Ausgabe 04|2022

Deutschland € 12,90 Österreich € 13,90 Schweiz sfr 22,20

4 194164 112900 04

IT Spektrum

vormals OBJEKTspektrum

Digitaler Wandel & Software-Architektur für Profis

Modernisierung – schrittweise und mit Alttechnologie-Know-how

DIP um IOSP ergänzen

Das Dependency Inversion Principle richtet Schaden an

Teamzusammenarbeit gestalten

Selbstorganisation – einfach mal ausprobieren

Neue Serie Architektur-Porträt

Was der Quelltexteditor Visual Studio Code kann

www.ITSpektrum.de

Modernisierung – schrittweise und mit Alttechnologie-Know-how

G 6540F

Microservices-Architekturen

Modernisierung – schrittweise und mit Alttechnologie-Know-how

Viele Unternehmen betreiben zum Teil über 20 Jahre alte Legacy-Systeme, die den aktuellen Digitalisierungsanforderungen nicht mehr gewachsen sind. Probleme hierbei sind insbesondere die mangelnde Wartbarkeit, Flexibilität, Stabilität und Skalierbarkeit bei zugleich hohen Kosten. Eine „Big-Bang"-Ablösung eines solchen Systems ist mit einem hohen Risiko verbunden. Eine Alternative hierzu ist die schrittweise Ablösung mithilfe von Microservices in Kombination mit fundiertem Alttechnologie-Know-how.

Quelle Headerbild: Adobe Stock_iuriimotov, Collage: Consist Software Solutions

Der Begriff Legacy-System hat seinen Ursprung im Englischen („Erbschaft") und bezeichnet eine historisch gewachsene Unternehmenssoftware. Es existiert keine allgemein akzeptierte Definition, welche Kriterien ein System zu einem Legacy-System machen. Die Gartner Inc. definiert ein Legacy-System wie folgt: „An information system that may be based on outdated technologies, but is critical to day-to-day operations" [Gar]. Was das in der Praxis für Auswirkungen für ein Unternehmen hat, beleuchten wir in einem ersten Schritt.

Welche Probleme bringt der Einsatz von Legacy-Systemen mit sich?

Legacy-Systeme wurden meist in einer Zeit entwickelt, als die Anforderungen an IT-Systeme und auch deren technische Möglichkeiten deutlich andere waren als heutzutage. Die Systeme waren konzeptionell Inhouse-Systeme mit einer überschaubaren Benutzeranzahl und verhältnismäßig wenigen funktionalen Änderungen in langen Time-to-Market-Zyklen (z. B. 4 Releases pro Jahr). Auch die nicht funktionalen Anforderungen im Hinblick auf Verfügbarkeit (z. B. nur 6 bis 20 Uhr) und Sicherheit (keine Internetanbindung) waren andere als heutzutage.
Hieraus resultieren aktuell im Zeitalter der Digitalisierung mit einer weitreichenden Vernetzung und einem sich schnell verändernden Markt und Umfeld erhebliche Probleme. Die Systeme sind nicht flexibel genug, um schnell und zuverlässig funktional weiterentwickelt zu werden. Eine Öffnung für Internet-Nutzer (z. B. Kunden) und die beschleunigte und häufigere Ausführung von Geschäftspro-

StranglerFigApplication

Das „StranglerFigApplication"-Pattern ist ein von Martin Fowler in 2004 beschriebenes Muster. Er beschreibt dort, wie ihm bei einem Australien-Besuch die Würgefeigen („StranglerFig") als Metapher für ein Vorgehen beim Modernisieren von kritischen Systemen in den Sinn kamen. Die Würgefeige wächst von der Krone eines Baumes zum Boden und umschließt ihn mit ihren Wurzeln. Irgendwann ist der Wirtsbaum abgestorben und durch die Feige ersetzt.

Die Modernisierung eines Systems erfolgt hierbei schrittweise durch das Verlagern von Funktionalitäten aus dem Altsystem in die Strangler-Applikation, bis diese das Altsystem vollständig ersetzt.

