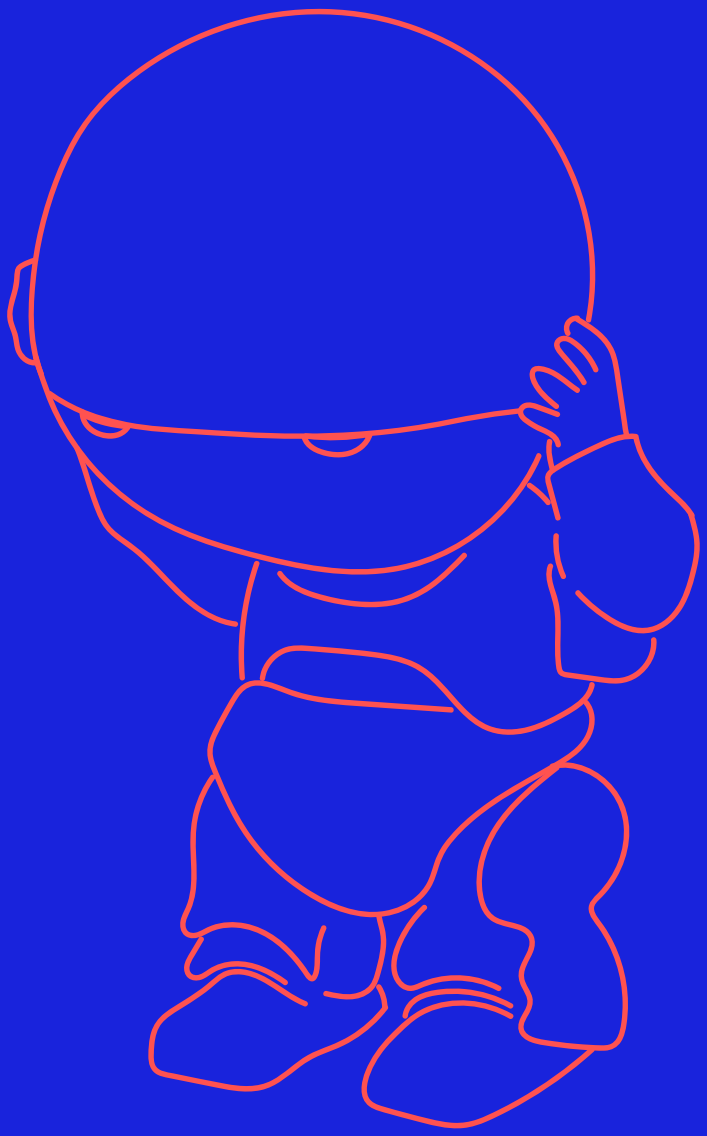


Team Cognitive Load

Der Arbeitsspeicher des Teams

Wer schon in einem Entwicklungsteam gearbeitet hat, der kennt das bestimmt: Schattenboxen mit Problemen, die konkrete Schmerzen bereiten, aber deren Ursache schwer greifbar ist. “Ich komm zu nix!”, “Warum wurde dieser Service schon seit Jahren nicht geupdatet?”, “Bei dem Thema sind wir jetzt der Flaschenhals”, “Diesen Sprint haben wir wieder nicht viel geschafft, es waren zu viele Ablenkungen”. Die schwer greifbare Ursache ist nicht selten die zu hohe kognitive Last - Cognitive Load - des Teams. Aber was ist das genau? Woher kommt das? Und was macht man dagegen? Das wollen wir uns in diesem Artikel mal genauer anschauen.

Ein Artikel von Arnold Franke

Arnold Franke (franke@synyx.de) wirkt als Entwickler und Berater für die Kunden der synyx GmbH & Co KG in Karlsruhe. Dabei baut er als Komplexitätsvermeider und Software Craftsman nachhaltige, pragmatische Lösungen mit sauberem Code.

Vereinfacht gesagt besteht die Cognitive Load eines Teams aus allen Dingen, die die Teammitglieder im Kopf haben müssen, um die Aufgabe des Teams zu erfüllen. Welche Services entwickelt das Team? Welche Programmiersprachen muss es dafür beherrschen? Wie funktionieren die dabei eingesetzten Technologien? Wie finden Deployment und Betrieb statt? Wie analysiert man Fehler? Welche Kommunikation mit anderen Teams und Stakeholdern ist nötig?

