»From SOA2WOA«

Leitfaden 2016

www.bitkom.org

bitkom

Herausgeber

Bitkom e. V.
Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e. V.
Albrechtstraße 10 | 10117 Berlin

Ansprechpartner

Frank Termer | Bitkom e. V.
T 030 27576-232 | f.termer@bitkom.org

Autoren

- Markus Mayer | Axway GmbH
- Matthias Mertens | BLUECARAT AG
- Prof. Dr. Olaf Resch | HWR Berlin
- Dr. Frank Simon | BLUECARAT AG
- Frank Termer | Bitkom e.V.
- Christoph Wiechmann | Axway GmbH

Verantwortliches Bitkom-Gremium

AK Software Architektur

»From SOA2WOA«

Leitfaden

Inhaltsverzeichnis

Abbildungsverzeichnis

Vorwort

In diesem Bitkom-Leitfaden »From SOA2WOA« soll das hochaktuelle Thema der schnittstellenbasierten, unternehmensübergreifenden IT-Kommunikation aus Sicht des Consumers und des Providers erläutert werden. Ziel des Leitfadens ist, sowohl dem Manager einige business-relevante Einblicke in das Thema zu geben, als auch dem Techniker für die spätere Umsetzung besonders kritische Entscheidungspunkte aufzuzeigen.

Das Dokument ist in drei Abschnitte aufgeteilt:

- Nach einer Erarbeitung einer präzisen Nomenklatur, die gerade für diese junge Disziplin noch ganz wesentlich ist, wird im »Status Quo«-Bereich eine Übersicht über den aktuellen Stand gegeben: Was ist das veränderte Business-Modell, was sind ggfs. Deutschland-spezifische Besonderheiten, welche Risiken sind offenkundig, und welche Klassifikationen für den Einsatz einer WOA lassen sich finden?

- Im zweiten Quo Vadis wird der konstruktive Charakter des Leitfadens deutlich, in dem konkret aufgezeigt wird, welche Vorteile sich für die einzelnen Branchen ergeben können, wenn einige technische Besonderheiten geschickt und frühzeitig festgelegt werden.

- Im dritten Abschnitt wird versucht, für die Transition vom Status Quo zum Quo Vadis einige Lessons Learned aus der Vergangenheit heranzuziehen und Parallelen zu anderen, ähnlich gelagerten Innovationen herzustellen.

1 Begriffswirrwarr: Eine erste Einordnung

1.1 Zur Notwendigkeit einer Nomenklatur

Die Auseinandersetzung mit einem Thema bedarf grundsätzlich einer einheitlichen Sprache, damit alle handelnden Personen bei der Verwendung derselben Begriffe damit auch die gleiche inhaltliche Bedeutung verbinden. Das Festlegen dieser Namen und Bezeichnungen wird als Nomenklatur bezeichnet. Diese verbindliche Sammlung von Begriffen hilft zum einen, Transparenz über ein bestimmtes Themengebiet zu schaffen, in dem inhaltlich verwandte Begriffe beschrieben und jeweils zueinander in Beziehung gesetzt werden. Gleichzeitig werden hierdurch auch die Grenzen des Gebietes definiert. Diese negative Abgrenzung kann entweder durch Weglassen oder explizites Ausschließen von Begrifflichkeiten erfolgen. Eine Nomenklatur bettet sich zudem in eine Terminologie ein, welche alle relevanten Begriffe eines Fachgebietes beschreibt.

Dieses erste Kapitel des Leitfadens dient nun dazu, eine solche Nomenklatur als Verständnisgrundlage bereitzustellen. Dabei erfolgt eine Orientierung an aktueller (Standard-)Literatur bzw., wo eine solche Literatur derzeit nicht zur Verfügung steht, wird ein Begriff aus der gängigen Handhabung in der täglichen Praxis heraus definiert. Grundsätzlich bleibt festzuhalten, dass Definitionen weder richtig noch falsch sein können, sondern immer situationsbezogen einem Zweck dienlich sein müssen. Daher werden Begriffe teilweise in unterschiedlicher Detailliertheit definiert, je nachdem, welche Wichtigkeit und welchen Stellenwert diese für den vorliegenden Leitfaden haben. Wo es möglich ist, wird auf entsprechend genutzte Literatur verwiesen.

Die Vorstellung der Nomenklatur erfolgt in drei Wellen:

1. Zutatensammlung: Die später zu definierenden Begriffe wie WOA und Software-Ecosysteme bestehen aus Kompositionen anderer Begriffe (z. B. API, Schnittstelle). Diese werden als spätere Zutaten im ersten Schritt definiert.

2. Definition: Auf Basis der Zutaten werden anschließend die für dieses Dokument relevanten Begriffe definiert.

3. Abgrenzung: Hier werden verwandte und explizit irrelevante Begriffe zu den vorher definierten Begriffen abgegrenzt bzw. Parallelen aufgezeigt.

1.2 Zutatensammlung

(Web-)Service

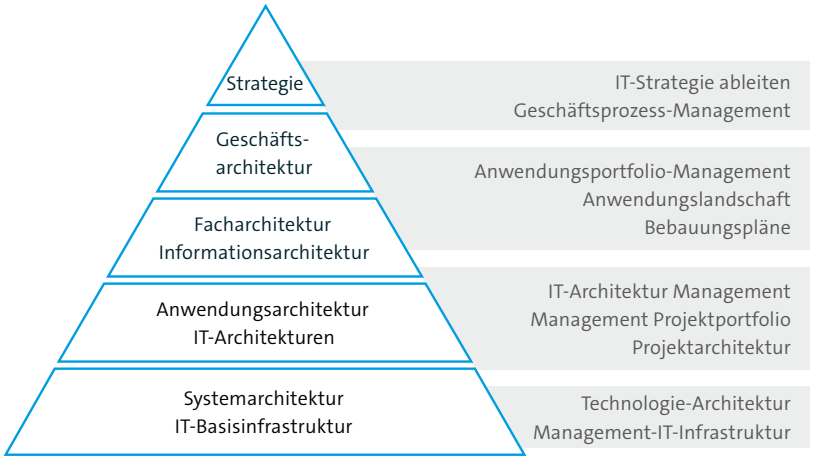
Allgemein beschreibt ein Service ein Softwareelement mit einer fest definierten Leistung (Krafzig et al. 2005). Ist dieses Softwareelement durch ein Netzwerk erreichbar und kann es mittels nachrichtenbasierter Kommunikation aufgerufen werden, so wird von einem Web-Service gesprochen. Dabei bietet der Web-Service seine Funktionen nach außen über eine wohldefinierte Schnittstelle an, welche durch eine eindeutige Spezifikation beschrieben ist. Somit ist der Web-Service entweder öffentlich oder für eine definierte Zielgruppe zugänglich (Marx Gomez 2015).

Web-Services sind im Zusammenhang mit SOA weitverbreitet und werden mit den Komponenten des SOA-Dreiecks (Service-Nutzer, Service-Anbieter und Service-Verzeichnis) umgesetzt.

Architektur

Eine Architektur beschreibt die grundlegende Anordnung von Elementen eines Systems, sowie deren Beziehungen (Verknüpfungen) untereinander, aber auch die Beziehungen zur Umwelt des Systems (Winter und Aier 2015). Im Rahmen des Architekturmanagements wird neben der Beschreibungsaufgabe auch die Gestaltungsaufgabe adressiert, die Prinzipien zur Konstruktion, Weiterentwicklung und Nutzung der Architektur angeben. Eine Architektur erlaubt eine ganzheitliche Betrachtung eines Systems und kann durch unterschiedliche Abstraktionsebenen ein System in verschiedenen Detaillierungsgraden abbilden.

Für das Abbilden der Architektur von Informationssystemen (IS) hat sich die Verwendung der Architekturpyramide als eine wesentliche Möglichkeit etabliert (siehe Abbildung 1). Diese beschreibt verschiedene Schichten bzw. Betrachtungsebenen einer ganzheitlichen IS-Architektur.



http://winfwiki.wi-fom.de/images/9/91/Architekturpyramide_dern.jpg

Abbildung 1: Architekturpyramide und zugehörige Aufgaben im IT-Management (Dern 2009)

Wichtig ist an dieser Stelle darauf hinzuweisen, dass alle Sichten gleichzeitig berücksichtigt werden müssen: Es genügt daher nicht, das Thema API-Management und Software-Ecosysteme nur in einer IT-Basisinfrastruktur zu berücksichtigen, ohne die Geschäftsarchitektur zu involvieren. Das Sichtenkonzept von Architekturen hilft genau hier, offenzulegen, welche Aspekte von einer Änderung betroffen sind. Auch dieser Leitfaden versucht, alle Architektursichten gleichermaßen anzusprechen.

Informationssystem
Ein Informationssystem (IS) ist ein soziotechnisches System, welches aus den Komponenten Mensch, Aufgabe und Technik besteht (Gabriel 2013). Die Komponente Mensch kann weiter in verschiedene Rollen und Anspruchsgruppen des Systems unterschieden werden, wobei sich eine Betrachtung von IS-Konstrukteur, IS-Operateur und IS-Nutzer etabliert hat.

Web-Service-Basistechnologien (Standards)
Zur technischen Umsetzung der Konzepte von Services und Web-Services werden verschiedene Standards genutzt. Hierzu gehören SOAP (Simple Object Access Protocol), WSDL (Web Service Description Language) und UDDI (Universal Description, Discovery and Integration), ebenso wie JSON (JavaScript Object Notation) und REST (Representational State Transfer).

(Software-)Ecosystem
Ein Ökosystem beschreibt grundsätzlich eine Gemeinschaft von lebenden Organismen, die in einem bestimmten räumlichen Gebiet zusammenleben und interagieren, wobei sich das Ökosystem in die Umwelt einbettet, zu der ebenfalls entsprechende Beziehungen bestehen. In einer konkreten Betrachtungsweise werden in einem menschlichen Ökosystem Akteure, deren Beziehungen zueinander, die Handlungen der Akteure und die Transaktionen der Akteure entlang der Beziehungen subsumiert. In einem geschäftlichen Ökosystem sind die Teilnehmer konkret Unternehmen, deren Kunden und Lieferanten, und die Transaktionen können Informationen und Wissen, aber auch Geld- und Warenflüsse sein. Von dieser Betrachtungsweise leitet sich der Begriff des Software-Ecosystems ab, welches eine Menge von Softwarelösungen umfasst, die die Handlungen und Transaktionen in einem geschäftlichen Ökosystem ermöglichen, unterstützen und automatisieren. (Bosch 2009, S. 2)

API/Schnittstelle
Ein Softwaresystem enthält logische Berührungspunkte, die den Austausch von Befehlen und Daten zwischen verschiedenen Komponenten ermöglichen. Diese werden als Schnittstellen bezeichnet. Eine Programmierschnittstelle (Application Programming Interface; API) ist ein Teil eines Softwareprogramms, welches anderen Programmen als Möglichkeit der Anbindung an das Softwaresystem angeboten wird. Zu jeder API wird eine Dokumentation bereitgestellt, welche die von einer Schnittstelle angebotenen Funktionen beschreibt und insbesondere die Parameter, die zum erfolgreichen Aufruf einer Schnittstelle benötigt werden, angibt. Durch die Nutzung von Protokollen können APIs unabhängig von etwaiger Hardware und dem Betriebssystem implementiert werden.

Consumer/Provider/Prosumer
Sowohl in einer WOA als auch in einer SOA treten die Rollen des Dienstanbieters (Service-Provider) und des Dienstnutzers (Service-Consumer) auf. Organisationen, die sowohl Dienste anbieten (also Provider sind) als auch nutzen (und damit Consumer sind), werden als Prosumer bezeichnet. Durch die zunehmende Fokussierung von Organisationen in ihren Aufgaben und der Möglichkeit, auf Basis von Services neue Geschäftsmodelle umzusetzen, ist zu beobachten, dass Organisationen zunehmend als Anbieter von Services auftreten. Hierdurch steigt die Anzahl angebotener Services insgesamt und Organisationen können diese steigende Zahl von Diensten nutzen. Damit lohnt es sich für Organisationen, lediglich solche Dienste anzubieten, in denen sie originäres Expertenwissen einfließen lassen haben. Andererseits werden alle anderen Funktionalitäten von anderen Anbietern bereitgestellt. Damit verschwimmt die klassische Trennung in Provider und Consumer und Organisationen treten zunehmend als Prosumer auf.

1.3 Definition

API-Management
Als API-Management wird das Veröffentlichen von Application Programming Interfaces (API) in einer sicheren und skalierbaren Umgebung bezeichnet. Ergänzt wird die Veröffentlichung durch den Support und die Dokumentation der API. Ziel des API-Managements ist es, den Lebenszyklus einer API verfolgen zu können, um sie so anbieten zu können, dass sie Nutzern und Programmierern den bestmöglichen Nutzen stiftet. Dem API-Management werden verschiedene Teilaufgaben zugeordnet. Diese sind Sicherheit, Test, Deployment, Versionierung, Authentifizierung und Autorisierung.

API-basierte Software-Ecosysteme
API-basierte Software-Ecosysteme bestehen aus leichtgewichtigen Schnittstellen sowie einem globalen Technikstack, über den diese einfach anzubieten und zu nutzen sind. Der Consumer kann die geschickte Schnittstellen-Orchestrierung effektiv für die eigene Wertschöpfung verwenden, um darauf effizient eigene Marktvorteile zu implementieren. Der Provider kann über die Plattform effizient bestehen und neue (Zwischen-)Produkte an existierende und neue Kunden anbieten. Ein Teilnehmer im API-basierten Software-Ecosystem tritt meist als Prosumer auf, ist also gleichzeitig Provider und Consumer (Simon und Kraft 2015).

Web-orientierte Architektur (WOA)
Eine Web-orientierte Architektur ist die technische Umsetzung eines API-basierten Software-Ecosystems unter Zuhilfenahme von Web-Technologien. Eine allgemein anerkannte Definition von WOA liegt bisher nicht vor. Den vielversprechendsten Ansatz liefert Gartner (Gall et al. 2008 und Gartner 2015): »WOA is an architectural substyle of SOA that integrates systems and users via a web of globally linked hypermedia based on the architecture of the Web.
This architecture emphasizes generality of interfaces (UIs and APIs) to achieve global network effects through five fundamental generic interface constraints:

1. Identification of resources (e.g., uniform resource identifier [URI])
2. Manipulation of resources through representations (e.g., HTTP)
3. Self-descriptive messages (e.g., Multipurpose Internet Messaging Extensions [MIME] types)
4. Hypermedia as the engine of application state (e.g., links)
5. Application neutrality«

Zusammen mit API-Management und einem DevOps-Ansatz stellt WOA die Voraussetzung für eine »Industrially Designed Infrastructure« dar (Reymer 2015).

Dabei sind WOA und SOA keineswegs als sich gegeneinander ausschließende Ansätze zu verstehen, sondern oft schafft erst die Kombination von beiden Ansätzen eine wirkungsvolle Grundlage für die Schaffung von unternehmensübergreifenden API-basierten Ökosystemen. Insbesondere das Fehlen eines eigenen hochentwickelten Security- und Identity-Stacks bei WOA macht SOAP, und die SOA zugrunde liegenden WS*-Standards für die Realisierung vieler webbasierten Anwendungsfälle für Unternehmen heute noch unverzichtbar. Diesen Zusammenhang stellt auch Chuan-Jun Su in (»Web-Oriented Architecture (WOA) Enabled Customer-Centric Collaborative Commerce Platform«) deutlich heraus: »WOA is not suitable for all applications, and high-end enterprise applications in particular require the more sophisticated portions of the SOA stack.« Dieser sinnvollen und pragmatischen Sichtweise folgt auch der vorliegende Leitfaden, wenn im weiteren Verlauf die unterschiedlichen Aspekte von API-basierten Ökosystemen betrachtet werden.

1.4 Abgrenzung

Microservice
Microservices sind ein Architekturstil, bei dem komplexe Anwendungen aus sehr kleinen und unabhängigen Diensten erstellt werden, die über Schnittstellen verbunden sind. Diese Dienste sind so gestaltet, dass jeder lediglich eine sehr kleine Aufgabe umsetzt, die insbesondere deutlich kleiner ist, als eine durch einen Web-Service realisierte Aufgabe. Aus der Komposition der Dienste ergibt sich eine lediglich geringe bzw. schwache Kopplung, so dass insbesondere die unabhängige Weiterentwicklung der Dienste ermöglicht wird. So können sie bspw. verschiedene Programmiersprachen und Technologien nutzen, unabhängig von anderen Diensten in Produktion gebracht werden und bei Bedarf einfach komplett ausgetauscht werden. Damit unterstützen Microservices Konzepte wie bspw. Continuous Delivery, Skalierbarkeit und Agilität. Nachteile von Microservices sind auf der anderen Seite zusätzliche Komplexität durch Lastverteilung und Fehlertoleranz, erhöhte Schwierigkeiten bei der Softwareverteilung und dem Softwaretest sowie Verschieben der Abhängigkeiten zwischen Diensten von der Deployment-Ebene auf die Netzwerkebene.

Service-orientierte Architektur (SOA)
Eine Service-orientierte Architektur (SOA) wird definiert als »offene, agile, erweiterbare, verbundene, komponierbare Architektur, die aus autonomen, [...], interoperablen, auffindbaren und potenziell wiederverwendbaren Services besteht« (Erl 2005).

Einer SOA liegen verschiedene Konstruktionsprinzipien zugrunde:

- **Lose gekoppelte Services:** Services bieten in sich abgeschlossene Funktionalitäten an und werden grundsätzlich nur über eine Schnittstelle angesprochen. Zur Verwendung eines Service wird eine temporäre Beziehung zwischen Dienstnutzer und Dienstanbieter hergestellt und anschließend wieder gelöst. Damit sind Services in einer SOA lediglich lose gekoppelt.

- **Entwurfsstandards:** Services werden nach einheitlichen Richtlinien entwickelt, um Interoperabilität zu gewährleisten. Hierzu zählen bspw. das Prinzip der Kapselung und der Aufruf über definierte Schnittstellen.

- **Modularität:** Services bilden unabhängige Teilsysteme, wobei jeder Service einen spezifischen Zweck erfüllt. Somit können beliebig neue Services zur SOA hinzugefügt aber auch teilnehmende Services aus der SOA entfernt werden.

- **Jeder Service kann Teil einer SOA werden:** Durch das Grundprinzip, dass jeder Service-Teil einer SOA werden kann, lassen sich redundante Angebote vermeiden und eine stärkere Wiederverwendung von Services erreichen.

