Sonderdruck aus OBJEKTspektrum

Die Wachstumsrate von Software

Wächst Software analog zu dem für Prozessoren geltenden mooreschen Gesetz?

Die Leistungsfähigkeit von Prozessoren verdoppelt sich alle 18 Monate. Aber wie verhält es sich eigentlich mit dem Wachstum von Software? Les Hatton, Diomidis Spinellis und Michiel van Genuchten sind der Frage nachgegangen und haben hierzu eine interessante Studie im Journal of Software veröffentlicht, welche wir hier vorstellen. Das Ergebnis lautet übrigens tatsächlich 42.

Vereinfacht ausgedrückt, besagt das mooresche Gesetz, dass sich die Leistungsfähigkeit von Computer-Prozessoren etwa alle 18 Monate verdoppelt (**siehe Abbildung 1**). Leistungsfähigkeit wird hierbei mit der Anzahl von Transistoren gleichgesetzt, was zwar eine Vereinfachung ist, aber eine allgemein akzeptierte. Das „Gesetz" ist auch eher als Faustformel zu verstehen, welche zwar einerseits empirisch belegt ist, sich aber andererseits auch einem Ende zu nähern scheint (vgl. [MITTR]).

Für Software fehlte so eine allgemein anerkannte Faustformel bislang, wie beispielsweise auch die New York Times in [TNYT] feststellte: *„There are no such laws in software"*. Gegenfalls wäre hier das Wirthsche Gesetz (vgl. [Wir95]) anzuführen, welches aber nur recht allgemein feststellt, dass *„die Software schneller langsamer wird, als die Hardware schneller wird"*. Demgegenüber stellte Marc Andreessen fest: *„Software is eating the world"* (vgl. [And11]).

Wie soll aber die Leistungsfähigkeit von Software beurteilt werden? Vielleicht durch die Anzahl ihrer Features? Dann stellt sich gleich die Frage, wie man Features misst und vergleicht, womit man zu aufwendigen Verfahren, wie etwa der Function Point-Analyse, kommt. Nach unserer Erfahrung wird selbige nur selten eingesetzt, weshalb sie sich für einen empirischen Nachweis vermutlich nicht eignet.

Natürlich ist Softwareentwicklung komplex und lässt sich nur schwerlich auf eine einfache Metrik reduzieren. Eine Vereinfachung analog zum mooreschen Gesetz könnte aber vielleicht auch für Software eine Lösung sein.

Michiel van Genuchten und Les Hatton zeigten einen entsprechenden Ansatz erstmals 2012 in ihrem Artikel „Compound Annual Growth Rate for Software" im IEEE Software Magazin (vgl. [Gen12]) auf.

Lines of Code

Die Autoren umgehen die Frage der Leistungsfähigkeit von Software hier und messen stattdessen, wie sich die Codebasis von verschiedenen Softwaresystemen hinsichtlich ihrer Größe in Anzahl der Codezeilen oder auch „Lines of Code" (LoC) über die Zeit entwickelt. In Anlehnung an die Anzahl Transistoren des mooreschen Gesetzes ist dieser Ansatz durchaus nachvollziehbar.

