# Vom Monolithen zur Service-Architektur mit Hilfe von Graphen

Andres Koch (akoch@objeng.ch), Remo Koch (rkoch@objeng.ch)
Object Engineering GmbH, CH-8142 Uitikon-Waldegg

## Abstract

Monolithen sind nicht nur bei Mainframe-Applikationen auffindbar. Im Gegenteil, Applikationen der «Moderne», welche mit neueren objektorientierten Technologien, Sprachen und Frameworks erstellt wurden, evolvierten nach einigen Jahren zu noch komplexeren Gebilden, als man es je gekannt hat. Zukunftssichere und flexible Systeme wurden seit vielen Jahren basierend auf Service-Komponenten gebaut. Ein Reengineering, das aus Graphen-Metadaten von bestehenden, komplexen Applikationen mittels gängigen Graphen-Algorithmen sinnvolle Komponenten-Grenzen findet, stellt einen wirkungsvollen, teils heuristischen Ansatz dar.

## Service-Architekturen

Aus langjähriger Erfahrung mit Systemen, die auf Service-Komponenten basieren, konnten wir feststellen, dass solche Systeme flexibel genug sind, um neuen, funktionalen und technologischen Anforderungen auch nach Jahren noch gerecht zu werden. Die Grundlage dazu wurde in IT-Urzeit von David L. Parnas in seinem ACM-Artikel «On The Criteria To Be Used In Decomposing Systems Into Modules» im Dezember 1972 [1] beschrieben.

Gemäss diesem Prinzip kann der Service als das grundlegende Konstruktions-Element für jede, heute typisch verteilte Umgebung, betrachtet werden.

Ein Service kapselt eine Funktionalität, mit oder ohne persistente Daten, welche einer Business-Domäne entspricht, oder eine generell verwendbare Funktionalität. Diese wird über eine oder mehrere Schnittstellen (API pro Aspekt) für den Konsumenten verfügbar gemacht.

Trotz der Grundlagen von D. L. Parnas ist der Funktions-Umfang pro Service (Modul) seit Jahrzehnten eine nicht wirklich klar beantwortete Fragestellung. Die Aufteilung mit Hilfe von Domain Driven Design (DDD) ist ein guter Ansatz, auch wenn dies in der Praxis immer noch eine Herausforderung darstellt. Die Problemstellung besteht aus dem Aufteilen von grossen, bereits existierenden Gebilden in logische Module von besser verwaltbarem Umfang.

## Monolithen

Auch Monolithen wurden meistens modular entworfen. Die Programmierer konnten aber ohne wirkliche Hindernisse (Service-Schnittstellen) praktisch jede Funktion innerhalb des Monolithen aufrufen. Genau diese «Freiheit» ergab über die Jahre der Weiterentwicklung ein komplexes und verworrenes Gebilde. Der Aufwand solche Gebilde zu verstehen und aufzubrechen, respektive durch ein Re-Design wieder wartbar zu machen, kann ohne automatisierte Unterstützung in Grössenordnungen schnellen, die keinem Wartungsbudget mehr gerecht werden [4].

Die Verschiebung von Funktionalität aus der vorgesehenen Architektur-Ebene in eine andere, kann nicht selten beobachtet werden. So wird oft Geschäftslogik mitten im Benutzerschnittstellen-Code anzutreffen sein, statt in den dafür ursprünglich gedachten Business-Logik-Modulen. Selbst «moderne» Architektur-Muster weisen nach einer gewissen Entstehungszeit diese versteckte Abhängigkeits-Vernetzung auf [2].

## Aufbrechen von Monolithen

Ohne manuelles Zutun ist das Aufbrechen von solchen, aufgrund der unterschiedlichen Entwurfs-Generationen der Entwickler-Teams entstandenen Gebilde, praktisch nicht machbar.

Dem gegenüber steht die Komplexität und auf der Ebene von Programmen üblicherweise ein immenser Code-Umfang, der eine rein manuelle Umstrukturierung in der Regel nicht zulässt.

Wir verfolgen den Ansatz: «Der Mensch bringt die Intelligenz und Strategie, die Maschine die Geschwindigkeit und Präzision», woraus der folgende Prozess entsteht:

1. Maschinelle Analyse von Abhängigkeiten
2. Manuelles Festlegen von Kategorien/Gewichtung
3. Maschinelle Attributierung der Artefakte
4. Maschinelles Aufteilen mit geeigneten Algorithmen
5. Manuelle Bereinigung, Aufwandabschätzung und definitive Aufspaltung

Die Resultate der maschinellen Analyse [5] werden in einem Graphen-Repository abgelegt.

Danach erfolgt die manuelle Grob- und Detail-Sichtung des Gesamtbildes, um eine Domänen-orientierte Kategorisierung zu machen.