- **Service-Broker:** Der Service-Broker wählt bei Bedarf aus mehreren Services einen passenden aus, bindet diesen in die SOA ein und stellt die Verbindung zwischen Services in einer SOA her.

- **Service-Bus:** Der Service-Bus stellt die Kommunikation zwischen den Services sicher und garantiert gleichzeitig die sichere Zustellung und Ereignisbearbeitung von Nachrichten.

- **Service-Dokumentation:** In der standardisierten Service-Dokumentation sind eine fachliche Beschreibung der Funktion des Service und eine technische Beschreibung des Aufrufs des Service enthalten.

- **Service-Katalog:** Der Service-Katalog liefert eine Übersicht, welche Services grundsätzlich zur Verfügung stehen und dokumentiert ebenso, welches Services bereits in einer SOA verwendet werden.

Im Vergleich zu SOA zeichnet sich WOA durch folgende Unterschiede aus (Hinchcliffe 2011):

- Eine SOA besitzt typischerweise eine geringe und gut definierte Anzahl von Endpunkten, über welche Daten ausgetauscht werden. Eine WOA hingegen enthält eine große und offene Anzahl von Endpunkten, i. d. R. einen für jede teilnehmende Instanz einer Ressource, die durch einen URI angesprochen werden kann.

- Eine traditionelle SOA erzeugt Nachrichten auf Basis von HTTP und SOAP. Eine WOA hingegen nutzt einzig HTTP und verwandte Mechanismen, die sich auf der gleichen Abstraktionsebene befinden.

- SOA wurde top-down durch verschiedene Anbieter entwickelt. WOA hingegen entstand bottom-up durch die Community der Web-Entwickler.

- SOA nutzt WS-Security und andere eher hochentwickelte Standards für Sicherheitsmechanismen. WOA nutzt hingegen einfache Verfahren wie HTTPS, OAUth und HMAC-SHA-1.

Mashup
Mashups sind Anwendungen, deren Mehrwert zu einem großen Teil durch »importierte« Ressourcen entsteht, wobei insbesondere offene APIs anderer (Web-)Anwendungen genutzt werden (Koch 2015). Die genutzten Ressourcen können Inhalte, Daten oder Funktionalitäten sein. Im Unterschied zu den »Composite Applications« nutzen Mashups webbasierte Anwendungen und freie Datenquellen.

Der Begriff des Mashups kann auf drei verschiedenen Ebenen weiter konkretisiert werden:

- **Endbenutzer-Mashups:**
  Hier erfolgt die Integration der importierten Ressourcen auf Ebene der Benutzerschnittstelle, wenn bspw. beliebige Anwendungen in einem Portal hinzugefügt werden können, welche auf Datenquellen des Portals oder auf außenstehende Datenquellen zugreifen.

- **Daten-Mashups:**
  Hierbei werden Daten aus verschiedenen Quellen kombiniert und wiederum als Dienst bzw. Datenquelle zur Verfügung gestellt. Es werden dadurch tatsächlich neue Funktionalitäten bereitgestellt.

- **Mashups in Betriebsprozessen:**
  In diesem Fall werden Web-Services in einem Workflow miteinander verbunden und bilden einen neuen Web-Service.

Durch Mashups können mit vergleichsweise geringem Aufwand, insbesondere auch ohne Programmierkenntnisse, neue Anwendungen kreiert werden. Grundvoraussetzung bleibt jedoch das Bereitstellen von Diensten und Daten in einer standardisierten und einfach

zugänglichen Form. Hierin liegt auch der enge Bezug zu SOA. Mashups beinhalten somit die schnelle und einfache Integration von Komponenten, häufig über offene APIs und Datenquellen, um so neuartige Ergebnisse zu generieren, für welche die ursprünglichen (Roh-)Daten nicht gedacht waren.

Wichtig festzuhalten ist aber, dass Mashups nicht mit API-basierten Software-Ecosystemen gleichzusetzen sind, da Mashups i. d. R. der Content-Aggregation dienen, wohingegen Software-Ecosystems durch den Austausch von Informationen, Ressourcen und Artefakten gekennzeichnet sind.

Enterprise Service Bus (ESB)
Ein Enterprise Service Bus (ESB) stellt typischerweise das Rückgrat einer SOA dar, da er Services aus verschiedenen Quellen miteinander verbindet. Als Vermittler stellt er damit eines der wesentlichen Definitions- und Konstruktionsmerkmale einer SOA dar, auch wenn die praktische Umsetzung einer SOA nicht zwingend die Nutzung eines ESB als Integrationswerkzeug zu nutzen. Einsatzziel eines ESB ist das Verhindern von direkten physischen Kopplungen auf der Datenebene. Bei einer WOA wird hingegen grundsätzlich kein ESB eingesetzt und es gibt folglich keine zentrale Steuerungseinheit zwischen den Teilnehmern auf globaler Ebene. In einer WOA werden APIs direkt Punkt zu Punkt verbunden.

2 WOA im Industrialisierungskontext

2.1 WOA: Status-Quo

Das Themenfeld der Web-orientierten Architekturen (WOA) ist keine technische Variation oder Neuerung, sondern ermöglicht grundsätzlich neue Wertschöpfungsketten. Um die Tiefe dieser Innovation besser einschätzen zu können, soll im Folgenden kurz ein historischer Abriss der Industrialisierung dargestellt werden, an dessen Ende dann die Einführung webbasierter Architekturen zur Einrichtung von neuen Software-Ecosystemen steht.

2.2 Von ad-hoc-Wertschöpfungsketten zur Manufaktur

Noch im Mittelalter dominierten Wertschöpfungsketten, die eher ad-hoc geprägt waren. Es wurde das geschaffen, was gerade möglich war und was im nächsten Umfeld auch direkt einen Markt hatte. Angebot und Nachfrage hatten einen sehr kleinen gemeinsamen regionalen Kreis. Das, was gerade vorhanden war, wurde mit anderem, was ebenfalls gerade vorhanden war, zur Erstellung eines höherwertigen Produktes verwendet. War eine der Zutaten nicht mehr vorhanden, wurde der andere Rohstoff zur Erzeugung eines anderen Produktes verwendet.
Erst der Industrialisierungsschritt der Manufaktur hat dies fundamental geändert:
In der Manufaktur wurden verschiedene fest definierte Handwerke zu einem Arbeitshaus unter einer gemeinsamen Zielvorstellung zusammengelegt. Es gab eine wohldefinierte Modularisierung, und jedes einzelne Modul hatte anschließend die Möglichkeit der weiteren Spezialisierung. Die Wiederholung und Optimierung sowohl der Einzelarbeitsschritte als auch der Orchestrierungen haben zu einem steuerbaren Qualitätslevel geführt.

2.3 Von der Manufaktur zur Fabrik

Die Einführung von Maschinen hat den Charakter der Manufakturen grundlegend verändert: Die zunehmende Automatisierung, damals noch mit viel Dampf und Schmutz verbunden, hat den Wert der Manufaktur als Lebensraum nach und nach gesenkt. Die damit einhergehende Trennung von Wohnen und Arbeiten hat, zusammen mit einer deutlicheren Unterstützung automatisierbarer Arbeitsschritte durch Maschinen, die Fabriken geschaffen. Die Maschinen haben in den Schnittstellen auch zu präziseren Formulierung von Ein- und Ausgangsschnitt-stellen geführt: Konnten Menschen in den Manufakturen bestimmte nichterfüllte Kriterien eines vorherigen Arbeitsschrittes noch pragmatisch kompensieren, so haben Maschinen diese Variabilität nicht. So entstand in den Fabriken nach und nach das Konzept von Quality-Gates, die eine Mindestqualität von allen Zwischenprodukten sichergestellt haben. Auch das Konzept von Service-Leveln hat hier Einzug gehalten, da der Durchsatz der neuen Maschinen meist die Taktzahl des gesamten Arbeitsprozesses definierte. Leerzeiten galt es aufgrund der hohen Investitionskosten der Maschinen bestmöglich zu vermeiden.

2.4 Von der Fabrik zum IT-nutzenden Unternehmen

Aus Sicht der Industrialisierung ist die Einführung von meist unverbundener IT »lediglich« eine beschleunigte Fortführung der Fabrik: Die IT ermöglichte, immer mehr Arbeitsschritte, die zur Erzeugung des Endproduktes notwendig waren, zu automatisieren. Für den Kunden des Produktes war es häufig gar nicht klar, inwieweit die Wertschöpfungskette IT-unterstützt oder nur maschinell-automatisiert ausgerichtet war. Da die mit der IT mögliche Effizienzsteigerung allerdings enorm war, war sie nach wenigen Jahren bereits so stark in den meisten Unternehmen ausgeprägt, dass eigene IT-Organisationseinheiten eingerichtet wurden, die den Betrieb des Werkzeugs IT fortwährend sicherstellten. Diese Stärkung der IT hat dazu geführt, dass sie immer normativer den Arbeitsablauf selbst gestaltet hat: Da die IT häufig fertig eingekauft wurde, mussten die beinhalteten Schnittstellen zwangsläufig vom Unternehmen bedient werden.

Das IT-nutzende Unternehmen hat über die IT immer mehr Produktionsexpertise eingekauft, die massiven Einfluss auf die Standardisierung der Wertschöpfungskette hatte. Die frühen SAP-Systeme sind ein typisches Beispiel für diese zwangsweise Standardisierung. Die Systeme wurden dabei jeweils eingekauft und zentral im eigenen Unternehmen betrieben. Der Fabrik-Gedanke mit seinem zentralisierenden Grundcharakter wurde hier fortgesetzt, in dem immer mehr unterstützende Systeme in das eigene Unternehmen integriert wurden.

Noch heute (2016) dominiert in vielen Unternehmen dieses IT-Verständnis. Deutlich wird dies vor allen Dingen an den vielen Medienbrüchen, wenn verschiedene Unternehmen zusammenarbeiten müssen. So sehen laut einer von Ferrari electronic Ende 2011 durchgeführten Befragung 82 Prozent der Unternehmen das Fax als »wichtig für ihr Unternehmen« an. (Lück 2011) Dieser Trend ist auch 2013 z. B. für die Techniker Krankenkasse immer noch gültig (Ferrari electronic AG 2013): Dort wird sogar eine »Tendenz steigend« vorausgesagt. Das Fax ist dabei das verbindende Medium zweier Unternehmen, die jeweils ihre eigene unvernetzte IT einsetzen.

2.5 Vom IT-nutzenden Unternehmen zum Software-Ecosystem

Die Vernetzung der einzelnen IT-Unternehmen ermöglicht einen weiteren Schritt der Industrialisierung: die Fokussierung auf eine eigene Kernkompetenz und das Einbeziehen dezentraler Leistungserbringer. Sowohl die Manufakturen als auch die Fabriken hatten eine große Wertschöpfungstiefe bzgl. der erstellten Produkte im eigenen Unternehmen. Nur sehr große Stückzahlen und längerfristige Teilorchestrierungen rechtfertigten das Ausgliedern bestimmter Komponenten. Die Automobilindustrie ist ein gutes Beispiel hierfür. Die zu bedienenden Schnittstellen sind allerdings sehr schwergewichtig und nur mit einem längerfristigen Horizont und einer entsprechenden Mindestnutzung der Verbindung sinnvoll.

Die Nutzung anderer IT-Systeme und der durch sie verfügbaren Funktionalität ist dagegen sehr viel leichtgewichtiger und vor allen Dingen globaler: Liegt die HR-Abrechnung nicht in der Kernkompetenz eines Unternehmens so kann dieser Bereich heute als Business Process Outsourcing (BPO) leichtgewichtig über Schnittstellen extern vergeben werden. Die Leistungserbringung selbst kann hierbei irgendwo auf der Welt erfolgen. Liegt die Zeiterfassung der Mitarbeiter nicht in der Kernkompetenz des Unternehmens, so kann auch diese Funktionalität sehr schnell extern bezogen werden.

Seit einigen Jahren steigt die Anzahl verfügbarer Schnittstellen dramatisch: Ein modernes Unternehmen sieht das eigene Business damit als Teil eines viel größeren Software-Ecosystems. Die zu nutzende Funktionalität muss dabei nicht mehr zentral selbst aufgestellt und betrieben werden, sondern kann ergebnisorientiert einfach genutzt werden. Auch die eigenen Absatzkanäle werden durch diese zunehmende Vernetztheit erweitert: Eigene Produkte und Zwischenprodukte können leichtgewichtig weltweit verkauft werden.

Software-Ecosysteme ermöglichen ein viel schnelleres Zusammenstellen neuer Wertschöpfungsketten. Die Innovationsfähigkeit von Unternehmen kann hiermit massiv positiv beeinflusst werden. Hierbei helfen vor allen Dingen die geringen Initialkosten (z. B. im Gegensatz zum IT-nutzenden Unternehmen, welches die komplette IT zentral vorhalten musste) sowie die in der Wertschöpfungstiefe immer höher liegenden Ergebnisse, deren Details für den Nutzer immer weniger relevant werden.

Damit hat die Industrialisierung einen neuen Meilenstein erreicht.
Die wesentlichen Charakteristika sind hierbei:

- Unternehmensgrenzen weichen nach und nach auf. Sowohl die Manufaktur, die Fabrik als auch das IT-nutzende Unternehmen waren autarke Organisationen. Externe Interaktionen waren schwergewichtig und durch Medienbrüche gekennzeichnet. In Software-Ecosystemen spielt die »Heimat« eines Teilproduktes kaum noch eine Rolle.

- Die eigene Wertschöpfungstiefe der Unternehmen nimmt immer weiter ab. Ein Großteil kann fertig von außen bezogen werden, so dass das Unternehmen sich effektiv dem eigentlichen Alleinstellungsmerkmal zuwenden kann. Dadurch wird auch das Startkapital, das für eine Unternehmensgründung notwendig ist, immer weiter reduziert (im Gegensatz zum Beispiel zur schwergewichtigen Fabrik).

- Die Time-to-Market wird kontinuierlich reduziert, da die verfügbaren Services immer höherwertiger und die Orchestrierung immer effizienter möglich wird.

Das Web ist dabei das Hauptübertragungsmedium dieser Software-Ecosysteme. Firmeninterne Service-Kataloge, wie sie in Service-orientierten Architekturen großer Unternehmen erstellt wurden, öffnen sich nach und nach anderen Unternehmen, Ländern und Kontinenten. Software-Ecosystemen liegt eher eine Shareconomy zugrunde, innerhalb derer der die größten Vorteile hat, der mit anderen intensiv interagiert. Die webbasierten Schnittstellen (WOA) spielen für diese Software-Ecosysteme eine fundamentale Rolle, da IT-Systeme nur über Schnittstellen miteinander interagieren können.

3 WOA und das spezifisch Deutsche

Das Konzept Web-orientierter Architekturen erfreut sich gerade in der New Economy größter Beliebtheit: So machen Ebay, Salesforce und Twitter bereits heute einen Großteil ihres Business nicht über die Endprodukte, also die GUI-behafteten Anwendungen, sondern über Schnittstellen zu (Zwischen-)Produkten ihrer Dienstleistung.

Ein typisches Beispiel, das sich immer weiter zunehmender Beliebtheit erfreut, ist die Teilnahme am webbasierten Software-Ecosystem als Consumer, um das Thema Authentifizierung fertig von anderen Anbietern übernehmen zu können. Ein solch typisches Beispiel ist in der folgenden Abbildung für das Registrierungsprocedere bei meetup (http://www.meetup.com/de/) dargestellt:

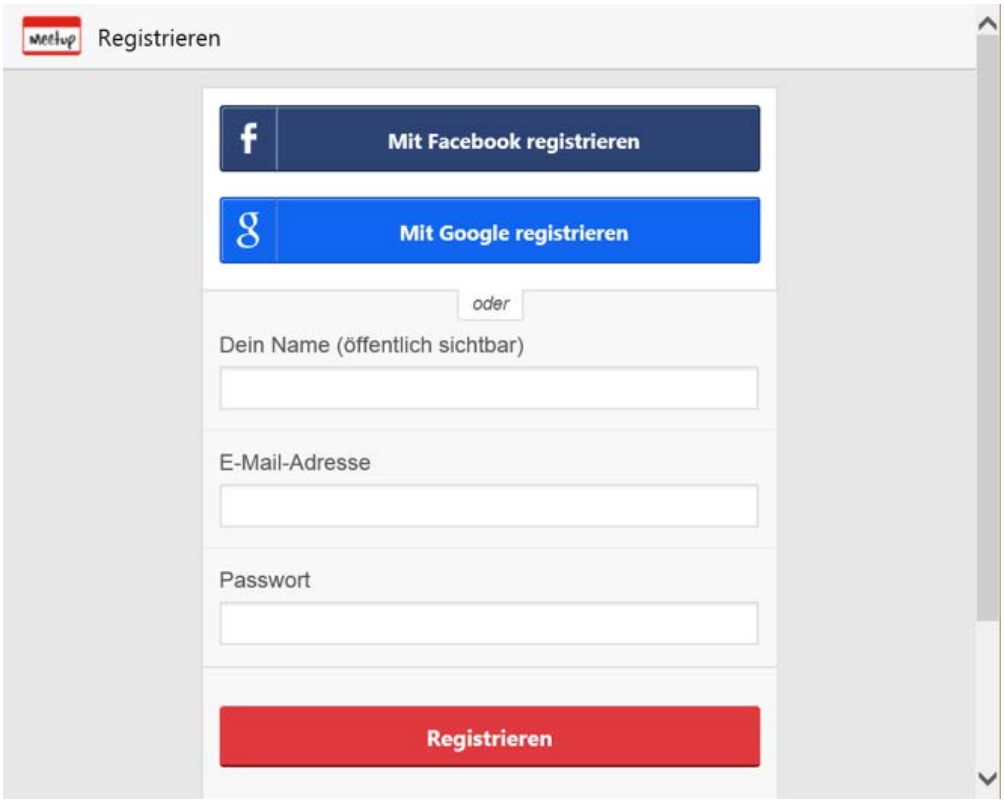


Abbildung 2: Nutzung von fremden Authentfiizierungslösungen als Consumer

Der durchschlagende Erfolg von schnittstellenbasierten Software-Ecosystemen lässt sich gut an den Verkaufszahlen der dafür notwendigen API-Management-Plattformen ablesen: So werden laut Forrester in 2015 bereits 140 Millionen Dollar in die Beschaffung solcher »API management solutions« investiert. Dieser Umfang wird sich laut Forrester bis 2020 auf 660 Millionen Dollar vervierfacht haben (Yamnitsky 2015). Die großen Marktplayer wie Oracle, HP, IBM, CA und Axway bespielen diesen Markt aktiv. Spannend ist hier allerdings die Restriktion, dass sich sowohl der Status Quo als auch der Ausblick lediglich auf den US-Markt bezieht.