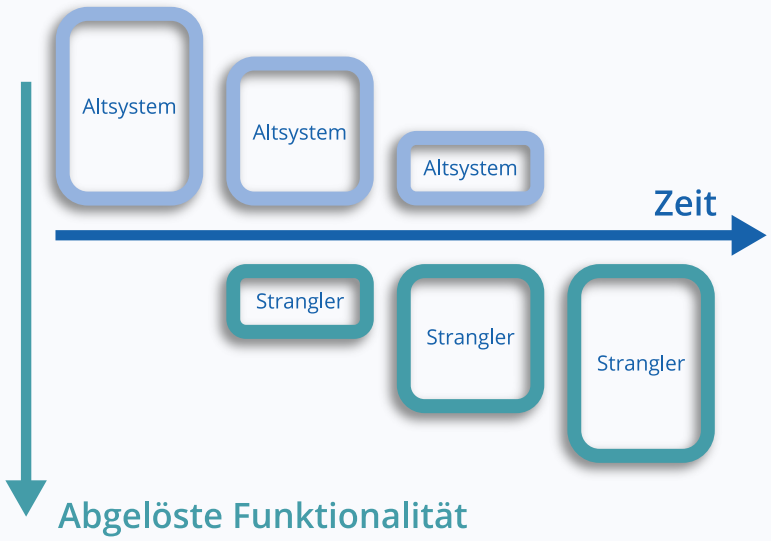

Abb. 1: StranglerFigApplication-Pattern, Quelle: Consist Software Solutions

zessen führen zu Skalierbarkeits- und Sicherheitsproblemen. Der Betrieb der Legacy-Hardware und Plattformen ist per se schon mit hohen Kosten verbunden. Ein unterbrechungsfreier 24x7-Betrieb potenziert diese Kosten noch neben den rein technischen Problemen. Die Silo-Datenhaltung der Altsysteme stellt eine Hürde für die unternehmensweite Nutzung der anfallenden Daten dar. Und abschließend seien noch die veralteten Benutzeroberflächen, die heutzutage zu erheblichen Akzeptanzproblemen führen, und die unflexiblen und schwerfälligen Schnittstellen-Technologien erwähnt.
Neben diesen rein objektiven Problemen, die der Einsatz von Legacy-Systemen mit sich bringt, schwebt darüber noch das Damoklesschwert des immer weiter schwindenden Know-hows über das System und dessen Technologien (z. B. Personal geht in den Ruhestand). Dies stellt für eine Vielzahl von Unternehmen ein erhebliches Risiko dar.
Im nächsten Abschnitt betrachten wir nun, was die Ablösung von Legacy-Systemen in der Praxis so schwierig macht.

Warum ist die Ablösung von Legacy-Systemen so schwierig?

Jede Softwareentwicklung wird begleitet von einer Reihe kleinerer und größerer Herausforderungen, die gelöst werden müssen. Dies bereitet in der Praxis selten „echte" Probleme, da ein eingespieltes Team mit fundiertem und aktuellem Know-how über das System vorhanden ist.

Bei der Ablösung von Altsystemen stellt sich die Ausgangssituation jedoch im Allgemeinen deutlich anders dar. Aufgrund der Kritikalität von Legacy-Systemen stellt eine Ablösung immer ein Risiko für die Funktionsfähigkeit des gesamten Unternehmens dar. Dies geht einher mit dem großen Funktionsumfang solcher Systeme in Kombination mit erodierten Strukturen, die aus einer jahrzehntelangen Entwicklungshistorie mit unterschiedlichsten Akteuren resultieren. Auch sind in den wenigsten Fällen eine flächendeckende und aktuelle Spezifikation und Dokumentation der Systeme vorhanden. Erschwerend kommt hinzu, dass oftmals auch der Sourcecode nur für „Eingeweihte" verständlich ist, was aufgrund des schwindenden Know-hows über diese Systeme das Verständnis weiter erschwert.
Neben diesen rein formalen Problemen eines Legacy-Systems stellt auch der eigentliche Ablösungsprozess während des Betriebs und der Weiterentwicklung des Altsystems eine Herausforderung dar.
Diese Probleme führen in Summe dazu, dass ohne ein strukturiertes und fundiertes Vorgehen die Ablösung eines Altsystems mit einem hohen Risiko bzgl. Zeitplan und Kosten behaftet ist. In extremen Fällen droht auch das komplette Scheitern der Ablösung. Diese Risiken führen in der Praxis häufig dazu, dass Entscheider in Unternehmen eine Ablösung hinausschieben oder sogar versuchen, diese zu vermeiden, insbesondere wenn die aktuelle Position nur eine kurze Stufe auf der Karriereleiter ist.
Als Appell aus der Praxis sei abschließend angemerkt, dass durch ein zu langes Festhalten an Altsystemen auch die wirtschaftliche Entwicklung eines Unternehmens im Vergleich zu Wettbewerbern mit zeitgemäßen IT-Systemen deutlich schlechter ist.
Als Quintessenz lässt sich feststellen, dass spätestens jetzt der richtige Zeitpunkt zur Ablösung von Altsystemen ist. Bevor wir hierfür eines der möglichen Lösungskonzepte vorstellen, betrachten wir kurz die architektonischen und technologischen Rahmenbedingungen für eine Neuentwicklung.

