Über Softwarentwicklungskultur und Projektmanagement

Luca Thomas

luca.thomas@tu-braunschweig.de

Lisa Glowczewski

l.glowczewski@tu-braunschweig.de

Inhaltsverzeichnis

1 Überblick

Software ist untrennbar mit dem Großteil der modernen gesellschaftlichen Fortschritte verbunden. Das lässt ihrer Qualität automatisch eine hohe Verantwortung zukommen. Bestehende Systeme müssen effizient und zukunftsorientiert gewartet werden und Innovationsimpulse in die richtigen Bahnen gelenkt werden. Vor allem in großen Projekten werden diese Prozesse von Managementstrukturen koordiniert. Wir beschäftigen uns hier mit dem Einfluss dieser auf die Qualität der Produkte und mit der generellen gesamtgesellschaftlichen Situation in der Softwareentwicklung. Hierzu werden sowohl spezifische technische Aspekte, als auch allgemeine kulturelle Dynamiken in großen Projekten beleuchtet.

2 Fallbeispiel

Airbus brachte um das Jahr 2010 mit seiner A320neo eine neue Linie an Flugzeugen heraus. Diese zeichnete sich besonders durch effizientere Triebwerke aus, die 15 bis 20% weniger Treibstoff verbrauchen als ihre Vorgängermodelle [1]. Um wettbewerbsfähig zu bleiben, begann der Marktkonkurrent Boeing als Reaktion auf die Ankündigung der A320neo Serie mit der Entwicklung eines Nachfolgers der eigenen Boeing 737, der Boeing 737 MAX. Kurz nachdem diese in Betrieb genommen wurde, machte sie allerdings aufgrund von mehreren Flugzeugabstürzen negative Schlagzeilen. Die Hintergründe hierfür finden sich in der unzureichenden Qualität von dort verwendeter Software.

In modernen Systemen in der Luftfahrt unterstützen softwarebasierte Assistenzsysteme an vielen Stellen die Piloten und sollen so unter anderem für eine erhöhte Sicherheit sorgen. In diesem Fall befasste sich ein Teil des Projekts bei der Entwicklung der 737 MAX mit der Entwicklung eines neuen Flugassistenzsystems.

Da Airbus seine treibstoffeffizientere Baureihe an Flugzeugen zuerst bekanntgegeben und früher mit der Entwicklung begonnen hatte, stand Boeing zeitlich im Nachteil und somit unter Zeitdruck bei der Entwicklung der 737 MAX. Falls das Projekt der Entwicklung der 737 MAX zeitlich in zu großen Verzug zu der Airbus A320neo gelangen würde, befürchtete Boeing einen Verlust an entscheidenden Marktanteilen. Aufgrund der Flugzeugarchitektur der Boeing 737 war es nicht einfach möglich die Konkurrenten von Airbus zu immitieren und eine einfache Erneuerung der Triebwerke durchzuführen. Hier kam eine Softwarelösung ins Spiel. Es war somit Priorität, das Projekt so schnell wie möglich abzuschließen und die notwendigen Mittel möglichst gering zu halten.

Die 737 war schon 20 Jahre älter und liegt tiefer als der Airbus A320. Da so der Abstand der Triebwerke zum Boden bei der Boeing 737 geringer war, erforderte die Vergrößerung der Triebwerke für die Boeing 737 eine Umpositionierung der Triebwerke. Die Triebwerke wurden daraufhin weiter vorne und höher auf den Tragflächen positioniert. Dadurch wurde aber auch der sogenannte „Angle of Attack“ (AoA), also der Winkel, in dem die Luft auf die Tragflächen trifft, vergrößert, was das Flugzeug anfälliger für einen Strömungsabriss aufgrund von zu steilem Steigflug machte [2, p. ~413]. Eine solche Änderung des Flugverhaltens bedarf normalerweise einer Umschulung für Piloten, welche wiederum wertvolle Zeit und Geld benötigt – Mittel, die Boeing nicht bereit war aufzuwenden.

2.1 MCAS

Das „Maneuvering Characteristics Augmentation System“, kurz MCAS, wurde daraufhin entwickelt, um der Anfälligkeit zum Strömungsabriss unbemerkt entgegenzuwirken. Die grundlegende Funktionsweise wird in Abbildung 1 dargestellt. Das MCAS bezieht Daten von einem der AoA-Sensoren des Flugzeugs. Wenn die Sensoren einen zu hohen AoA signalisieren, reguliert das MCAS über den Stabilisator des Hecks diese Lage, indem es das Heck des Flugzeugs höher steigen lässt und die Nase des Flugzeugs absenken lässt, sodass sich der Steigwinkel des Flugzeugs nicht mehr im kritischen Bereich befindet [3, p. ~2960].

Ein solches System war bis dahin noch in keinem anderen Passagierflugzeug zum Einsatz gekommen und existierte sonst in ähnlicher Form nur für militärische Flugzeuge [4, p. ~4]. Die Piloten waren auf eine solche Umstellung der technischen Verhaltensweisen nicht eingestellt, da sie keine Kenntnis von diesem System besaßen.

