Projekte drehen

Hallo! Willkommen zum Talk über das Drehen von Projekten

Wer war schon einmal in einem **Katastrophen-Projekt**?

Frage an das Publikum: Wer von Euch war schon einmal in einem Katastrophenprojekt? Projekte, in denen Dinge langsam werden, Interessen unklar sind und man sich gerne und viel streitet? Die Zielstellungen unklar sind?

Wir auch.

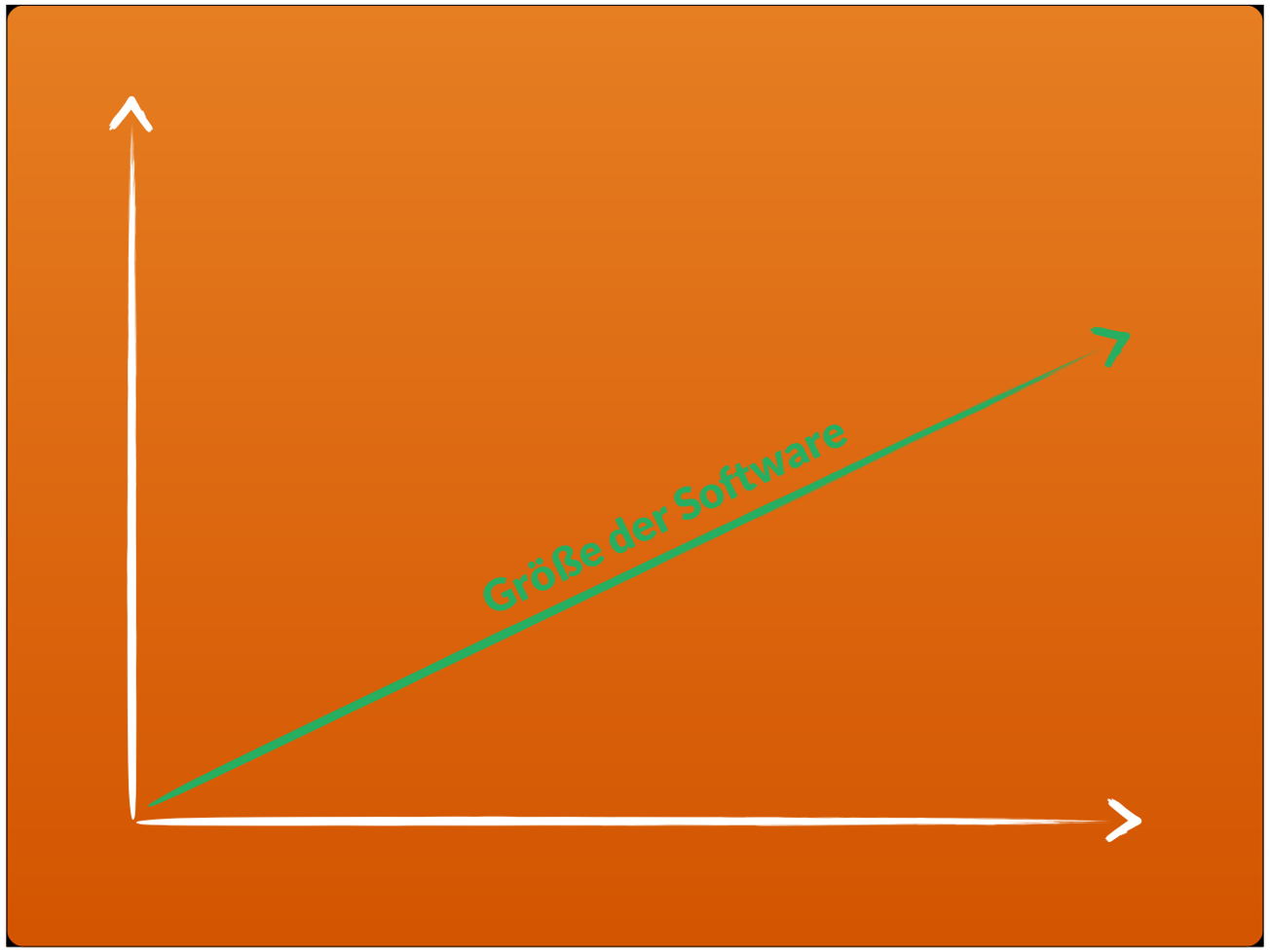
Wir auch - Vorstellung

Wie man mit lauter **richtigen Schritten** zur einer **kaputten Lösung** kommt.

