8 Teamorganisation verbessern

Um ein Legacy-System in einen besseren Zustand zu versetzen, muss mehr beachtet werden als »nur« die technischen und fachlichen Dimensionen von Domain-Driven Transformation. Häufig lohnt es sich, auch die Teamstrukturen und das Vorgehen zu überdenken. In vielen Organisationen ist agiles Vorgehen bereits gut etabliert, was uns immer wieder sehr freut. Ist das bei Ihnen noch nicht der Fall, ist jetzt der Moment, darüber nachzudenken, denn agile Transition und Domain-Driven Transformation gehen oft Hand in Hand.

In diesem Kapitel liefern wir Ihnen Konzepte und Argumente, wie Sie in Ihrer Organisation Diskussionen über die vorhandenen Teamstrukturen führen und das bei Ihnen sinnvolle Vorgehen für den Umbau Ihres Legacy-Systems einleiten können.

8.1 Software als soziotechnisches System

Nicht nur die Architektur eines Systems spielt eine wichtige Rolle, wenn es darum geht, das System über viele Jahre hinweg nutzbar, wartbar und erweiterbar zu halten. Entscheidend für den Erfolg sind auch die Organisations- und Kommunikationsstrukturen der Teams, die das System bauen und weiterentwickeln. Der Zusammenhang zwischen Teamstrukturen und Architektur wurde erstmals von Mel Conway formuliert:

> »[…] organizations which design systems […] are constrained to produce designs which are copies of the communication structures of these organizations« (s. [Conway 1968]).

Conway hat damit beschrieben, was man in vielen Organisationen beobachten kann: Die Grenzen von Softwaresystemen stimmen häufig mit den Grenzen zwischen Teams überein. Wobei die gelebten Teamgrenzen oft nicht mit den im Organigramm vorgegebenen Grenzen übereinstimmen. Weil dieser Zusammenhang so wichtig ist, wird Software heute nicht mehr nur als technisches System allein, sondern als **soziotechnisches System** betrachtet. Software existiert nicht unabhängig von den sie programmierenden und einsetzenden Menschen, deshalb sind Softwarearchitektur und Teamorganisation untrennbar miteinander verbunden.

Der Umbau einer Legacy-Software hin zu fachlichen Bounded Contexts kann daher nur gelingen, wenn auch Teams nach fachlichen Kriterien strukturiert sind.

Machen wir uns das an einem einfachen Beispiel mit einer Architektur klar, die aus drei Schichten besteht (s. Abb. 8–1): Frontend, Backend und Datenbank. In jeder dieser Schichten gibt es verschiedene Komponenten, in denen das System organisiert ist. Der Einfachheit halber haben wir sie hier mit Buchstaben benannt.

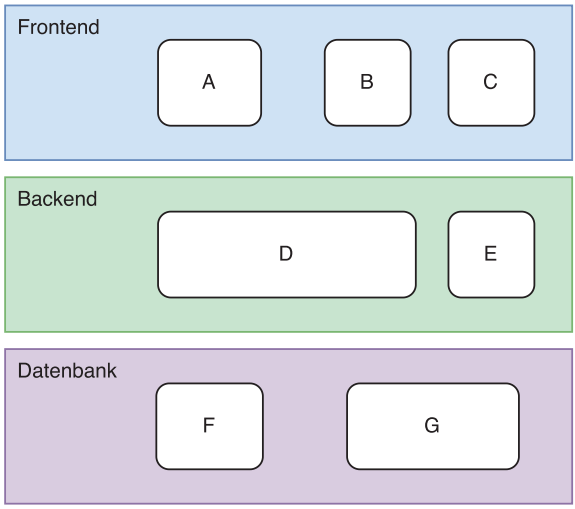

Abb. 8–1 *Schichten mit Komponenten*

8.1.1 Architekturschnitt/Teamorganisation horizontal

Teilt man diese Architektur nach technischen Kriterien auf Teams auf, so erhält man **Schichtenteams** (engl. **Layer Teams**). Der (erhoffte) Vorteil: Alle Expertinnen für ein Thema sind in einem Team versammelt. Hier in unserem Beispiel führt das jeweils zu einem *Frontend*-, *Backend*- und *Datenbank*-Team (s. Abb. 8–2).