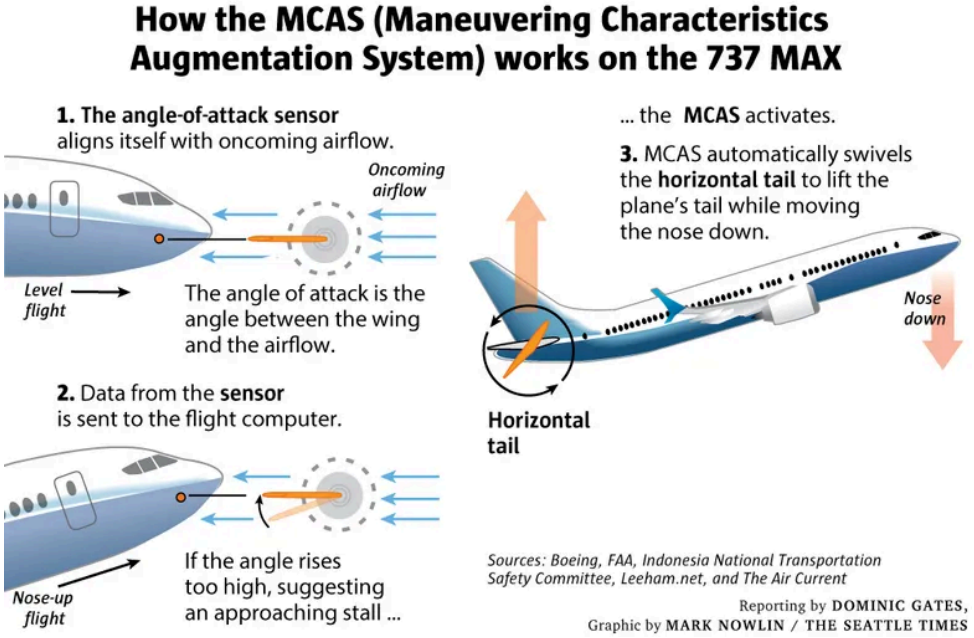


Abbildung 1: Die Funktionsweise des MCAS [5].

2.2 Flugzeugabstürze und Folgen

Etwa eineinhalb Jahre nach Inbetriebnahme der 737 MAX stürzte 2018 ein Lion Air Flug nach nur 13 Minuten nach der Landung ab. Vier Monate später stürzte 2019 ein Ethiopian Airlines Flug 6 Minuten nach dem Start ab. Die Ursache der Abstürze war, dass das im Hintergrund aktive MCAS System bei beiden Flugzeugen mehrmals ungewollt den Sinkflug eingeleitet hatte. Das System erhielt falsche Werte eines AoA-Sensors und verließ sich lediglich auf die Werte eines einzelnen AoA-Sensors, welcher zufällig bei Start ausgewählt wurde. Die Piloten konnten sich den Eingriffen des Systems nicht widersetzen und den Absturz verhindern. Das MCAS System war nicht bekannt, kam nicht in den Dokumentationen vor und wurde auch in den Trainings der Piloten nicht behandelt [3, p. 2958-2959].

Nach diesen Vorfällen brachte Boeing ein Update heraus, nach welchem nun nicht mehr nur ein AoA-Sensor als alleiniger Indikator für einen zu steilen Steigflug genutzt wurde und nun Signale zur Erkennung einer solchen fehlenden Übereinstimmung der AoA-Werte standardmäßig enthalten sind. Des Weiteren wurde die Möglichkeit installiert, dass Piloten das Flugzeug anstelle mit MCAS manuell kontrollieren können. Die Software sei lange analysiert, sowohl im Simulator, als auch in der Realität getestet worden, und nun auch strenger von der Amerikanischen Flugbehörde beaufsichtigt. Neben der Änderung der Software wurden auch die Trainings der Piloten für die 737 MAX erweitert und das MCAS als fester Bestandteil dieses Trainings aufgenommen [4, p. 6-7].

2.3 Unternehmens- und Projektkultur

Boeings Ingenieure hatten ursprünglich viel Einfluss auf die Entwicklung der Flugzeuge – die Kompetenzentscheidung wurde geschätzt und respektiert.

Aufgrund des Wettbewerbs mit Airbus kaufte Boeing in den 1990ern die Firmen McDonnell Douglas und Rockwell Interantional [6, p. ~7] auf. Besonders die Fusion mit McDonnell Douglas sorgte für Konflikte in der Unternehmensphilosophie und für Schwierigkeiten in der Zusammenarbeit [6, p. ~35]. Dabei bestand der Kernkonflikt zwischen Boeings Ingenieurskultur und McDonnell Douglas' börsenorientierter Unternehmenskultur. Zwar kaufte Boeing McDonnell Douglas und das resultierende Unternehmen trug auch den Namen von Boeing, allerdings war die seitdem vorherrschende Unternehmenskultur die von McDonnell Douglas [7, p. 1]. Es kam nun hauptsächlich darauf an, die Kosten gering zu halten [8, p. ~162], auch zu Lasten der Qualitätssicherung.

Abgesehen von den Prioritäten des Unternehmens veränderte sich auch die Unternehmensführung. Der ehemalige Chef von McDonnell Douglas wurde zum Chef des gemeinsamen Boeing-Konzerns. Die Ingenieure hatten nun keine sichere und gute Vertragssituation mehr. Es gab seit der Fusion Androhungen von Gehaltskürzungen, falls die vom Management gesetzten Ziele nicht erreicht werden konnten [8, p. ~163]. Boeing missachtete bewusst Beschwerden von Mitarbeitern, die Sicherheitsbedenken äußerten und stellte damit den Zeitplan und Gewinn über Sicherheit und Qualität [3, p. ~2962].

3 Ein kultureller Standpunkt

Solche Entwicklungen sehen wir nicht zum ersten Mal. Viele Menschen wundert es überhaupt nicht, das sich ein solcher Vorfall überhaupt ereignen kann. Software hatte lange Zeit einen gewissen Sicherheitsstatus: „Ein Computer verhält sich immer genau so wie vorgegeben". Diese Zeiten sind jedoch vorbei und eine gegenteilige Einstellung hat sich in der breiten Masse etabliert. Mittlerweile stürzen Flugzeuge ab, und das einzig und allein aufgrund von schlechter Software. Wie sind wir in der Gesellschaft in eine derartige Situation geraten, wie sehen die generellen kulturellen Dynamiken in diesem Bereich aus und welche Folgen kann es langfristig haben, wenn dem nicht entgegengewirkt wird? Wie kann man eine solche Entwicklung aufhalten und einen positiven Impuls in die richtige Richtung setzen? Im Folgenden werden wir uns mit diesen Fragen auseinandersetzen und dafür den Blickwinkel auf das zuvor betrachtete Fallbeispiel erweitern.

