wie zum Beispiel das Jetty-Maven-Plug-in von Eclipse und Mojos Cassandra-Maven-Plug-in existieren. Sie automatisieren eine Einbettung und die gesamte Installation.

Das Bündeln der gesamten Anwendung mitsamt Webserver und Datenbank in einem Docker-Container macht diese unabhängig von Hardware und Umgebung lauffähig. Auch können wir damit das System problemlos in einer Private Cloud bereitstellen.

In puncto Design-Entscheidungen muss auch hier eine Frage geklärt werden: Warum embedded Cassandra – steht die Einbettung nicht einer Clusterbildung entgegen?

Bei dieser Frage galt es für uns, zwischen zwei möglicherweise in Konflikt stehenden Vorteilen abzuwägen: Eine konsistente Umsetzung des Einweggebrauchsprinzips erforderte die Einbettung der Datenbank Cassandra. Ihre Einbettung in die Anwendung, die später in einem Container läuft, würde eine Clusterisierung möglicherweise erschweren. Schließlich müssen sich die eingebetteten Cassandra-Instanzen idealerweise per Automatismus erkennen und verbinden.

Abgesehen davon, dass dies mittels Discovery-Mechanismen auf Treiberebene möglich ist – wenn auch nur für Fortgeschrittene –, überwog in diesem Projekt der Vorteil des Einweggebrauchsprinzips. Die mit Cassandra gelieferte Spaltenorientierung und JSON-Unterstützung waren aus unserer Sicht deutliche Vorteile für das Vorhaben.

Mobilfähige Webanwendung mit REST, AngularJS und D3.JS

Die Webanwendung läuft in einem (mitgelieferten) JSR 315-konformen Servlet-Container, dem Jetty, der somit das Servlet 3.0 API unterstützt. RESTful Webservices setzen wir mit dem Framework Jersey um, das eine Referenzimplementierung des JSR 339 ist. Mittlerweile existieren sogar Abspaltungen, die den JSR 370, sprich JAX-RS 2.1, implementieren.

Das Frontend entwickeln wir mit dem Framework AngularJS (Version 4) von Google, das auf TypeScript 2 und damit auf Microsofts graduell typisierte JavaScript-Spracherweiterung baut. Es erlaubt den Einsatz von Vorlagen und bidirektionales Databinding, um auf Client-Seite eine saubere Model-View-Controller (MVC)-Trennung zu schaffen [Höll17, MCLT17].

Das Kompilieren und Bündeln von AngularJS-Artefakten bewerkstelligen wir mit dem Werkzeug Angular CLI, das wir ebenfalls in Maven und somit als Teilschritt im gesamten Erstellungsprozesses integriert haben. Die visuelle Analytik setzen wir mit dem JavaScript-Framework D3.JS um, das sehr gut in AngularJS integrierbar ist [LePo14]. Es erlaubt eine interaktive Datenvisualisierung und entspricht damit den Anforderungen an unser Wissensmanagementsystem.

Die abschließende Paketierung aller Webanwendungsartefakte einschließlich der REST-Services und der Angular-spezifischen Kompilate zu einem Web Application Archive (WAR) übernimmt schließlich für uns das Apache-Maven-WAR-Plug-in.

Fügen wir die Puzzlesteine zu einer Gesamtarchitektur zusammen

Wie sehen denn nun aber die Gesamtarchitektur und das pilotierte Wissensmanagementsystem aus? **Abbildung 1** stellt die Gesamtarchitektur des in der BA entwickelten Wissensmanagementsystems dar. Gut erkennen lassen sich die Synergien durch das Zusammenspiel der Teilsysteme. So kann die NoQSL-Daten-