Architektur und Technologie bei Modernisierungen – eine Kurzbetrachtung

Der Fokus dieses Artikels liegt in erster Linie auf einem praxisorientierten Vorgehensmodell für die Neuentwicklung von Legacy-Systemen und nicht auf den hierfür notwendigen Architekturen und Technologien. Dennoch haben diese Konsequenzen für das Vorgehen, die wir im folgenden Exkurs kurz beleuchten.
Aus unserer Erfahrung heraus haben Legacy-Systeme im Allgemeinen eine Komplexität und Größe, die den Einsatz einer Microservices-Architektur sinnvoll macht. Neben deren Vorteilen seien aber auch die damit einhergehenden Nachteile erwähnt. Microservices basieren auf einem verteilten System, das „Eventually Consistent" ist. Das erhöht die Programmierkomplexität und erschwert unter Umständen die Abbildung von Business-Transaktionen. Darüber hinaus erfordern Microservices, um diese in der Praxis effizient zu betreiben, das Vorhandensein

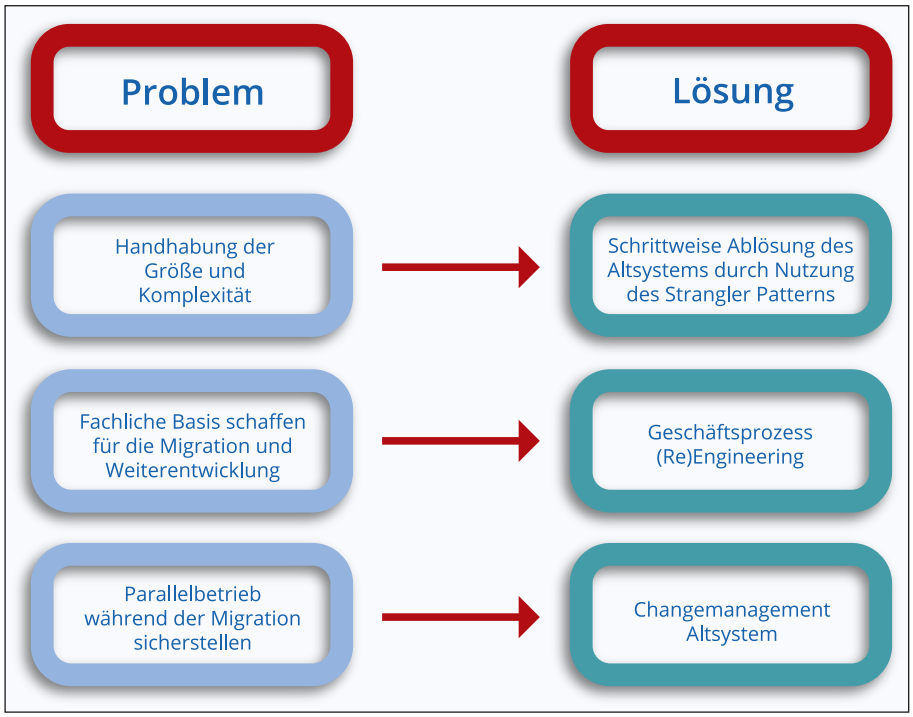

Abb. 2: Ansatz zur Ablösung von Altsystemen – grundlegende Mechanismen, Quelle: Consist Software Solutions

einer automatisierten DevOps-Pipeline. Eine architektonische Alternative zu Microservices, insbesondere für „kleinere" Systeme, stellt deshalb der „Structured Monolith" dar.