3.1 Historie und Innovation

Wirft man nur einen Blick in unsere direkte Vergangenheit, dann fällt auf, dass die Menschheit regelmäßig ihre Fähigkeit zur technologischen Innovation unter Beweis gestellt hat. Anfang der 60er Jahre lagen die Vereinigten Staaten im Zuge des kalten Krieges wiederholt zurück hinter der Sowjetunion im sogenannten „Wettlauf ins All“ [9]. 1957 schoss die Sowjetunion den ersten Satelliten „Sputnik“ in die Erdumlaufbahn und veranlasste die USA dazu nachzuziehen. 1961 wurde Juri Gagarin dann der erste Mensch im Orbit. Präsident John F. Kennedy erklärte darauf hin im Mai 1961, eine große Menge Mittel seien nötig, um in diesem Wettrennen zu gewinnen, und kündigte an, noch vor Ende des Jahrzehnts einen Menschen auf den Mond bringen zu wollen. Auch wenn dieses Ziel zunächst verrückt erschien, gelang es dann tatsächlich im Jahr 1969 nach extrem kurzer Zeit im Rahmen der Apollo 11 Mission genau das zu verwirklichen. In den darauffolgenden Jahrzehnten wurde das Weltraumprogramm der USA zwar weitergeführt, aber neue Technologien wie das Space Shuttle bestanden nicht den Test der Zeit. Dessen Nutzung war unzuverlässig und sehr teuer. Dieser Trend des technologischen Verfalls mündete dann darin, dass die USA zunehmende auf Russland und deren Sojus angewiesen war, um überhaupt Menschen ins All zu befördern. Dass sich das Raumfahrtprogramm heutzutage wieder in einem Aufwärtstrend befindet ist nur darauf zurückzuführen, dass eine Privatperson sich dazu entschieden hatte ein Raumfahrtunternehmen zu gründen und sehr viel Geld und Energie dort hinein zu investieren. Technologie erfährt nicht von selber Innovation, sondern bedarf stetiger Intention und rigoroser Arbeit, um nicht dem sonst automatisch eintretenden Verfallstrend zu erliegen [10].

Auch in der entfernteren Vergangenheit finden sich Dynamiken, deren Verständnis uns in der Betrachtung heutiger Vorgänge behilflich sein kann. Leute haben oft eine leicht zynische Vorstellung alter Zivilisationen, als ob es damals nicht wirklich bemerkenswerte technologische Errungenschaften gegeben hätte, die mit heutigem Verständnis noch interessant wären. Viele dieser Technologien sind mit der Zeit verloren gegangen. Selbst heute verstehen wir noch nicht alle vollends. Gute Beispiele hierfür sind der Lycurgus-Kelch aus dem antiken Rom, der durch die Behandlung mit Edelmetallnanopartikeln bei der Bestrahlung mit Licht die Farbe ändert, oder die Antikythera aus dem antiken Griechenland: ein vollständiger astronomischer Kalender aus Zahnrädern, der unter anderem Planetenbewegungen erfasst [10]. Alle diese Technologien erfordern konstante, sorgfältige Ingenieursarbeit und hohes Fachwissen um überhaupt geschaffen werden zu können. Eben diese Eigenschaft macht sie auch fragil. Im Prozess technologischen Fortschritts muss an jede neue Generation immer der bereits vorhandene Wissensstand übertragen werden. Das ist aufwendig und immer verlustbehaftet. Welche Auswirkungen das haben kann veranschaulicht eine Periode in der späten Bronzezeit um 1177 BC [11]. Zu dieser Zeit hatte sich unter den im Mittelmeerraum ansässigen Zivilisationen ein ausgefeiltes globalisiertes Handelsnetzwerk ausgebildet, das die Versorgung mit Zinn und Kupfer sicherte. Diese Rohstoffe hatten ihren Ursprung in weit voneinander entfernten Orten und wurden für die Produktion von Bronze benötigt – waren also essentiell. Dann, innerhalb einer Periode von ungefähr 100 Jahren, kollabierten alle diese Zivilisationen und verschwanden. Bis heute ist man sich unsicher, warum genau das passierte. Man geht von einer Reihe an Umweltfaktoren wie Fluten und Dürren aus, die dann mit Kriegen zum Zusammenbruch dieses Netzwerkes führten.

Heute finden wir uns in einer privilegierten Zeit, in der es einem vorkommt, als würde ein technologischer Fortschritt ganz von alleine dem anderen folgen. Software ist ein vergleichbar

integraler Bestandteil unserer Gesellschaft und hat ein Netz geschaffen, das Grundlage für den Großteil der informationsabhängigen Prozesse ist. Namenhafte Persönlichkeiten aus der Softwarewelt erkennen zunehmend, wenn auch nicht in der Mehrheit, die Probleme, die sich anbahnen könnten [10] und sehen einen Abwärtstrend in Software und Technologie – entgegen dem öffentlichen Eindruck, in Zeiten von KI und Internet-Start-Ups warte an jeder Ecke der nächste Fortschritt. Befindet man sich in einer solchen Periode des Verfalls, dann geschieht dies langsam und man bemerkt diese Entwicklung nicht zwingend direkt.