Es wird zwar in Aussicht gestellt, dass »international sales will take this figure over the billion-dollar mark« (Brennan 2015), aber bereits hier wird deutlich, dass schnittstellenbasierte Software-Ecosysteme auf dem alten Kontinent noch sehr vorsichtig beäugt werden. Ein Blick in die hiesige IT-Landschaft zeigt nämlich insgesamt bei der Bestrebung, über Schnittstellen Teil eines größeren Ecosysteme zu sein, für Deutschland ein sehr viel defensiveres Bild. Dies soll in den beiden nächsten Kapiteln beleuchtet werden.

3.1 Erwartungen in Deutschland

Die Erwartungen der deutschen Industrie an schnittstellenbasierte Software-Ecosysteme sind mit denen der internationalen Mitspieler vergleichbar. In einigen Workshops konnten folgende Schlaglichter erzeugt werden, die zwar keineswegs repräsentativ sind (n ~20), aber dennoch einige Indikatoren aufzeigen und deren Priorisierung belegen.

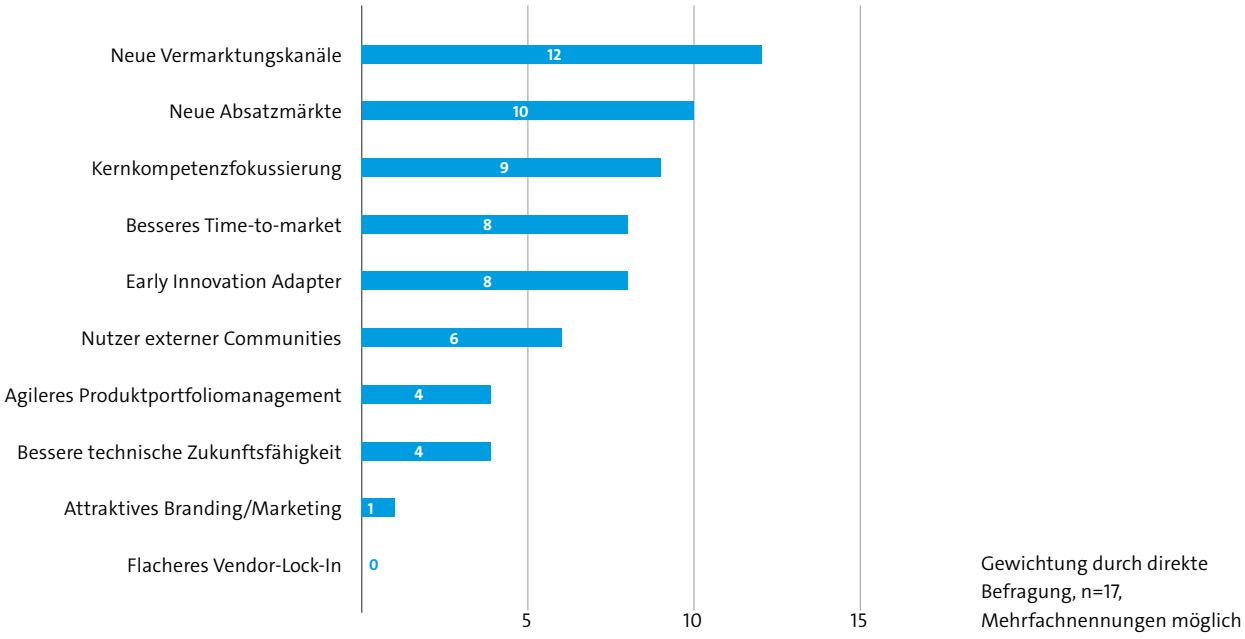


Abbildung 3: Erwartungen deutscher Unternehmen an schnittstellenbasierte Software-Ecosysteme

Die einzelnen Vorteile sollen hier kurz erläutert werden:

Vorteile	Erwartungshaltung	Zielsetzung
Neue Absatzmärkte	Über die APIs können ganz neue Adressaten angesprochen werden.	APIs werden hier als Enabler von zusätzlichem Business gesehen.
Neue Vermarktungs-kanäle	Bestehende Produkte können über neue APIs zusätzlich vermarktet werden.	APIs werden hier als Stärkung von bestehendem Business gesehen.
Flacheres Vendor-Lock-In	Durch die Modularisierung des eigenen Business-Prozesses in Sub-Prozesse und deren leichtgewichtige Bespielung durch externe Service-Provider kann die Abhängigkeit zu einzelnen Herstellern reduziert werden. Dies zeigt auf das Charakteristikum schnittstellenbasierter Software-Ecosysteme, nicht einige wenige große, sondern viele kleine Mitspieler zu besitzen.	APIs werden hier als Risiko-reduktion gesehen.
Nutzen externer Communities	Durch die Herausgabe von (Zwischen-)Produkten über Schnittstellen entstehen Communities, die als Multiplikator am Markt auftreten. Die oben aufgeführte Nutzung der Facebook-Authentifizierung ist ein gutes Beispiel für externe Communities, die über diesen Weg als Multiplikator des Schnittstellenproviders (im konkreten Fall also Facebook) auftreten.	APIs werden hier als Ressourcen-Pooling gesehen.
Attraktiveres Branding/ Marketing	Alleine durch das Anbieten von APIs wird die eigene Firma in der Außendarstellung gewinnen.	APIs werden hier als Marketing-Instrument gesehen.
Besseres Time-to-Market	Durch das effiziente Beziehen externer Services als Consumer kann das eigene Kernbusiness schneller realisiert werden.	APIs werden hier als Business-Beschleuniger gesehen.
Early Innovation Adopter	Durch die Nutzung externer APIs können Neuerungen zügiger in das eigene Unternehmen Einzug halten, d.h. der Austausch von Innovationen über das schnittstellenbasierte Software-Ecosystem wird vereinfacht.	APIs werden hier Innovations-treiber gesehen.
Agileres Produkt-portfoliomanagement	Durch die effiziente Orchestrierung externer Schnittstellen kann das eigene Portfolio dynamischer an neue Anforderungen angepasst werden, da die Kernkompetenzfokussierung zu deutlich kleineren eigenen Komponenten führt.	APIs werden hier Business Enabler gesehen.
Kernkompetenzfokus-sierung	Durch die Hinzunahme extern verfügbarer Services über Schnittstellen kann der eigene Entwicklungsumfang reduziert werden bzw. kann der Fokus verstärkt auf das eigene Alleinstellungsmerkmal gelegt werden.	APIs werden hier als Zuliefe-rungstechnik gesehen.
Bessere technische Zukunftsfähigkeit	Durch die Modularisierung der eigenen Wertschöpfungskette und durch das punktuelle Beziehen fertiger (Teil)-produkte über Schnittstellen kann die eigene Zukunftsfähigkeit gesichert werden.	APIs werden hier als Risiko-reduktion gesehen.

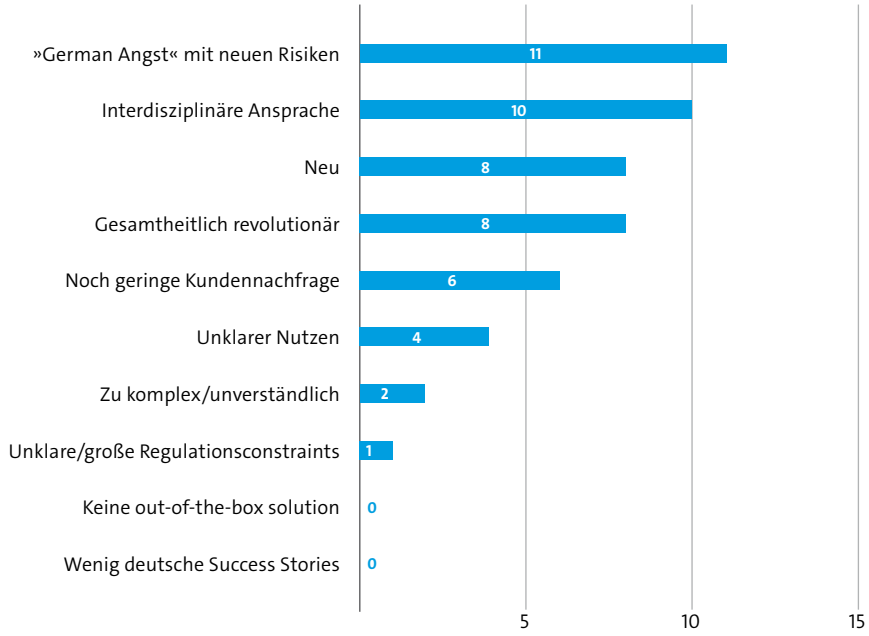
Die numerischen Angaben zeigen, dass die erwarteten Vorteile der Teilnahme an schnittstellenbasierten Software-Ecosystemen sowohl die Rolle als Consumer als auch die Rolle als Provider berücksichtigen. Die beiden größten Zustimmungen (neue Absatzmärkte, neue Vermarktungskanäle) beziehen sich auf die Rolle als Provider, die nächsten drei Vorteile (Kernkompetenzfokussierung, besseres Time-to-market und Early Innovation Adopter) beziehen sich auf die Rolle als Consumer.

In Summe bleibt damit festzuhalten, dass die in Deutschland mit der Einführung von schnittstellenbasierten Software-Ecosystemen verbundene Erwartungshaltung dessen Verständnis gut wiedergibt, in dem ein Teilnehmer dieser Ecosysteme sowohl als Provider als auch als Consumer auftritt.

3.2 Befürchtungen in Deutschland

Trotz dieser Einsichten nutzen aktuell in Deutschland noch unterdurchschnittlich viele Mitspieler die Möglichkeiten schnittstellenbasierter Software-Ecosysteme. Grundsätzlich muss eine solche Einführung als Innovation verstanden werden, und schon diese sind im Allgemeinen in Deutschland nicht sehr positiv besprochen. Bereits 2009 hat Bain & Company in einer Studie mit dem provokativen Titel »Deutsche Unternehmen sind innovationsfeindlich« (Janson 2009) aufgezeigt, dass die Strukturen und Prozesse in deutschen Unternehmen Innovationen nicht fördern. Konkret wurde als Innovationshinderung »die fehlende Zusammenarbeit von kreativen Vordenkern und pragmatischen Analytikern, geringe Weiterbildung- und Karrierechancen für Kreative sowie unzureichende Berücksichtigung der Kundenbedürfnisse im Innovationsprozess« beklagt. In einer ähnlichen Studie von 2011 mit dem vergleichbar klingenden Titel »Mehrheit deutscher Unternehmen ist innovationsfeindlich« wurden weitere Gründe erfasst. Demnach »dominiert das Vollkasko-Denken: Fehler vermeiden um jeden Preis«, auch wenn dies nach außen anders dargestellt wird: »70 Prozent aller Unternehmen kündigen Innovation zwar in ihren Hochglanzbroschüren an – konsequent umsetzen tun sie es aber nicht: Ideen werden totdiskutiert anstatt gefördert«. (Die Ideeologen 2015)

Dieses Vollkasko-Denken hat sich auch für den Bereich der schnittstellenbasierten Software-Ecosysteme in vielen Bitkom-Arbeitskreissitzungen gezeigt. Auch hier als Essenz wieder die Top-10 der intensiv diskutierten Risiken, die bei der Einführung von schnittstellenbasierten Software-Ecosystemen drohen:



Gewichtung durch direkte Befragung, n=17, Mehrfachnennungen möglich

Abbildung 4: Befürchtungen deutscher Unternehmen bei der Einführung schnittstellenbasierter Software-Ecosysteme

Die einzelnen Risiken werden im Folgenden jeweils kurz erläutert:

- **Interdisziplinäre Ansprache:** Mit diesem Risiko ist die Einsicht verbunden, dass schnittstellenbasierte Software-Ecosysteme viel mehr sind als nur eine neue Technik. Für ihren Erfolg ist demnach ganz wesentlich, dass viele unterschiedliche Stakeholder verschiedener Disziplinen miteinander kooperieren müssen, was entlang dieser Sicht nicht risikolos ist.

- **Neu:** Mit diesem Risiko ist die Einsicht verbunden, dass viele Menschen heute eine grundsätzliche Abwehrhaltung gegenüber Neuem haben, da jede Neuerung erst einmal das Verlassen der Komfortzone bedeutet, unabhängig davon, ob der Folgezustand ggfs. noch komfortabler ist.

- **Wenig deutsche Success Stories:** Mit diesem Risiko ist die Erkenntnis verbunden, dass ein signifikanter Großteil der Success Stories von API-basierten Software-Ecosystemen aus den USA stammt. Europäische oder gar deutsche Erfolgsgeschichten sind selten, was wiederum die Angst nährt, dass das Konzept eben nicht so trivial in Deutschland anwendbar ist.

- **»German Angst« mit neuen Risiken:** Dieses Risiko, bezogen auf das konkrete Phänomen API-basierte Software-Ecosysteme, bezeichnet die naturgemäße Unsicherheit des IT-Business und dessen infrastrukturellen Rahmens. Wie wird sich der Datenschutz entwickeln? Wie sehen die regulatorischen Vorgaben morgen aus? Welche Sicherheitslöcher werden ggfs. in Zukunft aufgedeckt werden. Diese Fokussierung auf mögliche K.O.-Argumente der Zukunft wird im Allgemeinen als German Angst bezeichnet und greift auch in diesem Bereich. Im Ausland wird

diese German Angst definiert als »Bedenkenträger [die] kaum einen Tag ohne Existenzangst verleben [...] Veränderungen hassen [...] und von einem Gefühl der permanenten Bedrohung getrieben sind« Sie lässt sich statistisch ermitteln und hat nach aktuellen Untersuchungen evtl. sogar epigenetische Ursprünge (Czycholl 2014).

- **Unklarer Nutzen:** Dieses Risiko bezeichnet die Unschärfe des mit API-basierten Software-Ecosystemen erwarteten Nutzens. Wie genau sehen zusätzliche Absatzmärkte aus, aus welchen Personen bestehen genau die Communities, die ggfs. Promotor einer neuen Schnittstelle werden.

- **Keine Out-of-the-Box-Solution:** Dieses Risiko bezieht sich auf die Notwendigkeit der Anpassung der Innovation an den jeweiligen Kontext. Es bedarf demnach noch des Customizings, um effektiv wirken zu können.

- **Gesamtheitlich revolutionär:** Dieses Risiko bezieht sich auf den disruptiven Charakter der Innovation und ihres möglichen Geltungsbereiches. Die Verwendung von Schnittstellen kann demnach revolutionär auf Arbeitsprozess, Organisationen und letztlich auch auf die Menschen wirken, deren Arbeitsumfeld und -weise sich durch die Einführung API-basierter Software-Ecosysteme dramatisch ändern können.

- **Unklare / große Regulationsconstraints.** Mit diesem Risiko ist das gerade in Deutschland besonders starke Regulationsnetz gemeint, das bezogen auf die schnittstellbasierten Software-Ecosysteme besonders durch Datenschutzaspekte (z. B. die Veröffentlichung personenspezifischer Daten über Schnittstellen), kartellrechtliche Fragen (z. B. die Bildung von Oligopolen innerhalb der Software-Ecosysteme), steuerliche Parameter (z. B. die Besteuerung von nutzungsbasierten Bezahlmodellen) und auch revisionsbezogene Regelwerke (z. B. die Einhaltung bestimmter Compliance-Anforderungen bei einer Schnittstellennutzung). Diese und weitere Aspekte können unterschiedlich stark auf die Innovation wirken und sich als Risiko durchaus negativ auswirken.

- **Noch geringe Kundennachfrage:** Hinter diesem Risiko steht die Wahrnehmung, dass das Angebot u. U. von nicht genügend Kunden wahrgenommen wird, so dass sich das Initialinvestment erst sehr langfristig – wenn überhaupt – rechnet.

- **Zu komplex / unverständlich:** Hinter diesem Risiko steht die Angst, das Phänomen API-basierter Software-Ecosysteme aufgrund dessen Komplexität nicht vollständig durchschauen zu können und damit weitere Risiken ggfs. zu übersehen.

Die numerischen Angaben zeigen, dass die erwarteten Risiken primär die Teilnahme an API-basierten Software-Ecosystemen als Provider berücksichtigen. Die reine Nutzung als Consumer wird nur durch einige wenige Aspekte besprochen.

In Summe zeigen beide Befragungen deutlich den aktuellen Status-Quo: Bei der Teilnahme eines Unternehmens als Provider innerhalb der API-basierten Software-Ecosysteme dominieren gerade in Deutschland noch viele Ängste. Die Ursachen liegen primär im universellen Charakteristikum des Neuen (interdisziplinäre Ansprache, Innovation, Änderungen) und weniger in der Thematik selbst. Die Teilnahme als Consumer wird dagegen mit deutlich weniger Risiken assoziiert, so dass hier die vielen Vorteile direkt zum Tragen kommen können. Dies erklärt, warum viele Unternehmen zwar Consumer von Schnittstellen meist amerikanischer Unternehmen sind, selbst aber die Vorteile als Provider erst sehr langsam angehen. Dieses Ungleichgewicht führt interessanterweise zu einer weiteren Stärkung API-basierter Software-Ecosysteme in den USA. Es bleibt abzuwarten, ob zum späteren Zeitpunkt, wenn auch die Rolle des API-Providers in Deutschland Fuß fasst, noch genügend Marktpotential vorhanden ist.

4 WOA: Architektur-Blueprints

4.1 WOA: Quo Vadis

Bisher haben sich Unternehmen durch externe & interne Firewalls gegen Angriffe auf das interne Unternehmensnetzwerk abgesichert. Die Grundlage für klassische Firewalls bildet das TCP/IP Transport-Protokoll mit entsprechenden IP-Adressen und TCP/UDP-Ports. Dienste, wie zum Beispiel eine Webseite oder eben eine API werden also auf einer IP-Adresse und einem entsprechenden Port angeboten. So ist zum Beispiel unter der IP: 139.2.165.93 und dem Port: 80 die Webseite des Bitkom zu erreichen.

