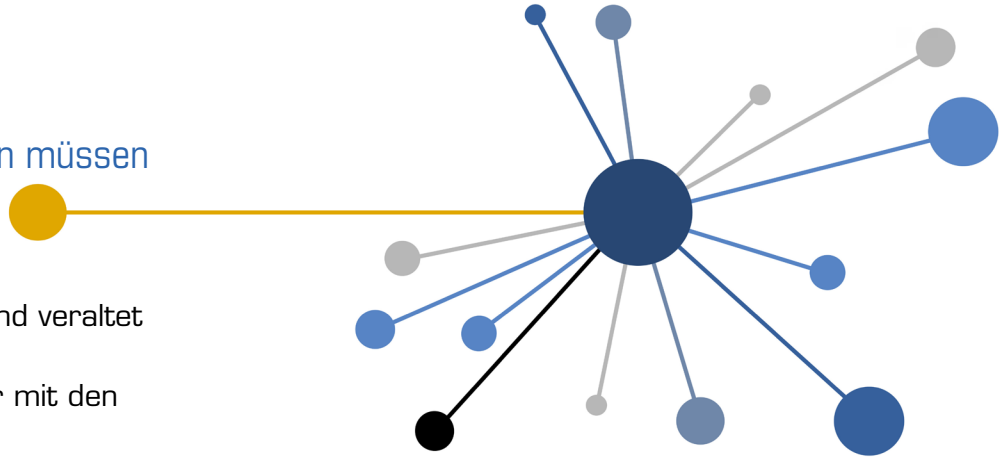
Dreh- und Angelpunkte für die IT-Modernisierung

Warum wir APIs als unternehmenskritischen Erfolgsfaktor betrachten müssen



Die rasante Entwicklung der IT hat uns in eine neue Ära katapultiert: Unternehmen und Organisationen müssen ihre bisherigen Konzepte für Modernisierung und Datenaustausch über Bord werfen und Raum schaffen für neue Ansätze. In diesem Kontext spielen APIs eine zentrale Rolle. Denn APIs fungieren als Brücke zwischen verschiedenen Anwendungen, Plattformen und Systemen. Sie ermöglichen einen reibungslosen und sicheren Austausch von Daten. In diesem Artikel sehen wir uns die Rolle und die Funktionsweise von APIs bei der IT-Modernisierung genauer an. Dafür werfen wir unter anderem einen Blick in ein aktuelles Modernisierungsprojekt.

Modernisierung ist alternativlos

Fakt ist: Unternehmen brauchen heute moderne IT-Systeme, um langfristig wettbewerbs- und innovationsfähig zu bleiben. Nur dann sind sie in der Lage, mit den rasanten Veränderungen am Markt Schritt zu halten. Aber warum ist das so?

Die Gründe, warum es für ein Unternehmen substanziell ist, sich mit dem Thema Modernisierung auseinanderzusetzen, sind breit gefächert. Sie hängen auch davon ab, wie lange ein Unternehmen bereits am Markt ist.

Häufige Gründe:

- Technologiegrundlagen sind veraltet
- Support läuft aus
- Niemand kennt sich mehr mit den Systemen aus
- Know-how-Träger haben das Unternehmen verlassen
- Es gibt ständig neue Geschäftsanforderungen, auf die agil reagiert werden muss

Wie aber geht es für Unternehmen oder Organisationen weiter? Wie kann das Geschäft weiterlaufen? Wie können Risiken und Aufwand minimiert werden? – Das fragen sich viele, wenn es darum geht, ein laufendes System durch ein neues zu ersetzen. Schließlich gibt es hier kein Plug-and-play. Dafür viele Optionen oder Strategien. Die richtig einzusetzen, darauf kommt es jetzt an.

5 Wege zur modernen IT

Im Grunde sprechen wir bei der Modernisierung über fünf Optionen oder Wege. Welcher Weg der Richtige ist, hängt von den individuellen Bedürfnissen und Ressourcen des Unternehmens, der Organisation ab, aber auch vom Leidensdruck.

1. **Continuation Monolith**
   Die Fortführung einer Legacy-Anwendung ist zwar nur bedingt hilfreich. Sie kann aber Sinn machen, wenn es bereits Pläne für eine vollständige Ablösung gibt, z. B. beim Einsatz einer Standardsoftware.