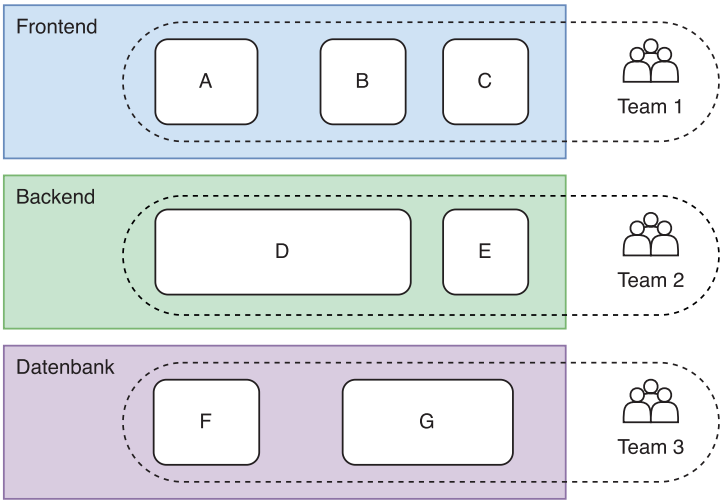

Abb. 8–2 *Führt zu Abhängigkeit: Schichten mit horizontalen Teams*

In so einer technischen Teamaufteilung ist das Frontend-Team davon abhängig, dass das Backend-Team die Frontend-Schnittstelle um benötigte Features erweitert. Das Backend-Team selbst hat auch keine Freiheit und muss seinerseits auf Anpassungen durch das Datenbank-Team warten. Neue Features betreffen in einer solchen Teamstruktur fast immer mehrere Teams, die bei der Umsetzung zeitlich voneinander abhängig sind und viel miteinander kommunizieren müssen, damit die Schnittstellen stimmen.

8.1.1.1 Spezialfall Framework-Team

Noch schwieriger wird diese Art der Teamorganisation in großen Unternehmen, in denen es häufig nicht nur Frontend-, Backend- und Datenbank-Teams gibt, sondern zusätzlich noch ein oder mehrere Framework-Teams, die für mehrere Softwareentwicklungen Basis-Features zur Verfügung stellen. Wenn eine Organisation eine solche Teamaufteilung gewählt hat, so sind Frontend-, Backend- und Datenbank-Team, die gemeinsam eine Applikation bauen, nicht nur voneinander, sondern auch vom Framework-Team abhängig.

Baut das Framework-Team nur technische Basisdienste, mag die Behinderung für die anderen Teams durch Wartezeiten auf neue technische Funktionalität nicht so sehr ins Gewicht fallen. Frontend-, Backend- und Datenbank-Team können trotz Verzögerung bei den technischen Basisdiensten einfach auf Grundlage der alten technischen Dienste weitere Features für die Anwenderinnen implementieren. Der Treiber für technische Änderungen ist in der Regel das Framework-Team selbst, das neue Technologien einführen will oder Sicherheits- bzw. Performance-Anforderungen umsetzen muss.

Gibt es in einer Organisation aber Framework-Teams, die fachliche Basisklassen und Dienste zur Verfügung stellen, z. B. die überall verwendete Kunden-Klasse, dann schlagen die Verzögerungen und die inhaltlichen Missverständnisse direkt auf die Teams durch, die neue Features für die Anwenderinnen bauen. Im Gegensatz zu den technischen Neuerungen erwachsen die fachlichen Neuerungen üblicherweise aus Feature-

Wünschen der Anwenderinnen. Das heißt, Frontend-, Backend- und Datenbank-Team treten mit einem Wunsch nach fachlicher Erweiterung und Veränderung an das Framework-Team für fachliche Basisdienste heran. Hier müssen einerseits die fachlichen Wünsche vermittelt werden und andererseits müssen Frontend-, Backend- und Datenbank-Team mit den zeitlichen Plänen des Framework-Teams klarkommen. Im Gegensatz zur Erweiterung der technischen Basisdienste sind in diesem Fall der fachlichen Basisdienste häufig die Teams die Treiber, die die Features für die Anwenderinnen bauen, also Frontend-, Backend- und Datenbank-Team. Einfach auf den vorhandenen technischen Basisdiensten aufzusetzen, ohne auf die Neuerungen zu warten, ist im Fall von fachlichen Erweiterungen häufig nicht möglich und verschärft die Abhängigkeit vom Framework-Team noch mehr.

8.1.1.2 Probleme mit dem horizontalen Schnitt

Mit so einer horizontalen Aufteilung hat man einerseits Teams, die ständig auf Lieferungen von anderen Teams warten, von denen sie abhängig sind. Andererseits fangen die Teams – Conway's Law folgend – ganz automatisch an, die Komponenten, an denen sie arbeiten, zu vereinheitlichen und zusammenzuführen.