Möchte ein Client nun eine Verbindung zu einem Service aufbauen, dann kontaktiert dieser die IP-Adresse und den Port des gewünschten Dienstes. Und genau hier setzen klassische Paket-Firewalls auf, welche auf der Netzwerk- & Transport-Ebene arbeiten. Anhand von festgelegten Regeln entscheidet die Firewall, ob ein TCP/IP Paket den Dienst kontaktieren darf oder geblockt wird. Diese Regeln basieren auf den Verbindungsdaten, also welche Quell-IP-Adresse (Client) versucht, auf welche Ziel-IP-Adresse & Port (Service) zuzugreifen.

Bei dem Beispiel der Bitkom Webseite handelt es sich um eine öffentliche Webseite, welche von jedem Anwender der Welt erreichbar sein soll. Die Firewall, welche den Webserver absichert, ist also so konfiguriert, dass jede IP-Adresse den Webserver unter dem HTTP-Standard-Port 80 ansprechen darf.

Paket-Firewalls können also den Datenverkehr auf der Transport- und Netzwerk-Ebene überprüfen und ggf. blockieren, jedoch ist eine Paket-Firewall nicht in der Lage, Anfragen auf Basis des Dateninhaltes zu überprüfen und zu blockieren. Web-APIs sind vergleichbare Dienste wie zum Beispiel die Bitkom-Webseite. Sie basieren ebenfalls auf dem TCP/IP Transport-Protokoll und werden unter einem bestimmten TCP/UDP-Ports angeboten.

Ein Beispiel für eine REST-API, die eine IP-Adresse in eine GEO-Location umsetzt, ist unter folgender Adresse zu finden: http://freegeoip.net/json/139.2.165.93. In diesem Beispiel wird die IP-Adresse (139.2.165.93) des Bitkom-Webservers in der URL an die API übergeben und als Ergebnis wird der Standort des Bitkom-Webservers in der folgenden Antwort von der API zurückgegeben:


```
{
    ip:   »139.2.165.93«,
    country_code:   »DE«,
    region_code:   »NW«
    region_name:   »Nordrhein-Westfalen«,
    city:   »Dortmund«,
    zip_code:   »Europe/Berlin«,
    latitude:   51.5198,
    longitude:   7.4378,
    metro_code:   0
    }
```


Quelle: http://freegeoip.net/json/139.2.165.93

Abbildung 5: Beispiel einer REST-API

Dieses einfache Beispiel für eine REST/JSON basierte API soll verdeutlichen, wie WebAPIs funktionieren.

Möchte ein Unternehmen nun Bestandteil eines schnittstellenbasierten Software-Ecosystems werden, also APIs anbieten oder konsumieren, dann ist es gezwungen, die vorhandenen Netzwerk-Firewall-Regeln so einzustellen, dass die APIs von externen Partnern auf der Netzwerk-Ebene erreichbar sind, also ganz ähnlich wie die beschriebene Bitkom Webseite, welche global erreichbar ist.

Diese notwendige Öffnung von Unternehmensgrenzen für APIs bietet selbstverständlich zusätzliche Chancen, erhöht aber zwangsläufig auch die Angriffsfläche für mögliche Angriffe gegen ein Unternehmen. Noch dazu stellen die angebotenen APIs nicht selten auch sensible Daten zur Verfügung, welcher vor Angriffen gesondert geschützt werden müssen. Diesen Schutz kann eine klassische Netzwerk-Firewall nicht bieten, da dieser Schutz auf Basis anderer inhaltsspezifischer Regeln und nicht auf TCP/IP-Paket-Ebene erfolgen muss.

Diese Rolle übernehmen in der Regel sogenannte API-Gateways bzw. API-Firewalls. Ein API-Gateway wird in der typischen Architektur in der DMZ des Unternehmens, also zwischen externer & interner Unternehmens-Firewall, platziert und dient als zentraler Einstiegspunkt für sämtliche API-Anfragen eines Unternehmens.

Die externe Netzwerk-Firewall untersucht den eingehenden Datenverkehr auf Netzwerkebene und entscheidet abhängig von der angeforderten IP-Adresse und Port, ob es sich um eine API-Anfrage handelt. Wenn ja, dann wird die Anfrage zum API-Gateway weitergeleitet.

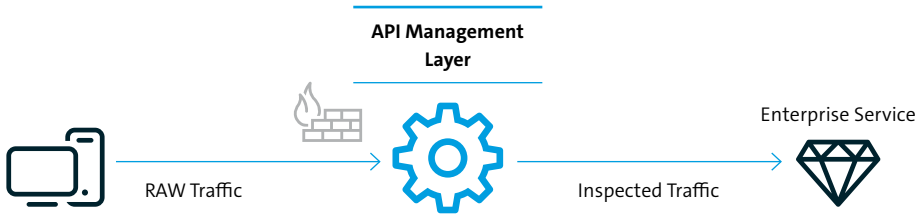


Abbildung 6: API-Management-Layer als zentraler Einstiegspunkt für API-Anfragen

Das API-Gateway arbeitet im Prinzip wie ein Proxy. Das heißt, dass es die API-Anfragen entgegen nimmt, die konfigurierten Regeln anwendet und danach die gesicherten Anfragen an das Backend weiter leitet. Das Thema Regeln und Möglichkeiten der Absicherung auf dem API-Gateway wird weiter unten noch beschrieben.

Diese Architektur gestattet es einem Unternehmen, folgende Aspekte zu gewährleisten:

- Absicherung von Anfragen auf Netzwerk-Ebene durch vorhandene Netzwerk-Firewalls
- zentrale Überwachung sämtlicher API-Anfragen
- zentraler Policy-Enforcement-Point für API-Anfragen
- zentrale Anwendung der Sicherheitsrichtlinien des Unternehmens

Aufgrund der Tatsache, dass der API-Layer eine zentrale Rolle für API-Anfragen einnimmt, also idealerweise sämtliche Anfragen eines Unternehmens entgegennimmt und verarbeitet, muss dieser mit der steigenden Anzahl von API-Anfragen wachsen bzw. skalieren können. Denn ein maßgeblicher Punkt für die Akzeptanz und den damit verbundenen Erfolg der angebotenen APIs ist eine möglichst schnelle Antwortzeit der angebotenen Services. Gerade im Bereich von mobilen Anwendungen, welche häufig auf APIs zugreifen, ist es wichtig, dass diese extrem responsive sind. Nur so ist sichergestellt, dass mobile Anwendungen und damit auch die zugrundeliegenden APIs erfolgreich sind.

Des Weiteren muss die Architektur des API-Layers auch sicherstellen, dass dieser hoch verfügbar ausgelegt ist. Auch schon kleinere Ausfälle im Minutenbereich führen bei erfolgreichen Endanwendungen zu negativen Erlebnissen und entsprechenden Bewertungen durch den Anwender.

Um beide Anforderungen, also Skalierbarkeit & Hochverfügbarkeit, abbilden zu können, empfiehlt sich eine Cluster-Architektur, wie in dem dargestellten Schaubild (siehe Abbildung 7). Dabei sollte jede Instanz aktiv sein, es sich also um einen sog. Active-Active-Cluster handeln.

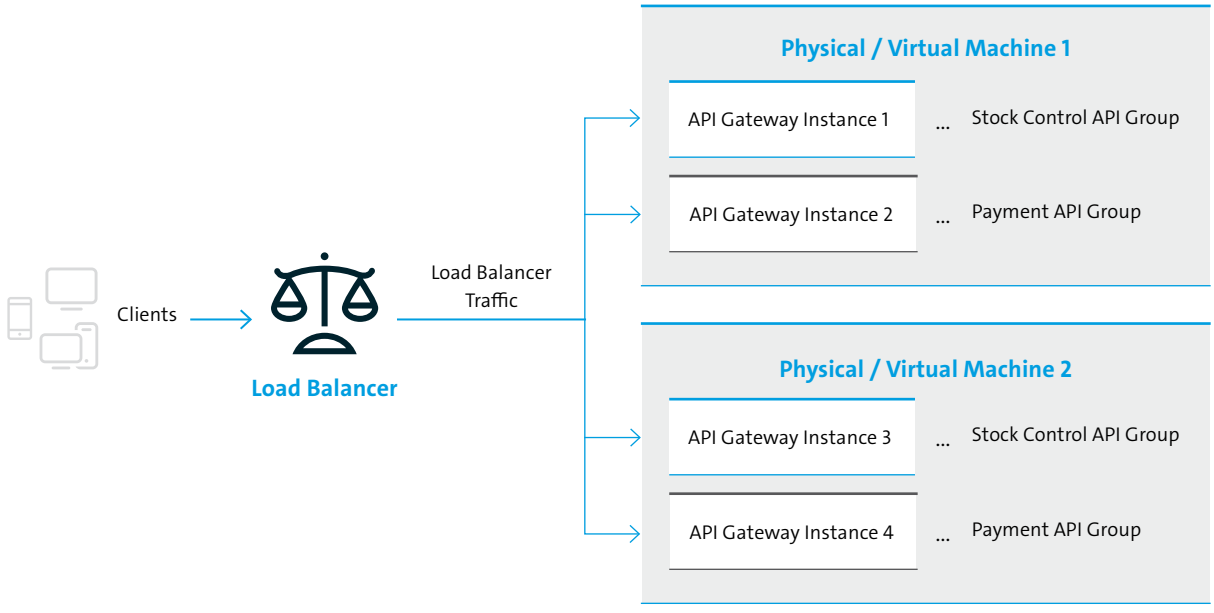


Abbildung 7: Umsetzung des API-Layers durch eine Cluster-Architektur

Hierbei werden API-Anfragen der externen Clients durch die externe Firewall nicht direkt auf das API-Gateway gesendet, sondern an einen vorgelagerten Load-Balancer. Dieser verteilt die Anfragen entweder gleichmäßig oder abhängig von der Auslastung an eine der API-Gateway Instanzen.

Von Vorteil ist es, wenn die API-Gateway Instanzen »stateless« ausgelegt sind. Das heißt, dass eingehende Anfragen von jeder beliebigen Gateway-Instanz ausgeführt werden können, da diese keinen Status (z. B. Session-Information) speichern. Damit ist der Load-Balancer in der Lage, die Anfragen völlig eigenständig zu verteilen und kann so mehr Durchsatz auf dem API-Layer erreichen.

Auch in einer so geplanten Cluster API-Gateway Architektur muss sichergestellt werden, dass dieser zentral administriert und überwacht werden kann.

Sind die zur Verfügung gestellten APIs erfolgreich, wird im Laufe der Zeit immer mehr Datenaufkommen auf den API-Layer treffen und muss verarbeitet werden. Ab einem gewissen Datenvolumen werden Anfragen mit den verfügbaren API-Instanzen nicht mehr die gewünschten Antwortzeiten erreicht. Der API-Layer muss nun weiter skalieren können, um das Datenvolumen bewältigen zu können. Das bedeutet, dass weitere Server dem vorhandenen Cluster hinzugefügt werden und so der Load-Balancer das Datenvolumen weiter verteilen zu kann.

Der an das Datenvolumen angepasste API-Gateway-Cluster ist jetzt in der Lage, deutlich mehr API-Anfragen zu verarbeiten. Da eine API in den meisten Fällen nicht ohne bereits verfügbare Business Logik auf dem API-Layer entsteht, sondern die Daten durch ein Backend bereitgestellt werden, hat man mit der Erweiterung des API-Clusters evtl. nur den Bottleneck Richtung API-Backend verschoben.

Viele Backend-Systeme sind nie dafür konzipiert worden, dass ihre Daten bzw. Dienste über APIs einer breiten Öffentlichkeit zugänglich gemacht werden. Damit ergibt sich die Gefahr, dass Millionen von API-Anfragen, die nun ohne Probleme den API-Layer passieren, das Backend negativ beeinflussen und sogar zum Absturz bringen.

Um dies zu vermeiden, muss ein API-Layer in der Lage sein, redundante Informationen zu cachen und API-Anfragen zu »throttlen«.

- Caching bedeutet in diesem Zusammenhang, dass das Backend nicht für jede API-Anfrage erneut kontaktiert werden muss, sondern die Information aus einem Cache auf dem API-Layer gezogen werden. Es gibt eine ganze Reihe von Informationen, die entweder vollständig read-only sind oder sich nur sehr selten ändern. Typische Beispiele sind Kunden- oder Artikelstammdaten. Das Gateway speichert diese einzelnen Entitäten fein granuliert in Caches mit verschiedenen Lebenszeiten und bezieht diese Entitäten bei der nächsten API-Anfrage wieder aus dem Cache.

- Für ein effektives Throttling werden auf dem API-Layer für unterschiedliche APIs verschiedene Schwellwerte festgelegt. Diese steuern entweder, wie viele Anfragen insgesamt oder pro Client auf eine API in einem definierten Zeitraum ausgeführt werden dürfen. Der System-Grenzwert dient hauptsächlich dazu, die Backend-Systeme vor einer Überflutung mit API-Anfragen zu schützen. Die Konfiguration von Schwellwerten pro Client ist ein Kontrollinstrument für den API-Anbieter, welche es zum Beispiel erlaubt zu steuern, dass wichtige Clients (z. B. Kunden) mehr API-Anfragen stellen können, als andere Partner.

Diese beiden Funktionalitäten sind heute für ein API-Gateway wesentlich. Ein reines Öffnen via APIs ohne solche Funktionen ist nicht nur sofort risikobehaftet, sondern auch nicht langfristig erfolgversprechend.

4.2 Security-Anforderungen an WOA

Wie beschrieben, ist der API-Layer, neben der externen Firewall, die »First-Line-of-Defence« für exponierte APIs. Er muss vor Angriffen auf APIs, welche teilweise empfindliche und geschäftskritische Daten bereitstellen, schützen. Ein erfolgreicher Angriff stellt einen immensen Schaden für ein Unternehmen dar und das beinhaltet nicht nur den Verlust von sensiblen Daten, sondern darüber hinaus einen Markenschaden oder rechtliche Konsequenzen.

Dienste, die durch WebAPIs, also SOAP oder REST-Services, bereitgestellt werden, beruhen in den meisten Fällen auf internen Backend-Anwendungen. Diese mitunter alten und über die Jahre gewachsenen Anwendungen sind nie dafür konzipiert worden, dass ihre Dienste über APIs einer großen und öffentlichen Anzahl von Anwendern bereitgestellt werden. Infolgedessen müssen diese Backend-Systeme vor Angriffen geschützt werden.

Da WebAPIs auf den gleichen Technologien wie klassische Web-Anwendungen aufbauen, sind es oft die gleichen oder ähnlich aufgebaute Angriffsvektoren. Diese Angriffsvektoren können in 3 verschiedene Ebenen aufgeteilt werden.

1. Die Netzwerk-Schnittstelle kann durch Denial-of-Service (DoS)-Angriffe zum Absturz des Backend-Systems führen. Wenn ein Unternehmen eine spezielle API für die Nutzung im schnittstellenbasierten Software-Ecosystem freigibt, so ist die Anzahl der diese Schnittstelle nutzenden Consumer häufig unbekannt. Eine übertriebene Nutzung, z. B. im Kontext einer DoS-Attacke oder durch die schier große Anzahl von Anwendern, würde dann zwangsläufig dazu führen, dass sämtliche von dieser Schnittstelle abhängigen Systeme in Mitleidenschaft gezogen würden. Da es sich hierbei dann allerdings schon um unternehmensinterne Systeme handelt, könnte das gesamte IT-Netz in Mitleidenschaft gezogen werden. Von daher ist eine wesentliche Aufgabe der API-Gateways die Nutzung einer bestimmten Schnittstelle quantitativ einzuschränken (Throttling, s. o.). Sowohl gegen beabsichtigte DoS-Angriffe oder durch ungewolltes »Friendly-Fire« einer schlecht programmierten Endanwender-Anwendung.

2. Eine ungenügende Absicherung von Zugriffsrechten kann es Angreifern erlauben, unberechtigt Daten zu erlangen. Firmeninterne Schnittstellen sind meist bzgl. der Zugriffsrechte einfach zu steuern, da es ein zentrales Identity- & Access-Management-System gibt. Dort sind alle Nutzer hinterlegt und jedes System, das aufgerufen wurde, prüft dort, ob der Consumer die entsprechenden Rechte hat. Consumer von Schnittstellen einer WOA können nicht immer zuvor ins eigene IAM-System übernommen werden, so dass ggfs. schwergewichtigere Mechanismen und Konzepte zum Einsatz kommen. Kapitel 5 führt genau für diesen Bereich neue Lösungsansätze ein.

3. Unerlaubte Datenmanipulation und Schadcode: Als letzte und tiefste Form des Angriffs sind Datenmanipulationen des mit der Schnittstelle verbundenen Backends aufzufassen: Hat ein Angreifer die API aufgerufen und bestimmte Zugriffsrechte erhalten, so besteht bei fehlender Absicherung der APIs theoretisch die Möglichkeit, die Daten im Backend-System zu manipulieren. Eine SQL-Injection-Attacke fällt genau in diese Richtung.

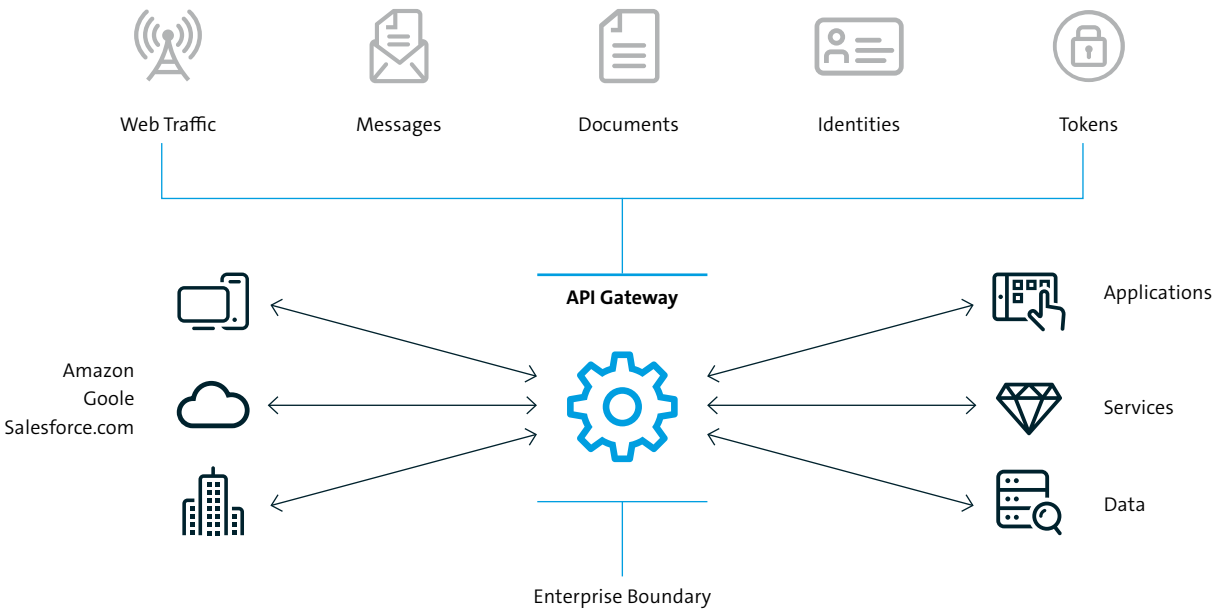


Abbildung 8: Rolle des API-Gateways bei schnittstellenbasierten Software-Ecosystemen

Das API-Gateway wird zur Absicherung auf allen diesen drei Ebenen als Policy-Enforcement-Point (PEP) bezeichnet: Dort werden Nutzer von API Anfragen authentifiziert, autorisiert und dort wird geprüft, dass gegen das API keine böswilligen Angriffe gefahren werden. Effektive Möglichkeiten eines geeigneten Monitorings von Volumen und Verhalten der Clients sind hier heute als De-facto-Standard zu nennen.