Die Modernisierung eines Legacy-Systems bietet darüber hinaus auch die Möglichkeit, eine standardisierte Ausführungsumgebung für alle Systeme in einem Unternehmen zu etablieren. Diese abstrahiert dann die verschiedenen Betriebs(system)-Umgebungen und vereinfacht so die Administration und den Betrieb der Systeme. Als De-facto-Standard hierfür hat sich die Containerisierung von Applikationen herauskristallisiert in Kombination mit einer Container-Orchestrierungsplattform (z. B. Kubernetes).

Prinzipieller Lösungsansatz zur Ablösung von Altsystemen

Der grundsätzliche Lösungsansatz basiert auf der Nutzung des Strangler-Patterns (StranglerFigApplication [Fow], siehe **Abbildung 1**), um so eine schrittweise Ablösung des Altsystems zu ermöglichen.

Ergänzt wird dieses Paradigma durch weitere Mechanismen in den Bereichen fachliche Dokumentation und dem Parallelbetrieb Alt-Neusystem. Der konkrete Ansatz ist in **Abbildung 2** dargestellt.

Der Strangler-Ansatz bietet die Möglichkeit, fortwährend alte Funktionalitäten/Features durch Neuimplementierungen zu ersetzen, bis die gesamte Anwendung modernisiert ist. Neben der Risikominimierung gegenüber einem „Big-Bang-Ansatz" bietet sich als weiterer Vorteil an, dass die Erfahrungen aus den jeweiligen Erneuerungsschritten auch direkt in die weitere Planung und Umsetzung einfließen können.

Neben der reinen Größe und Komplexität eines Altsystems stellt die mangelnde fachliche Spezifikation häufig eines der größten Probleme bei deren Ablösung dar. Dieses Problem wird durch einen hybriden Ansatz gelöst. Zum einen erfolgt eine fachliche Top-down-Analyse und Dokumentation der Geschäftsprozesse („Anwendersicht") und zum anderen eine Bottom-up-Analyse des Altsystems („Sourcecode") in Fällen von Unklarheiten/Unschärfen. Für den zweiten Schritt sind Spezialisten erforderlich, die die Legacy-Technologien beherrschen.

Im Rahmen dieses Prozesses besteht dann auch die Möglichkeit, Geschäftsprozesse zu überdenken und gegebenenfalls anzupassen.

Da im Rahmen einer Migrationsphase in der Regel auch noch Anpassungen und Erweiterung am Altsystem erfolgen, ist ein formales Änderungsmanagement für das Altsystem eine Grundvoraussetzung, um die Änderung dann auch im neuen System nachzuziehen.

Im nächsten Schritt überführen wird diesen prinzipiellen Lösungsansatz in ein spezifisches Vorgehensmodell.

Vorgehensmodell Analyse-Split-Strangle

Das Vorgehensmodell besteht aus den drei grundlegenden Phasen, die in **Abbildung 3** dargestellt sind. Die einzelnen Phasen stellen sich im Detail wie folgt dar.

Die Analyse-Phase – das Fundament

Ziel der Analyse-Phase ist es, die fachliche Basis für die Neu- und Weiterentwicklung des Systems zu schaffen. Wichtig ist hierbei, sich nicht in Details zu verlieren, sondern die grundlegenden Strukturen zu verstehen und zu dokumentieren. Diese sind dann die Basis für die weiteren Entscheidungen, zum Beispiel in welche (Micro-)Services das Neusystem zerlegt wird und welche Abhängigkeiten zwischen diesen bestehen.