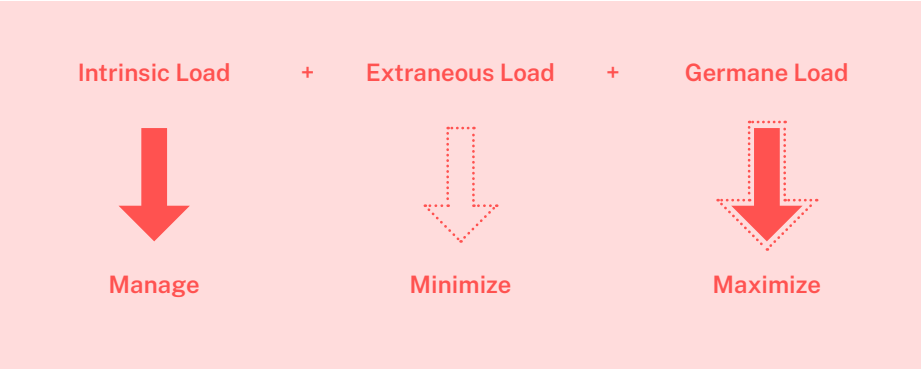


Abb. 1: Arten von Cognitive Load

Was ist Cognitive Load?

Das Problem ist, dass diese Liste nicht beliebig lang werden kann, denn die kognitive Kapazität des menschlichen Gehirns – und damit auch des Teams – ist begrenzt. Wird diese Grenze innerhalb eines Teams überschritten, kann das unerfreuliche Folgen haben. Wichtige Aufgaben werden vernachlässigt, die Entwicklungsgeschwindigkeit verlangsamt sich, Innovation kommt zum Erliegen, die Qualität leidet und die Motivation sinkt.
Diese Erkenntnis geht auf die Cognitive Load Theory des Psychologen John Sweller aus den 80er-Jahren zurück [Swe88]. Sie besagt, dass die Kapazität des Arbeitsgedächtnisses begrenzt ist und nur eine bestimmte Menge an Informationen aufrecht erhalten werden kann. Sie unterscheidet drei Arten von Cognitive Load (siehe Abbildung 1), die man gut auf die Arbeit eines Entwicklungsteams übertragen kann [Swe98]:

- Intrinsische (intrinsic) Cognitive Load betrifft direkt die Aufgaben, die ein Team lösen muss. „Wie lese ich ein Objekt aus der Datenbank?" „Wie schreibe ich einen Unittest?" „Wie funktioniert ein switch Statement in Java?". Diese Art der Cognitive Load gehört zur täglichen Arbeit und lässt sich daher nicht eliminieren. Man soltte aber darauf hinarbeiten, sie zu beherrschen und in ein leicht verwaltbares Maß zu optimieren.

- Extrinsische (extraneous) Cognitive Load betrifft die Umgebung, in der man die Aufgaben löst und die externen Voraussetzungen, die dafür benötigt werden. „Wie nehme ich ein neues Feature produktiv?" „Wo finde ich die Logs von Service X?". Man sollte als Ziel haben, die extrinsische Cognitive Load weitgehend zu eliminieren, zu Beispiel durch Automatisierung.

- Lernbezogene (germane) Cognitive Load betrifft das Lernen neuer Dinge und den Aufbau neuer Schemata im Gehirn. „Mit welcher Technologie lösen wir dieses neuartige Problem?" „Wie integrieren wir Service A mit Service B?" Für diese Art von Cognitive Load möchte man Platz schaffen, denn sie ermöglicht Innovation. Damit ist nicht gemeint, auf Teufel komm raus neue Technologien einzusetzen oder das Rad neu zu erfinden, sondern unter ständiger Verbesserung Mehrwert zu schaffen.

Das Thema ist im Bewusstsein der Developer Community stark in den Vordergrund gerückt. So ist es ein Hauptthema in Manuel Pais' und Mathew Skeltons wegweisendem Buch „Team Topologies" [SkePai19] und das Stichwort „Team Cognitive Load" hat es im Thoughtworks Technology Radar [Thought] in die Kategorie „Trial" geschafft.

Welche Probleme entstehen durch Cognitive Load?

Durch zu hohe Cognitive Load können in einem Team unterschiedliche Engpässe entstehen (siehe Abbildung 2). Ist die technische Infrastruktur eines Systems schwer beherrschbar und nicht automatisiert, dann verschwendet das Team viel extrinsische Cognitive Load darauf, anstatt die Software selbst weiter zu entwickeln. Kommen in der Software viele unterschiedliche Technologien und keine einheitlichen Patterns zum Einsatz, dann nimmt die intrinsische Cognitive Load unnötig viel Kapazität ein. Wenn die lernbezogene Cognitive Load dadurch zu kurz