Angriff auf die Netzwerk-Schnittstelle

Oft zielen Angriffe auf die Netzwerk-Schnittstelle auf Schwächen in der IT-Infrastruktur. So ist es häufig möglich, Netzwerk-Firewalls mit manipulierten SYN-Paketen zum Absturz zu bringen

und damit ein Unternehmen lahm zu legen. Firewall-Hersteller haben daraus gelernt und sind nun in der Lage, diese Angriffsformen zu erkennen und erfolgreich abzuwehren. Um eine API für die Öffentlichkeit frei zu geben, wird diese wie beschrieben auf der Netzwerk-Ebene für die Öffentlichkeit auf der externen Firewall zugänglich gemacht. Anfragen werden nun ungehindert von der externen Firewall auf den API-Layer weitergeleitet. Nun muss der API-Layer mit anderen Formen von Denial-of-Service-Angriffen umgehen können, ohne selber in Mitleidenschaft gezogen zu werden oder gar abzustürzen. Ein API-Gateway muss viele tausende HTTP-Anfragen pro Sekunde entgegennehmen, prüfen und ggf. die Verbindung beenden können, ohne selber abzustürzen. Ein Absturz des API-Layers würde dazu führen, dass die zur Verfügung gestellten APIs nicht mehr erreichbar sind, Endanwendungen würden nicht mehr funktionieren und es kommt zum Image-Verlust.

Absicherung von Zugriffsrechten
Bisher haben nur intern genutzte und kontrollierbare Anwendungen Dienste von Backend-Systemen verwendet. Die Authentisierung & Autorisierung erfolgte in diesem Fall über etablierte Identitätssysteme (s. u. unter Identity-Management). Werden diese internen Dienste nun als APIs der Öffentlichkeit oder Partnern zu Verfügung gestellt, dann muss diese Absicherung mit Hilfe von anderen Techniken realisiert werden. Dabei sollten die Zugriffsrechte für eingehende API-Anfragen direkt auf dem API-Layer und nicht in der Backend-Anwendung stattfinden, da der API-Layer andere Kontrollmöglichkeiten für den Datenfluss hat. Da der API-Layer als Proxy für API-Anfragen arbeitet und somit Kontrolle über jede Anfrage hat, kann sich dieser mit den vorhandenen Identitätssystemen des Unternehmens verbinden, um Anfragen zu authentisieren. Ferner können die beschriebenen Techniken wie Caching- & Session-Management oder eine Vorab-Validierung via IP-Adresse oder Client-Zertifikat für eine Entlastung der Identitätssysteme sorgen, da nur initiale & valide Anfragen vom Identitätssystem verarbeitet werden müssen.

Eine Authentisierung und Autorisierung externer Clients, genutzt wie bisher für interne Anwender, stellt wieder einen möglichen Angriffsvektor für einen Denial-of-Service-Angriff gegen die Identitätssysteme eines Unternehmens dar.

Unerlaubte Datenmanipulation und Schadcode
Ein weiteres großes Sicherheitsrisiko ist der mögliche Angriff über unerlaubte Datenmanipulation bzw. Schadcode. Welche Art Schadcode oder Manipulation zum Einsatz kommt, hängt von vielen Faktoren und dem Wissen des Angreifers über die Funktionsweise einer API ab. So ist es möglich, über Session-Hijacking die Session eines anderen Anwenders zu stehlen und in dessen Namen API-Anfragen auszulösen. Oder es ist möglich, eigene Session-Information mit erweiterten Attributen zu versehen, um Administratoren-Zugriff oder andere spezielle Rechte zu erlangen. Im Falle von Viren ist das Ziel des Angreifers, diese im IT-Netz eines Unternehmens zu platzieren. Ein erfolgreich platzierter Virus kann dann Daten, wie z. B. persönliche Informationen ausspähen oder dem Angreifer eine Hintertür für weitere Angriffsmöglichkeiten öffnen.

Viren können z. B. als Anlage für einen SOAP-Service mit gesendet werden. Ein Beispiel dafür wäre ein PDF-Dokument mit eingebettetem Virus zur Weiterverarbeitung, welcher dann durch einen Mitarbeiter durch öffnen des PDF-Dokuments installiert wird.

Schadcode für eine API kann ganz unterschiedlich eingeschleust werden. Dabei kann dieser Code entweder auf die Backend-Systeme des API-Anbieters abzielen oder auf andere Anwender der API, welche diese zu einem späteren Zeitpunkt aufrufen. Denkbar wäre hier ins API-Backend eingeschleuster Javascript-Schadcode, der bei anderen API-Konsumenten ausgeführt wird, um so zum Beispiel Sessions zu stehlen.

Ein API-Gateway muss Möglichkeiten bieten, diese vielfältigen Angriffsvarianten erkennen zu können. Dennoch ist das API-Gateway nur das Mittel zum Zweck, und es ist die Aufgabe des API-Anbieters, freigegebene APIs entsprechend zu sichern. So müssen für eine SOAP/XML-basierte API andere Regeln definiert werden:

- Ist die SOAP-Action wie erwartet?
- Nur XML-Nachrichten dürfen akzeptiert werden.
- Die XML-Nachrichten dürfen nur in einer bestimmten Größe und XML-Komplexität kommen.
- Ein XML-Schema zur Validierung der Nachricht wird eingesetzt.
- Nachrichten werden auf Viren überprüft.
- Der Playload beinhaltet keinen böswilligen Code, wie z. B. eine Injektion.

als für eine REST/JSON-basierte API:

- Ist das HTTP-Verb (GET, POST, PUT, etc.) korrekt?
- Der Nachrichteninhalt muss ggf. JSON sein.
- Die Nachrichtengröße wird überprüft.
- Die Nachricht wird mittels JSON-Schema validiert.
- Die Query- & Header-Attribute werden überprüft.

Beide Listen haben keinerlei Anspruch auf Vollständigkeit. Sie sollen deutlich machen, dass der API-Provider die richtigen Sicherheitsmechanismen für die entsprechenden APIs oder gar Backend-Systeme wählen muss. Es kann durchaus sinnvoll sein, spezielle Absicherungen auf dem API-Layer für eine API zu konfigurieren, die gegen ein bestimmtes Backend mit einer speziellen Gefährdung geht.

Es mag nicht direkt auf der Hand liegen, aber es kann absolut richtig sein, dass nicht nur die API-Anfrage vom Client, sondern auch die API-Antwort, welche vom Backend zurückkommt, überprüft wird. Hier sind Gegenstand der Prüfung zum Beispiel, ob sensible Daten das Unternehmen verlassen oder schlicht, ob auch die Antwort der erwarteten Größe entspricht. So lässt sich der Verlust von sensiblen Daten, wie zum Beispiel Kundendaten oder Unternehmens-Wissen erkennen und unterbinden.

Monitoring und Alerting

Für einen Angreifer ist es der Idealfall, wenn dieser seinen Angriff völlig unbemerkt durchführen konnte. Also das System hacken, die gewünschten Daten stehlen und danach seine Spuren wieder entfernen. So kann er jederzeit wieder zurückkommen, erneut einbrechen und Daten

stehlen. Oder einen Angriff auf andere Systeme mit Hilfe der gleichen Sicherheitslücke durchführen.

Nur wenn der API-Layer ein geeignetes Monitoring und Alerting bereitstellt, ist es möglich, laufende Angriffe auf einem API-Layer festzustellen und entsprechende Gegenmaßnahmen einzuleiten. Mögliche Szenarien, Angriffe zu erkennen sind, etwa unverhältnismäßig viele Anfragen auf eine API insgesamt oder von einem bestimmt Client zu erhalten. Eine weitere Anomalie ist eine ungewöhnlich hohe Anzahl von Fehlern auf einer API, die im normalen Alltag ohne Probleme funktioniert.

Das API-Gateway sollte die Möglichkeit haben, solche Unregelmäßigkeiten zu erkennen, zu loggen und sich mit aktiven Benachrichtigungen / Alerts in zentralen Monitoring-System zu integrieren. Das operative Monitoring wird damit in die Lage versetzt, Angriffe frühzeitig zu erkennen, diese zu analysieren während sie passieren und wenn nötig Schwachstellen in den angewendeten Sicherheitsregeln auf dem API-Gateway zu schließen.

5 Identitätsmanagement, die große Herausforderung

5.1 Grenzen des klassischen IAMs

Klassisches Identity and Access Management (IAM) verwaltet die digitalen Identitäten und Nutzungsrechte der Mitarbeiter eines Unternehmens. Dabei wird zum einen die Authentizität der Identität eines Nutzers (Authentifizierung) sichergestellt, zum anderen wird geregelt, wer in welchem Umfang auf welche Ressourcen und Anwendungen des Unternehmens zugreifen darf (Autorisierung). Die Administration des Lebenszyklus einer Mitarbeiteridentität – von der Einstellung bis zum Ausscheiden aus dem Unternehmen – erfolgt zentral, wobei das HR-System des Unternehmens meist die führende Datenquelle für die Bereitstellung der Nutzeridentitäten ist (Provisioning).

Ein unternehmensinternes Identity Management ermöglicht es Mitarbeitern, sich für die Nutzung eines Dienstes oder einer Ressource des Unternehmens zu authentifizieren. Nach einer erfolgreichen Authentifizierung erfolgt idealerweise die Autorisierung, die anhand von Rollen, die der Identität eines Mitarbeiters zugeordnet sind, oder mittels definierter Zugriffsregeln entscheidet, ob der Mitarbeiter berechtigt ist, einen Dienst oder eine Ressource zu nutzen. Sollen Dienste des eigenen Unternehmens für Nutzergruppen außerhalb der Unternehmensgrenzen geöffnet werden, muss eine Nutzung von Identitäten natürlich auch über die Grenzen des eigenen Unternehmens hinaus möglich sein. Identitäten sind dann nicht länger nur mit den internen Mitarbeitern des Unternehmens verbunden, sondern auch mit externen Nutzern wie Endkunden und Mitarbeitern von Partnerunternehmen. Als Konsequenz daraus ergibt sich, dass die Identität eines Nutzers nicht mehr zwingend da verwaltet wird, wo sich auch eine Schnittstelle oder Ressource befindet, auf die der Nutzer zugreifen möchte.

Als gradlinige Scheinlösung erfolgte gerade in den Anfängen der schnittstellenbasierten Software-Ecosysteme das Identitätskopieren: Jeder Consumer einer API einer Unternehmung musste sich dort in das firmeninterne IAM-System als Gastnutzer eintragen lassen, damit das API wie gewohnt das zentrale IAM-System als die finale Authentifizierung und Autorisierung eines jeden Nutzers verwenden kann. Da die Anzahl von Consumern allerdings im Vorfeld der Veröffentlichung einer Schnittstelle nur sehr schwer vorhersehbar ist und der konkrete Consumerkreis zudem extrem volatil ist, erzeugte dieses Lösungsmuster auf Provider-Sicht, der nämlich jeden neuen Consumer in das IAM-System eintragen muss, starken Zusatzaufwand.

Das klassische IAM ist daher für solche firmenübergreifenden API-Nutzungsszenarien nur wenig geeignet, und es werden passende neue IAM-Konzepte und -Technologien für API-basierte Software-Ecosysteme benötigt, die nicht mehr die zentrale Verwaltung der unternehmensinternen Mitarbeiter in den Mittelpunkt stellen, sondern einen nutzerzentrierten Ansatz verfolgen und dem Nutzer mehr Verantwortung für seine digitale Identitäten übertragen.

5.2 Erweiterte Konzept für API-basierte Ecosysteme

Klassische Identity-Management-(IM)-Ansätze basieren auf der zentralen Kontrolle und dem Besitz von Identitäten und sind daher nur wenig geeignet, um Identitäten zu authentifizieren, die nicht mehr im eigenen zentralen Identitätssilo verwaltet werden, sondern sich verteilt in unterschiedlichen Identitätsdomänen befinden. Um diesen geänderten Bedingungen gerecht zu werden, werden neue IM-Ansätze benötigt, die es ermöglichen, Authentifizierungen über die Grenzen von Identitäts- und Unternehmensgrenzen auszutauschen und nicht mehr ausschließlich auf dem Besitz von Identitäten basieren. Das Konzept der Vertrauensbeziehungen zwischen Organisationen bietet einen Lösungsansatz für genau diese Problemstellung. Dabei vertraut eine Organisation der Authentifizierung, die eine andere Organisation für einen Nutzer ausgestellt hat, ohne den Nutzer selbst zu kennen. Auf Basis dieser Vertrauensbeziehung kann dann die Authentifizierung eines Nutzers zwischen unterschiedlichen Sicherheitsdomänen, ohne die Notwendigkeit einer redundanten Verwaltung seiner Identität, erfolgen. Ein Benutzer kann so eine einzige digitale Identität universell für die Authentifizierung von Zugriffen auf APIs und Ressourcen bei voneinander unabhängigen Service-Anbietern nutzen. Dies ist exakt der Use Case, wie er für breit angewendete APIs schnittstellenbasierter Software-Ecosysteme zum Einsatz kommt.

Die untenstehende Abbildung verdeutlicht das Konzept. Zwischen dem zentralen Identity Provider (IDP) und den voneinander unabhängigen Identitätsdomänen A, B und C ist jeweils eine bilaterale Vertrauensbeziehung etabliert. Ein ID Token, der von dem IDP nach erfolgreicher Authentifizierung für den Nutzer ausgestellt wurde, kann von dem Nutzer wiederum für die Authentifizierung und die Nutzung von geschützten APIs bei den Identitätsdomänen A, B und C verwendet werden. Dabei ist der Nutzer nur dem zentralen IDP bekannt und die Akzeptanz des vom IDP ausgestellten ID Tokens erfolgt auf Basis der bestehenden Vertrauensbeziehungen.

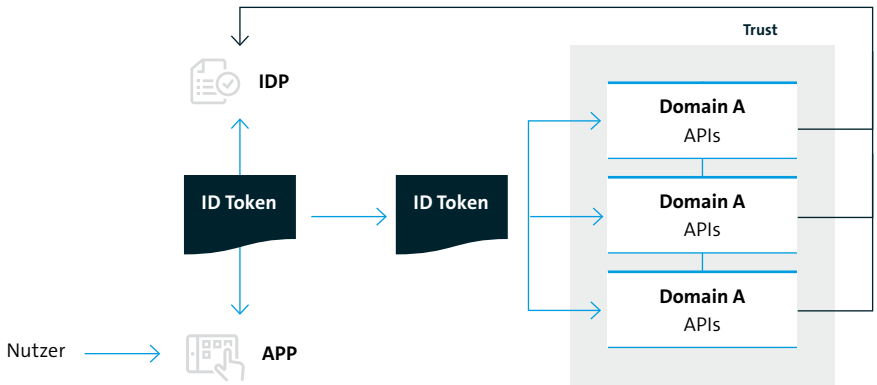


Abbildung 9: Trustbeziehungen zwischen App und IMs

Als Voraussetzung einer solchen Vertrauensbeziehung zwischen zwei voneinander unabhängigen Organisationen muss sowohl organisatorisch Vertrauen hergestellt werden, als auch dieses Vertrauen technisch umgesetzt werden. Für die technische Umsetzung einer Vertrauensbeziehung sind Standards wie SAML und WS-Federation in der unternehmensübergreifenden B2B-Kommunikation schon länger etabliert.

SAML und WS-Federation sind sichere und bewährte Federation-Standards, die heute von vielen Middleware Produkten unterstützt werden. Sie bergen aber auch einen hohen Grad an Komplexität, da sie auf den schwergewichtigen Standards des WS*-Stacks wie XML-Signature basieren. Insbesondere für den Einsatz auf ressourcenbeschränkten mobilen Endgeräten sind sie daher nicht geeignet, schließlich wurden sie bereits vor zehn Jahren für die Machine-to-Machine-Kommunikation in einer SOA entwickelt. In der mehr lose gekoppelten Kommunikation über Web-APIs im Kontext von WOA übernehmen daher neuere leichtgewichtigere Technologien wie OpenID Connect diese Rolle. OpenID Connect basiert dabei auf den gleichen Federation-Konzepten wie SAML und WS-Federation, setzen diese aber basierend auf leichtgewichtigen Standards wie z. B. JSON Web Token und JSON Web Signature um. Dadurch wird die Nutzung von unternehmensübergreifenden föderierten Authentifizierungen auch im Rahmen einer WOA unter Verwendung eines breiten Spektrums von Endgeräten ermöglicht.

Somit können auf Basis von Vertrauensbeziehungen Authentifizierungen zwischen unterschiedlichen Sicherheitsdomänen erfolgen, ohne die Notwendigkeit, eine redundante Verwaltung von Identitäten in unterschiedlichen Domänen erzwingen zu müssen. Ein Benutzer kann so eine einzige digitale Identität für die Authentifizierung von Zugriffen auf APIs und Ressourcen bei voneinander unabhängigen Service-Anbietern nutzen. Er muss seine Authentifizierung auch nur einmal belegen, da dieser Beleg dann an die angeschlossenen Trusted-Systeme weitergeleitet wird. In diesem Zusammenhang spricht man dann auch von Federated Single Sign On (SSO).