Die wichtigsten Bestandteile einer solchen Dokumentation sind das Datenmodell und die funktionale Beschreibung des Systems. Hierfür wird ein Fachklassenmodell erstellt, das die grundlegenden fachlichen

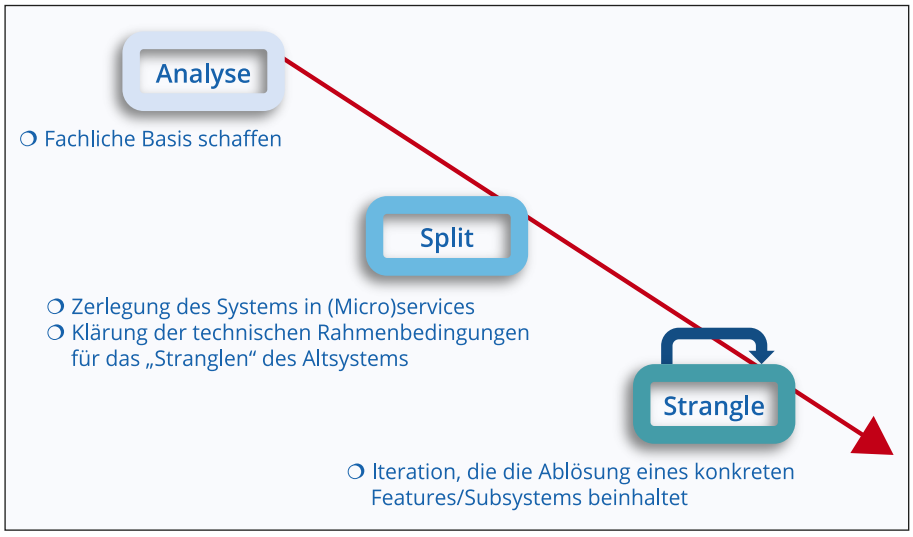

Abb. 3: Vorgehensmodell Analyse-Split-Strangle, Quelle: Consist Software Solutions