3.2 Status quo Softwaretechnologie

In einem Interview mit John Colwell, ehemaligem Chief Microprocessor Architect bei Intel, berichtet er von Problemen an Chips mit denen sie gearbeitet haben [12, p. ~69]. Dort heißt es:

„Rich Lethin and I made a pilgrimage down to TI in Richardson, Texas and we said ‚as best as well can tell many of your chips don‘t work properly., […] I half expected them to say, ‚what you are out of your mind! You‘ve done something wrong. […]‘ But no, they said ‚Yeah we know, let me see your list.‘ And they looked at the list and said ‚here is some more that you don‘t know about., […] Their parts were no worse than anyone else. Motorola‘s were no good, Fairchild’s were no good, they all had this problem. And so I asked TI, ‚how did the entire industry fall on its face at the same time¿ We are killing ourselves trying to work around the shortcomings in your silicon. And the guy said ‚the first generation of TTL was done by the old gray beard guys that really know what they are doing. The new generation was done by kids who are straight out of school who didn‘t know to ask what the change in packaging would do to inductive spikes.‚„

Das illustriert gut den zuvor angesprochenen Zusammenhang. Der generationsübergreifende Wissenstransfer ist mit Reibung verbunden. Ähnliche Entwicklungen gibt es in der Software. Technische Kompetenz macht Platz für Trivia: eine kulturelle Verschiebung. Das hat auch direkte Auswirkungen, nicht nur langfristige.

Aus der Sicht eines Konsumenten erleben wir eine Zeit rasanter technologischer Neuheiten. Allerdings fahren solche neuartigen Technologien oft Trittbrett auf Hardwareverbesserungen. Softwaretechnologie als solche hat in der letzten Zeit keine gleichwertige Innovation verzeichnen können. Die Algorithmen maschinellen Lernens sind beispielsweise nicht wirklich komplex. Sie profitieren hauptsächlich von der enormen Rechenleistung, die wir ihnen zu Verfügung stellen. Die tatsächlich komplexen Dinge sind die, die wir oft für selbstverständlich halten und auf denen alles andere aufbaut.

Die Entwicklungen bei etablierten Produkten ist ebenfalls nicht gut. Die Trends im Bereich Computerleistung in Abbildung 2 verzeichnen allein im Zeitraum 2000 bis 2020 im Bereich Single-Core-Performance eine Verbesserung um einen Faktor 20.

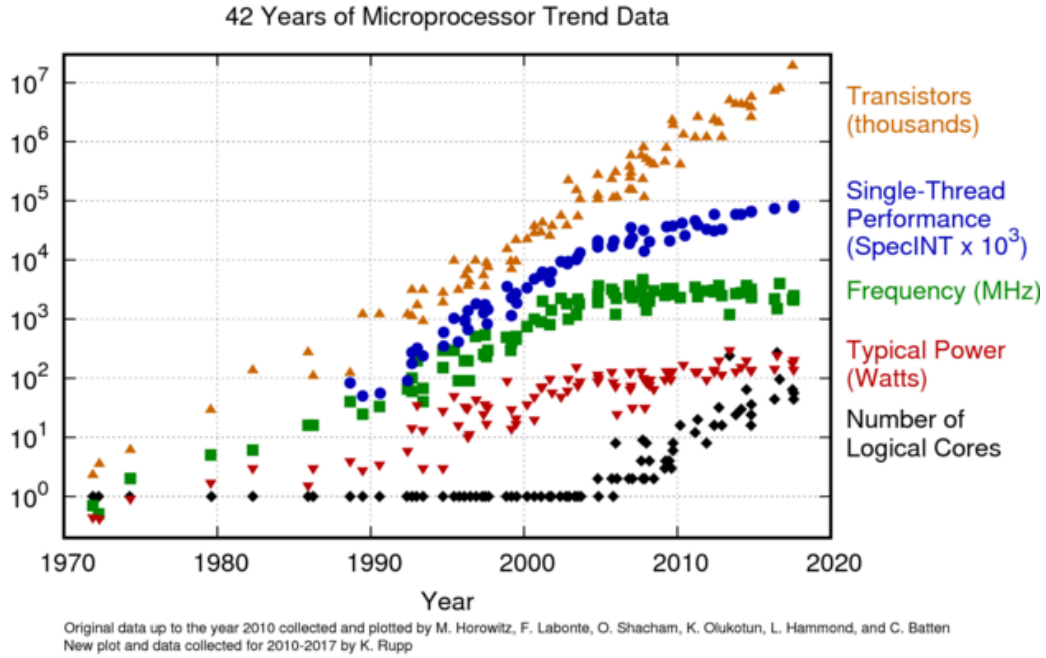


Abbildung 2: Entwicklung Mikroprozessorleistung [13].

Man sollte so eigentlich davon ausgehen können, dass auch Software, die wir regelmäßig verwenden, dementsprechend schneller geworden ist. Jeder, der regelmäßig mit Software interagiert, weiß jedoch um das Ausbleiben einer solchen Entwicklung in der Breite – eher das Gegenteil ist der Fall. Wir erwarten eigentlich sogar, dass Software unzuverlässig oder langsam ist.

Bemerkenswert ist ebenfalls, dass Single-Core-Performance und Taktfrequenz zu stagnieren scheinen, während die Anzahl an Transistoren und Kernen stetig exponentiell steigt. Die Single-Core-Performance ist nach Amdahl'schem Gesetz [14] immer relevant – die meisten Probleme sind nicht endlos parallelisierbar. Wir verzeichnen sowohl auf Software-, als auch auf Hardwareebene eine Entwicklung nach dem gleichen Profil. Moderne Softwareinnovation bedarf Hardwareverbesserungstrends, deren Fortbestehen allerdings nicht sicher ist. Auf Hardwareseite sind dies größtenteils physikalische Limitierungen, aber auf Softwareseite sind sie anderer Natur.