2. **Plattformshift**
   Um effektive Veränderungen herbeizuführen, die spürbare Auswirkungen haben, können Legacy-Lösungen, die oft in veralteten Technologien wie Cobol implementiert sind und auf alternder, teurer Hardware wie Mainframe-Systemen betrieben werden, auf eine modernere Plattform migriert werden.

3. **Generator-Concepts**
   Wenn ein Platformshift beispielsweise aufgrund technischer Limitierungen nicht möglich ist, muss die zugrundeliegende Technologiebasis geändert werden. In diesem Zusammenhang können Generatoransätze zum Einsatz kommen.

Mit Option 2 und 3 sparen Sie kurzfristig Kosten ein. Jedoch bleiben die ursprünglichen Probleme auf lange Sicht bestehen. Wie zum Beispiel mangelnde Wartbarkeit und fehlende Flexibilität.

4. **Monolith Crushing**
   Eine zukunftsgerichtete Modernisierung kann durch ein inkrementelles Aufbrechen erfolgen. Hierbei spechen wir von Monolith Crushing. Beim Monolith Crushing wird die Legacy-Anwendung, dem Strangler-Pattern [1] folgend, inkrementell zerlegt und nach und nach abgelöst. Ziel ist hierbei meist eine Microservices-Architektur oder ein „Modulith" – also ein „guter Monolith" mit sauberer Modulstruktur.

5. **Reengineering**
   Anders als beim Monolith Crushing werden beim Re-Engineering die Funktionalitäten des Altsystems neu implementiert. Zu einem definierten Zeitpunkt wird schließlich in einem „Big Bang"-Vorgehen auf das neu entwickelte System umgestellt.

Sie sehen, hier gibt es nicht die eine Option, die für alle passt. Auch nicht für alle Systeme einer einzelnen Organisation. Welche Option die Richtige ist, hängt beispielsweise davon ab, wie angreifbar oder komplex das Altsystem ist. Das muss situativ entschieden werden.

Ein System von der Stange?

Zusätzlich sollte geprüft werden, ob es für bestimmte Funktionalitäten eine Lösung von der Stange geben könnte: Ein *„commercial off-the-shelf"* (COTS). Insbesondere wenn es um Anforderungen geht, die nicht marktentscheidend sind und keine oder wenig Innovationskraft besitzen, kann ein Standardsystem zukunftsfähig sein: Es erfüllt sicher alle Standardanforderungen und setzt im Modernisierungsprozess Ressourcen frei, die anderweitig eingesetzt werden können. Zudem ist dies meist die kostengünstigste Option, wenn die abgebildeten Prozesse gut zu den Ist-Prozessen passen.

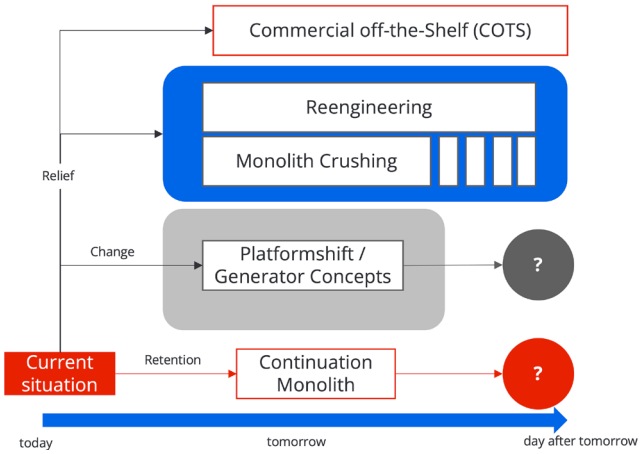


Abbildung 1: Strategische Modernisierungsoptionen

Welche Rolle spielen APIs bei der Modernisierung?

Wie Gelenke, die unsere Knochen miteinander verbinden und uns beweglich machen, so sind APIs die Dreh- und Angelpunkte bei der Modernisierung von IT-Architekturen.