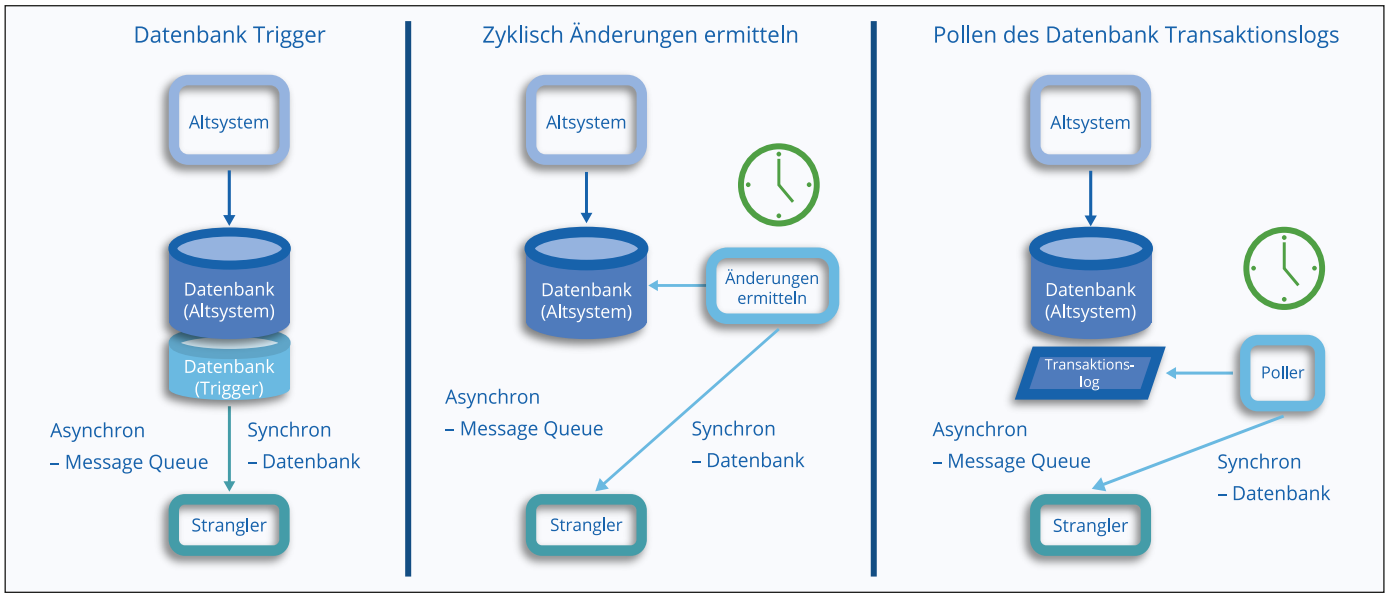

Abb. 4: Mechanismen zum Datenabgleich Altsystem – Strangler-Applikation, Quelle: Consist Software Solutions

Entitäten/Klassen und deren strukturelle Beziehung beschreibt. Ergänzt wird dieses durch einen Datenkatalog, der die wesentlichen Attribute der Fachklassen beschreibt. Im nächsten Schritt erfolgt dann eine grundlegende Beschreibung aller Geschäftsprozesse des Systems. Diese besteht aus der Beschreibung der wesentlichen Abläufe der einzelnen Geschäftsprozesse (einzelne Schritte ohne Details oder alternativ ein UML-Aktivitätsdiagramm) und welche anderen Prozesse/Services dabei genutzt werden.
Die Ergebnisse dieser Phase spiegeln in etwa den Umfang eines groben Lastenhefts wider. Sie sind das Fundament für den nächsten Schritt, in dem das System für die Ablösung in einzelne, in sich autarke Subsysteme zerlegt wird.

Die Split-Phase – die Planung

Der erste Schritt innerhalb der Split-Phase ist die Zerlegung des Gesamtsystems in (Micro-)Services. Hierfür hat sich die Nutzung von Domain-Driven Design [Ver13] als Methodik etabliert. Dabei wird das Gesamtsystem in sogenannte (fachliche) Bounded Contexts zerlegt, die jeweils über eine private Datenhaltung verfügen und nur über öffentliche, definierte Schnittstellen kommunizieren (Microservices). Hierdurch wird die Handhabbarkeit der Gesamtkomplexität des Systems erreicht, da die Entwickler immer nur den kleinen Teil des Systems kennen und überblicken müssen inklusive dessen Schnittstellen zu den anderen Microservices, den sie aktuell bearbeiten. Dieser Schritt ist einer der wichtigsten konzeptionellen Tätigkeiten bei der Ablösung eines Altsystems, insbesondere wegen seiner weitreichenden Konsequenzen für die Struktur der Neuimplementierung. Ziel ist die Überführung der fachlichen Anforderungen in eine möglichst lose gekoppelte Systemstruktur.
Im zweiten Schritt erfolgen dann die Evaluierung und Konzeptionierung der konkreten (Schnittstellen-)Techniken für den Strangle-Prozess. Ziel ist es, für die Strangler-Übergangsphase eine möglichst einfache und aufwandsarme technische Integration der Altsystembestandteile mit den neuen Komponenten zu realisieren und so einen reibungslosen Betrieb während der Migration sicherzustellen. Dieser Prozess ist nicht trivial und erfordert sowohl einiges an Kreativität auf der Entwicklungsseite als auch Kompromissbereitschaft auf der Nutzerseite (z. B. in Form von zwei Benutzeroberflächen während der Strangler-Phase).
Grundlegende Voraussetzung für diesen Schritt ist qualifiziertes Personal, das die Alttechnologien beherrscht und entsprechende Analysen und Anpassungen am Altsystem vornehmen kann.
Die Gestaltung der Strangler-Schnittstellen sollte möglichst geringe Änderungen und Eingriffe am Altsystem erfordern und auch nicht die Geschäftslogik verändern. Darüber hinaus muss die Möglichkeit eines Parallelbetriebes Alt-Neu für eine Test-/Übergangsphase gegeben sein. Im Anschluss an diese muss eine einfache Abschaltung der Funktionalität im Altsystem möglich sein.
Ein erster und einfacher Schritt hierzu ist die Sichtung der vorhandenen Schnittstellen des Altsystems und inwiefern diese für das „Strangeln“ verwendet werden können. Eine Integration auf Benutzeroberflächen-Ebene ist bei „alten Altsystemen“, die über eine Terminal-Emulation (z. B. IBM 3270) als Benutzeroberfläche verfügen, mit einer neuen modernen Weboberfläche schwierig. Eine in der Praxis weitverbreitete Lösung ist daher die Integration auf Datenebene, in der die Datenänderungen im Altsystem in das neue System (Strangler-Applikation) propagiert werden. Hierfür existieren die in **Abbildung 4** dargestellten technischen Lösungsansätze.
Welches das geeignete Verfahren in einem konkreten Anwendungsfall ist, hängt von den jeweiligen Rahmenbedingungen ab. Die Lösung über Datenbank-Trigger ist relativ einfach zu realisieren und ist dann praktikabel, wenn man mit wenigen Triggern auskommt. Wenn das nicht der Fall ist, wird diese Lösung schnell kompliziert und schwer wart- und betreibbar.
Der Ansatz der zyklischen Ermittlung von Änderungen sieht bei erster Betrachtung einfach aus, hat aber seine Tücken in der Ermittlung der Daten, die wirklich geändert wurden. Hierfür sind gegebenenfalls Eingriffe am Datenbankschema erforderlich (z. B. Timestamp des Transfers), die das Verfahren in der Praxis kompliziert machen. Der Lösungsansatz durch Pollen des Datenbank-Transaktionslog ist die einfachste Lösung, wenn man auf ein bestehendes Framework/Werkzeug aufsetzen kann, das die eingesetzte Datenbank-Technologie unterstützt. Der Datentransfer in die Strangler-Applikation kann in allen Verfahren entweder synchron erfolgen oder wenn eine entsprechende Messaging-Infrastruktur vorhanden ist, bevorzugt asynchron zur losen Kopplung.
Das letztendliche Ergebnis dieser Analyse ist ein technisches Lösungskonzept mit den entsprechenden Schnittstellen, mittels derer das Altsystem „gestrangelt“ werden