Im kulturellen Corpus rund um die Softwareentwicklung stößt man oft auf immer wiederkehrende Ideen und etablierte Vorstellungen darüber, wie eben dieser Prozess ablaufen sollte. Der stetige Aufstieg auf der Leiter der Abstraktionen wird als allgemein gut verstanden. Technische Fragen rund um Implementierungsdetails, z.B. darüber wo der Speicher, der verwendet wird, eigentlich herkommt, seien lästig und hielten einen in Designfragen nur auf. Im Zuge dessen haben sich Metriken und Leitlinien für Codequalität entwickelt, die fast jeder in seiner Bildungslaufbahn zu Gesicht bekommt: Konzepte wie SOLID, DRY und diverse „Clean Code" Prinzipien dabei an erster Stelle. Die meisten dieser Konzepte haben allerdings auch viele Nachteile, allen voran im Punkto Performance. Das sukzessive Missachten solcher Konzepte kann hier ohne weitere Optimierungen in einem Minimalbeispiel eine Geschwindigkeitsverbesserung um einen Faktor 15 erreichen [15]. Die Nutzung jener Konzepte entspricht hier einem Rückschritt in Sachen Hardwareleistung auf den Stand von 2008. Algorithmen an sich sind nicht das Problem, sondern die Art und Weise wie diese miteinander verknüpft sind. Das Thema Performance wird in einigen Teilen der Softwarewelt häufig nicht derart ernst genommen; es gäbe nicht wirklich ein akutes Problem. Aber es gibt nicht nur Performance-Implikationen. Jeder einzelne Entwickler

hat so ein zunehmend weniger globales Verständnis von dem, woran er arbeitet. Das macht die Kompetenzverteilung zunehmend fragiler und fördert Risiken.

Für diese Entwicklung im Bereich Software gibt es viele Entschuldigungen. Es gäbe keinen Bedarf sich über Softwareperformance Sorgen zu machen, da Hardware und Compiler so gut seien, dass die Benutzererfahrung am Ende immer ausreichend schnell ist, unabhängig von der Umsetzung. Selbst wenn es einen Unterschied macht, sei dieser zu klein, als dass er den Mehraufwand rechtfertigen würde, den es verursacht, sich über Performance Gedanken zu machen – vor allem, wenn es um finanziellen Mehraufwand geht. Die Realität ist allerdings nicht so eindeutig [16]. Im Jahr 2010 kündigte Facebook an, sie wollten ihr Produkt „doppelt so schnell machen“. Sie hatten Untersuchungen angestellt, bestätigt von Google und Microsoft, die zeigen, dass Benutzer der Services mehr Seiten besuchten und höheren Mehrwert aus ihrer Erfahrung zogen, wenn sie schneller war. Dieses organisationsweite Projekt umfasste nicht nur Arbeit an einigen wenigen Hotspots, sondern auch die Entwicklung komplett neuer Systeme und vollständige Rewrites ganzer Komponenten – extrem aufwendig. Es finden sich viele weitere solcher Beispiele. Es ist also durchaus wirtschaftlich, sich Gedanken um Performance zu machen. Abgesehen davon ist die Entwicklung auf Makroebene, die hier in Abschnitt 3 thematisiert wird, nicht eingepreist – es gibt kein allgemeines Bewusstsein dafür.

Die versprochene höhere Produktivität beim Festhalten an diesen Ideen ist auch nicht wirklich identifizierbar. Große Firmen stellen mehr und mehr Entwickler ein, aber der Produktumfang spiegelt das nicht gleichermaßen wieder. Die Produktivität pro Entwickler sinkt. Hier sind diverse Faktoren relevant.

3.3 Dynamiken großer Projekte

Das Amdahl’sche Gesetz ist eines der Grundprinzien, die die Softwarewelt leitet [17].

$$T(n) = b + \frac{T(1) - b}{n}$$

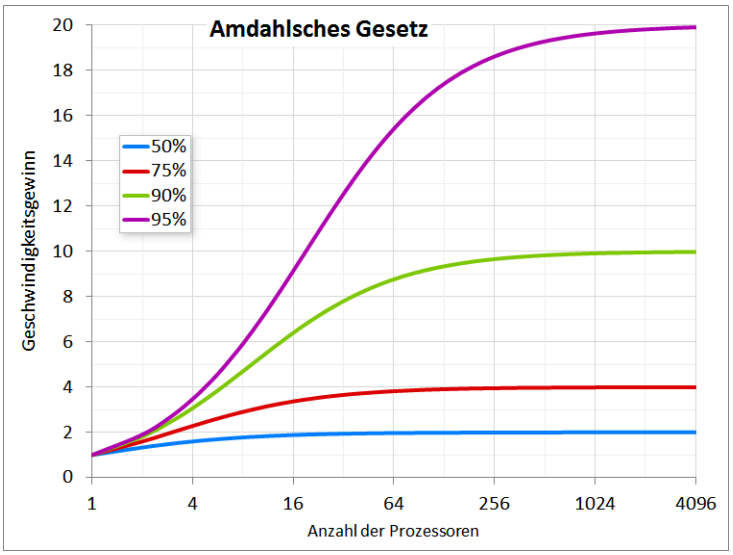
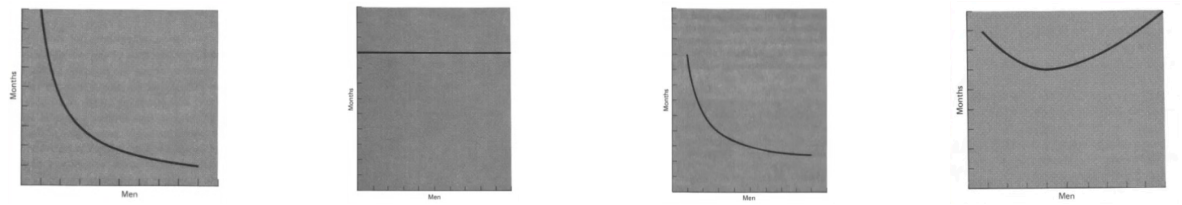


Abbildung 3: Amdahlsches Gesetz Graph [18].