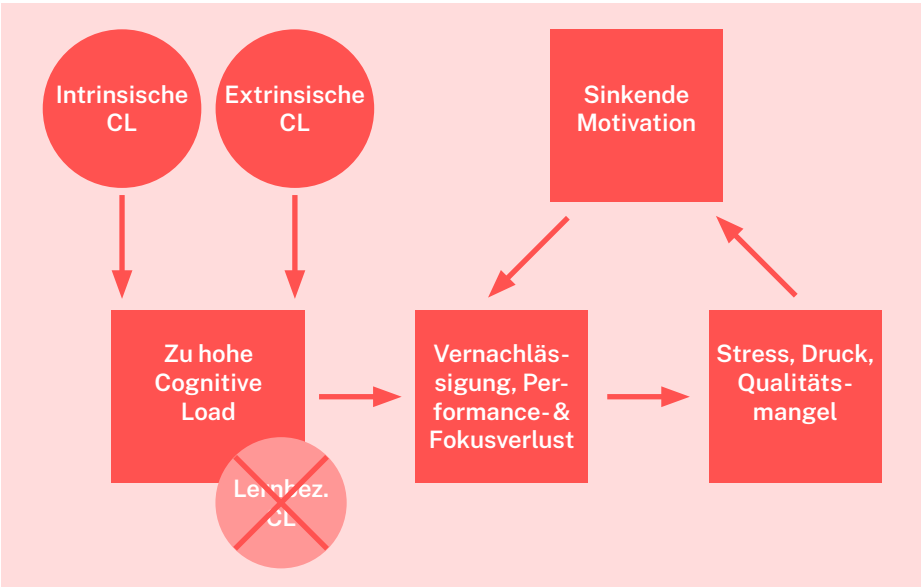


Abb. 2: Folgen hoher Cognitive Load

kommt, dann dauern Neuentwicklungen und Erweiterungen deutlich länger und die Innovationskraft sinkt.

Mit dem Mangel an kognitiver Kapazität einhergehende Probleme sind Vernachlässigung wichtiger Aufgaben, häufiger Fokuswechsel und das Entstehen einer Stressatmosphäre, was sich nicht nur auf die Qualität negativ auswirkt. Steigender Druck schränkt die Autonomie des Teams ein, zu wählen wann und wie es an welchen Aufgaben arbeitet. Die Sinnhaftigkeit der Arbeit geht verloren, wenn man nur noch hinterher hechelt, anstatt gezielt Mehrwert zu schaffen. Damit bröckeln die Grundlagen der intrinsischen Motivation, Autonomy, Mastery und Purpose (nach Daniel Pink [Pink09]), was sich in sinkender Motivation der Teammitglieder niederschlägt. Das wiederum verstärkt die negativen Effekte und führt zu einer Abwärtsspirale.
Ich habe viele dieser Effekte an echten Teams live beobachten können. Da gab es zum Beispiel das performante Feature-Team, das eine große, komplexe Domain mit vielen Services betreute. Es lieferte dennoch schnell und in hoher Qualität Ergebnisse und erweiterte laufend die Komplexität seiner technischen Landschaft. Allerdings wurden bereits produktive Services selten aktualisiert und Modernisierungsmaßnahmen oft verschoben. Als eine Infrastrukturmigration anstand, war das Team auf einmal zu so vielen Modernisierungs-und Migrationsaufwänden gezwungen, dass sich das Verhältnis von Weiterentwicklungs- zu Betriebsaufwänden innerhalb kurzer Zeit von 80/20 auf 20/80 umkehrte und über Jahre so stecken blieb. Das Vorzeigeteam wurde quasi über Nacht zum Flaschenhals. Teammitglieder wurden unzufrieden und wanderten ab, wichtige Themen mussten auf lange Zeit verschoben oder ganz verworfen werden. Die hohe Cognitive Load des Teams konnte also lange Zeit beherrscht werden, wurde ihm am Ende aber zum Verhängnis

Wie erkennt man zu hohe Cognitive Last?

Leider ist es schwer, rechtzeitig zu erkennen, wann die kognitive Kapazität eines Teams überschritten ist. Einfache Kennzahlen wie Lines of Code, Anzahl der Services oder Ähnliches sind dafür nicht geeignet. Man kann stattdessen auf die typischen Anzeichen achten wie geringe Performance, lange Reaktionszeiten, Vernachlässigung wichtiger Aufgaben oder der typische „Ich komm zu nix"-Effekt. Empirische Methoden zur Quantifizierung der Cognitive Load sind schwer zu bestimmen. In Team Topologies [Ske-Pai19] werden zwei methodische Ansätze vorgeschlagen.

Der erste Ansatz ist die Ermittlung der Domain Complexity. Man kann bestimmen, für wie viele Domänen ein Team verantwortlich ist und deren relative Komplexität bewerten. Wenn ein Team die Verantwortung für mehr als eine komplexe (große und unübersichtliche) beziehungsweisekomplizierte (erhebliches Spezialwissen erfordernde) Domäne trägt, dann ist die Gefahr für zu hohe Cognitive Load gegeben. Die Faustregel ist: Nicht mehr als eine komplexe/komplizierte Domäne oder bis zu drei einfache Domänen pro Team. Dieser Ansatz liefert aber keine absoluten Erkenntnisse, da sowohl Domänen als auch die kognitive Kapazität von Teams variabel sind.