kann. Dieses kann einen signifikanten Anteil an den Aufwänden, der Dauer und damit auch Kosten eines Projektes ausmachen.

Die Strangle-Phase – die Realisierung

In der Strangle-Phase erfolgt die eigentliche Ablösung des Altsystems. Diese besteht aus einzelnen Strangle-Iterationen, in denen jeweils Teile der Funktionalität neu entwickelt werden.

Diese Phase basiert auf dem Einsatz des Strangler Fig Application Pattern's. Sam Newman hat in seinem Buch [New19] die Anwendung des Patterns für eine Microservices-Architektur in drei Schritten beschrieben. Der erste Schritt hiervon, die Identifikation von gut isolierten Funktionalitäten, ist bereits in der Phase 2 (Split) unseres Vorgehensmodells erfolgt.

Der zweite Schritt bei diesem Pattern ist die Verlagerung spezifischer Funktionalitäten in einen Microservice. Jede Strangle-Iteration beginnt deshalb mit der Auswahl des Bounded Context, der abgelöst werden soll. Für diesen Kontext erfolgt dann die Detaillierung/Vervollständigung der fachlichen Dokumentation. Basis hierfür sind die Ergebnisse aus der Analyse-Phase. Diese werden dann ergänzt durch eine detaillierte Beschreibung der Geschäftsprozesse und den diesen zugrunde liegenden Geschäftsregeln, sodass die Datenstrukturen und Prozesse des Kontextes eindeutig beschrieben sind. Anschließend wird mit der Realisierung des korrespondierenden Microservices begonnen. Das bedeutet je nach Konstellation und Technologie, dass bestehender Programmcode entweder so weit wie möglich transferiert oder komplett neuentwickelt wird. Darüber hinaus erfolgt (wenn notwendig) auch die Erweiterung des Altsystems um die Strangler-Schnittstellen für die Kommunikation zwischen diesem und dem neuen Microservice. Das Altsystem wird in der Regel in dieser Phase noch die externen Anfragen entgegennehmen und diese dann an die Neuimplementierung weiterleiten.