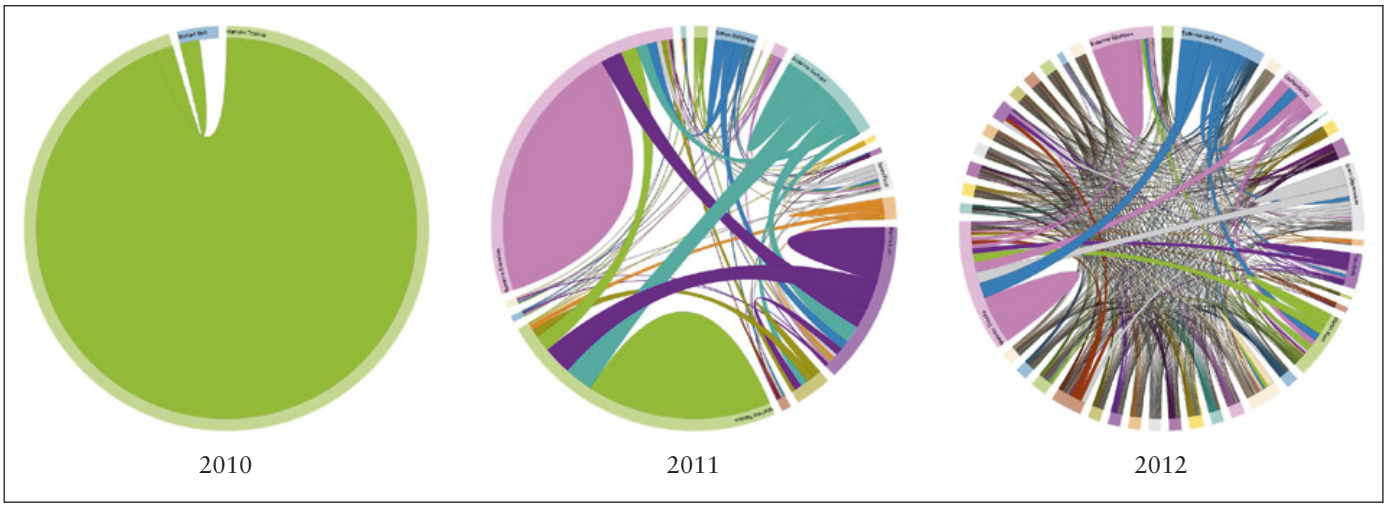


Abb. 2: Historisierte Ansicht für Wissensteilung

bank Cassandra ihre Ergebnisse bereits im JSON-Format zurückgeben, die der REST-Service einfach nur wiedergibt. Die entsprechende Servlet-Methode ist gemäß Servlet 3.0 API lediglich mit

@Produces (

MediaType.APPLICATION_JSON)

annotiert und gibt – wie in **Abbildung 1** dargestellt – das von der Cassandra-Datenbank erhaltene Ergebnis direkt zurück. Hierbei ist kein Objekt-Relationales Mapping (ORM) im Spiel und auch nicht nötig. Der auf Client-Seite implementierte Angular-Service nimmt das Ergebnis ebenfalls, ohne dass der Entwickler weitere Konvertierungen vornehmen muss, entgegen und bildet es mittels automatischer Typenumwandlung auf die entsprechende JSON-Entitätsklasse ab. TypeScript bietet hierfür eine stark vereinfachende „as"-Syntax an. Damit entfällt viel Verbindungscode, auch Glue Code genannt. Und der Datentransport sowie die Konvertierung erfolgen nach dem Konvention-vor-Konfiguration-Prinzip automatisch.

„Weniger ist mehr" – dieses Prinzip lebt beispielsweise SAP HANA vor. Dieses In-Memory-Datenbanksystem schlägt vor, die Geschäftslogik mittels JavaScript in SQL-Prozeduren zu formulieren und ebenfalls per JavaScript im Frontend aufzurufen. Das waren bereits die ersten Signale dafür, dass das Ende einer Ära der oft als obligatorisch empfundenen, vielschichtigen Architektur eingeläutet wurde. Eine exzessive Vielschichtigkeit erlaubt zwar, Frameworks auszutauschen – unabhängig auf welcher Ebene (Daten-, Zugriffs-, Business- oder Präsentationsschicht). Als Beispiele seien hier das ORM-Framework (Hibernate, OpenJPA, EclipseLink) oder eine Template-Engine (Velocity, Freemarker) genannt. Das hat jedoch nicht nur zur Folge, dass die Entwickler viel Verbindungscode schreiben mussten, sondern es kostete auch Systemleistung.