Diese wichtige Rolle verdeutlicht auch Gartner in seinem Zielbild des Composable-Enterprise-Ansatzes. [2] In der Idee des Composable Enterprise besteht IT-Architektur immer mehr aus modularen Bausteinen, die flexibel kombiniert und erweitert werden können. Unternehmen ist es so möglich, agil auf sich ändernde Anforderungen zu reagieren und innovative Lösungen schneller an den Markt zu bringen.

- Als Verbindungsglieder sind APIs der Schlüssel für eine moderne und dynamik-robusten IT-Architektur [3], die den Anforderungen der digitalen Transformation gerecht wird.
- Neue Services und Applikationen sind Cloud-native [4] und sollten über eine intuitive Business API verfügen,
- idealerweise nach einem API-first Ansatz [5].

Das heißt, im Kontext ihrer Modernisierungsbestrebungen kommen Unternehmen nicht daran vorbei, sich mit APIs und einer durchgängigen API-Strategie zu beschäftigen.

API-Architekturstile

Viele verstehen unter einer API immer noch eine REST API im technischen Sinne. Stattdessen müssen wir APIs heute als ein Konzept begreifen

- für einen interoperablen, konsistenten Austausch von Daten,
- deren Format
- sowie den Zeitpunkt, wann Daten wo zur Verfügung stehen sollen.

Unabhängig von Technologien und Protokollen.

Neben REST sind heute weitere Stile von Bedeutung:

Stil	Beschreibung	Vorherrschendes Interaktionsmuster
REST	Ressourcen-orientierter Datenaustausch, meist basierend auf JSON-Format	sync
SOAP	XML-basierter, streng typisierter Datenaustausch der auf Operationen basiert	sync
GraphQL	Query Language die es erlaubt nur benötigte Daten abzufragen; streng typisiert	sync
gRPC	Modernes Remote Procedure Call (RPC) framework für den performanten Datenaustausch	bi-directional streaming
WebSocket	Statusbehafteter Datenaustausch zwischen Client und Server; für Real-time Use Cases geeignet	bi-directional streaming
Webhook	Event-basierte Kommunikation auf Basis von http; Serverapplikationen können Clients über Änderungen informieren	async
MQTT **AMQP** **Kafka**	Asynchrone Publish/Subscribe-Protokolle, die event-basierte Kommunikation ermöglichen	pub/sub

Wie die konkrete technische Umsetzung am Ende aussieht, ist abhängig von verschiedenen Faktoren, wie Antwortzeit, Datenaktualität oder Zuverlässigkeit. Anhand dieser Kriterien wird entschieden, welcher API-Architekturstil zu verwenden ist. Der am weitesten verbreitete API-Architekturstil ist nach wie vor REST [6].

Damit in der Architektur nicht alles drunter und drüber geht und ihre Konsistenz sichergestellt ist, muss klar definiert werden, welcher API-Architekturstil für welche Use Cases zu verwenden ist. Wir brauchen also eine API Governance, beispielsweise über einen Architecture Decision Record (ADR).

Eat your own dogfood! – Wenn API-getriebene Modernisierung im eigenen Haus erforderlich wird

Heute haben die meisten Unternehmen mit Modernisierungsherausforderungen zu kämpfen; wir als Consulting-Unternehmen sind da keine Ausnahme: Vor zwei Jahren haben wir uns entschieden, unser eigenentwickeltes Kernsystem abzulösen, das Projektverfolgungssystem PVS.

Die Gründe für die Ablösung waren typisch für ein über Jahrzehnte gewachsenen System. Sie reichten

- von einer gewachsenen Komplexität
- bis zur mangelnden Transparenz über Integrationen und Datenflüsse.

Eine veraltete Technologieplattform hinderte uns daran, neue Anforderungen effizient umzusetzen. Technische Basis war Oracle Forms, also ein datenbankzentrisches, monolithisches Applikationsframework.

Fachlich gesehen unterstützte das PVS alle wesentlichen Prozesse entlang der Unternehmenswertschöpfung in diversen Unternehmensbereichen. Von der Lead-Generierung bis zur Rechnungsstellung. Natürlich gab es hier zahlreiche eigenentwickelte Integrationen mit weiteren Systemen, die für die Abwicklung des Geschäfts benötigt werden.