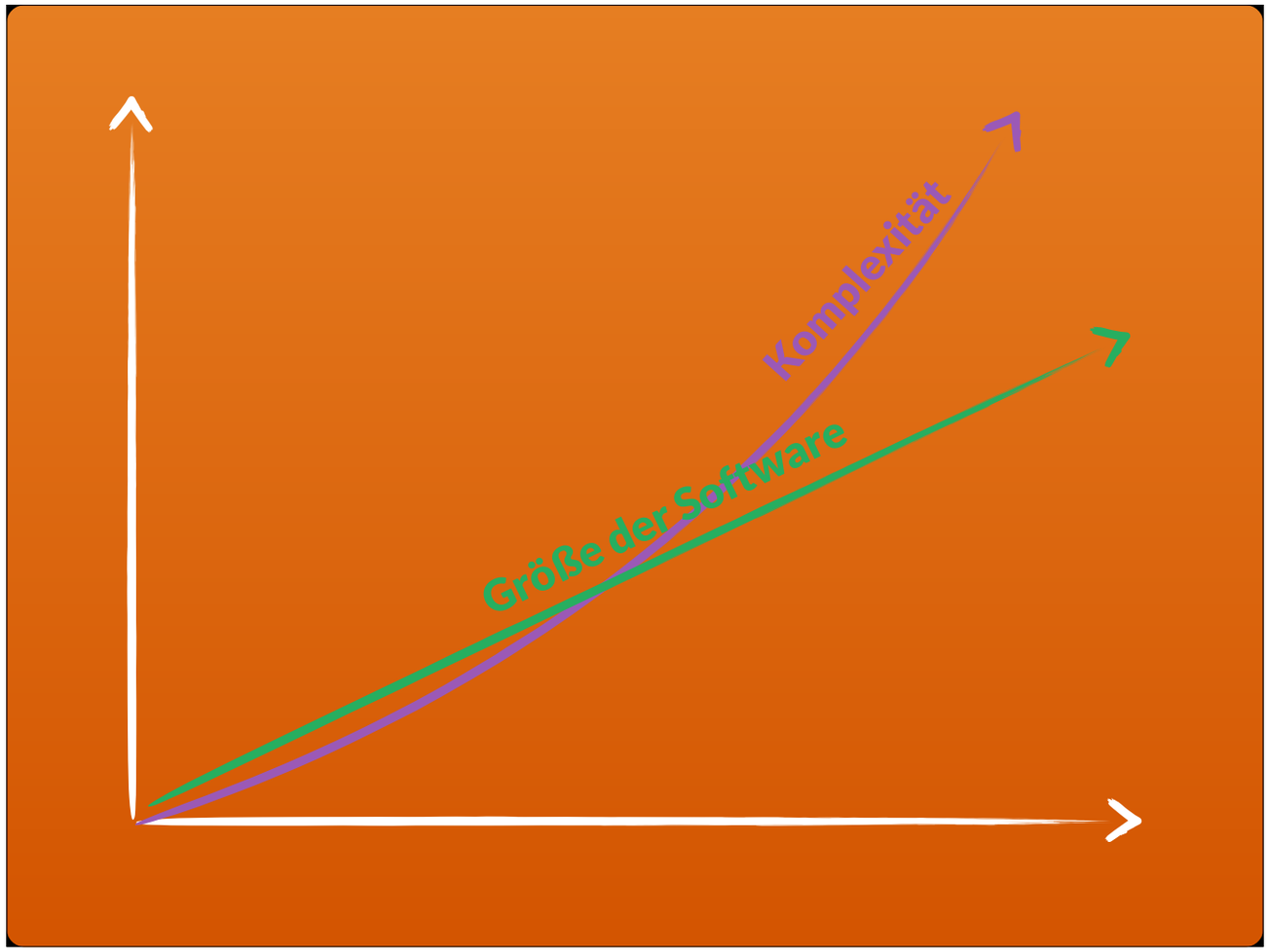
Katastrophen passieren nie mit Absicht und nur selten aus Dummheit. Und auch in unserem Fall war jeder einzelne Schritt sinnvoll, auch wenn die Summe aller Schritte zu einer schwer wartbaren Software geführt hat.

Wenn man mit einer neuen Software, möglichst in Greenfield oder auf Basis bestehender Komponenten, dann entwickelt man sehr schnell.