Heutzutage bilden sich in der unendlich erscheinenden Framework-Landschaft Schwergewichte wie etwa AngularJS heraus, die sich als De-facto-Standard durchsetzen und die Landschaft polarisieren. Viele Frameworks sind beispielsweise eine Abspaltung eines Schwergewichts und damit konform zu einem Standard, der sich durchgesetzt hat.

Nachdem wir uns mit der Architektur auseinandergesetzt haben, betrachten wir nun das Ergebnis unseres Pilotsystems. Die **Abbildungen 2 bis 4** zeigen Screenshots der Wissensteilungs-, Individualwissensentwicklungs- und Brennpunkt-Ansicht.

Gemeinsame Wissensentwicklung clustern und visualisieren

Über die Wissensteilungsansicht (**siehe Abbildung 2** können Entwickler, Projektmanager und Führungskräfte das entwickelte Wissen in gruppierten Wissensgebieten identifizieren und visualisieren. Dort sehen wir, wie das Team kollektives und vernetztes Wissen seit drei Jahren gemeinsam entwickelt hat.

Die äußere Kante des Kreises beinhaltet die Wissensträger, in diesem konkreten

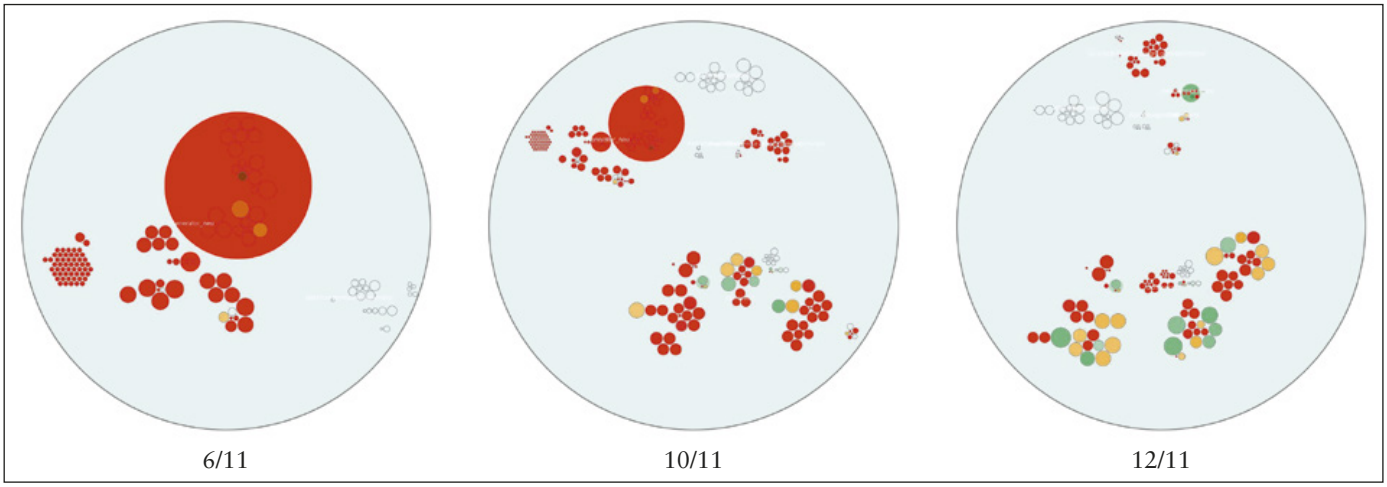


Abb. 3: Historisierte Ansicht für individuelle Wissensentwicklung

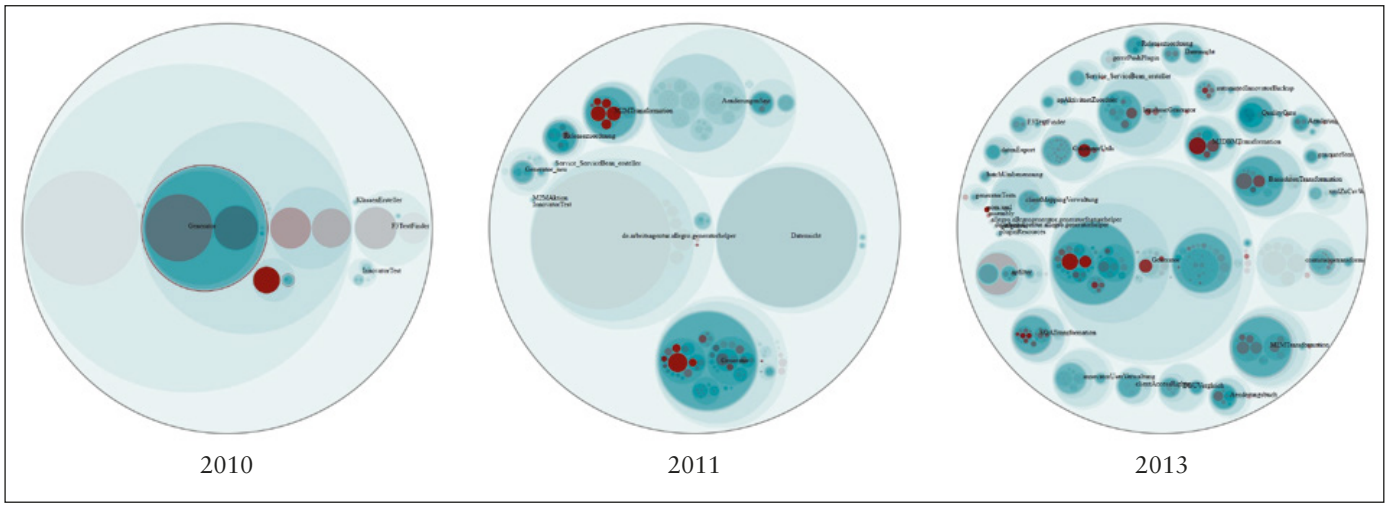


Abb. 4: Historisierte Ansicht für Brennpunkte