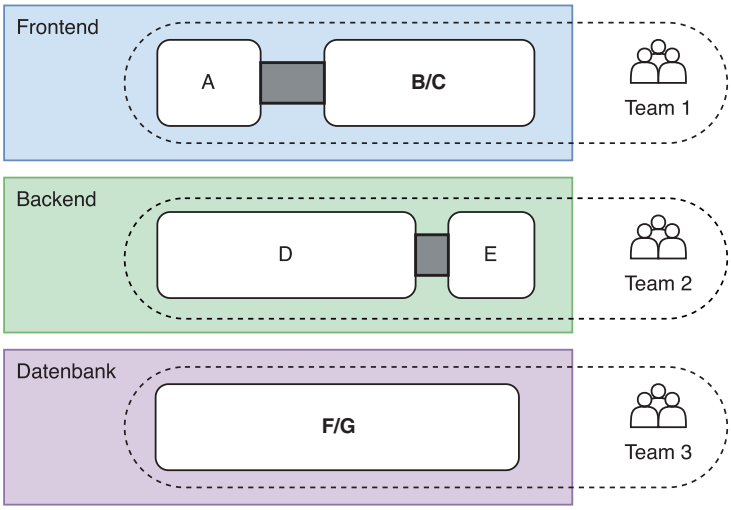

Abb. 8–3 *Problematisch: Schichten mit zusammenwachsenden Komponenten bei horizontal organisierten Teams*

In Abbildung 8–3 ist angedeutet, was Teams üblicherweise tun:

- Sie finden Gemeinsamkeiten in den von ihnen bearbeiteten Komponenten und legen sie komplett zusammen, wie B + C sowie F + G.
- Außerdem schaffen sie in ihren Komponenten gemeinsame Frameworks und Bibliotheken, die die einzelnen Teile weiter miteinander verbinden und aneinanderkoppeln, wie die Verbindung zwischen A und B/C sowie zwischen D und E.

Will man diese Effekte vermeiden, muss man seine Teams vertikal organisieren.

8.1.2 Architekturschnitt/Teamorganisation vertikal

In Abschnitt 2.3.1 im Grundlagenkapitel zu Domain-Driven Design haben wir erfahren, dass ein Bounded Context vertikal geschnitten sein sollte – also quer zu der technischen Schichtung. Folgt man der Logik von Conway, dass Architektur die Teamstruktur abbildet, dann müssen auch die Teams, die Bounded Contexts betreuen, vertikal geschnitten sein.

Dann brauchen wir in jedem Team Mitglieder mit Know-how aus allen Schichten und nicht »nur« z.B. Backend-Entwicklerinnen oder Frontend-Entwicklerinnen. In der agilen Sprechweise sagt man dazu, dass die Teams *cross-funktional* zusammengesetzt sein sollen.

Eine vertikale Aufteilung von Teams nach fachlichen Kriterien macht es möglich, dass ein Team für einen Bounded Context in der Software zuständig ist, der sich durch alle technischen Schichten von der Oberfläche bis zur Datenbank und den Frameworks zieht (s. Abb. 8–4).

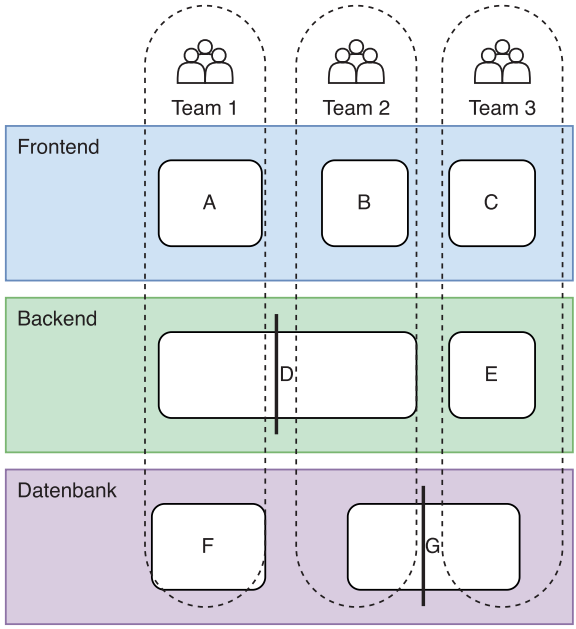

Abb. 8–4 *Macht Unabhängigkeit möglich: Schichten mit cross-funktionalen Teams*

Damit Team 1 und Team 2 wirklich getrennt entwickeln können, muss in unserer Architektur das Modul D in der Backend-Schicht in zwei getrennte Komponenten aufgeteilt werden. Auch in der Datenbank müssen die Datentabellen, die zusammen G ausmachen, aufgeteilt werden. Wie das geht, werden wir in den folgenden Kapiteln zu strategischer Domain-Driven Transformation erfahren.