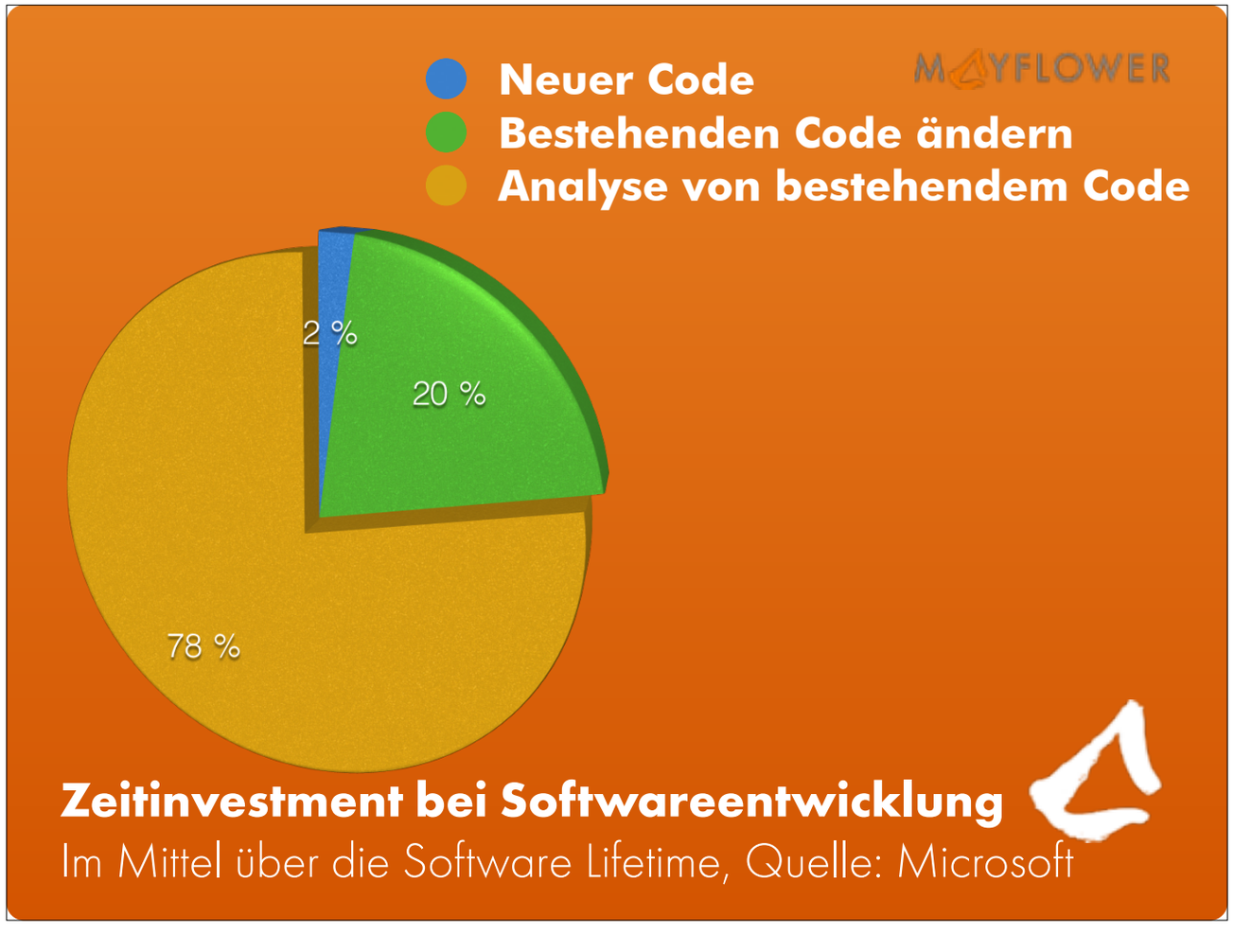
In dieser Phase macht Software Spass - man bewegt sich sehr schnell, Features sind schnell integriert & gebaut, Bugs eine Frage von Minuten.