Fall die Entwickler des Quellcodes von ALLEGRO, der zur Leistungsberechnung von Arbeitslosengeld II eingesetzten Applikation in deutschen Jobcentern. Eine diagonale Linie zwischen zwei oder mehreren, auch Committern genannten Entwicklern, bedeutet, dass diese Personen in demselben Bereich effektiv zusammengearbeitet und kollokierte Codefragmente wie Klassen oder Methoden bearbeitet und committed haben.
Die in den Abbildungen dargestellten Ausbuchtungen stellen Wissensinseln dar, sprich nicht geteiltes Wissen. Das bedeutet, dass hier nur eine Person innerhalb eines Wissensgebiets Artefakte und Know-how aufgebaut hat.

Individuale Wissensgenerierung identifizieren und visualisieren
Die Ansicht zur Individualwissensentwicklung dient dem Identifizieren und Visualisieren des Wissens, das von einer Person entwickelt wurde. Im Zentrum steht das Individuum – wie in **Abbildung 3** dargestellt. Die Kreise repräsentieren das durch Commits ermittelte Individualwissen. Der Durchmesser ist dabei das Maß für die Anzahl geänderter Codezeilen in einem Wissensbereich. Kreise, die sich im gleichen Umkreis befinden, sind Klassen innerhalb desselben Java-Packages, das durch den umgreifenden Kreis repräsentiert wird.
Zur Farbgebung gilt zu wissen: Die Farbe Rot bedeutet, dass der Code nur von einer Person geschrieben wurde. Die Farbe Gelb bedeutet, dass mehrere Personen an diesem Code beteiligt sind – Wissen wird geteilt – und visualisiert somit kollektives Wissen. Die Farbe Grün bedeutet, dass viele Personen beteiligt sind und somit das Wissen ausreichend innerhalb eines Kollektivs geteilt ist.