Im Folgenden sollen die beiden unterschiedlichen Sichtweisen auf Identitäten (zentrale Identitätsverwaltung mit klassischem IAM und verteilte Identitätsverwaltung für unternehmensübergreifende API-basierte Software-Ecosysteme) an dem Beispiel Single Sign On (SSO) verdeutlicht werden.

SSO ermöglicht es Nutzern, mehrere separate geschützte Anwendungen nach einer einmaligen Authentifizierung zu verwenden. Im Rahmen eines SSO-Prozesses wird ein Nutzer mit einer einmaligen Authentifizierung automatisch für mehrere separate geschützte Anwendungen authentifiziert, für die er die geeigneten Zugangsberechtigungen besitzt. Es besteht aber ein Unterschied hinsichtlich der Motivation zwischen dem als Enterprise SSO (ESSO) bezeichneten Single Sign On in den Grenzen eines Unternehmens und Federated SSO.

Enterprise SSO (ESSO)
Unternehmensintern bedeutet die Einführung von SSO vor allem eine Steigerung der Produktivität, indem eine wiederholte Eingabe von Authentifizierungsdaten durch den Benutzer unnötig wird. Auch ein kostenreduzierender Effekt durch eine Minimierung der Anzahl passwortbezogener Helpdesk Calls kann in diesem Zusammenhang angenommen werden.

Dieses SSO funktioniert nur bedingt für APIs von schnittstellenbasierten Ecosystemen: Wie weiter oben formuliert, ist es theoretisch zwar möglich, jeden Consumer eines API in das zentrale API aufzunehmen, aber der Mehraufwand auf Seiten des Providers ist nicht unerheblich. Ein SSO würde in diesem Fall dann auch nur für die APIs des einen Providers gültig sein; sobald ein API eines anderen Providers aufgerufen würde, bedürfte es wieder der separaten manuellen Authentifizierung. Dieses Vorgehen ist zwar grundsätzlich möglich, aber für die Zukunft sicher nicht empfehlenswert:

- Die Pflege der Einträge von Consumern ist zeitaufwändig und aufgrund der fehlenden Governance auf die Menschen nur bedingt möglich. Während ein Mitarbeiter wohldefiniert über die HR-Abteilung den Job verlässt und damit für die interne IT nicht mehr als Nutzer in Frage kommt, ist dieser strukturierte Prozess bei externen Consumern kaum umzusetzen.

- Die Menge von Consumern, die in das zentrale IAM-System eingepflegt werden müssen, ist im Vorfeld nur schwer abschätzbar. Schnittstellen von Twitter und SalesForces werden heute aber von mehreren Millionen Consumern intensiv genutzt, was die Leistungsfähigkeit von »normalen« IAM-Systemen an seine Grenzen führt.

- Kein SSO möglich. Insbesondere WOAs zeichnen sich durch eine hohe Leichtgewichtigkeit aus: Existieren zwei APIs mit ähnlicher Funktionalität, dann wird diejenige verwendet, die ein echtes unternehmensübergreifendes SSO ermöglicht.

Federated SSO

Für externe Identitäten stellt Enterprise-übergreifendes SSO die höchste Reifestufe für die meisten Use Cases dar, indem Benutzer mit einer Authentifizierung bei ihrem eigenen Unternehmen oder einem unabhängigen Identitäts-Provider auf die APIs und Ressourcen eines anderen Unternehmens (Service-Provider) ohne erneute Registrierung und Anmeldung zugreifen können. Diese Nutzung von Authentifizierungen über unterschiedliche Sicherheitsdomänen hinweg ermöglicht somit erst den effizienten, sicheren Gebrauch von geschützten APIs und den Aufbau von leichtgewichtigen WOA-basierten API-Ecosystemen.

Die Verwendung des Logins von sozialen Medien, wie etwa Facebook oder Twitter für die Authentifizierung bei Services Dritter, ist eine bereits weit verbreitete Anwendung von Federated SSO. Da viele anmeldepflichtige Dienste im Internet die Verwendung von Social Logins akzeptieren, kann ein Nutzer durch die Verwendung einer einzigen universellen digitalen Identität eine Vielzahl von Internet-Angeboten aufrufen, ohne die Notwendigkeit sich immer wieder erneut zu authentifizieren. Solch eine Nutzung einer universellen digitalen Identität wird oft auch als Bring-your-own-Identity (BYOID) bezeichnet.

In diesem Zusammenhang werden auch die Begriffe »User Centric« und »Consumer Identity« gebraucht. Damit ist gemeint, dass der einzelne Nutzer mehr in den Mittelpunkt des Identifizierungsprozesses rückt. Die Verantwortung für seine digitale Identität behält der Nutzer und wird nicht einem zentralen Identity Management übertragen. Der Nutzer soll selber entscheiden können, welche Identitätsinformationen ein Service-Anbieter von ihm bekommen darf.

Nutzerzentrierte Ansätze leisten somit auch einen wichtigen Beitrag zum Schutz von Identitätsinformationen und somit auch für den Schutz der Privatsphäre.

Die Nutzung von digitalen Identitäten über unterschiedliche Sicherheitsdomänen hinweg bedeutet aber auch, dass die Verantwortung für Identitäten und Berechtigungen nicht mehr zentral in einem Unternehmen liegt, wie in klassischen IAM-Szenarien üblich. Daher müssen Nutzer ihre Identitätsdaten und Rechte auch selbst verwalten können, um auf diese Weise die Kontrolle über ihre eigenen Identitätsdaten zu behalten.

Am Beispiel des populären Security-Standards OAuth 2.0 soll im Folgenden die Bedeutung von nutzerzentrierten Identity-Management-Ansätzen verdeutlicht werden.

OAuth 2.0 adressiert dabei ein Problem, das in Web-orientierten Architekturen heute als ein Standardszenario angesehen werden kann. Eine Applikation greift im Namen eines Nutzers ohne Kenntnis eines Passwortes auf eine geschützte Ressource des Nutzers zu. Das Szenario kann in Anlehnung des Beispiels aus der OAuth-2.0-Spezifikation (RFC 6749) verdeutlicht werden. Ein Nutzer möchte seine bei einem Cloud-Anbieter gespeicherten Fotos von einem Fotoprint-Service ausdrucken lassen. Der Print-Service soll dabei direkt auf die Fotos in der Cloud zugreifen können, ohne das Passwort des Nutzers für dessen Cloud-Account zu kennen.

OAuth 2.0 definiert in diesem Zusammenhang 4 Rollen:

1. Resource Owner (RO): Im Print-Service-Beispiel wird diese Rolle von dem Nutzer eingenommen, dessen Fotos ausgedruckt werden sollen.

2. Client: Eine Portal-Anwendung oder eine Mobile-App des Print-Services, die dem Nutzer die Auswahl der auszudruckenden Fotos ermöglicht, nimmt im Print-Beispiel diese Rolle ein.

3. Authorization Server (AS): Ein Authorization Server fungiert dabei als ein sogenannter Security Token Service (STS), der Security Token als Berechtigungsnachweis für definierte Zugriffe auf Ressourcen oder APIs ausstellt.

4. Resource Server (RS): Hierbei handelt es sich um einen Service, der die eigentliche Ressource – in dem Beispiel die Fotos des Nutzers – über APIs zur Verfügung stellt.

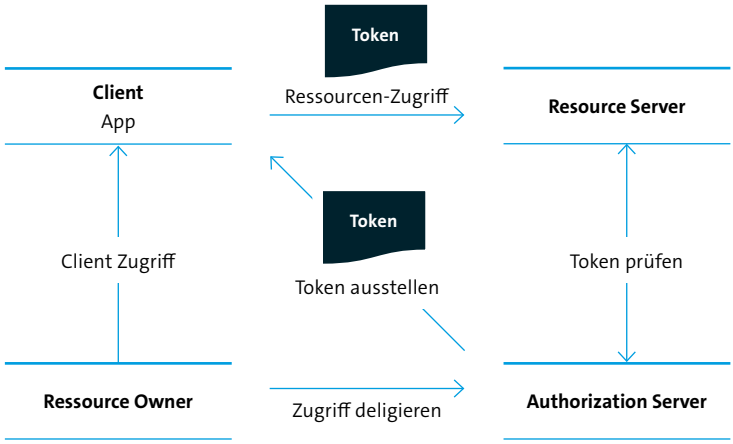


Abbildung 10: Beispielarchitektur einer OAuth-Kooperation

Um dem Client des Print-Services Zugriff auf die Fotos seines Cloud-Accounts (Ressource Server) zu gewähren, muss der Nutzer (RO) den Client dafür berechtigen. Zu diesem Zweck wird der Nutzer vom Client des Print-Service zu einem Authorization Server umgeleitet. Nachdem der Nutzer sich beim Authorization Server authentifiziert hat, wird ihm nun ein sogenannter Consent-Dialog angezeigt. Mittels des Consent-Dialogs kann der Nutzer den Client für eine beschränkte Zeit Zugriff auf eine definierte Anzahl von seinen Fotos gewähren. Dafür wird dem Client ein Security Token ausgestellt, das genau die vom Nutzer gewünschten Zugriffsberechtigungen enthält. Der Client kann das Security Token dazu verwenden, auf die Fotos des Nutzers beim Cloud-Anbieter zuzugreifen und diese auszudrucken.

Der Ablauf ist stark vereinfacht dargestellt und soll vor allem das OAuth 2.0 zu Grunde liegende Prinzip verdeutlichen. Als Vorbedingung des beschriebenen Ablaufs muss der Client im Rahmen einer Registrierung dem Authorization Server bekannt gemacht werden. Darüber hinaus muss eine organisatorische und technische Vertrauensbeziehung zwischen Ressource Server und dem Authorization Server etabliert sein.

Der beschriebene Vorgang wird im Allgemeinen als Access Delegation bezeichnet und ermöglicht es einem Nutzer, einer Applikation zeitlich begrenzt nur genau die Berechtigungen zu übertragen, die notwendig sind, um eine bestimmte Aufgabe auszuführen. Klassische IAM Ansätze bieten für dieses Szenario keine geeigneten Lösungen. Da im Kontext einer WOA die Wertschöpfung zunehmend verteilt und unternehmensübergreifend erfolgt, sind Lösungen für das Delegieren von Zugriffsberechtigungen, wie OAuth es ermöglicht, unverzichtbar, um Endkunden, Lieferanten und Partner kontrollierten Zugriff auf Web-Schnittstellen und Ressourcen zu ermöglichen.

6 Lessons Learned aus verwandten Disziplinen

6.1 Transition vom Status-Quo zum Quo vadis

Auch wenn schon entlang der Beschreibung der obigen Nomenklatur und der Forderung nach gänzlich neuen Identitätsverwaltungslösungen deutlich geworden ist, wie »anders« und »neu« die Sicht auf API-basierte Software-Ecosysteme ist, so garantiert diese Neuartigkeit noch keineswegs den Erfolg einer Einführung. Die Gründe hierfür sind meist vielfältig und können auf oberster Ebene den allgemeinen Risiken jeder Änderung zugesprochen werden (Change Resistence). Diese sind allerdings ebenso generisch bei jeder Innovation immer zu berücksichtigen, so dass Fragen nach der Einbindung aller Mitarbeiter, nach einer möglichst hohen Management-Attention und auch nach intensiven Schulungen grundsätzlich relevant sind.

Es gibt allerdings noch einen weiteren wichtigen Pool an Risiken bei der Einführung API-basierter Software-Ecosysteme: Dieser entstammt dem inhaltlich nageliegenden Bereich der Einführung Service-orientierter Architekturen (vgl. Kapitel 1): Das Konzept selbst ist nun über 20 Jahre alt, hat auch eine Schnittstellensicht und hat mittlerweile eine ausgeprägte Community, die sich der Fragen rund um Gründe für den häufig fehlenden Erfolg des Konzeptes annimmt.

Im Folgenden wird die Vielzahl existierender Analysen der Einführung von SOAs auf drei Kernrisiken konzentriert, deren Berücksichtigung ebenfalls existentiell für den Erfolg einer Einführung API-basierter Software-Ecosysteme ist.

6.2 Technik-Falle

Der Status-Quo Service-orientierter Architekturen ist nicht konsensfähig festzustellen. Alleine auf der jährlich stattfindenden SOA-Konferenz BSOA (Schmietendorf und Kunisch 2015) wird regelmäßig vom faktischen Tot der grundsätzlich guten Idee berichtet. Eine SOA als »Kommunismus der IT« wird dort als grundlegend positiv dargestellt, aber aufgrund unterschiedlicher menschlicher und eben nicht-technischer Facetten als sehr schwierig. Das passt zur bereits 2009 von der Analystin Anne Thomas Manes von der Burtongroup gemachten Aussage »SOA is dead; long live Services« (Manes 2009). Auch hier geht es um das Bekenntnis zu Services und nicht zu konkreten Techniken, die ebenfalls die Einführung einer SOA ausmachen.

Die Einführung Service-orientierter Architekturen ist aus vielerlei Gründen häufig in der Praxis gescheitert. Die Nähe zu einer WOA lässt vermuten, dass ein Lessons Learned der SOA-Fehlschläge sich konstruktiv auf die Einführung einer WOA übertragen lässt.

Das in diesem Abschnitt beschriebene Risiko beschreibt den Umstand, dass die Einführung einer WOA wie die Einführung einer SOA KEINE technische Herausforderung ist. Am deutlichsten wird dieses Risiko durch das sogenannte »SOA-Manifest« (Josuttis 2012) deutlich, das 2009 von einigen hochrangigen Experten in diesem Bereich erstellt wurde und unter anderem folgende Werte und Prinzipien vorschlägt:

- **Wert:** Geschäftswert über technische Strategie.
- **Prinzip:** Respektiere die Sozial- und Machtstruktur der Organisation.
- **Prinzip:** Produkte und Standards alleine werden weder SOA liefern noch das Service-orientierte Paradigma umsetzen.
- **Prinzip:** Services und deren Ausgestaltung sollten sich anhand der Art und Weise, wie sie wirklich genutzt werden, entwickeln.

Allen diesen Lessons Learned wohnt die Erkenntnis inne, dass derartige Innovationen zum Scheitern verurteilt sind, sobald sie lediglich auf die Technik ausgerichtet sind. Vielmehr müssen sich WOAs auf den Geschäftswert fokussieren, der sich allerdings durch die Einführung von WOAs ebenfalls verändert (vgl. Kapitel 2). Auch die Organisation selbst wird zweiseitig durch die Einführung einer WOA beeinflusst: Auf der einen Seite soll sie bei der Einführung von schnittstellenbasierten Software-Ecosystemen berücksichtigt werden, auf der andere Seite wird sie selbst massiv durch diese Innovation selbst verändert.

Dieses Risiko gilt ebenso für die Einführung einer WOA: Als rein technische Herausforderung verstanden, ist die Idee höchstwahrscheinlich sehr zügig gescheitert. Eine WOA kann erst dann ihre volle Kraft ausspielen, wenn mit geänderten Prozessen für die Wertschöpfung, geänderten Organisationsstrukturen und auch Arbeitsergebnissen Geld verdient werden kann.

Ein auch aus dem SOA-Kontext bereits erfolgreich eingesetztes Modell, das diese Ganzheitlichkeit der von einer SOA bzw. WOA berührten Aspekte modelliert, ist das EFQM-Modell, das auf der obersten Ebene die sich ändernde Organisation, die sich ändernden Prozesse sowie die sich ändernden Ergebnisse in Form von Arbeitsergebnissen darstellt. Auch alle Zwischen-Aspekte sind für die Einführung von WOA relevant.

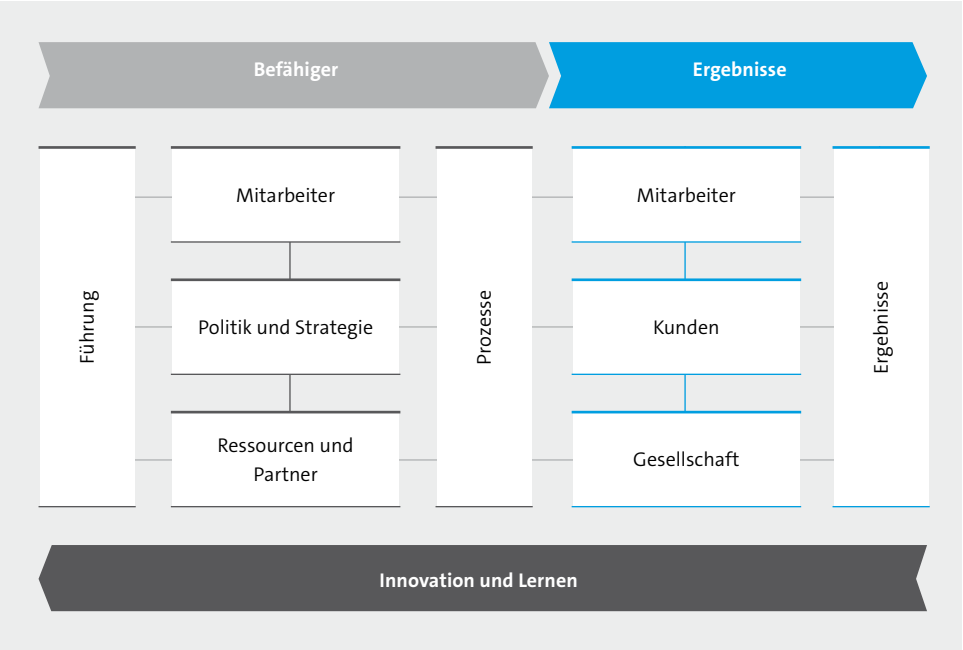


Abbildung 11: EFQM-Modell zur Abhängigkeitsanalyse der WOA-Einführung

6.3 Security-Falle

Ein weiteres Risiko bei der Einführung eines schnittstellenbasierten Software-Ecosystems ist die durch die Öffnung via APIs gegebene Erweiterung der Security-Angriffsfläche: Viele Unternehmen haben aufgrund der Vielzahl von Security-Vorfällen (Cyber-Kriminalität, Malware, Bot-Netze u. A.) sich maximal von der umgebenden Infrastruktur abgekapselt: Meist funktioniert lediglich das Browsen via http und das Versenden via Exchange. Sämtlicher weiterer Traffic wird durch die Firewall restriktiv verboten. Falls sichere Daten übertragen werden ist als maximaler Sicherheitslevel noch das https-Protokoll etabliert.