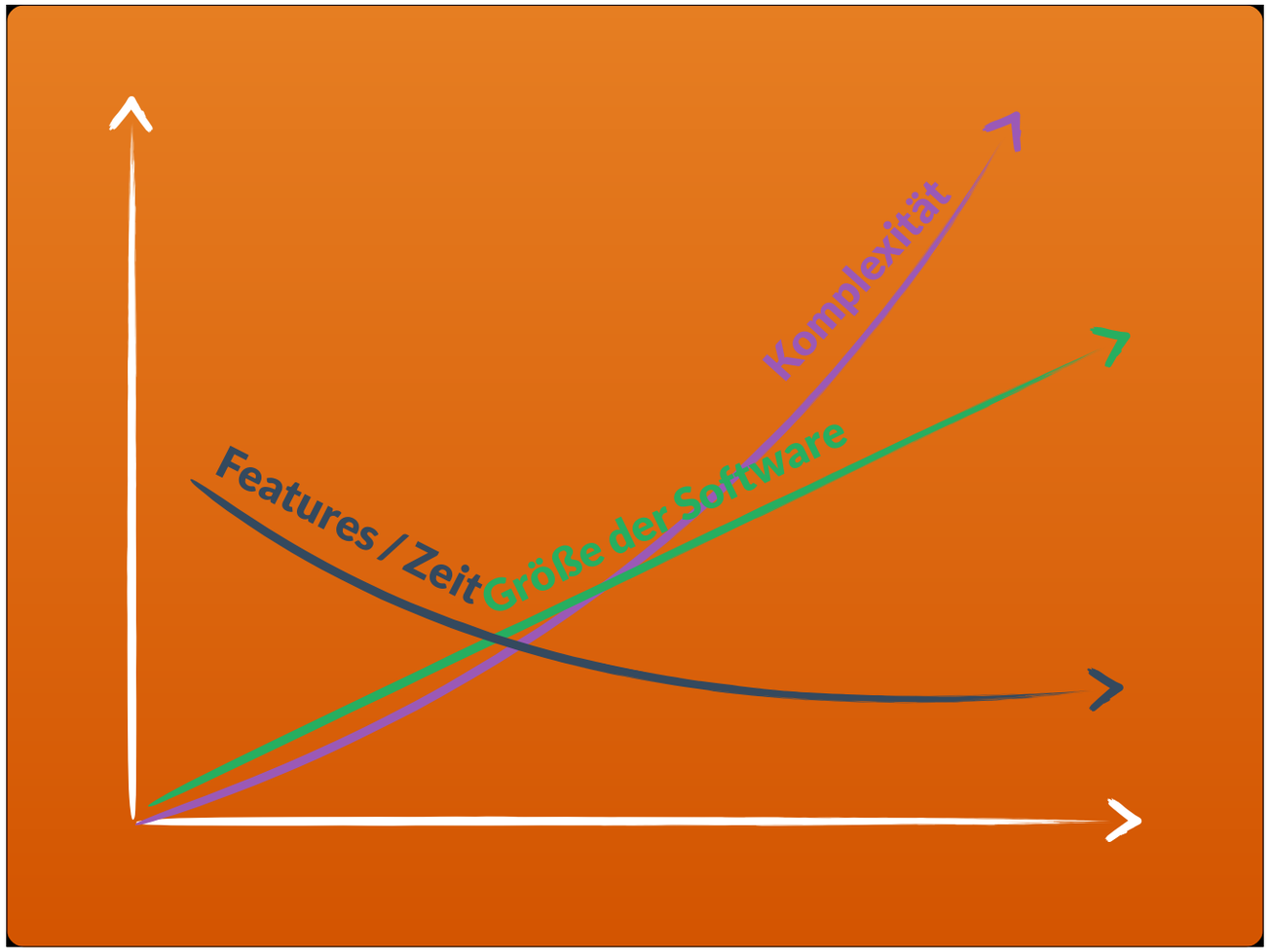
Aber die Software wird kontinuierliche grösser. Und noch etwas anderes passiert.



Die Komplexität der Software steigt ebenfalls. Weil mehr Elemente in der Software auf mehr Elemente zugreifen handelt es sich hier um kubisch bis exponentiell wachsende Funktion.



Die meiste Zeit geht bei der Softwareentwicklung in die Analyse von bestehenden Code - und wächst proportional zur Komplexität, also zur Vernetzung innerhalb der Software.



Im Resultat wird Softwaredevelopment über Zeit immer langsamer. Bei guter Software stetig langsamer, bei schlechter Software kann die Verlangsamung des Entwicklungsprozess auch deutlich steiler ausfallen.

Und das wird auch von aussen wahrgenommen - das Business, das bislang stetige Lieferungen gewöhnt ist, stellt ebenfalls fest, dass das Development langsamer geworden ist. Faktisch ist das ein zwangsläufiger Prozess, aber kein verstandener.

Was passiert also? Die Developer werden aufgefordert wieder mehr zu liefern, jetzt kommt es nämlich darauf an.

Die Entwickler können das aber nicht - sie sind nicht wirklich flexibel in Ihrer Bewegung, und sie können nur im Rahmen dessen navigieren was die Legobausteine der alten Systeme erlauben.
Dazu kommen technische Schulden - immer wenn es eilig wird geht die Qualität verloren. Technical Debt macht die ganze Entwicklung nicht nur langsamer, sondern erzeugt auch mehr Bugs in der Arbeit - die dann wiederum selbst noch für eine Verlangsamung sorgen.

Das Entwicklung bringt also nicht die Performance die es braucht - erster Lösungsansatz ist mehr Druck. Das funktioniert auch kurzzeitig, versagt aber auf der mittleren Strecke. Und zwar meist schon nach sehr kurzer Zeit.

Resultat:

Die **Performance sinkt** nach einem initialen Anstieg **weiter**.