8.1.3 Wie führt man diese Teamreorganisation durch?

Organisatorische Domain-Driven Transformation bedeutet oft, horizontal organisierte Teams (s. Abb. 8–2) zu vertikal organisierten Teams (s. Abb. 8–4) weiterzuentwickeln. Um den Teams einen Anhaltspunkt zu geben, wie die geplanten Veränderungen ablaufen werden, haben wir die folgenden soziotechnischen Refactorings[1] beschrieben:

- *Form Cross-Functional Team out of Layer-Team Members*[2]
- *Form Second Cross-Functional Team out of Partly Layer-Team and First-Team Members*[3]
- *Form Second Team out of Only Layer-Team Members*[4]
- *Assign Context to Existing Team*[5]

Organisatorische Domain-Driven Transformation geht oft mit strategischer Domain-Driven Transformation Hand in Hand. Für ein Beispiel verweisen wir daher auf Abschnitt 10.5.

In den meisten Organisationen ist diese einfache Sicht auf Architektur und Teamorganisation ein guter Ausgangspunkt bzw. ein gutes erstes Erklärmodell, um den Anstoß für Diskussionen über die Organisation zu geben. Um einen nachhaltigen Wandel in den Teamstrukturen zu erreichen, braucht es allerdings mehr. Das Buch von Matthew Skelton und Manuel Pais mit dem Titel *Team Topologies* (s. [Skelton & Pais 2019]) hat an dieser Stelle einen weit beachteten Beitrag geleistet. Im nächsten Abschnitt stellen wir den Ansatz vor und verknüpfen ihn mit DDD, sodass Sie beide Ansätze für die Veränderungen in Ihrem Unternehmen in Kombination verwenden können.

8.2 Team Topologies

Nach Skelton und Pais sollte es das Ziel einer Organisation sein, schlagkräftige und effiziente Teams zu haben, um möglichst viel Mehrwert für die Organisation zu erwirtschaften. Teams sind dann schlagkräftig und effizient, wenn sie bei der Softwareentwicklung in den *Flow* kommen und so neue Ideen und Features schnell in die Umsetzung und in Produktion bringen. Genau das, was sich jede Anwenderin wünscht: Eine neue Idee wird vom Team aufgenommen und ist in kurzer Zeit im Produkt zu finden.

---

1. Wir verwenden hier den Begriff »Refactoring« in einem sehr weiten Sinne. Alternative Begriffe wären: »Reorganisation« oder »soziotechnische Transformation«.
2. *https://hschwentner.io/domain-driven-refactorings/socio-technical/form-cross-functional-team-out-of-layer-team-members*.
3. *https://hschwentner.io/domain-driven-refactorings/socio-technical/form-second-team-out-of-partly-layer-team-and-first-team-members*.
4. *https://hschwentner.io/domain-driven-refactorings/socio-technical/form-second-team-out-of-layer-team-only*.
5. *https://hschwentner.io/domain-driven-refactorings/socio-technical/assign-context-to-existing-team*.

8.2.1 Arbeiten im Flow

Für dieses Arbeiten im Flow nennen Skelton und Pais zwei wichtige Voraussetzungen:

1. Man muss die **kognitive Last** für die Teammitglieder im Griff behalten, sodass sie sich auf ihre Aufgabe und deren zügige Umsetzung konzentrieren können. Ziel ist es, dass alle möglichst viel ihrer gedanklichen Energie darauf verwenden können, einen Wert für die Kundinnen zu schaffen – und ihre Energie nicht auf Nebenschauplätzen, wie Abstimmungen mit Abteilungen oder akzidentelle Komplexität (s. Abschnitt 1.3) aller Art verschwenden.
2. Es ist wichtig, den Teammitgliedern zur größtmöglichen **Autonomie** bei ihrer Arbeit zu verhelfen, indem man die Abhängigkeiten nach außen reduziert und lokale Entscheidungen ermöglicht. Bei hoher Autonomie können sie ihre Aufgaben in ihrem eigenen Rhythmus und zu dem Zeitpunkt, an dem sie Zeit haben, umsetzen, ohne durch Abhängigkeiten gebremst zu werden.