Der zweite Ansatz besteht aus der Befragung des Teams, um eine aktive Selbsteinschätzung zu erhalten. Man kann das Team bitten, zu quantifizieren, ob es seinen Aufgaben noch effektiv und in vernünftiger Zeit nachkommen kann. Hierfür schlagen die Autoren einen Fragebogen vor, auf dem die Teammitglieder ihre Erfahrungen bezüglich verschiedener Teile ihrer Arbeit wie Entwicklung, Testing, Deployment, Betrieb, Support bewerten können – unter Berücksichtigung von Detailaspekten wie zum Beispiel „Wie leicht sind Fehler zu analysieren?", „Wie einfach sind Deployments durchzuführen?" oder „Wie stressig sind nächtliche Produktionsprobleme?". Einen beispielhaften Fragebogen haben die Autoren neben vielen anderen Ressourcen auf GitHub (https://github.com/TeamTopologies/Team-Cognitive-Load-Assessment) veröffentlicht.
Letztlich liegt es vor allem in der Verantwortung des Teams, derartige Probleme eigenständig feststellen und Alarm zu schlagen und vor allem in der Verantwortung der Organisation, die Probleme anzugehen. In einer Organisation mit hoher Eigenverantwortung und einer gesunden Fehlerkultur sollte dieser Weg dorthin kein Problem sein – schließlich liegt es im Interesse aller, dass ein Team effizient und motiviert arbeiten kann.

Was kann man als Team tun?

Einen Großteil der Faktoren, die zur Erhöhung der Cognitive Load führen, haben die Teammitglieder selbst in der Hand. Sie bauen die Software und entscheiden über die Komplexität der Lösungen. Die Wissensverteilung und der Entwicklungsprozess liegen unter dem Einfluss des Teams. Auch wie ein Team kommuniziert, kann es selbst entscheiden. Durch Eigeninitiative und gewissenhafte Arbeit kann es vielen Problemen vorbeugen.

Komplexitätsmanagement

Cognitive Load entsteht unter anderem durch Komplexität, die man im Kopf behalten muss, zum Beispiel komplexes Softwaredesign. Je mehr Abstraktionsebenen, Sonderlocken, Abhängigkeiten und verschiedene Technologien ein Team kennen muss, desto voller sind die Köpfe. Das Tolle ist: Die Komplexität der Software hat ein Team größtenteils in der eigenen Hand!
Ich erinnere mich an ein Beispiel aus einem meiner letzten Teams. Es sollte eine schlanke Webapp gebaut werden, um manuelle Excel/E-Mail-Prozesse abzulösen. Gelegentlich kommt es vor, dass zwei User die Anwendung gleichzeitig im Browser offen haben und Daten bearbeiten. Das Entwicklerteam wollte dafür eine coole Lösung bauen und entschied sich für Echtzeitupdates via Websockets auf alle Clients bei jeder Änderung.Die Umsetzung gelang und das Feedback der User war gut.
Leider hatte man im ersten Schritt nicht berücksichtigt, dass man die Websockets auch noch zwischen allen Backend-Instanzen synchronisieren muss. Also hat das Team noch einen Verteilmechanismus über ein Kafka-Topic ergänzt. Eine zweite Webapp interessierte sich auch für die Daten, und damit der User ein durchgängiges Echtzeit-Erlebnis hatte, wurde die Websocket-Lösung für die Kommunikation dorthin erweitert. Große Echtzeit-Abhängigkeitsgraphen entstanden.
Nach ein paar Monaten Produktionsbetrieb wurde von einem Infrastrukturteam eine Komponente ausgetauscht, die plötzlich das Websocket-Protokoll nicht mehr unterstützte. Daraufhin musste das Team alles auf HTTP Long Polling umbauen, um

weiter das Websocket-Verhalten zu emulieren. Während der ganzen Zeit war der immer komplexere Websocket-Mechanismus ein Faktor, den das Team bei jeder Weiterentwicklung berücksichtigen musste, und zugleich eine häufige Fehlerquelle. Jeder Entwickler musste den ganzen Mechanismus verstanden und im Kopf haben, um die Software weiterzuentwickeln.
Die Moral von der Geschicht': Das Team hat sich hier durch eine technische Entscheidung ohne Not viel Komplexität und damit Intrinsische Cognitive Load aufgebürdet. Diese Entscheidung war nicht zwingend schlecht, denn das Feedback der User war durchgehend positiv. Je nach Nutzungsszenario wären die User aber mit einem einfachen Reload/Locking-Mechanismus ebenso zufrieden gewesen und man hätte viel Komplexität gespart. Diesen Trade-off sollte ein Team stets bei der Lösungsfindung berücksichtigen und im Zweifelsfall lieber erst mal den Feldweg bauen anstatt sofort die Autobahn.