Druck auf Softwaredeveloper hilft nur begrenzt weiter - am Anfang steigt die Lieferfähigkeit, und auch Überstunden resultieren in mehr Features. Das geht etwa 4 Wochen gut, dann flaut die Mehrleistung nicht nur ab, sondern man landet sogar unterhalb dem Torniveau - denn die technische Qualität ist einmal mehr deutlich gesunken.

Die zweite Strategie ist in der IT auch nicht eben unüblich - wenn ich mehr Features über Zeit will, erhöhe ich die Zahl der beteiligten Leute.

Brooks Law:

„Adding manpower
to a late software project
makes it later"

(Un)bekannt seit 1975.

Eigentlich ist Brooks Law schon von 1975, doch seltsamerweise hat es sich noch lange nicht in deutsche Managementetagen herumgesprochen. Ich kann den Featurefluss nicht schnell erhöhen indem ich neue Leute in das Projekt nehme. Ganz im Gegenteil - es wird langsamer, und diese Verlangsamung kann durch für ein halbes Jahr gelten, bevor das Projekt durch die neuen Kollegen an Fahrt gewinnt, manchmal geschieht dies auch nie.

Resultat:

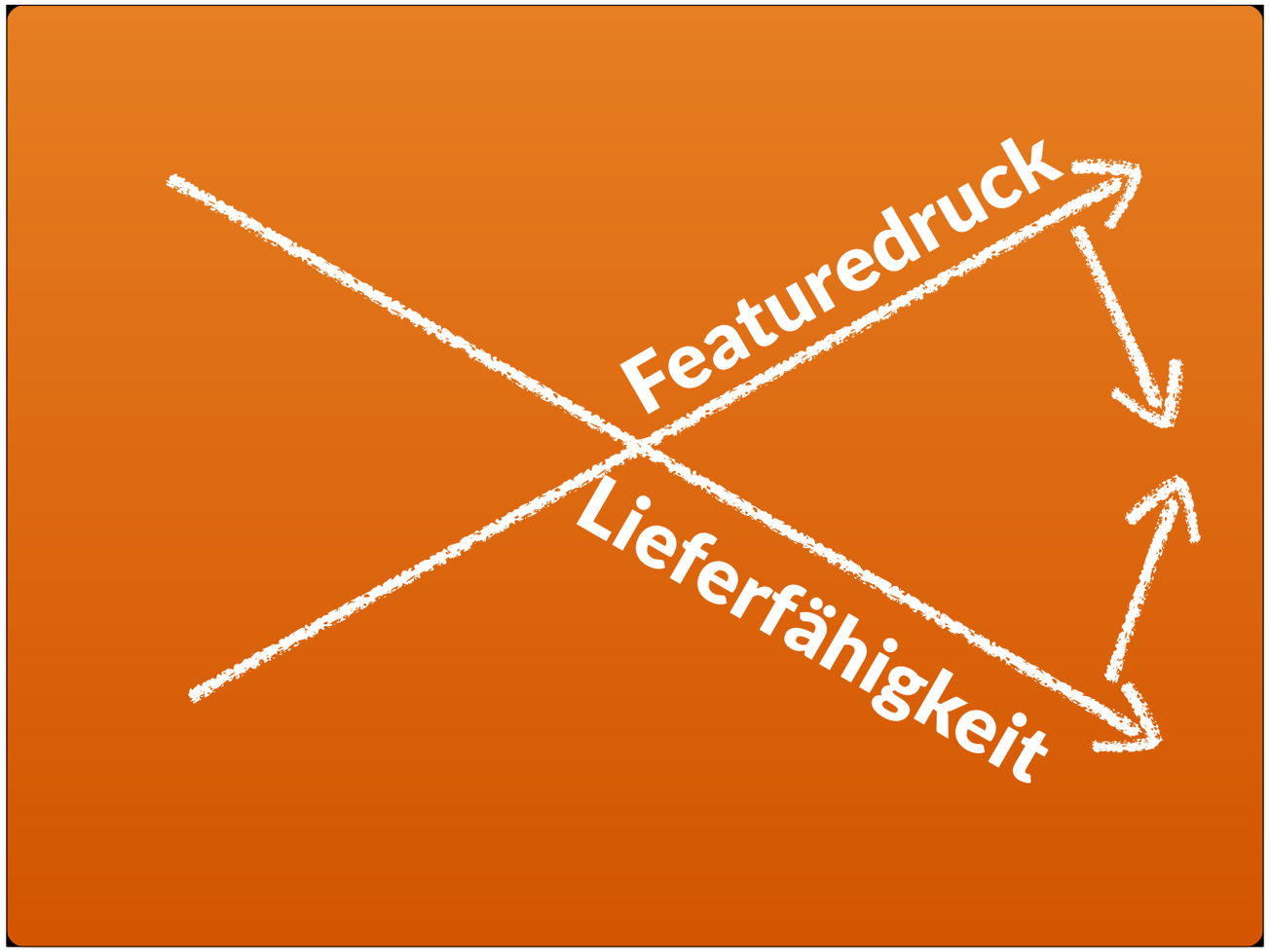
Die **Performance sinkt** weiter, jetzt aber **teurer**.