Die Gesamtlaufzeit T in Abhängigkeit von der Anzahl parallel arbeitender Parteien (Threads beispielsweise) n strebt an und ist immer beschränkt durch die Laufzeit des nicht parallelisierbaren Teils b (Abbildung 3). An der Genauigkeit dieses Modells gibt es auch Kritik [14]. Beispielsweise hat die Menge an für die CPU verfügbarem schnellen Cachespeicher auch eine Auswirkung auf die Schnelligkeit eines Prozesses. Die generellen Dynamiken bleiben jedoch die selben.

Diese sind allerdings nicht beschränkt auf die Softwarewelt, sondern haben auch direkte Korrespondenzen im Projektmanagement. Hierfür betrachten wir das Brook'sche Gesetz. Fred Brook veröffentlichte 1975 erstmals das Buch „The Mythical Man-Month“ [19], in dem er eine typische Reaktion seitens Managements großer Projekte, bei Abweichungen vom Zeitplan mehr Leute auf ein Projekt anzusetzen, behandelt. Zentral ist das Konzept des „Man-Month“, basierend auf der Idee, eine Arbeitskraft sei gleich einer Ressource, die man in ein Projekt investieren und so die Laufzeit beeinflussen kann. Wenn komplexe Probleme allerdings Kommunikation zwischen Teams erfordern, dann können mehr Arbeitskräfte auch kontraproduktiv sein. In Abhängigkeit von der Höhe dieses Kommunikationsniveaus ergeben sich dann unterschiedliche Zusammenhänge (Abbildung 4).



Keine Kommunikation. | Keine Parallelisierung. | Kommunkikationsbarriere. | Erhöhter Kommunikationsbedarf.

Abbildung 4: Einfluss Interteamkommunikation und Parallelisierung [19].

Wenn zwischen Teams kein Kommunikationsbedarf besteht, dann kann jeder an dem ihm zugeteilten Part arbeiten und die mögliche Laufzeitreduzierung ist mehr oder weniger unerschöpflich. Wenn alle Arbeitsschritte sequenziell ablaufen müssen und keine Möglichkeit zur Parallelisierung besteht, dann haben mehr Arbeitskräfte keinen Einfluss. Es kann aber auch sein, dass es einen gewissen sequenziellen Teil gibt oder einen Kommunikationsflaschenhals, der zwar Verschnellerung erlaubt, aber eine Schranke beinhaltet. Dieser Graph ist vergleichbar mit dem Amdahlschen Gesetz, wenn man ihn auf den Kopf dreht. Im schlimmsten Fall führt die zusätzliche Menge an Arbeitskräften zu einer erhöhten Kommunikationslast, die zu einer im Vergleich zur Ausgangslage sogar erhöhten Laufzeit führt. Zentral, vor allem bei komplexeren Projekten, ist immer der Teil, der nicht von mehr Arbeitskraft profitiert, da dieser die Gesamtlaufzeit definiert.

Wenn man sich die Frage stellt, wie genau dieser Teil nun zum Tragen kommt, dann hilft ein Blick auf ein weiteres Gesetz: das Conway'sche Gesetz. Grundlage ist hier die Arbeit von Melvin E. Conway [20]. Sie befasst sich mit der Interaktion zwischen den Organisationsstrukturen großer Unternehmen und der Architektur der Produkte, die daraus hervorgehen. Kleine Teams und im Extremen Einzelpersonen unterliegen keinem Einfluss von Kommunikationsgeschwindigkeit. Sie können mehr oder weniger unendlich schnell Ideen austauschen und ihre Arbeit koordinieren. Wenn man jedoch große Projekte betrachtet, die viele Arbeitskräfte benötigen, dann muss schon

vor dem Start des Designprozesses eine ganze Reihe an Vorannahmen über die Architektur des Produkts getroffen werden, auch wenn vielleicht noch gar nicht bekannt ist, welcher Aufbau am Ende sinnvoll ist. Das ist vor allem relevant für das Organigramm bzw. den Teamaufbau des Projekts. Das bedeutet, man definiert vorab in welcher Struktur man arbeitet, obwohl man, wenn man an einem halbwegs bedeutungsvollen Projekt arbeitet, noch nicht weiß, welche das sein wird. Durch diese Strukturierung werden auch vorab automatisch Urteile darüber gefällt, welche Kommunikationswege mehr Reibung haben als andere. Die Produktarchitektur gleicht sich so der Architektur der Teamorganisation an. Dies ist maßgeblich dadurch beeinflusst, dass die Kommunikation und Koordination zwischen Teams immer aufwendiger ist, als innerhalb eines Teams. Je mehr Leute nun an einem Projekt arbeiten, desto mehr Kompromisse müssen eingegangen werden, die sich auch in der Software selber niederschlagen: Festlegung von Interfaces zwischen verschiedenen Komponenten, und wie Änderungen an bestehendem Code koordiniert werden (z.B. Versionsverwaltung).

In so einem Umfeld ist diese Organisationsstruktur aber nur der asymptotisch bestmögliche Richtwert für die Produktarchitektur, der erreicht werden kann. Nun kommt noch eine weitere Dimension hinzu: zeitliche Entwicklung und Legacy Code. Wenn eine Organisationsstruktur verändert wird, dann ist die resultierende Produktarchitekturänderung nicht nur ein Abbild der neuen Struktur, sondern berücksichtigt auch bereits lange existierende Produktstrukturen. Diese Verschmelzen dann in gewisser Weise miteinander und schaffen Querverbindungen (Abbildung 5).