Das Gleiche gilt auf allen Ebenen von der Systemlandschaft bis zum Code. Jedes Kästchen und jeder Pfeil, den man auf dem Systemdiagramm sparen kann, ist ein Stück freie kognitive Kapazität. Jede vermiedene Sonderlocke, jede Abstraktionsebene, die man einspart, und jedes „if", das man weglassen kann, ist Code, den niemand mehr lesen und verstehen muss. Teams, die nach Einfachheit streben, halten sich den Kopf frei für die Lernbezogene Cognitive Load und damit Innovation und Effizienz.
Nicht nur Einfachheit, auch Struktur hat einen großen Einfluss auf die Cognitive Load. Carola Lilienthal hat in ihrem Buch „Langlebige Software-Architekturen" [Lil20] analysiert, welche Erkenntnisse der kognitiven Psychologie sich auf Methoden in der Softwareentwicklung übertragen lassen. Ganz oben stehen dabei die Prinzipien der Modularisierung, wie man sie in einem sauber geschnittenen Modulithen [Fra20] oder einer gesunden Microservice-Landschaft wiederfindet. Separation of Concerns, Single Responsibility, geringe Kopplung, hohe Kohäsion, explizite, minimale Schnittstellen folgen dem psychologischen Prinzip des „Chunking", das unserem Gehirn durch sinnvolle Aufteilung der Informationen deren Verarbeitung erleichtert.
Als weiteren Punkt nennt Carola Lilienthal die Nutzung der Mustererkennung unseres Gehirns. Durch die Verwendung geläufiger Patterns fällt es Entwicklern leichter, eine Software zu verstehen, und sie müssen weniger Kapazität für das Durchdringen unüblicher Lösungen verwenden.
Zu guter Letzt kommt unser Gehirn sehr gut mit hierarchischen Strukturen klar, weshalb es so wichtig ist, auf unidirektionale, zyklenfreie Abhängigkeiten in der Software zu achten (siehe Abbildung 3). All das hat das Entwicklerteam selbst in der Hand und damit auch die Möglichkeit, seine Cognitive Load bewusst und nachhaltig zu reduzieren.

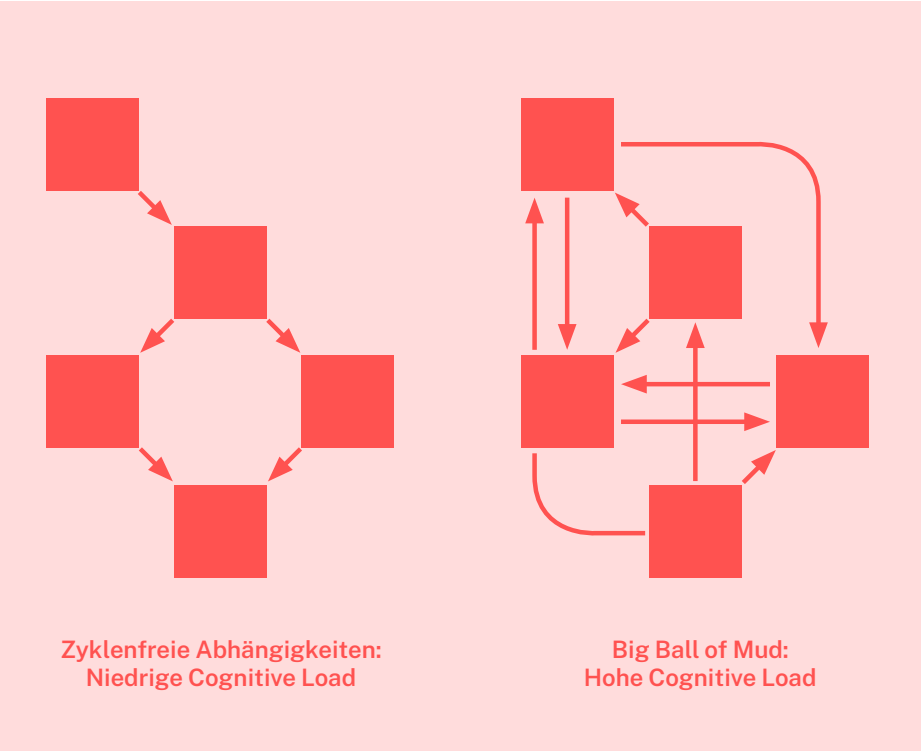


Abb. 3: Cognitive Load durch Abhängigkeiten

Cognitive Load bewerten

- Typische Anzeichen erkennen
- Bewertung anhand relativer Domänenkomplexität
- Selbsteinschätzung durch Befragung des Teams
- Eigenverantwortliche Selbsteinschätzung des Teams