Fazit daraus - die Performance sinkt immer noch, die Kosten sind allerdings gestiegen.

Und am Ende befindet man sich in einer Situation, in der der Featuredruck immer mehr steigt, die Lieferfähigkeit immer kleiner wird.
Die übliche Antwort darauf: Aktionismus - es werden mehr Dinge probiert, andere Leute, andere Techniken. Solche Projekte werden auch Deathmarch, also Todesmarschprojekte genannt - da gibt es sogar ein Buch mit genau diesem Titel zu.

Als Management denkt man: Liebes Development, jetzt wird es gerade wirklich wichtig, und Ihr performed überhaupt nicht. Was soll ich denn bitte noch tun?

Und wenn Du lange genug im Chaos steckst und nichts nachhaltig den gewünschten Erfolg bringt - dann tut sich faktisch eine Chance auf.



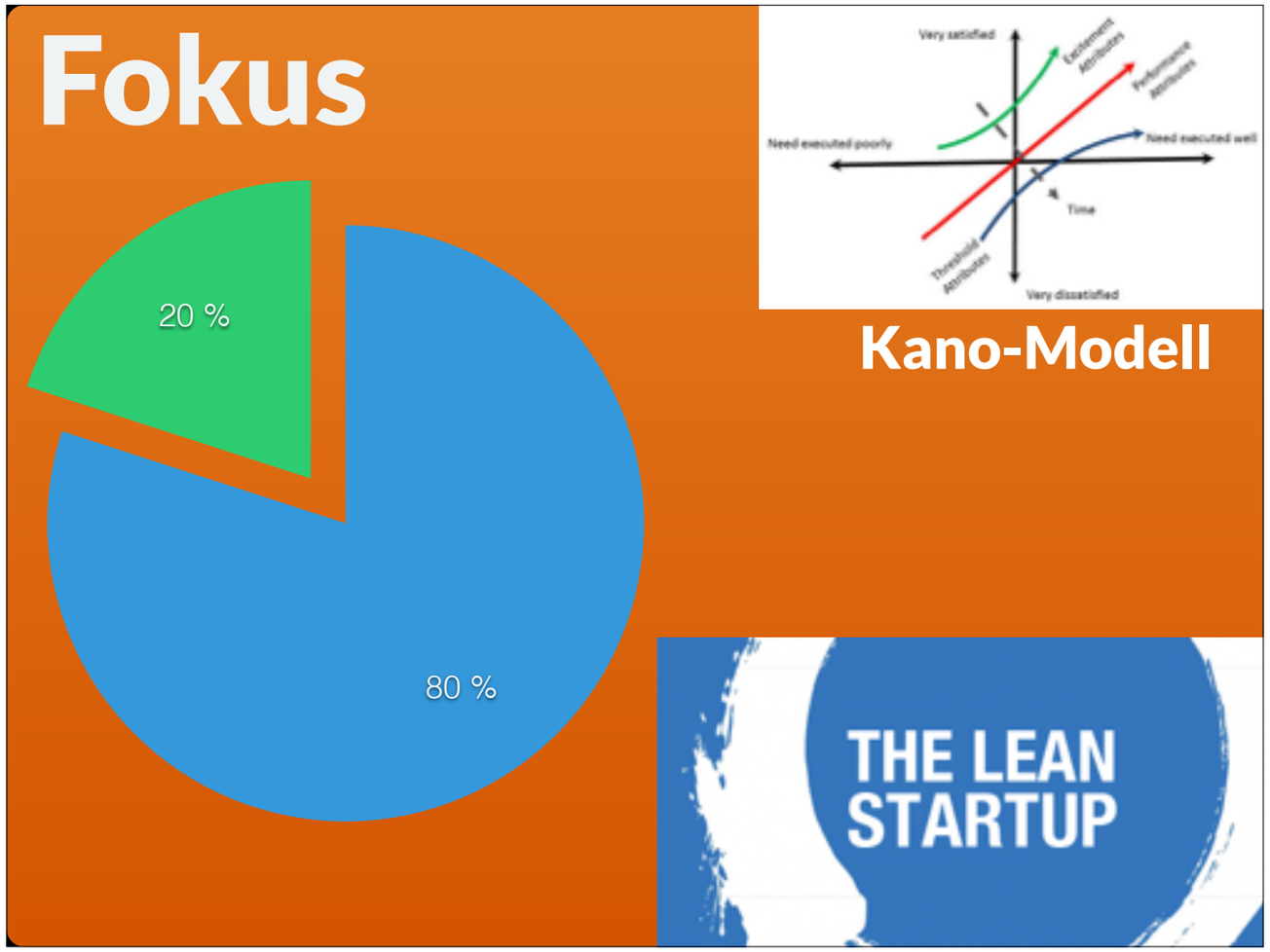
Wenn ich einen Schritt zurücktrete ist die Lösung doch eigentlich ganz offensichtlich: Ich muss die Lieferfähigkeit nach oben bringen und den Bedarf reduzieren. Das sagt sich jetzt leicht. Wie erreiche ich das?