Unser Weg: Inkrementelle Ablösung

Wir entschieden uns für eine inkrementelle Ablösung des PVS. Also gemäß Option 4, dem Ansatz des Monolith Crushing. (Vgl. oben)

- Dabei sollten Standardfunktionalitäten aus den Bereichen CRM, ERP oder HR mithilfe von Standardsoftware umgesetzt werden.
- Andere Funktionalitäten, wie die Kantinenverwaltung, wurden in Form autonomer Applikationen implementiert.

Ziel: Dynamikrobust statt direkte Kopplung

Unser Ziel war eine dynamikrobuste Integrationsplattform [3]. Als Rückgrat der IT-Landschaft soll diese unsere Systeme, Daten und Prozesse miteinander verbinden. Dabei ist sie so aufgebaut, dass zukünftige Anforderungen flexibel und sicher, oder anders ausgedrückt: dynamisch und robust, umgesetzt werden können. Mithilfe von APIs werden Daten aus unterschiedlichen Systemen zur Verfügung gestellt, in einer klar definierten Struktur.

Schritt 1: Architekturstil kann später erweitert werden

Bezugnehmend auf die vorgestellten API-Architekturstile setzen wir im ersten Schritt auf eine Kombination aus asynchronen, eventbasierten Schnittstellen und synchronen REST APIs. Durch die flexible Architektur der Integrationsplattform, ist eine Erweiterung um andere API-Architekturstile, wie GraphQL, um ein Backend-for-Frontend Pattern [7] zu implementieren, problemlos möglich.

Schritt 2: Funktionalitäten werden nach und nach abgelöst

In der ersten Iteration wurde, die CRM-Funktionalität als Capability abgelöst. (Siehe Abbildung 2) Damit werden unsere Sales- und Marketing-Prozesse künftig durch die CRM-Software Salesforce unterstützt und nicht mehr durch das PVS. Trotzdem müssen neue Daten und auch Änderungen an bestehenden Daten ins PVS zurücküberführt werden, um zu gewährleisten, dass nachgelagerte Prozesse wie die Rechnungsstellung nahtlos funktionieren.

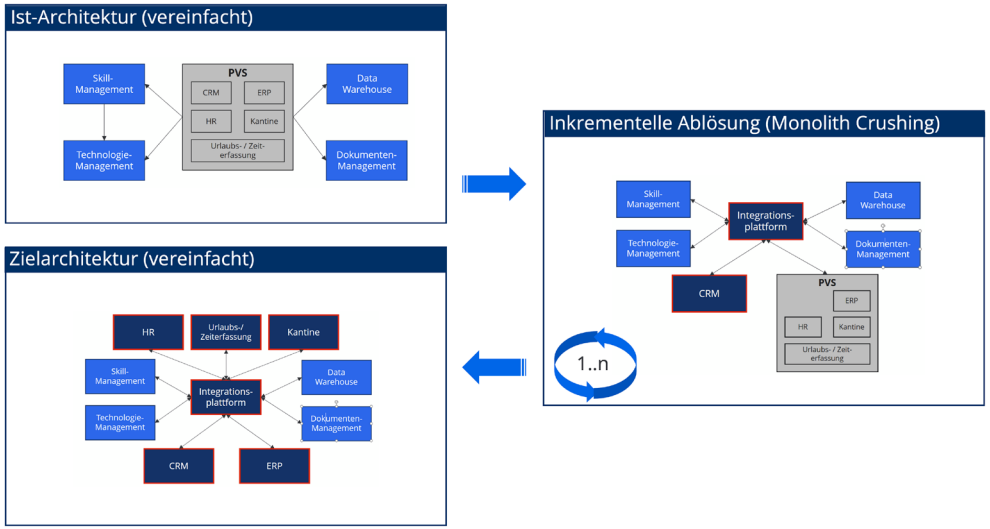


Abbildung 2: Inkrementelle Ablösung (Monolith Crushing)

Schritt 3: Datenformat ist wichtig für eine gute Entkopplung

Um Quell- und Zielsystem möglichst gut voneinander zu entkoppeln, haben wir innerhalb der Integrationsplattform mit einem einheitlichen Datenformat, im Sinne einer „Ubiquitous Language“ [8], gearbeitet. Daraus ergibt sich ein Datenmodell für unsere Business APIs, über das Daten effizient abgerufen werden können.
Das einheitliche Datenformat ist systemunabhängig. Deshalb braucht es auf Consumer-Seite kein Systemwissen, um das Datenmodell zu verstehen. Die Überführung in das einheitliche Format passiert dabei möglichst nah am Quellsystem. (Siehe Abbildung 3, Seite 17)

Schritt 4: Connectivity Services

Für die Implementierung der Transformationen sowie die Herstellung der Verbindungen zwischen Quell- und Zielsystemen gibt es Connectivity Services, die in Form Cloud-nativer Services bereitgestellt und betrieben werden können.