Im dritten und letzten Schritt werden dann auch die externen Aufrufe vom Microservice entgegengenommen und dieser ist dann vollumfänglich für die Funktionalität zuständig. Hierbei kann es dann auch notwendig sein, die funktionale Weiterentwicklung am Altsystem, die mittels des Changemanagements erfasst wurde, nachzuziehen. Um einen konsistenten Stand zu erreichen, muss dies in einem atomaren Schritt erfolgen, in dem keine weiteren Änderungen am Altsystem erfolgen.

So wird dann Stück für Stück Funktionalität aus dem Altsystem herausgelöst und neu implementiert (Strangler-Iteration). Schließlich ist dann der Punkt der letzten Iteration erreicht, in dem die noch verbliebene Funktionalität abgelöst wird. Wie diese sich genau gestaltet, hängt von verschiedenen Faktoren ab. Zum einen kann die verbliebene Funktionalität vom Umfang so gering sein, dass sie als Ganzes abgelöst werden kann, oder der Rest ist so miteinander verwoben, dass es keine Teile mehr gibt, die sich mit vertretbarem Aufwand im Altsystem isolieren lassen.

Fazit und Ausblick

Die Ablösung unternehmenskritischer Altsysteme durch eine Neuentwicklung ist aufgrund der Komplexität, der Größe und der Entwicklungshistorie nie einfach. Wie immer gibt es nicht nur den einen richtigen Weg, dieses zu erreichen, sondern viele. Im Rahmen des Artikels wird ein 3-phasiges, iteratives Vorgehensmodell (Analyse-Split-Strangle) zur Ablösung von Altsystemen vorgestellt, das auf den Kernelementen Strangler-Pattern, Businessprozess (Re)Engineering und Changemanagement für das Altsystem fußt.

Eine generell unterschätzte Grundvoraussetzung für die erfolgreiche Ablösung eines Altsystems ist die Verfügbarkeit von qualifiziertem Personal, das die Technologien des Altsystems beherrscht. Insbesondere durch den Strangler-Ansatz, auf dem das beschriebene Vorgehen basiert und der in der Regel mit Eingriffen am Altsystem einhergeht, ist die Erfüllung dieser Voraussetzung von essenzieller Bedeutung. Darüber hinaus sind weitere Erfolgsfaktoren ein effizientes Anforderungsmanagement zur Erstellung der fachlichen Dokumentation und die adäquate Zerlegung des Gesamtsystems in losegekoppelte Microservices, zum Beispiel mittels Domain-Driven Design.

Neben der rein technischen Ablösung eines Altsystems durch eine Neuentwicklung, ergibt sich hierdurch für ein Unternehmen auch eine völlig neue technische Basis. Diese ermöglicht die einfache Integration aktueller Technologien, wie zum Beispiel Cloud, Data Science usw., und eröffnet so auch völlig neue Möglichkeiten für die geschäftliche Entwicklung eines Unternehmens. ||

Literatur & Links

[Fow] M. Fowler, StranglerFigApplication, 29.06.2004, siehe: **https://martinfowler.com/bliki/StranglerFigApplication.html**

[Gar] Gartner Glossary, Legacy Application Or System, siehe: **https://www.gartner.com/en/information-technology/glossary/legacy-application-or-system**

[New19] S. Newman, Monolith to Microservices, O'Reilly Media, Inc., 2019

[Ver13] V. Vernon, Implementing Domain-Driven Design, Addison-Wesley Professional, 2013

Die Autoren

Christian Bergert
(bergert@consist.de)
arbeitet als Softwarearchitekt für die Consist Software Solutions GmbH. Sein aktueller Tätigkeitsschwerpunkt liegt im Bereich Software-Engineering mit dem Fokus auf Microservices-Anwendungen und Serverless Computing.

Markus Scheil
(scheil@consist.de)
arbeitet als Softwarearchitekt für die Consist Software Solutions GmbH. Nach langjähriger Entwicklungstätigkeit für ein Versicherungsunternehmen im Bereich Mainframe und Cobol ist sein aktueller Tätigkeitsschwerpunkt im Bereich Microservices und DevOps.