Hosen runter & Shareholdersupport

In unserem Fall haben wir den gesamten, angehäuften Bedarf schonungslos auf den Tisch gelegt und damit ein Bewusstsein geschaffen dass wir hier auf ganz viele lose Enden blicken. Konkret waren da Features für B2C Kunden, Features für B2B Kunden, ellenlange Wunschlisten der angeschlossenen Fahrer, aus dem BI, dem Marketing, dem Accounting, der Operations etc. etc.
Ab diesem Moment wussten wir unseren Shareholder hinter uns - und das gibt einem erst den Freiraum, um wieder aus dem Sumpf herauszukommen.

Störungen reduzieren

Wie haben wir es gemacht?
Natürliche Verknappung auf der Maintenance-Seite, und die klügsten Leute mit Domainwissen so freigeschaufelt. Wir haben Business und Technik räumlich getrennt - das soll man eigentlich nicht machen, war in dieser Situation aber notwendig.



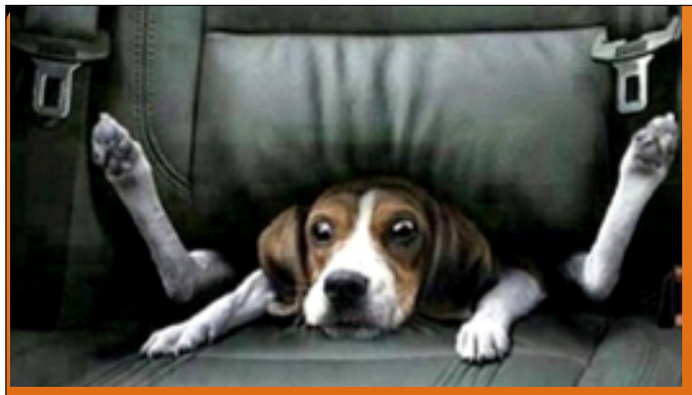
Und während man den Featuredruck nicht reduzieren kann, kann man ihn wenigstens fokussieren. Es gilt die Pareto-Regel: mit 20% des Aufwands kann man 80% des Nutzens erreichen. Wir haben das im konkreten Fall mit dem Kano-Modell gemacht, und den Rewrite über die Internationalisierung als Lean Startup gemacht - also ein MVP, ein kleinstmögliches Produkt in einem anderen Land aufgebaut und dann nach und nach die übrigen Features ergänzt.

Sequentiell > Parallel

Time to Market:
3 Monate im Mittel
Fokus: **unklar**

Time to Market:
2 Monate im Mittel
Fokus: **klar**

In Deathmarchprojekten sind oft sehr viele Dinge parallel in Arbeit - man hofft die Time to Market zu verbessern, indem man früh beginnt.
Faktisch ist es aber offensichtlich, dass die Time to Market bei sequentieller Bearbeitung deutlich besser ist, denn abgeschlossene Tätigkeiten werden früher geliefert.
Und nicht nur das - durch den besseren Fokus liefert die Entwicklung schneller.



DEV Ops

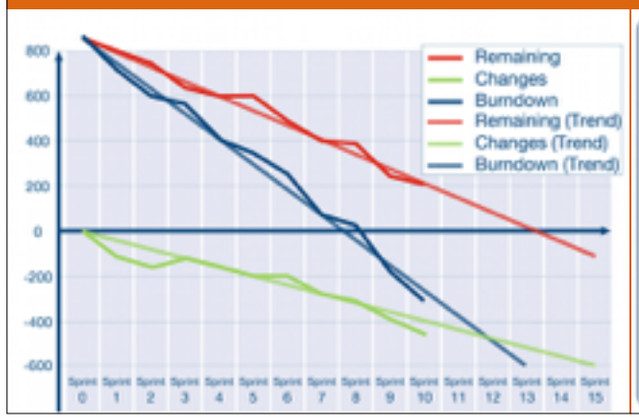
Immer QA statt **QA-Phasen**
Deploy in **5 Minuten**
Cloud statt **Server**
Infrastructure as **Code**