Erkennen und Darstellen von Brennpunkten
Die Brennpunkt-Ansicht in **Abbildung 4** visualisiert sich stark entwickelnde Wissensgebiete. Dabei repräsentiert der Durchmesser das Volumen der Codezeilen in einem Wissensbereich. Die Farbe ist dunkler, je mehr Commits in diesem Bereich existieren. So lässt sich leicht erkennen, wie sich der Fokus von einem Bereich auf einen anderen verlagert, indem wir die Zeitachse nicht additiv visualisieren.
Von Vorteil ist, dass die Verantwortlichen jedes Jahr separat analysieren können, indem sie keine Commits aus dem Vorjahr einschließen. Eine weitere Brennpunkt-Metrik ist ein Quotient aus der Gesamtzahl der Commits und der Anzahl der Commits in einem Bereich. Dies dient der Analyse und Darstellung von Kausalwissen.
Wird die Farbe dunkler, der Kreis aber nicht größer, dann deutet dies auf hohe Qualitätssicherungsanforderungen in diesem Wissensbereich hin, weil dort ständig Änderungen stattfinden, sich der Quellcode aber nicht vergrößert – es handelt sich also um ständige Revidierungen (und nicht Erweiterungen).

Erkennen von Hauptakteuren und Engpässen
Die Analyse von durchsuchten Dokumenten, Commit-Historien und Codereviews liefert eine Übersicht über die wichtigsten Personen, die stillschweigendes oder explizites Wissen in bestimmten Bereichen aufgebaut haben. Auf diese Weise identifizieren wir Experten und wichtige Akteure sowie Engpässe.

Identifikation der Lücke zwischen Wissensnachfrage und -angebot
Die Auswertung von Suchanfragen, deren Häufigkeit beziehungsweise die Häufigkeit gesuchter Schlüsselwörter und Anzahl von Suchergebnissen im organisationsweiten Wiki, Confluence und Wissensportal gibt Aufschluss über ein Delta zwischen Wissensangebot und -bedarf. Daran erkennen wir im IT-Verfahren ALLEGRO, in welchen Wissensbereichen wir nachjustieren müssen.
Zusätzlich macht ein Vergleich von Git mit Confluence die Kluft zwischen stillschweigendem Wissen und explizit niedergeschriebenem oder dokumentiertem Wissen deutlich. Wenn viel in einem Wissensbereich beigetragen wird, sollte sich das auch in Confluence widerspiegeln. Aus einer solchen Kluft leiten wir ab, dass Nachdokumentationen nötig und gegebenenfalls auch schon überfällig sind.

Identifikation von Wissen, das sich über verschiedene Teams verteilt
Hierbei geht es darum, Wissen bereichsspezifisch zu identifizieren, das für unterschiedliche Teams wie Fachkonzept, Entwicklung und Test relevant ist oder sein wird. Dies geschieht zum Beispiel durch das automatisierte Durchsuchen von Dokument-Metadaten in der Dateifreigabe, die häufig vom Fachkonzept-, Test- und Entwicklungs-Team verwendet werden. Anschließend werden Verbindungen dieser Metadaten mit den Commits im Git-Repository durch Suche nach Übereinstimmung von Schlüsselwörtern identifiziert.
So können wir für die nächste Version Engpässe oder nicht abgedeckte Wissensbedarfe im Entwicklungsteam vorhersagen. Das System extrahiert vielfältige Wissensgebiete aus den Fachkonzeptartefakten wie etwa den Modellen und Konzeptdokumenten und gleicht diese mit den Artefakten und Wissensträgern im Entwicklungsteam ab.

Neu entstehendes Wissen erkennen und anzeigen

Durch die Analyse der generierten Inhalte und Inhaltshistorien im internen Wiki, Confluence und Wissensportal erzeugt unser Wissensmanagementsystem ein Gesamtbild der Wissensentwicklung – in vorhandenen und vor allem neuen Gebieten, die für verschiedene Teams relevant sein werden. Hieran erkennen wir die Dynamik in der Wissensentstehung. Das Erkennen neuen Wissens beschränkt sich nicht nur auf die Inhaltsanalyse, sondern beinhaltet auch die Analyse von Modellen, die vom Fachkonzeptteam mit der Modellierungsplattform Innovator erstellt werden.

Personalplanung verbessern

Weil unser Wissensmanagementsystem in der Lage ist, Wissensgebiete hervorzuheben, die unzureichend abgedeckt sind, kann Nachwuchs in solchen Bereichen gezielt gesucht und eingestellt oder qualifiziert und für bestimmte Aufgaben eingesetzt werden.