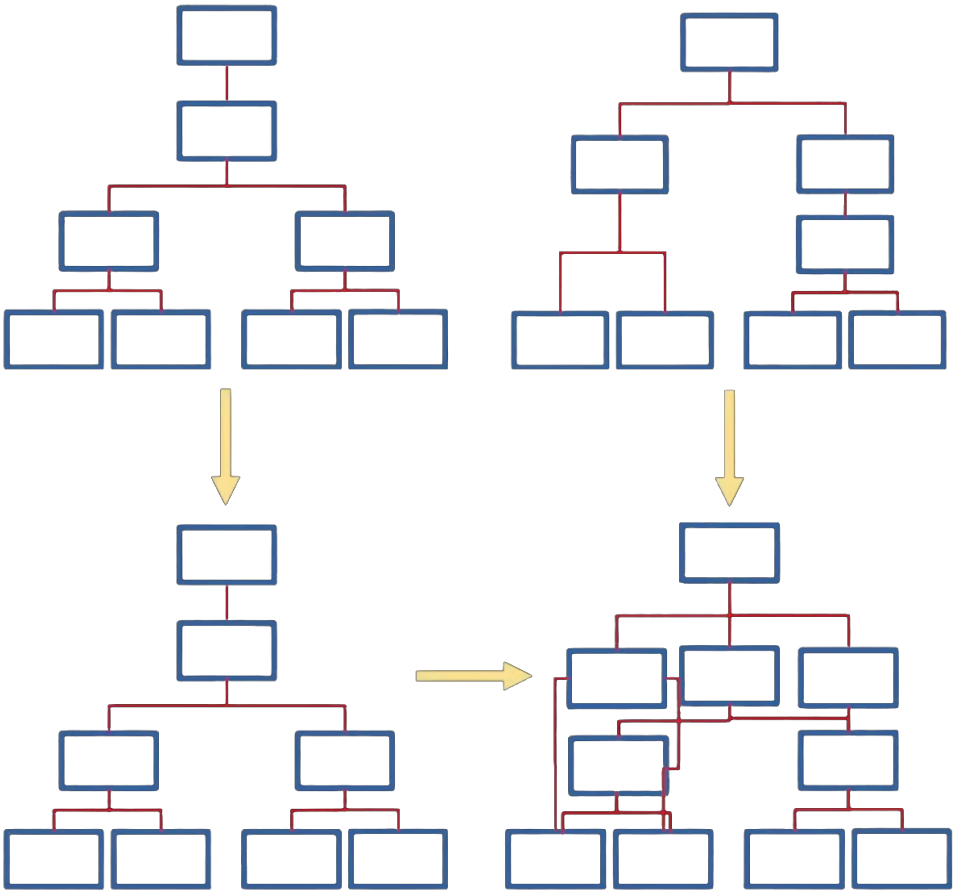


Abbildung 5: Restrukturierung mit Legacy-Architektur [17].

Das hat mehrere Effekte. Zum einen werden die gleichen Funktionen unter Umständen an mehreren Stellen gleichzeitig implementiert, da die Abhängigkeiten zu groß sind, um alles zu vereinheitlichen. Zum anderen ist die Aufteilung der Zuständigkeiten zwischen den Teams nicht mehr optimal. Ein praktisches Beispiel hierfür sind die vier verschiedenen Lautstärkeregler in Windows; alle unabhängig voneinander und ursprünglich aus verschiedenen Komponenten und Windowsversionen.

3.4 Einfluss von Softwarearchitektur

Diese Effekte werden allerdings noch verstärkt durch die Softwarearchitektur innerhalb der Teams. Konzepte wie objektorientierte Programmierung haben exakt die gleichen Dynamiken inne und sind fester Teil der aktuellen Softwarekultur. Genau dieser Prozess der Verantwortungsaufteilung wird als gut empfunden, dabei ist er eher ein notwendiges Übel. Das führt dazu, dass die Strukturen innerhalb einzelner Knoten in Abbildung 5 nach dem selben Prinzip geformt werden. Ein bezeichnendes Beispiel ist hier das Klassendiagramm der Operator-Klasse in LLVM in Abbildung 6. Kritiker zweifeln an der Nachhaltigkeit davon [17].