Kasten 1

Team Interface

Letztlich kann das Team durch bewusste Kommunikation Cognitive Load vermeiden. Welche Kommunikation ist wertvoll und welche ist eher hinderlich? Wie kommuniziert das Team mit anderen Teams und Stakeholdern? Team Topologies nennt als Methode, um die Kommunikation zu klarifizieren, die Definition eines „Team Interface" [SkePai19]. Dies ist die Menge aller Kommunikationskanäle eines Teams. Dazu gehören schon einfache Dinge wie Versionsnummern. Durch Semantic Versioning kommuniziert ein Team beispielsweise klar und ohne Aufwand, ob ein Stück Software abwärtskompatibel ist oder nicht. Durch gut gepflegte öffentliche Dokumentation kann es Klärungsaufwand verringern. Durch Ticketsysteme kann es Anforderungen kanalisieren und kommunizieren, woran es arbeitet. Ein öffentlicher Chatroom lädt zu spontanen Nachfragen ein. Andere Teams könnten durch einen Pull-Request-Prozess Änderungen am Code einkippen.
Klare Rollen bündeln Kommunikation auf bestimmte Ansprechpartner. So kann zum Beispiel ein dedizierter Product Owner die Kommunikation mit Stakeholdern auf sich bündeln, um dem restlichen Team Fokus zu verschaffen. Bei manchen Teams ist temporäre enge Zusammenarbeit und Pair Programming mit anderen Teams ein wichtiger Teil der Kommunikation.
Das Team sollte selbst bestimmen, welche Kommunikationskanäle zu seinem Team Interface gehören und – ebenso wichtig – welche nicht dazu gehören. Denn ent-

gegen weitverbreiteter Ansicht gibt es auch so etwas wie zu viel Kommunikation. Jede Anfrage auf jedem Kanal sofort beantworten, kann die kognitive Kapazität stark belasten. Ständig Support leisten zu müssen, ist anstrengender, als einen leicht verständlichen Self-Service bereitzustellen. Ziel des Team Interface sollte möglichst hohe Einfachheit und Effizienz der nötigen Kommunikation bei möglichst geringem Fokusverlust sein, ohne essenzielle Kommunikation zu verhindern.

Was kann man als Organisation tun?

Als Entscheider in einer Organisation hat man ganz andere Möglichkeiten, die Cognitive Load von Teams zu beeinflussen. Man kann Teams zusammenzustellen und verändern und kann ihnen Verantwortlichkeiten übertragen. Das sollte man nutzen, um die oben bereits erwähnte Regel sicherzustellen: Kein Team sollte für mehr als eine komplexe beziehungsweise komplizierte Domäne verantwortlich sein (siehe Kasten 2). Klar definierte Domänengrenzen sorgen dabei für Fokus auf das Wesentliche und verhindern, dass die Cognitive Load durch das Hinzunehmen von nicht zugehörigen Dingen steigt. Überhaupt ist Klarheit eine der förderlichsten Hilfen, die man als Entscheider beisteuern kann. Klarheit sorgt für Fokus und damit für geringe Cognitive Load. Zum Beispiel Klarheit über die Rollen verschiedener Teams und die Beziehungen beziehungsweise die angestrebte Art der Interaktion zwischen ihnen.

Teams und Domänen

- Team Cognitive Load anhand relativer Domänenkomplexität bewerten
- Nicht mehr als eine komplexe/ komplizierte Domäne pro Team
- Klar definierte Domänengrenzen
- Domänengrenzen an Team Cognitive Load angleichen

Kasten 2

In diese Kerbe schlägt Team Topologies mit seinem Baukastenmodell von Teamrollen und Interaktionsarten [SkePai19]. Darin gibt es bewusst nur wenige, klar definierte Rollen: Stream aligned Team, Platform Team, Subsystem Team und Enabling Team – jede mit eigenen Charakteristiken und Verhaltensweisen (siehe Abbildung 4). Interaktionen zwischen den Teams können die Ausprägungen Collaboration, Facilitating und As-a-Service haben. Diese Begriffe ausführlich zu erklären sprengt hier leider den Rahmen und ich muss für weitere Erklärung auf das Buch „Team Topologies" verweisen. Die Quintessenz für diesen Artikel ist: Ein Entwicklungsteam kann sich viel besser autonom und ohne Ablenkungen auf seine wesentliche Verantwortlichkeit fokussieren, wenn seine Rolle sowie die Grenzen und Beziehungen zu anderen Teams klar definiert sind.