Graphbasierte Wissensabfragen

In unserer Cassandra-Datenbank ist das gesamte ALLEGRO-Wissen importiert und gespeichert. So können alle interessierten Mitarbeiter von der Führungskraft bis zum Wissensmanager oder Entwickler Wissensabfragen nach spezifischen und miteinander verbundenen Wissensobjekten formulieren. Beispielsweise ist es möglich, Wissensträger zu suchen einschließlich ihrer Verbindungen beziehungsweise Personen, mit denen sie ihr Wissen teilen.

Resümee

Moderne Softwarearchitekturen erfüllen keinen Selbstzweck, auch wenn beispielsweise aktuell relationale Datenbanksysteme noch rund 90 Prozent der Anwendungsfälle für einen Datenbankeinsatz abdecken. Das vorgestellte Projekt zeigt konkret, wo die Vorteile einer modernen Architektur liegen und wie diese in der Praxis umgesetzt wurden.

Gerade im öffentlichen Sektor ist der Einsatz zukunftssicherer und skalierbarer Architekturen von großer Bedeutung, denn: Klassischerweise arbeiten die Verantwortlichen bei einer Migration – spätestens auf Datenebene – mit sehr vielen und teils auch kritischen Daten. Zudem sind Migrationen von Großsystemen, wie in dem Verfahren ALLEGRO, in der Regel mit hohem Aufwand verbunden. ||

Literatur & Links

[12FA] A. Wiggins, The Twelve-Factor App, siehe: **https://12factor.net/de/**

[Höll17] Ch. Höller, Angular: Das umfassende Handbuch, Rheinwerk Verlag, 2017

[LePo14] A. Lerner, V. Powell, D3 on AngularJS: Create Dynamic Visualizations with AngularJS, Kanada: Leanpub, 2014

[MCLT17] N. Murray, F. Coury, A. Lerner, C. Taborda, ng-book 2: The Complete Guide to Angular, USA: Fullstack.io, 2017

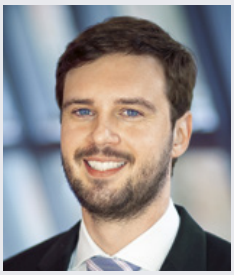
[Wiki] „Arbeitslosengeld II Leistungsverfahren Grundsicherung Online", siehe: **https://de.wikipedia.org/wiki/ALLEGRO#Verwaltungssoftware**

[Yara17] S. Yarabarla, Learning Apache Cassandra, Packt Publishing Ltd., 2017 (zweite Auflage)

Die Autoren

Dr. Eldar Sultanow
(eldar.sultanow@capgemini.com) ist Architekt bei Capgemini. Seine Schwerpunkte sind moderne Softwarearchitekturen, Digitalisierung und Unternehmensarchitekturmanagement.

Marinho Tobolla
(marinho.tobolla@arbeitsagentur.de) arbeitet seit 2006 bei der Bundesagentur für Arbeit, seit 2010 ist er an der Entwicklung von ALLEGRO beteiligt. Dort beschäftigt er sich neben der Entwicklung mit dem Schwerpunkt Java auch mit den Themen Wissensmanagement und Clean Code.



André Ullrich
(andre.ullrich@wi.uni-potsdam.de) arbeitet am Lehrstuhl für Wirtschaftsinformatik, insbesondere Prozesse und Systeme der Universität Potsdam. Schwerpunkte seiner Arbeit sind wandlungsfähige Architekturmerkmale von Organisationen sowie organisationales Wissensmanagement im Zuge der Digitalisierung.

Dr. Gergana Vladova
(gergana.vladova@wi.uni-potsdam.de) ist wissenschaftliche Mitarbeiterin am Lehrstuhl für Wirtschaftsinformatik, insbesondere Prozesse und Systeme an der Universität Potsdam, wo sie auch promovierte. Ihre Forschungsschwerpunkte umfassen das Wissens- und Innovationsmanagement, Industrie 4.0 und die Auswirkungen der Digitalisierung.