Zur Bewältigung der kognitiven Last und zur Schaffung von Autonomie müssen im Team-Topologies-Ansatz folgende Basisvoraussetzungen erfüllt werden:

- **Teamgröße**
  Ein Team sollte nicht mehr als fünf bis neun Personen umfassen, damit der Kommunikationsbedarf zwischen den Teammitgliedern nicht überhandnimmt. In großen Teams wird die kognitive Last durch mehr Kommunikation erhöht und auch dadurch, dass sie größere Stücke Software gemeinsam bearbeiten.
- Teams sollten **langfristig** zusammenarbeiten, um effiziente Arbeitsabläufe entwickeln und optimieren zu können. Arbeitet ein Team langfristig zusammen, so reduziert das die kognitive Last, weil man sich gut kennt und nicht immer wieder den Forming-Storming-Norming-Performing-Zyklus[6] durchlaufen muss – und hoffentlich auch regelmäßig die Schmerzpunkte in Retrospektiven ausräumt.
- Jedes Team sollte klare **Zuständigkeiten** und **Grenzen** für seine Arbeit haben, damit die Aufgaben und das Ziel für alle Teammitglieder verständlich und verfolgbar bleiben. Das schafft Autonomie und verringert die kognitive Last, weil weniger über unklare Randbedingungen nachgedacht werden muss und Entscheidungen schneller getroffen werden können.

8.2.2 Arten von Topologien

Aufbauend auf diesen Grundlagen beschreiben Skelton und Pais vier verschiedene Arten von Topologien:

- **Stream-aligned Team**
  Diese Teams schaffen den Wert für die Kundin entlang des Value Stream. Sie sind cross-funktional zusammengesetzt, damit sie mit diesem Kompetenzmix schnell

6. Auch »Tuckman's stages of group development« genannt und zuerst beschrieben in [Tuckman 1965].

signifikante Inkremente liefern können. Schnell geht es allerdings nur, wenn diese Teams arbeiten können, ohne auf andere Teams warten zu müssen.

Diese Topologie sollte in einem Unternehmen am häufigsten anzutreffen sein. Im Zusammenhang mit DDD würde man ein solches Team an einem Bounded Context (oder mehreren) exklusiv arbeiten lassen, da so die Zuständigkeiten und Grenzen sehr klar bestimmt werden können.

Im Programmkino-Beispiel hatten wir in Abbildung 2–9 eine Context Map entwickelt. Unser Zielbild für die Arbeit an den verschiedenen Kontexten ist, dass wir, wie von *Team Topologies* vorgeschlagen, Stream-aligned Teams für die Kontexte haben. Die Softwareentwicklung für unser Kinosystem hatte bisher eine Teamorganisation mit Frontend-, Backend- und Datenbank-Team. In einem teamübergreifenden Workshop diskutieren wir mit den Entwicklerinnen und Architektinnen über eine neue Teamzusammensetzung mit cross-funktionalen Teams. Wir können vier cross-funktionale Teams bilden und bitten die Teams, sich selbst Namen zu geben. Schließlich diskutieren wir, welches Team am besten zu welchem fachlichen Kontext passt. Abbildung 8–5 zeigt das Ergebnis.