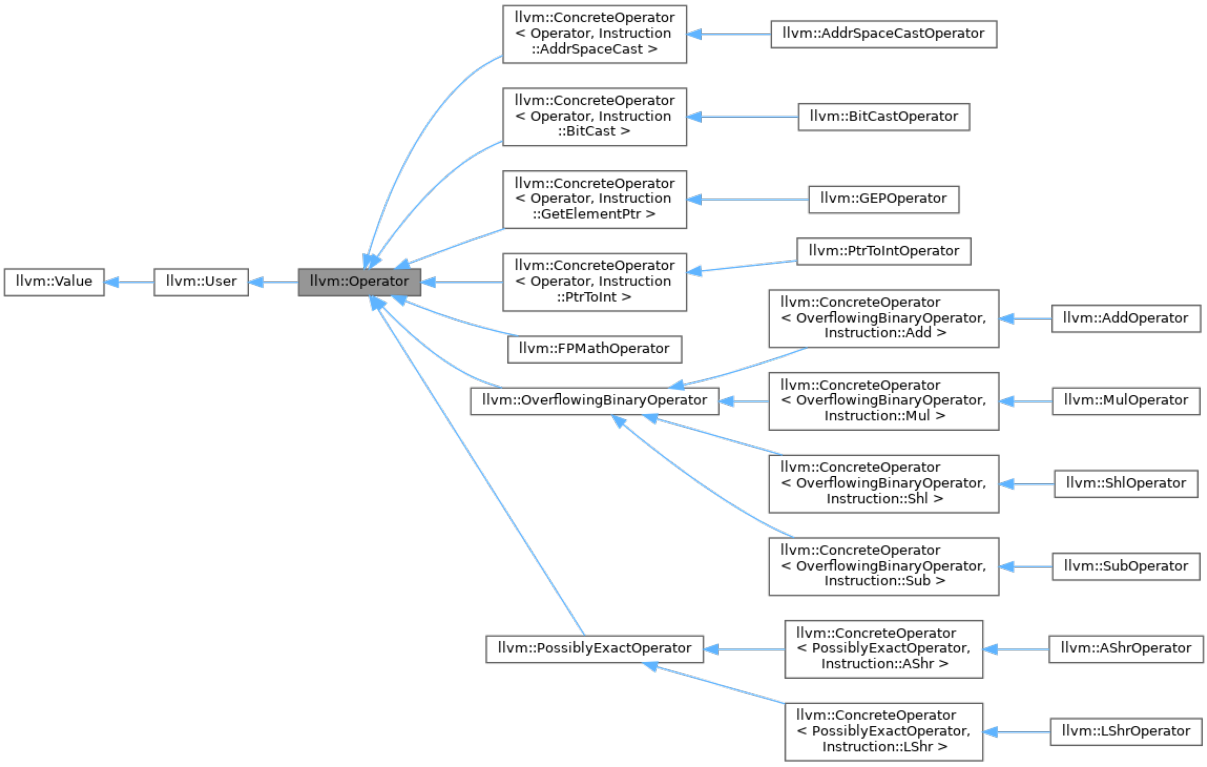


Abbildung 6: Klassendiagramm Operator-Klasse LLVM [21].

Dazu kommt, dass die modern Produktwelt geprägt ist von ähnlichen Strukturen, sodass neue Produkte sich zusätzlich auch in bestehende Umgebungen solcher Art einbetten müssen: Engines, Container, native Programme mit Webbasis, Installer und dergleichen. Gegen solche Dynamiken anzugehen ist mit Risiko verbunden und wenn der Ertrag nicht als ausreichend wertvoll erachtet wird, dann ist jeder Manager gut beraten, das Risiko zu minimieren. So erhält man einen sich selbst verstärkenden Effekt.

Es gibt hardware- und organisationsstrukturbedingte Limitierungen, die nicht einfach zu überwinden sind. Kulturelle Trends, vor allem in der Softwareentwicklung, riskieren ungeachtet dessen einen zivilisatorischen Zerfall. Software in der breiten Masse ist dafür bekannt, nicht richtig zu funktionieren oder extrem langsam zu sein, obwohl wir in den letzten Jahrzehnten unglaubliche Fortschritte in der Hardware erleben durften und Computer unfassbar schnell sind. Nach direkten Beweisstücken muss man nicht lange suchen. Es reicht seinen Alltag zu bestreiten und einfach genau hinzuschauen. Auf der anderen Seite muss ein Neuling, der sich im Internet über Engine-Technologie in der Videospielwelt erkundigen will und lernen will, wie solche performancekritischen Systeme im Detail funktionieren, sich anhören, dass er seine Zeit verschwende. Nachwuchskräften mangelt es zunehmend an technischem Grundverständnis und Begriffe wie „Vibe-Coding“ kommen vermehrt im Internet auf. Am Ende muss die Gesellschaft für sich entscheiden, welchen Einfluss man in der Welt haben möchte, und daraus die nötigen Schlüsse ziehen.

Literatur

[1] Wikipedia, [Online]. Verfügbar unter: https://en.wikipedia.org/wiki/Airbus_A320neo_family

[2] V. Shrivastava, „A study on the crash of Boeing 737 MAX“.

[3] J. Herkert, J. Borenstein, und K. Miller, „The Boeing 737 MAX: Lessons for engineering ethics“, *Springer*.

[4] P. Johnston und R. Harris, „The Boeing 737 MAX saga: lessons for software organizations“, *American Society for Quality*.

[5] Seattle Times, [Online]. Verfügbar unter: https://www.seattletimes.com/seattle-news/times-watchdog/the-inside-story-of-mcas-how-boeings-737-max-system-gained-power-and-lost-safeguards/

[6] A. Mhatre, C. Kamble, A. Shah, und S. Malim, „Reviving The Boeing Project By Using Project Management Methodologies“.

[7] N. Frost, „The 1997 merger that paved the way for the Boeing 737 Max crisis“.

[8] G. P. Shea und G. Allon, „Boeing: wait, there's more“, *Emerald Publishing Limited*.

[9] Wikipedia, [Online]. Verfügbar unter: https://de.wikipedia.org/wiki/Wettlauf_ins_All

[10] DevGAMM, „Preventing the Collapse of Civilization / Jonathan Blow (Thekla, Inc)“. [Online]. Verfügbar unter: https://youtu.be/ZSRHeXYDLko

[11] NCASVideo, „1177 BC: The Year Civilization Collapsed (Eric Cline, PhD)“. [Online]. Verfügbar unter: https://youtu.be/bRcu-ysocX4

[12] [Online]. Verfügbar unter: https://www.sigmicro.org/media/oralhistories/colwell.pdf

[13] K. Rupp, „42 Years of Microprocessor Trend Data“. [Online]. Verfügbar unter: https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/

[14] Wikipedia, [Online]. Verfügbar unter: https://de.wikipedia.org/wiki/Amdahlsches_Gesetz

[15] C. Muratori, „'Clean' Code, Horrible Performance“. [Online]. Verfügbar unter: https://youtu.be/tD5NrevFtbU

[16] C. Muratori, [Online]. Verfügbar unter: https://www.computerenhance.com/p/performance-excuses-debunked

[17] C. Muratori, „The Only Unbreakable Law“. [Online]. Verfügbar unter: https://youtu.be/5IUj1EZwpJY

[18] C. B.-S. 4. Von Frank Klemm - Eigenes Werk, „Wikimedia“. [Online]. Verfügbar unter: https://upload.wikimedia.org/wikipedia/commons/1/1e/Amdahl.png

[19] F. P. Brooks Jr, *The mythical man-month: essays on software engineering*. Pearson Education.

[20] M. E. Conway, „How do committees invent“.

[21] LLVM Documentation, [Online]. Verfügbar unter: https://llvm.org/doxygen/classllvm_1_1Operator.html