Damit eine Aufteilung in logische Komponenten (Service) sinnvoll gemacht werden kann, sollte man nach den Business-Domänen und deren Grenzen su-

chen. Dies ist bei der umfangreichen Komplexität von Monolithen eine grosse Herausforderung. Man sucht typischerweise in folgenden Bereichen nach den Business-Domänen:

- Benutzer-Schnittstellen (Masken- und Menu-Gruppen)
- Datenbank-Schemas (oft nach der Geschäftslogik gebaut)
- Geschäftslogik-Funktionen

Für eine Verdichtung der eingelesenen Daten ist es wichtig, Kategorien zu definieren und den Artefakten zuzuordnen. Dazu wird, wenn vorhanden, die Architektur-Beschreibung des Systems beigezogen, oder man schaut sich die bestehende interne oder vermutete Modularisierung des Systems an. Als weitere Klassifizierungen können Modul- und Filenamen und Programmier-Konventionen dienen.

Man entwickelt dazu eine Strategie, wie die Kategorien den einzelnen Komponenten zugeordnet werden sollen. Diese Strategie wird in Form von Algorithmen dem maschinellen Prozess übergeben, der diese dann den Knoten zuordnet und Gewichtungen (Wichtigkeiten) zuweist. Diese helfen in einem späteren Teilprozess, die Kohäsionstärke zu ermitteln.

## Aufbrechen mit Graphen-Algorithmen

Mit speziellen Graphen-Algorithmen kann man dann in den annotierten Graphen mögliche Domänen erkennen und hervorheben. Das heisst noch nicht, dass daraus dann automatisch ein einer Service-basierten Architektur entsprechendes System generiert werden kann, aber es hilft bei der Erstellung der «neuen» Architektur. Die folgenden Graphen-Algorithmen [3] helfen bei der Festlegung der Grenzen:

- **Degree Centrality:** hilft für spätere Auswertung durch Gewichten der Zentralität von Komponenten.
- **Closeness Centrality:** dient zum Finden von zentralen Komponenten, die viele eingehende Referenzen haben. Es gibt verschiedene Varianten, je nach Umfang des Graphs.
- **Betweenness Centrality:** dient dem Erkennen von Komponenten, welche auf dem kürzesten Pfad liegen und somit kritisch bei der Herauslösung sind. Dafür ist die Kategorieen-Gewichtung hilfreich.
- **Page Rank:** Spezieller Zentralitäts-Algorithmus für den transitiven oder gerichteten Einfluss einer Komponente.

Die bevorzugte Verwendung dieser Algorithmen war bisher Kommunikations- und soziale Netzwerke zu untersuchen und zu werten; zum Beispiel um zentrale Komponenten in Kommunikations-Netzwerken zu identifizieren und erwartete Konsequenzen bei deren Ausfall zu erkennen. Dieser Ansatz hat uns dazu bewogen, diese Algorithmen für das Aufbrechen

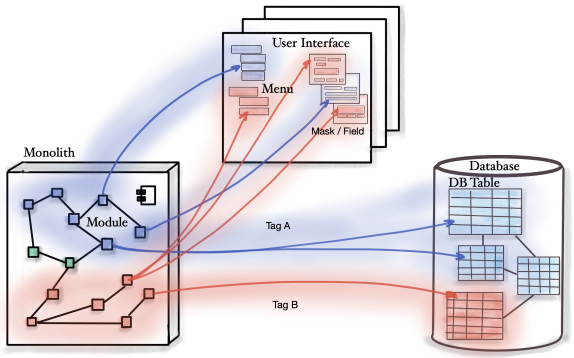


Abbildung 1: Konzeptionelle Kategorisierung

von Monolithen zu verwenden, um enge «Freunde» und «Gruppen von Freunden» herauszuschälen, womit Architektur-Entscheide unterstützt werden können.

Die Herausforderung ist die richtige Attributierung und Gewichtung der aus den eingelesenen Programm-Artefakten gebildeten Knoten zu wählen. Das Verfahren profitiert davon, dass die Implementation dieser Algorithmen bereits als Teil von populären Graphen-Datenbanken (Apache Spark, Neo4J) zur Verfügung steht [3].

## Fazit

Service basierte Architekturen bilden die Grundlage für über längere Zeit betreibbare und wartbare Systeme. Eine Auftrennung von Monolithen in einzelne Komponenten kann bei den vorhandenen Grössen der Code-Basis praktisch weder manuell, noch voll automatisiert gemacht werden. Graphen und dazugehörige Algorithmen unterstützen den manuellen und intellektuellen Prozess der Auftrennung in diesem Hybrid-Ansatz.

## Referenzen

[1] David L. Parnas, *On The Criteria To Be Used In Decomposing Systems Into Modules*. Communications of the ACM, Dezember 1972.

[2] Dave Nicolette, *THE URGE TO STRANGLE,*. https://www.leadingagile.com/2018/10/the-urge-to-stranglethe-strangler-pattern/, 2018.

[3] Mark Needham and Amy E. Hodler, *Graph Algorithms, Practical Examples in Apache Spark and Neo4*. O'Reilly, ISBN 978-1-492-05781-9 2019.

[4] Kai-Uwe Herrmann, Bison Schweiz AG. *Teilautomatisiertes Architektur-Reengineering in einem Java-EE-Monolithen*. WSRE, 2019.

[5] Andres Koch, Remo Koch, Object Engineering GmbH. *Metadaten basiertes, teilautomatisiertes Software-Reengineering*. WSRE, 2018.