Auch direkte Eingriffe in die personelle Zusammensetzung von Teams sind ein Instrument, um Cognitive Load zu managen (siehe Abbildung 5). Eine typische Teamentwicklung ist das Feature Team, das nach und nach immer mehr Software für seine Domain produziert, bis die Menge an Dingen, die es kennen und pflegen muss, zu groß wird. In manchen Fällen kann man ein Team dann vergrößern, um die Last auf mehr Schultern zu verteilen.

Dabei gibt es zwar keinen großen Spielraum, da man ab einer Größe von ca. acht Teammitgliedern wieder in ganz andere Probleme hineinläuft – aber ein achtköpfiges Team hat in Summe eine höhere kognitive Kapazität als ein vierköpfiges.

Eine andere Möglichkeit ist die Teilung der Domäne in Subdomänen und damit auch die Trennung des Teams in zwei neue Teams, die dann die bestehende Software und zukünftige Verantwortlichkeiten – und damit die Cognitive Load – unter sich aufteilen.

Der dritte Weg ist eine Transformation des Teams. Das passiert auch unfreiwillig, wenn man nichts tut: Das Team verwendet einen immer größer werdenden Teil seiner Kapazität für die Wartung und den Support bestehender Software und hat immer weniger Zeit für Weiterentwicklungen. Das muss nicht schlecht sein, wenn man diese Transformation plant und bewusst in Kauf nimmt. Dazu gehört eine realistisch angepasste Erwartungshaltung an den

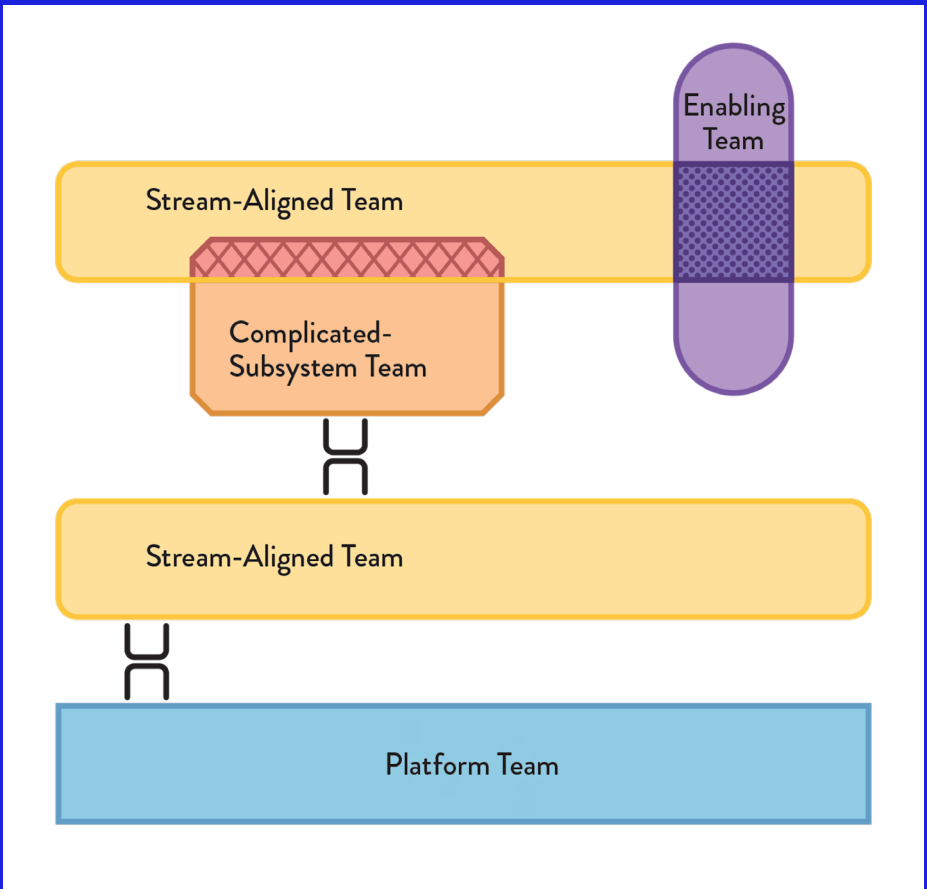


Abb. 4: Beispiel Team Topologies Diagramm
(Aus Team Topologies von Manuel Pais und Mathew Skelton)