Die APIs für das schnittstellenbasierte Software-Ecosystem bedürfen nicht nur der Öffnung der APIs über die Firewall, denn dieses Vorgehen genügt nicht: Hat ein Angreifer die API im Zugriff, so hat er die in den Unternehmen wesentliche Sicherheitsschranke, die Firewall, bereits hinter sich gelassen. Es ist extrem wichtig für einen Provider, dass er daher zusätzlich zur Firewall-öffnung alle notwendigen Security-Maßnahmen einleitet, die aufgrund der Öffnung über APIs notwendig sind. In Kapitel 4 sind bereits einige repräsentative Angriffsvektoren dargestellt und es wird erläutert, wie diese bekämpft werden können.

Die APIs als eigenständiges Opfer von Security-Angriffen sind seit einiger Zeit auch im Internet entsprechend präsent. So gibt es mittlerweile einige Internet-Seiten, so z. B. das umfangreiche Werk (Lensmar 2014) »API Security Testing – How to Hack an API and Get away with it« – in denen das API-Security-Testing intensiv beleuchtet wird und typische Schwachstellen identifiziert werden.

Wichtige Punkte, die zum Bekämpfen der Security-Falle bearbeitet werden müssen, sind:

- Zusätzliche Security-Infrastruktur zu der klassischen Security-Firewall. Die wesentlichen notwendigen Features sind oben unter dem Punkt API-Gateway beschrieben.

- Message-Formate, die jeweils eigene Security-Probleme bergen. Sowohl JSON als auch XML, als die prominentesten Message-Formate, bergen einzelne Security-Risiken, die mittels geeigneter Infrastruktur bekämpft werden können.

- Verschlüsselung und Signaturen als Mittel der Wahl für mehr Vertraulichkeit und Integrität: Wurden ggfs. bisher schon in einer SOA verwendete Services nur intern verwendet und bedurften daher keiner besonderen Verschlüsselung, so kann diese Sicht in der externen Nutzung grundlegend falsch sein. Viele relevante Daten wie z. B. Kreditkartendaten oder personenspezifische Daten sollten in keinem Fall so übertragen werden, dass sie von Außenstehenden ohne Legitimation eingesehen werden können.

- Robustheit: Aus einer Robustheitsbetrachtung ist es für ein API anbietendes System etwas anderes, ob bekannte Nutzer innerhalb einer Organisation eine Schnittstelle verwenden oder beliebige, evtl. gar nicht konkret bekannte Personen. Ungültige, extreme oder fehlende Parameter in beliebiger Intensität gehören daher genauso zum Angriffsszenario wie professionelle Werkzeuge wie Fuzzing oder Brute-Force-Attacken.

- Injection und Cross-Site-Attacken: Eng verbunden mit dem Punkt der Robustheit ist das sehr sorgfältige Analysieren von Eingabeparametern. Injection-Techniken wie SQL-Injection oder Cross-Site-Attacken resultieren zumeist aus der unreflektierten Weitergabe von Eingabedaten an das Back-End. Dieses Risiko muss für öffentliche APIs in jedem Fall vor einer Veröffentlichung beseitigt werden.

- Verfügbarkeit: Eine API kann nur dann effektiv in einem Software-Ecosystem verwendet werden, wenn sie eine gewisse Zuverlässigkeit erreicht. Hierzu gehört insbesondere eine aktive Absicherung nach Überlast: Diese kann »böse« durch Denial-of-Service-(DoS)-Attacken generiert oder schlichtweg durch zu viele echte Nutzer erzeugt werden. In beiden Fällen ist der Consumer nicht zufrieden, was sich mittelfristig als Risiko für den Provider darstellen lässt.

6.4 Identity-Falle

Die Notwendigkeit neuer Konzepte für das Identity-Management ist bereits in Kapitel 5 intensiv besprochen worden. Als Risiko-Falle zeigt es, dass das bisherige, auf ein zentrales IAM-System ausgerichtete Identity und Access Management, mit den neuen Ansätzen nicht mehr funktioniert. Sämtliche Versuche, alle externen Nutzer einer API ins eigene IAM zu überführen, sind mittelfristig zum Scheitern verurteilt und daher mit einer Absenkung des Security-Levels verbunden.

Wie oben beschrieben, ist das Trusting zwischen verschiedenen Identity-Providern ein vielversprechender Lösungsansatz, der allerdings nicht nur technische Vorbedingungen, sondern insbesondere organisatorische Rahmenparameter benötigt, um zu funktionieren.

6.5 Big-Bang-Falle

Einer der wesentlichen Vorteile der schnittstellenbasierten Software-Ecosysteme ist dessen Leichtgewichtigkeit: Im Gegensatz zu einer SOA werden leichtgewichtige Web-Technologien verwendet, die zudem keinen Enterprise-Service-Bus oder zu administrierende Service-Kataloge benötigen. Die Schwergewichtigkeit einer SOA mit ihrer Vielzahl technisch notwendiger Einzelbestandteile hat häufig zum DOA-Konzept: Dead on Arrival geführt.

Für die Einführung schnittstellenbasierter Software-Ecosysteme bedeutet diese den großen Vorteil, mit vielen kleinen Iterationen der Endausbaustufe näherzukommen: Anstatt monatelang in die Infrastruktur zu investieren, können APIs leichtgewichtig nach Installation eines API-Gateways verprobt werden. Ein einzelnes API genügt hier häufig schon, um damit den Impact auf Organisation und die eigenen Prozesse zu erfahren.

Anstelle eines Big-Bangs eignen sich also mehrere Iterationen, über die das System nach und nach reifer wird. Da gerade APIs eine hohe Volatilität erfahren, wird hier in der Regel nie von einer Endausbaustufe die Rede sein. Vielmehr befindet sich das API-Management fortwährend in einem Zyklus kontinuierlicher Verbesserung.

Der Vorteil der iterativen Vorgehensweise, der mit schnittstellenbasierten Software-Ecosystemen im Gegensatz zu SOA-Techniken möglich ist, ist in der folgenden Abbildung noch einmal dargestellt:

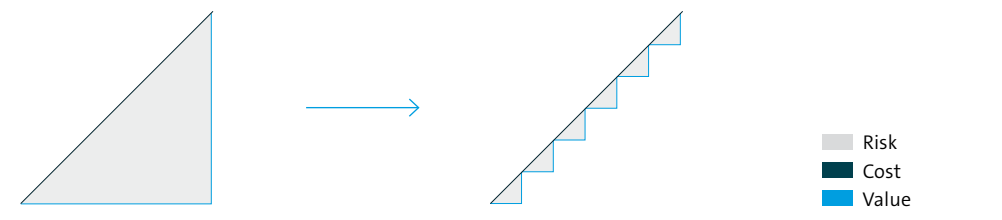


Abbildung 12: Iteratives Vorgehen anstelle von Big-Bang

Es hat sich in vielen Projekten zur Einführung extern sichtbarer APIs gezeigt, dass gerade für die Berücksichtigung der Ganzheitlichkeit des Konzeptes schnittstellenbasierter Software-Ecosysteme grundlegende Prozesse zur schrittweisen Verbesserung der eingesetzten Technik sinnvoll sind: Hier ist als de-facto-Standard auf das seit 1984 etablierte Quality Improvement Paradigm von Basisli & Weiss verwiesen, das als Prozess in der folgenden Abbildung noch einmal dargestellt ist:

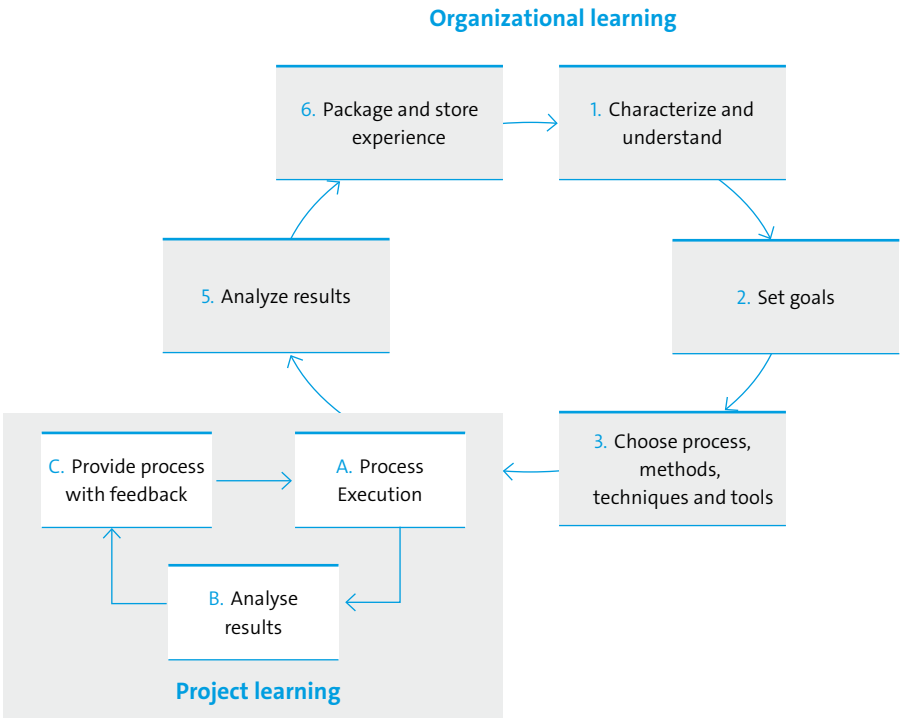


Abbildung 13: Organisational Learning mit APIs

Dieser Prozess stellt sicher, dass die Offenlegung von APIs nicht ausschließlich durch einzelne Projekte erfolgt, sondern in jedem Fall die Ergebnisse in die Organisation insgesamt zurückgetragen werden muss. Dieses Vorgehen erfüllt zweierlei Anforderungen:

- **Es ist schnell:** Über ein individuelles Projekt können bestimmte Schnittstellen veröffentlicht werden und so eine aktivere Teilnahme am Software-Ecosystem angestrebt werden.
- **Es ist organisationsweit:** Aufgrund der übergeordneten Organisational-Learning-Ebene erfährt jedes Einzelprojekt die notwendige Governance: Hierzu gehören bestimmte Prozesse, Methoden, Techniken und Tools ebenso wie das abgestimmte Setzen von anzustrebenden Zielen.

Dieses Wechselspiel aus kleinem Projekt-Learning-Kreis und großen Organisational-Learning-Kreis ist gerade für die Einführung schnittstellenbasierter Software-Ecosystem von großen Vorteilen, da er hochiterativ und leichtgewichtig arbeitet, ohne wesentliche Aspekte der Governance – insbesondere aus Security-Gesichtspunkten – aus dem Blick zu verlieren.

7 Businessmodelle und Hürden für APIs

Das Thema Businessmodelle von schnittstellenbasierten Software-Ecosystemen ist sicher eines der schwierigsten: Es fehlen aktuell noch referenzierbare und glaubwürdige Daten von großen etablierten Unternehmen, die die Transition gegangen sind. Es gibt genügend Businessmodelle von jungen Start-ups, die sofort Teil der Software-Ecosysteme sind; allerdings müssen diese Firmen bei den Businessmodellen nicht die Änderung von bereits existierenden Wertschöpfungsketten (vgl. Kapitel 2) berücksichtigen, sondern die Etablierung von solchen Wertschöpfungsketten, die ex ante bereits auf schnittstellenbasierte Software-Ecosysteme ausgerichtet sind.

Eine aus heutiger Sicht brauchbare Alternative ist die des Open Data für insbesondere öffentliche Institutionen: Hier hat der Gesetzgeber sehr früh mit seinem E-Government-Gesetz (BMI 2013) und dem konkreten Programm Digitale Verwaltung 2020 (BMI 2014) dazu beigetragen, dass die öffentlichen Institutionen ihre Daten über APIs anbieten müssen und damit das für bereits etablierte Unternehmen wichtige Umdenken in Wertschöpfungsketten zwecks klarer Businessmodelle bereits stattgefunden hat. Daher wird im Folgenden dieses eine typische Businessmodell herangezogen und erläutert. Es ist an dieser Stelle aber klarzumachen, dass viele weitere Businessmodelle möglich sind; hier besteht also keinerlei Anspruch auf Vollständigkeit oder Repräsentativität.

7.1 Open Data als Beispieldisziplin

Unter Open Data werden i. d. R. die folgenden Informationen verstanden:»Open Data refers to the information that can be freely used, modified and shared by anyone for any purpose. It must be available under an open license and provided in a convenient and modifiable form that is machine readable.« (E-Government-Gesetz des Bundes)

Das Thema Open Data ist für den Bereich WOA sehr relevant, da es entsprechende Gesetzesvorgaben gibt, wie bestimmte Institutionen mit Open Data umzugehen haben. Das sogenannte E-Government-Gesetz (E-Government-Gesetz des Bundes), das seit dem 1. August 2013 in Kraft ist, fordert z. B. in § 2: »Jede Behörde ist verpflichtet, auch einen Zugang für die Übermittlung elektronischer Dokumente [...] zu eröffnen.« Weiter heißt es »Stellen Behörden über öffentlich zugängliche Netze Daten zur Verfügung, an denen ein Nutzungsinteresse, insbesondere ein Weiterverwendungsinteresse im Sinne des Informationsweiterverwendungsgesetztes, zu erwarten ist, so sind grundsätzlich maschinenlesbare Format zu verwenden.« Bereits hier ist der Grundgedanke von Software-Ecosystemen, in denen auch Zwischenprodukte in diesem Fall von Behörden nutzungsstiftend veröffentlicht werden sollen, erkennbar.

Noch konkreter wird es dann im Projekt »Digitale Verwaltung 2020«, das am 8. April 2014 beschlossen wurde. Dort heißt es u. a. »Die Digitalisierung innovativer öffentlicher Dienstleistungen und Prozesse erleichtert und erfordert die weitere Öffnung staatlicher Geo-, Statistik- und anderer Datenbestände (open Data).« Dass diese Öffnung nicht über reine Datensenken, in denen die Daten direkt zum Download bereitstehen, sondern über Schnittstellen zu erfolgen hat, die kontrolliert und ggfs. auch mit Mehrwerten angereichert werden können, ist obligatorisch.

Behörden sind folglich per Gesetz zur Teilnahme an schnittstellenbasierten Software-Ecosystemen angehalten, wenn auch größtenteils in der Rolle des Providers.

Bzgl. derBusinessmodelle auf Basis solcher schnittstellenbasierter Software-Ecosysteme kann zwischen zwei Marktgrößen unterschieden werden:

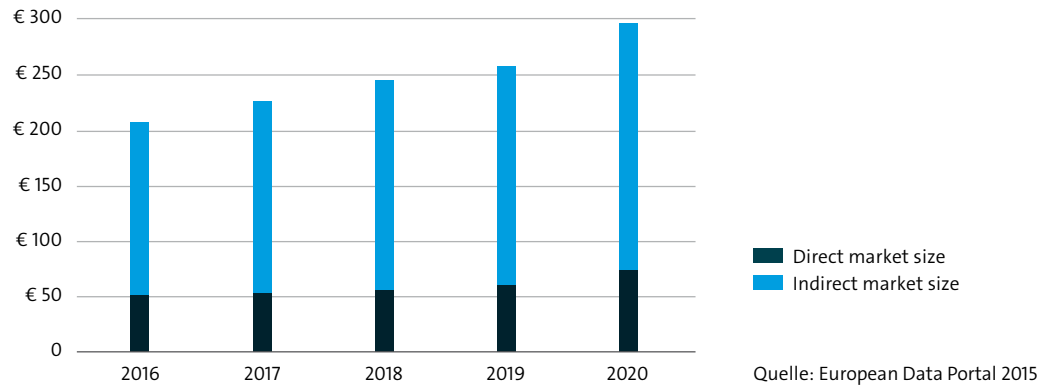
- **Direkter Markt**, der durch die Verwendung der über Schnittstellen verfügbaren Informationen entsteht. Ein Beispiel hierfür ist eine etwaige Lizenzgebühr für eine Applikation, die öffentlich verfügbare Hochwasserstandsdaten aufbereitet und lokalisiert den Nutzern anbietet.

- **Indirekter Markt,** der durch den mittels des direkten Marktes effektiv möglichen Mehrwertes geprägt ist. Ein Beispiel hierfür ist die Einsparung von Hochwasserschäden, die durch eine Anwendung der entsprechenden App des direkten Marktes erreicht werden kann.

Die aktuelle Situation plus die Prognose für die Zukunft dieser beiden Märkte ist in der folgenden Abbildung dargestellt:

Umfang des direkten und indirekten Marktes von Open Data als Businessmodellgrundlage

Total Market size Open Data EU28+, in Milliarden Euro



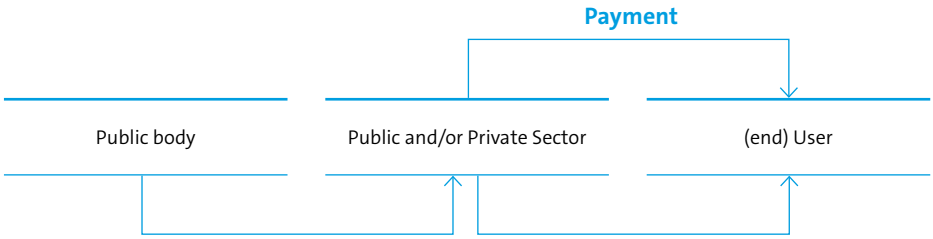
Quelle: European Data Portal 2015

Abbildung 14: Umfang des direkten und indirekten Marktes von Open Data als Businessmodellgrundlage (European Data Portal 2015)

Der Marktzuwachs zwischen 2016 und 2020 beläuft sich für beide Märkte auf fast 40 Prozent. Der direkte Markt ist dabei keineswegs auf den eigentlichen Provider der Schnittstellen beschränkt, sondern umfasst Finance, ICT, Transport und Landwirtschaft ebenso (European Data Portal 2015). Software-Ecosysteme, hier nun auf Basis von über Schnittstellen verfügbar gemachten Open Data, funktionieren hier domainunabhängig.

Die heute führende Rolle der öffentlichen Behörden in diesem Bereich lässt weitere Analysen zu: So arbeiten bereits heute 75.000 Mitarbeiter in sogenannten Open Data Jobs (European Data Portal 2015), d. h. ihr Job hängt ganz wesentlich von der Existenz von Software-Ecosystemen ab. Diese Zahl wird entlang dieser Studie in 2020 auf 100.000 angestiegen sein.