Schritt 5: Die Rückintegration

Abbildung 3 zeigt den schematischen Ablauf bei der Rückintegration:

1. Eine Änderung in Salesforce führt zu einem Event,
2. Das Event wird in ein Domänen-Event umgewandelt.
3. Das Domänen-Event wird in ein PVS-Format überführt und an das PVS übermittelt.
4. Durch die Verwendung eines einheitlichen Datenformats können weitere Systeme, die ebenfalls an Änderungen in Salesforce interessiert sind, leicht integriert werden.

Schritt 6: Erweiterung um eine API Gateway

In Abbildung 3 ist zu sehen, dass wir die Plattform um ein API-Gateway erweitert haben. Das war wichtig, um zu gewährleisten, dass Daten mittels synchroner REST APIs intern und extern sicher bereitgestellt werden (5).

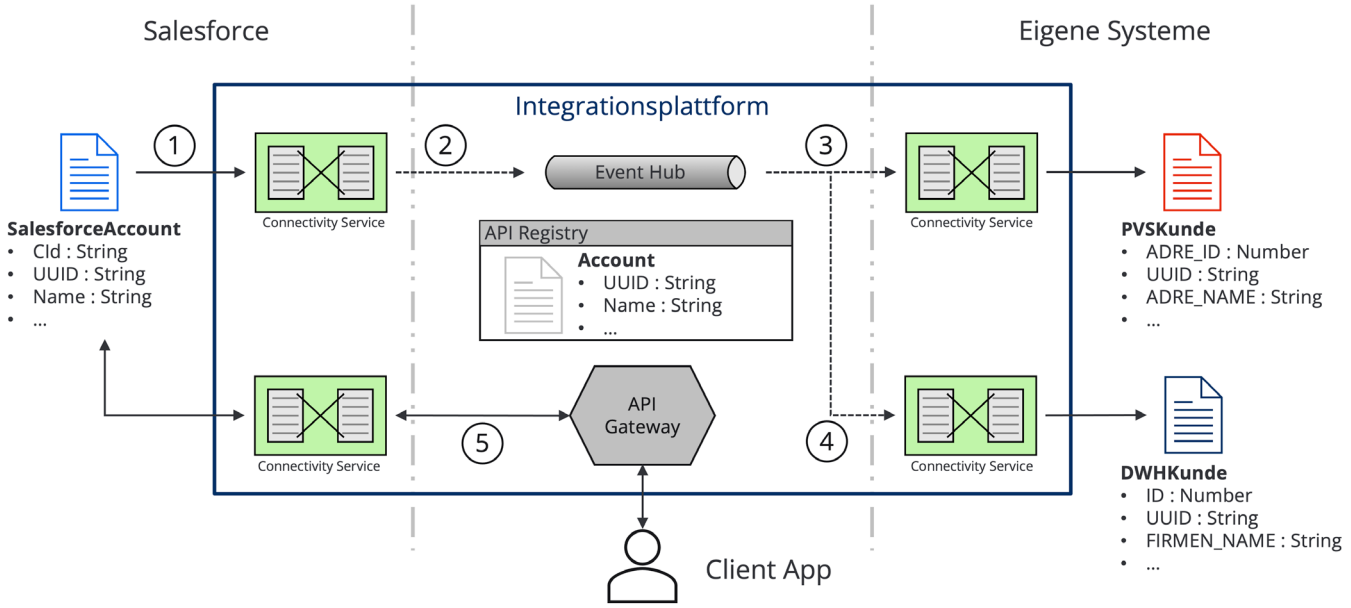


Abbildung 3: API-basierte Integration

Fazit

Dieser Beitrag zeigt Wege und Möglichkeiten zu einer modernen IT-Landschaft auf. Vor allem ging es aber darum, zu zeigen, wie wichtig APIs für moderne Architekturen sind. Sie sind Dreh und Angelpunkte, die unsere IT-Landschaften beweglich und sicher machen. Ihnen haben wir es zu verdanken, wenn wir direkte Systeme-zu-System-Kopplungen mit all ihren Nachteilen hinter uns lassen können.