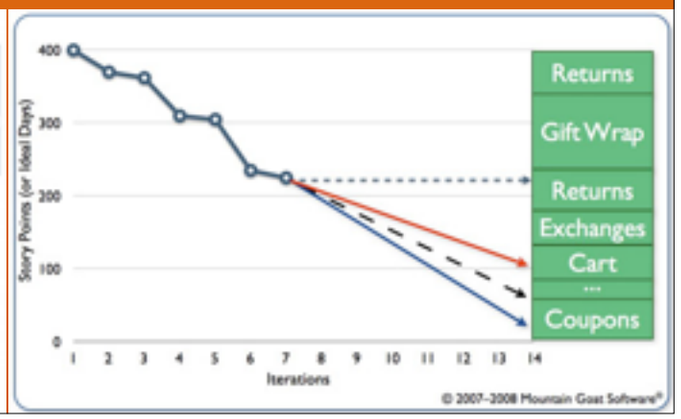
Nächster Schritt war Development auf Speed bringen. Da gehört nicht nur Fokus dazu, auch das Environment muss stimmen. Dafür haben wir DevOps-Methoden benutzt - gleich in die Cloud, automatisiertes Deployment, automatisierte Tests. Es gibt keine QA-Phasen, weil kontinuierlich - automatisch und manuell - getestet wird. Die komplette Plattform kann in 45 Minuten neu aufgebaut werden, wenn es einen Ausfall gäbe. Die gibt es jetzt natürlich nicht mehr.

Kontrollverlust

Das Management muss zulassen, dass die Entwicklung sich neu sortiert und darauf vertrauen, dass sie in absehbarer Zeit wieder lieferfähig wird.
Wenn ich da in einen Modus des Micromanagement und der Planung übergehe - was in so einer Situation ein nachvollziehbarer Reflex ist - dann ersticke ich jedes noch so kleine Quäntchen auf Erfolg. Statt dessen ist meine Aufgabe das Alignment von Development und Businessanforderungen. Jeder muss verstehen, worum es wirklich geht. In unserem Fall war ich derjenige der zwar das Domainwissen besaß aber die Lösung genauso wenig kann wie alle anderen. Ich kann nur empfehlen sich mitten in diesen Prozess hinein zu begeben, darauf zu vertrauen dass die Entwicklung die Lieferfähigkeit sehr bald wieder herstellen wird und die eigene Energie dafür einsetzen den Weg von jeglichen Blockaden zu befreien.

Kontrollgewinn über **Empirie**





Auf der anderen Seite gibt es auch den Kontrollgewinn durch Empirie. In den chaotischen Phasen war es meist nicht möglich, empirisches Projektmanagement zu machen. Diesen Umstand kann man beim Einfangen nutzen. Indem man wieder Daten sammelt und auf diesen Erfahrungen arbeitet kann man wieder Verlässlichkeit erzeugen, zunächst auf kleinem Niveau, aber wachsend.

Eigentlich ist es offensichtlich, aber Softwareteams halten sich trotzdem nur selten daran. Wenn ich bei meinem Kunden gewinnen möchte, brauche ich ein taugliches Erwartungsmanagement. Wenn ich also die empirischen Daten kenne muss ich nicht nur auf ihnen Arbeiten und die enthaltene Ungewissheit - in Form von Volatilität - berücksichtigen, sondern auf dieser Basis mehr Liefern als ich verspreche - schlicht um wieder Vertrauen herzustellen. Das setzt aber voraus, dass ich vorher in der Lage bin, die Erwartung deutlich zu reduzieren. Aber: gelingt mir das nicht, werde ich am Ende ohne verlieren.

Der zweite Aspekt von Erwartungsmanagement ist erkaufte Verlässlichkeit. Ich liefere nicht mehr alles in der Sekunde aus, in der es fertig ist. Ich nutze Feature Flags und Dark Launches, um im Hintergrund die Features gegen wenige Nutzer zu validieren. Der offizielle Launch folgt später, wenn sicher ist, dass die Features funktionieren.

Fazit: Es gibt keine Projektsanierung bei vollem Featurestream. Die Aufräumarbeiten müssen stattfinden, damit die Software wieder funktionieren kann.
Wenn man diese Durststrecke des Aufräumen aber aushält, dann hat man am Ende eine Software, die nicht nur schnell ist, sondern auch verlässlich - und nicht zuletzt verlässlich schnell bleibt.