Welche unterschiedlichen Rollen an der gesamten Wertschöpfungskette der Monetarisierung öffentlicher Schnittstellen als Medium von Open Data wie beteiligt sind, wird in der folgenden Abbildung modelliert:



Quelle: European Data Portal 2015

Abbildung 15: Wertschöpfungskette von Open Data und darauf basierenden Services (European Data Portal 2015).

Für den Kontext dieses Leitfadens ist wichtig darauf hinzuweisen, dass bereits der Datenfluss des »public body« zum »Public and / or private Sector« über öffentliche (open) Schnittstellen (= APIs) erfolgt. Das Businessmodell einer WOA ist für den öffentlichen Bereich, der selbst Provider ist, also zweigeteilt: Im ersten Bereich der Verfügbarmachung der Daten kann kein direktes Geld verdient werden. Im nachgelagerten Schritt, diese Basis-Schnittstellen zu höherwertigen Services und Produkten zusammen zu orchestrieren und mit weiterer Business Logik anzureichern, kann dann die Bezahlung mit den oben genannten Marktgrößen erzielt werden.

In der Studie selbst werden viele konkrete Beispiele für positiv evaluierte Businessmodelle angegeben. Diese stammen aufgrund der meist strengen Verschwiegenheit echter Zahlen im privatwirtschaftlichen Sektor zwar größtenteils aus dem Public Sector, lassen aber den erreichbaren Mehrwert des neuen Denkens als Teil von schnittstellenbasierten Ecosystemen erkennen.

7.2 Aktuelle Risiken

Zu einer Analyse von Businessmodellen gehört selbstverständlich auch die Berücksichtigung eines entsprechenden Risikos. Die Gründe für das Eintreten dieser Risiken sind ähnlich wie die Risiken selbst vielschichtig. Eine vollständige Analyse existiert bisher für den Bereich der schnittstellenbasierten Software-Ecosysteme nicht.

Prof. Dr. Olaf Resch von der Hochschule für Wirtschaft und Recht Berlin hat allerdings eine punktuelle Analyse anhand eines konkreten Praxisbeispiels für solche Risiken auslösenden Barrieren durchgeführt. Diese ist vollständig in den Proceedings der Bewertung Service-orientierter- und Cloudbasierter Architekturen (BSOA)-Konferenz des Jahres 2015 erschienen (Schmietendorf und Kunisch 2015). Dieses Kapitel ist im Folgenden dem Artikel von Resch (2015) entnommen.

Die API-Economy ist keineswegs unkritisch zu sehen und erfordert insbesondere von etablierten Unternehmen ein Umdenken. Beispielsweise ist Google sehr erfolgreich im Aufmerksamkeitsgeschäft, konkret dem Vermitteln von Werbung. Dazu benötigt Google Top-Index-Daten als Rohstoff. Diesen Rohstoff stellt Google sehr erfolgreich selber her und verwendet ihn quasi ausschließlich und wiederum sehr erfolgreich für das eigene Aufmerksamkeitsgeschäft. Sollte Google den wertvollen Rohstoff via einer API auch (ernsthaft) Dritten anbieten, würde dies ein neues Geschäftsfeld mit erheblichen Gewinnchancen eröffnen, gleichzeitig aber auch die Konkurrenten im etablierten Aufmerksamkeitsgeschäft stärken. Diesen Schritt geht Google zumindest aktuell noch nicht. Etwas anders sieht die Situation bei Yahoo aus, das über Yahoo BOSS durchaus auch den wertvollen Rohstoff anbietet. Bei Yahoo geschieht das jedoch ebenfalls recht halbherzig, und es wird deutlich, dass auch Yahoo sich eher im Aufmerksamkeitsgeschäft sieht.[1]

Eine erste Barriere für die API-Economy sind daher Geschäftsmodelle, die dem Anbieten wertvoller Kompetenzen via APIs entgegenstehen.

Auch auf der anderen Seite der sehr kleinen Anbieter finden sich durchaus wertvolle Kompetenzen, die auch gerne über APIs zur Verfügung gestellt werden. Hier wirken die Geschäftsmodelle aber häufig wenig durchdacht. Eine kostenlose Nutzung ist auf den ersten Blick positiv, allerdings nicht geeignet, die gerade für die API-Economy kritische Vertrauensbasis zu etablieren. Zumindest im B2B-Bereich ist »kostenlos« zudem eine Illusion, da auf Seiten der API-Konsumenten immer Aufwand für die Einbindung und Evaluation entsteht.

Eine zweite Barriere für die API-Economy sind unklare und wenig vertrauenserweckende Geschäftsmodelle.

1 Mittlerweile hat Yahoo angekündigt, den Service zum 31.3.2016 ganz abzuschalten.

Es gibt durchaus ernsthafte API-basierte Geschäftsmodelle, die wertvolle Kompetenzen anbieten, aber dennoch alten Denkweisen verhaftet sind. Das manifestiert sich insbesondere in starren Preismodellen. Ein wesentlicher Aspekt der Dynamik in der API-Economy entsteht durch die flexible Nutzung der jeweils benötigten Kompetenzen von unterschiedlichen Anbietern. Dies bedingt jedoch auch eine flexible Abrechnungsmöglichkeit und keine durch grobgranulare Preismodelle erzwungene Bindung an einzelne Anbieter.

Eine dritte Barriere für die API-Economy sind grobgranulare Preismodelle, die einer dynamischen Integration von Kompetenzen entgegenstehen.

Vertrauen ist ein Grundpfeiler jeglichen Wirtschaftens und senkt Transaktionskosten. Ein bewährtes Instrument, um Vertrauen zu etablieren, sind persönliche Beziehungen, die meist über den Vertrieb und den Kundendienst aufrechterhalten werden. In der API-Economy ist dieses Instrument allerdings nicht adäquat und würde zu unverhältnismäßig hohen Transaktionskosten führen. Somit sind neue Instrumente der Vertrauensbildung erforderlich. Diese müssen entwickelt und akzeptiert werden, was eine Veränderung sehr grundsätzlicher menschlicher Denkstrukturen bedingt.

Eine vierte Barriere für die API-Economy sind fehlende Instrumente der Vertrauensbildung in vergleichsweise anonymen Kompetenznetzwerken.

Ein dynamisches Kompetenznetzwerk mit vielen hochspezialisierten Partnern wird schnell so komplex, dass herkömmliche Steuerungsinstrumente an ihre Grenzen stoßen. Diese Komplexität kann auch nicht verringert werden, ohne dass die Vorteile der dynamischen Kompetenzintegration dadurch beeinträchtigt werden. Deshalb werden für die API-Economy neue Steuerungsinstrumente benötigt, welche die Komplexität beherrschbar machen.

Eine fünfte Barriere für die API-Economy sind fehlende Instrumente zur Steuerung hochkomplexer Kompetenznetzwerke.

Die skizzierten Barrieren entstammen einer ökonomischen Sichtweise auf die API-Economy, was allerdings keineswegs andeuten soll, dass bereits alle technischen und architektonischen Probleme der Systemintegration mithilfe von APIs gelöst sind.

8 Zusammenfassung

Eine Web-orientierte Architektur ist die technische Umsetzung eines API-basierten Software-Ecosystems unter Zuhilfenahme von Web-Technologien. Sie stellt die gemeinsame Plattform von Consumern, also Nutzern von Services über Schnittstellen, und Providern, also Anbietern von Services über Schnittstellen, dar. Der disruptive Charakter einer WOA wird besonders an einer Analyse im Kontext der üblichen Wertschöpfungsketten deutlich: Die WOA ist die bewusste Fokussierung auf die eigene Kernkompetenz, das ebenso bewusste Nutzen von Kernfunktionalitäten anderer Anbieter und die Nutzung von APIs als weiterem Absatzkanal von (Zwischen-)Produkten. Die Mitspieler an einem solchen schnittstellenbasierten Ecosystem werden daher üblicherweise gleichzeitig Services nutzen und anbieten und bezeichnen sich als Prosumer. Dieser durch eine WOA mögliche grundlegende Wandel trifft allerdings gerade in Deutschland auf viel Vorbehalt: Zwar sind die Vorteile im Gros bekannt, wie z. B. die deutlich verbreiterten Absatzmärkte sowie die Möglichkeit, auch eigene Zwischenprodukte direkt zu vermarkten, allerdings dominieren »German Angst« und eine grundsätzliche Skepsis bzgl. solcher Änderungen, die unternehmensweit, also nicht innerhalb eines organisatorisch abgeschirmten Teil-Bereiches, relevant sind. Hier sind die Haupthürden keineswegs technisch, sondern liegen im organisatorischen, prozessualen und menschlichen Bereich.

Von der technischen Seite eine WOA zu etablieren, kann als allgemeiner Stand der Technik aufgefasst werden. Die besonderen Aufgaben, die durch die Öffnung der Unternehmen für die Teilhabe an schnittstellenbasierten Software-Ecosystemen anfallen, können heute von modernen API-Gateways sicher abgearbeitet werden. Das Ziel ist immer eine maximale hohe Verfügbarkeit der APIs selbst, bei einer maximalen Absicherung der dahinter liegenden Backend-Systeme sowie der Möglichkeit, sich über das Gesamtzusammenspiel jederzeit einen Überblick zu verschaffen.

Ein technisch besonders relevanter Punkt ist ein idealerweise fundamental neuer Umgang mit Identitäten: Statt alle Nutzer von Systemen in einem eigenen IAM-System zu pflegen, gehört die Zukunft eher solchen Architekturen, in denen die Nutzer ihre eigene Identität mitbringen und der API-Provider diesem Identitätsprovider vertraut. Eine WOA funktioniert initial zwar auch ohne solche Konzepte, jedoch sind sie keineswegs zukunftsfähig, sobald die APIs tatsächlich in größerem Maße genutzt werden.

Die Einführung einer WOA weist Ähnlichkeiten zu der Einführung anderer Innovationen aus der Vergangenheit auf. So haben die vielen Jahre der Einführung von SOAs viele Indizien geliefert, die heute idealerweise bei der Einführung einer WOA proaktiv berücksichtigt werden. Hierzu gehört insbesondere eine übertriebene Techniksicht bei einer WOA-Einführung, die fehlende Sensibilität für Security, das Ignorieren der Notwendigkeit neuer Identitätskonzepte sowie das prozessorale Big-Bang-Risiko.

Dass das Konzept der WOAs erfolgversprechend ist und in Zukunft weiter sein wird, belegen unterschiedliche Beispiele. Besonders deutlich und heute schon umgesetzt ist dies für den Bereich von Open Data im öffentlichen Bereich: Hier hat der Gesetzgeber bereits früh die Weichen gestellt, sodass gerade öffentliche Institutionen heute bereits starke API-Provider sind.

Quellen

BMI (2013) E-Government-Gesetz des Bundes.
http://www.bmi.bund.de/DE/Themen/IT-Netzpolitik/E-Government/E-Government-Gesetz/e-government-gesetz_node.html
(Abgerufen am 27.01.2016)

BMI (2014) Digitale Verwaltung 2020 – Regierungsprogramm 18. Legislaturperiode.
http://www.bmi.bund.de/SharedDocs/Downloads/DE/Broschueren/2014/regierungsprogramm-digitale-verwaltung-2020.pdf
(Abgerufen am 27.01.2016)

Bosch, J (2009) From Software Product Lines to Software Ecosystems. 13th International Software Product Line Conference.
http://www.janbosch.com/jan_bosch/Composition_files/SPLC09-SoftwareEcosystems-Accepted.pdf
(Abgerufen am 27.01.2016)

Brennan, M W (2015) API Management Solutions Market Predicted to Quadruple by 2020.
http://www.programmableweb.com/news/api-management-solutions-market-predicted-to-quadruple-2020/elsewhere-web/2015/06/17
(Abgerufen am 27.01.2016)

Czycholl, H (2014) Die »German Angst« steckt tief in unseren Genen.
http://www.welt.de/wissenschaft/article132728527/Die-German-Angst-steckt-tief-in-unseren-Genen.html
(Abgerufen am 27.01.2016)

Dern, G (2009) Management von IT-Architekturen – Leitlinien für die Ausrichtung, Planung und Gestaltung von Informationssystemen. 3. Aufl., Vieweg+Teubner Verlag, Wiesbaden.

Die Ideeologen (2015) Mehrheit deutscher Unternehmen ist innovationsfeindlich.
http://www.ideeologen.de/fileadmin/ideeologen/Medienordner/Downloads/Wirtschaft+weiterbildung_181011.pdf
(Abgerufen am 27.01.2016)

Erl, T (2005) Service-oriented architecture. concepts, technology, and design. Prentice-Hall, Upper Saddle River, New Jersey [u. a.].

European Data Portal (2015) Creating Value through Open Data.
http://www.europeandataportal.eu/sites/default/files/edp_creating_value_through_open_data_0.pdf
(Abgerufen am 27.01.2016)

Ferrari electronic AG (2013) 1 Millionen Faxe im Jahr – Tendenz steigend: Techniker Krankenkasse (TK) optimierte mittels Unified Messaging Lösung ihre Kommunikationsprozesse. https://www.ferrari-electronic.de/downloads/files/801012/2015/DE_CS_TK_w.pdf (Abgerufen am 27.01.2016)

Gabriel, R (2013) Informationssystem. http://www.enzyklopaedie-der-wirtschaftsinformatik.de/lexikon/uebergreifendes/Kontext-und-Grundlagen/Informationssystem/ (Abgerufen am 27.01.2016)

Gall, N; Sholler, D; Bradley, A J (2008) Tutorial: Web-Oriented Architecture: Putting the Web Back in Web Services. https://www.gartner.com/doc/797713 (Abgerufen am 27.01.2016)

Gartner (2015) Gartner IT-Glossary – Web-Oriented Architecture (WOA). http://www.gartner.com/it-glossary/web-oriented-architecture-woa (Abgerufen am 27.01.2016)

Hinchcliffe, D (2011) Web-Oriented Architecture (WOA). http://de.slideshare.net/thetechnicalweb/weboriented-architecture-woa (Abgerufen am 27.01.2016).

Janson, S (2009) Deutsche Unternehmen sind innovationsfeindlich! http://berufebilder.de/2009/studie-deutsche-unternehmen-innovationsfeindlich/ (Abgerufen am 27.01.2016)

Josuttis, N M (2012) SOA Manifest (SOA-Manifesto auf deutsch). http://soa-manifest.de/ (Abgerufen am 27.01.2016)

Koch, M (2012) Mashups. http://www.enzyklopaedie-der-wirtschaftsinformatik.de/lexikon/daten-wissen/Informationsmanagement/IT-Infrastruktur/Informations--und-Kommunikationstechnologien/computer-supported-cooperative-work-cscw/mashups (Abgerufen am 27.01.2016)

Krafzig, D; Banke, K; Slama, D (2005) Enterprise SOA: service-oriented architecture best practices. Prentice Hall, Upper Saddle River, New Jersey.

Lensmar, O (2014) API Security Testing – How to Hack an API and Get away with it. http://blog.smartbear.com/readyapi/api-security-testing-how-to-hack-an-api-and-get-away-with-it-part-1-of-3/ (Abgerufen am 27.01.2016)

Lück, F (2011) Fax für 82 Prozent der Unternehmen wichtig: Fax schlägt E-Mail in punkto Sicherheit.
http://www.crn.de/markt/artikel-92299.html
(Abgerufen am 27.01.2016)

Manes, A T (2009) SOA is dad; long live services.
http://apsblog.burtongroup.com/2009/01/soa-is-dead-long-live-services.html
(Abgerufen am 27.01.2016)

Marx Gomez, J C (2015) Software Orientierte Architektur.
http://www.enzyklopaedie-der-wirtschaftsinformatik.de/lexikon/is-management/Systementwicklung/Softwarearchitektur/Architekturparadigmen/Serviceorientierte-Architektur
(Abgerufen am 27.01.2016)

Resch, O (2015) API-Economy – eine Situationsbestimmung. In: Schmietendorf, A/Kunisch, M (Hrsg.) Proceedings BSOA/BCloud 2015, Shaker-Verlag, Aachen.

Reymer, D (2015) Gartner: Top 10 Technology Trends 2015.
http://de.slideshare.net/denisreimer/gartner-top-10-technologytrends2015
(Abgerufen am 27.01.2016)

Schmietendorf, A; Kunisch, M (2015) BSOA/BCloud 2015: Proceedings des 10. Workshops »Bewertungsaspekte service- und cloudbasierter Architekturen«, Shaker Verlag, Aachen.

Simon, F; Kraft, T (2015) Das Schnittstellen Paradoxon: Weniger Eigenentwicklung führt nicht zu weniger Testaufwand. In: Schmietendorf, A/Kunisch, M (Hrsg.) Proceedings BSOA/BCloud 2015, Shaker-Verlag, Aachen.

Winter, R; Aier, S (2015) Informationssystem-Architektur.
http://www.enzyklopaedie-der-wirtschaftsinformatik.de/lexikon/daten-wissen/Informationsmanagement/Information-/Informationssystem-Architektur/index.html
(Abgerufen am 27.01.2016)

Yamnitsky, M (2015) API management solutions market will quadruple by 2020.
http://www.zdnet.com/article/the-api-management-solutions-market-will-quadruple-by-2020/
(Abgerufen am 27.01.2016)

Bitkom vertritt mehr als 2.300 Unternehmen der digitalen Wirtschaft, davon gut 1.500 Direktmitglieder. Sie erzielen mit 700.000 Beschäftigten jährlich Inlandsumsätze von 140 Milliarden Euro und stehen für Exporte von weiteren 50 Milliarden Euro. Zu den Mitgliedern zählen 1.000 Mittelständler, 300 Start-ups und nahezu alle Global Player. Sie bieten Software, IT-Services, Telekommunikations- oder Internetdienste an, stellen Hardware oder Consumer Electronics her, sind im Bereich der digitalen Medien oder der Netzwirtschaft tätig oder in anderer Weise Teil der digitalen Wirtschaft. 78 Prozent der Unternehmen haben ihren Hauptsitz in Deutschland, 9 Prozent kommen aus Europa, 9 Prozent aus den USA und 4 Prozent aus anderen Regionen. Bitkom setzt sich insbesondere für eine innovative Wirtschaftspolitik, eine Modernisierung des Bildungssystems und eine zukunftsorientierte Netzpolitik ein.

Bundesverband Informationswirtschaft,
Telekommunikation und neue Medien e.V.
Albrechtstraße 10
10117 Berlin
T 030 27576-0
F 030 27576-400
bitkom@bitkom.org
www.bitkom.org

bitkom