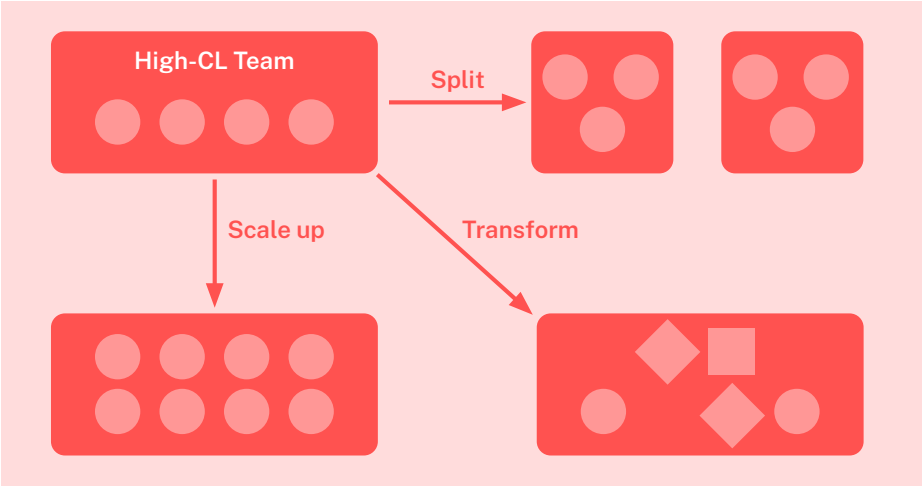
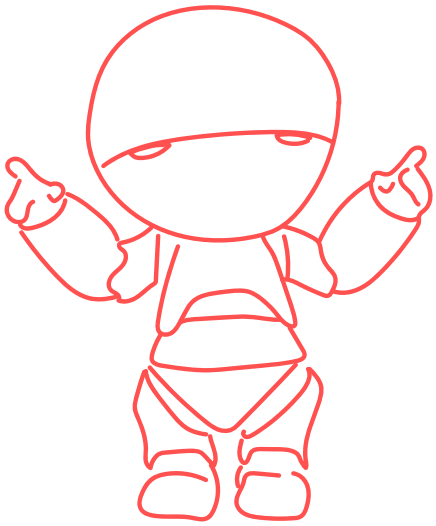


Abb. 5: Cognitive-Load-Maßnahmen durch Teamzusammensetzung

Output des Teams, auch damit es nicht unter Druck gerät. Außerdem sollte dann die Zusammenstellung des Teams bewusst angepasst werden, denn ein erhaltendes Team braucht andere Rollen und zieht andere Charaktere an als ein innovationstreibendes Team.
Letztendlich ist es bei all diesen strukturierenden Maßnahmen wichtig, Conway's Law [Con67] im Hinterkopf zu haben: Die Software, die am Ende herauskommt, wird maßgeblich durch die Kommunikationsstruktur der Organisation beeinflusst. Ziehen Sie daher schon bei der Strukturierung der Teamlandschaft Softwarearchitekten hinzu!

Was fange ich damit jetzt an?

Zu hohe Cognitive Load ist ein oft übersehenes Risiko für Teams. Wissenschaftlich ergründet und in der IT-Fachliteratur als verbreitetes Problem anerkannt, sollte man sie nicht als nebensächlich abtun. Machen Sie sich bewusst, wie stark die kognitive Kapazität des Teams, in dem Sie arbeiten, beansprucht ist. Ist noch Luft, dann seien Sie beruhigt. Häufen sich die Anzeichen, dass das Team zu viel auf dem Teller hat, dann sehen Sie es als Ihre eigene Verantwortung und in Ihrem eigenen Interesse, das zu ändern.
Wenn Sie leitende Verantwortung über eines oder mehrere Teams haben: Strecken Sie die Fühler aus und versuchen Sie frühzeitig zu erkennen, falls die Cognitive Load eines Teams an ihre Grenzen kommt. Sowohl Ihre Mitarbeiter als auch Ihre Organisation werden davon profitieren.
Ich hoffe, ich konnte mit diesem Artikel einen Denkanstoß und einen Einstieg in die Methodiken liefern, die Ihnen dabei helfen werden.



Literatur & Links

[Con67] Conway's Law, siehe: https://www.thoughtworks.com/insights/articles/demystifying-conways-law
[Fra20] A. Franke, Architekturpatterns in Modulithen, in: Java Magazin 12/20, 1/21, 2/21
[Lil20] C. Lilienthal, Langlebige Software-Architekturen, dpunkt.verlag, 3. Auflage. 2020
[Pink09] D. Pink, Drive: The surprising Truth About What Motivates us, Riverhead Books
[SkePai19] M. Skelton, M. Pais, Team Topologies – Organizing Business and Technology for Fast Flow, It Revolution Press, 2019
[Swe88] J. Sweller, Cognitive Load During Problem Solving: Effects on Learning, in: Cognitive Science, 12 (2), S. 257–285
[Swe98] J. Sweller, J. J. G. van Merrienboer, F. G. W. C. Paas, Cognitive Architecture and Instructional Design, in: Educational Psychology Review, 10 (3), S. 251–296
[Thought] Thoughtworks Technology Radar, siehe: https://www.thoughtworks.com/radar/techniques