Wollen wir digitale Produkte sicher und zuverlässig anbieten, müssen wir APIs als das behandeln, was sie sind: Ein kritischer Erfolgsfaktor und Enabler für die Modernisierung. Auch bei der Modernisierung von gewachsenen IT-Systemlandschaften spielen APIs eine tragende Rolle. Das konnten wir bei der Ablösung unseres Projektverfolgungssystems PVS beobachten. Der wichtigste Grund für die Schlüsselposition der APIs ist jedoch, dass sie uns helfen, die „alte“ und die „neue“ Welt zusammenzubringen. Mithilfe klar definierter, standardbasierter Schnittstellen können Integrationsanforderungen im Kontext heterogener Technologiestacks oder Architekturstile adressiert werden. Das ist vor allem deshalb wichtig, weil die Modernisierung gewachsener Systemlandschaften ein langer Prozess ist.

Mit Bedacht vorgehen

Wir wollen nicht verschweigen, dass auch bei der Verwendung von APIs entsprechende Rahmenbedingungen vorgegeben werden müssen. Hierfür bedarf es einer entsprechenden API-Strategie. Denn als Key Asset digitaler Unternehmen haben APIs einen Lebenszyklus, der sauber und nachvollziehbar zu managen ist. Zu einer API-Strategie gehören neben der zu verwendenden Technologieplattform zum Beispiel auch Leitplanken für API-Design, Governance und Security.

Auf den Punkt gebracht lässt sich sagen:

- **Wir müssen APIs strategisch betrachten**
  Es ist von entscheidender Bedeutung, APIs aus einer strategischen Perspektive zu betrachten und sie auf Basis einer durchdachten und konsistenten API-Strategie zu implementieren.

- **Wir sollten die Menschen mitnehmen**
  Wir dürfen nicht vergessen, die gesamte Organisation in diesen Prozess einzubeziehen – vor allem aber die Menschen, die eine Technologie, eine Software, ein System nutzen.

Wenn wir diese Punkte berücksichtigen, können wir die Herausforderungen der Modernisierung entmystifizieren und sie auf eine leichtere, erfolgreichere und effizientere Weise bewältigen.

Quellen:

[1] https://microservices.io/patterns/refactoring/strangler-application.html
[2] https://www.gartner.com/en/doc/465932-future-of-applications-delivering-the-composable-enterprise
[3] https://www.opitz-consulting.com/kompetenz/architecting-digital-world
[4] https://github.com/cncf/toc/blob/main/DEFINITION.md#deutsch
[5] https://www.postman.com/state-of-api/api-first-strategies/#api-first-strategies
[5] https://smartbear.com/state-of-software-quality/api/design/
[6] https://www.postman.com/state-of-api/api-technologies/#api-technologies
[6] https://smartbear.com/state-of-software-quality/api/tools/
[7] https://microservices.io/patterns/apigateway.html
[8] https://martinfowler.com/bliki/UbiquitousLanguage.html

Sven Bernhardt

ist ein Technologie-Enthusiast und arbeitet für OPITZ CONSULTING als Chief Architect im Corporate Development Team. In seiner Rolle ist er für das Management des Technologieportfolios und die Entwicklung von Best Practices und Richtlinien verantwortlich. Darüber hinaus unterstützt Sven seine Kollegen bei der Implementierung von Softwarelösungen für Kunden. Als Speaker spricht er regelmäßig auf Konferenzen über Technologie- und Architekturthemen und teilt seine Erfahrungen in Artikeln und Blogbeiträgen.

OPITZ CONSULTING

OPITZ CONSULTING Deutschland GmbH
Kirchstraße 6
51647 Gummersbach (Nochen)
+49 2261 6001-0
www.opitz-consulting.com