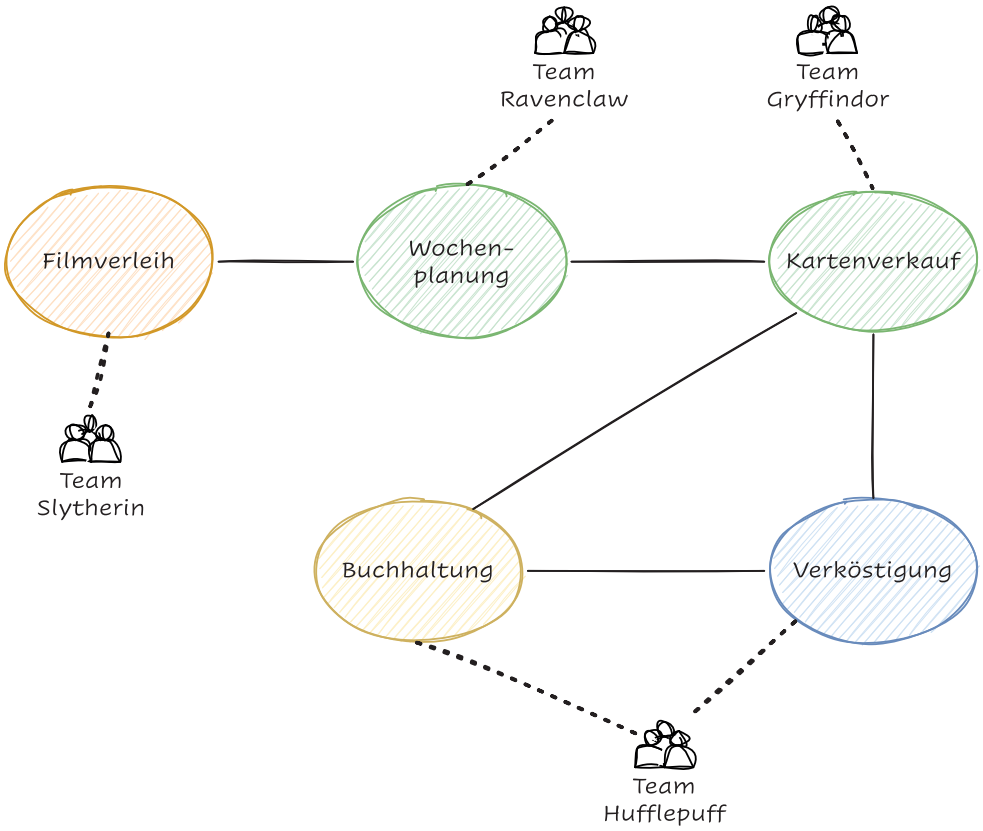

Abb. 8–5 *Stream-aligned Teams in der Context Map für das Kino*

Die beiden Kontexte »Kartenverkauf« und »Wochenplanung« haben wir in Abschnitt 2.2.3 als Core Domains identifiziert und in diesen beiden Kontexten wird es beim Umbau von CineSmall viel Arbeit zu erledigen geben. Deshalb sehen wir für diese beiden Kontexte jeweils ein eigenes Team vor. Die Supporting Domain »Verköstigung« und die Generic Domain »Buchhaltung« übergeben wir dem dritten Team. Dem externen Team für den »Filmverleih« geben wir ebenfalls einen Namen, auch wenn diese Entwicklerinnen nicht zu unserer Organisation gehören.

- **Platform Team**
  Ein solches Team stellt eine zugrunde liegende Plattform zur Verfügung, um die Stream-aligned Teams zu unterstützen. Entscheidend dabei ist, dass die Plattform Services anbietet, die die Komplexität der darunterliegenden Technologie oder Fachlichkeit vereinfacht und so die kognitive Last der Stream-aligned Teams verringern, um es ihnen zu ermöglichen, schnell zu arbeiten.

  Ziel eines Platform Team sollte sein, eine Plattform zu schaffen, die »gerade gut genug« ist. Man sagt auch, es solle die »Thinnest viable platform (TVP)« werden, also nicht nur eine dünne Plattform, sondern die dünnste Plattform, die möglich ist.

CineSmall ist aktuell noch ein Monolith und wir haben keine zugrunde liegende Plattform, die wir von einem eigenen Team bearbeiten lassen können. In Abschnitt 8.2.5 werden wir sehen, dass erst durch die Zerlegung von CineSmall ein Platform Team entstehen wird.

- **Complicated Subsystem Team**
  Diese Teams haben einen speziellen Auftrag für ein Teilsystem, das zu kompliziert ist, um von einem normalen Stream-aligned Team oder einem Platform Team bearbeitet zu werden.

  Wichtig ist, dass diese Teams bei Bedarf gebildet und nach kurzer Zeit wieder aufgelöst werden, denn auch hier ist das Ziel, die Stream-aligned Teams kognitiv zu entlasten. Diese Teams arbeiten an einem Teil eines Bounded Context, der später in den Verantwortungsbereich des Stream-aligned Team integriert wird, das für den Bounded Context zuständig ist.

Bei dem Bankbeispiel wird ein KI-Modul für die Bewertung der Kundenbonität beim Abschluss von Kreditverträgen eingeführt. Um dieses KI-Modul aufzusetzen, ist es sinnvoll, KI-Expertinnen für einen begrenzten Zeitraum in einem Team zusammenzuziehen, um diese Aufgabe zu lösen.