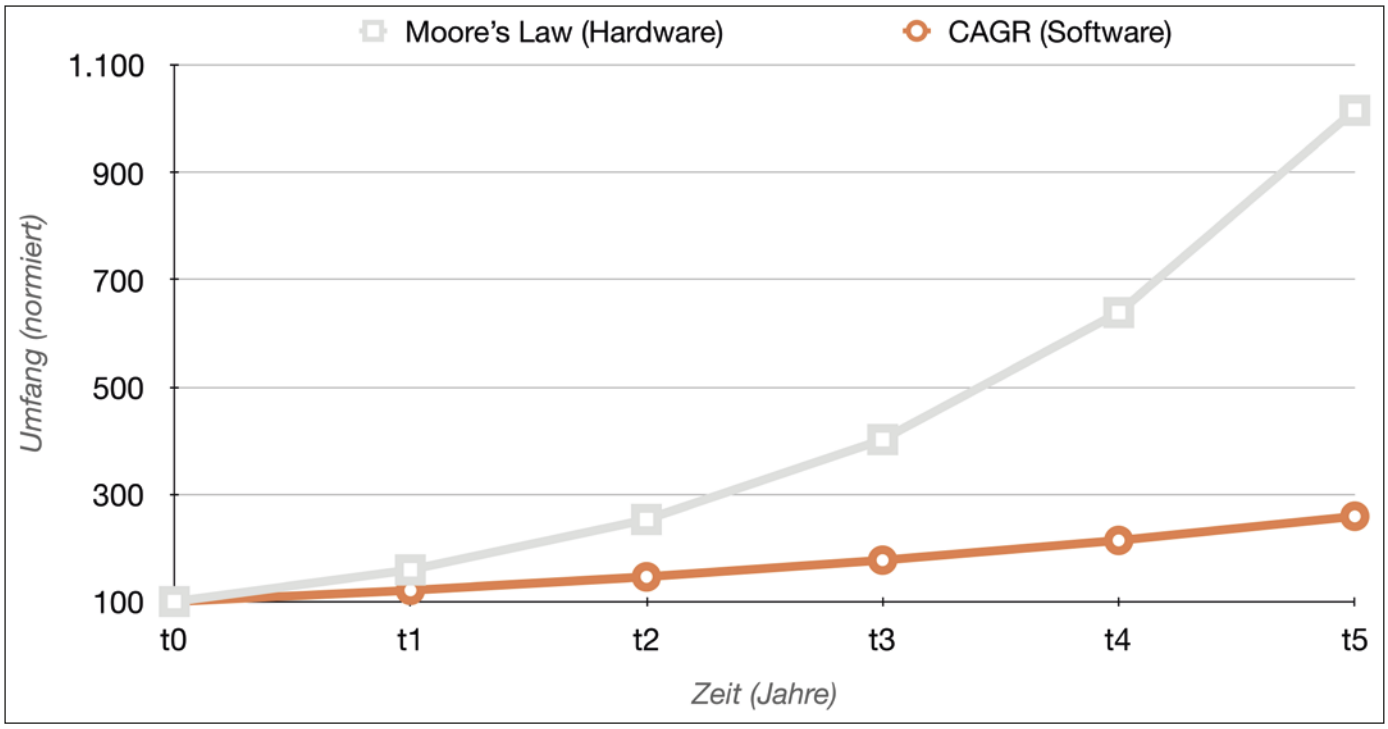

Abb. 1: Vergleich Moore's Law und Wachstumsrate der Software

Software	Art	Zeitraum	Anzahl Jahre	Umfang in Millionen LOC	CAGR
Magnetresonanz-tomograph	Closed Source	2002-2010	8	3,5 -> 9,5	1,13
Büro-Kopierer	Closed Source	1998-2008	10	2 -> 10,5	1,18
KFZ-Navigation	Closed Source	2004-2011	7	1,5 -> 6,5	1,23
Ölförderung	Closed Source	1993-2012	19	0,3 -> 2,2	1,11
Workflow-Management	Closed Source	2004-2011	7	0,02 -> 0,125	1,29
Flug-Management	Closed Source	1978-2010	22	0,05 -> 1	1,15
GIMP	Open Source	1999-2008	9	0,309 -> 0,903	1,12
Safer C Toolset	Open Source	2001-2006	5	0,035 -> 0,065	1,12

Tabelle 1: Übersicht der initial untersuchten Softwaresysteme

Die zugrunde liegende Codebasis umfasste zwei Open- und sechs Closed-Source-Systeme (**siehe Tabelle 1**). Wie anhand der Ausgangsgröße erkennbar, befanden sich alle untersuchten Softwaresysteme hinsichtlich ihres Lebenszyklus durchweg in der Wartungs- beziehungsweise Weiterentwicklungsphase.

Dies reichte zwar vielleicht noch nicht für eine statistisch relevante Aussage, das Ergebnis erschien mit einem Faktor von 1,16 aber schon recht stabil. Demnach verdoppelt sich Software also ungefähr alle 5 Jahre im Umfang:

"If we don't know anything about a certain software system at hand, as a good starting point, we could assume it has a CAGR [siehe weiter unten] of the median value 1.16 and therefore doubles in size every four to five years."

Interessanterweise stellen wir bei Diskussionen um die Messgröße Lines of Code in der Softwareentwicklung häufig sehr dogmatisch ablehnende Haltungen fest, ohne dass dann aber Alternativen aufgezeigt werden können. Natürlich sind Lines of Code eine Vereinfachung der komplexen Softwareentwicklungswelt. Aber alle Metriken sind Reduzierungen und können und wollen nicht die ganze Komplexität in einer Größe zeigen – es sind Modelle.

Wenn es um Fitness oder Gesundheit geht, schauen wir uns genauso vereinfachend zunächst häufig das Gewicht in Kilogramm einer Person an. Was aber nun, wenn jemand mehr Sport macht, dabei an Muskelmasse zunimmt und entsprechend sein Gewicht erhöht; ist das jetzt gut oder schlecht? Auch hier reicht das eindimensionale Körpergewicht einer Person als Kennzahl alleine nicht mehr aus, ist aber ein guter Einstieg in die Diskussion und weitere Analyse. Genauso sollten wir es mit der Kennzahl LoC in der Softwareentwicklung halten.

Compound Annual Growth Rate

Die Compound Annual Growth Rate (CAGR) oder jährliche Wachstumsrate (vgl. [Wiki]) ist eine „in der Betriebswirtschaft und Volkswirtschaft […] wesentliche Kennziffer". Sie ist insbesondere im Finanzsektor etabliert, um zum Beispiel das komplexe Wachstumsverhalten von Finanzinstrumenten abzubilden. Da die CAGR auch Eigenschaften aufweist, die für das komplexe Verhalten von Software relevant sind, spricht vieles dafür, sie entsprechend zu übernehmen:

„Sie stellt das durchschnittliche jährliche Wachstum einer zu betrachtenden Größe dar. […] Tatsächliche Ausschläge der Folgejahre in der Zwischenzeit wirken sich dabei nicht aus, die Wachstumsrate ist konstant."

Auf Software angewandt, gleichen sich Effekte, wie zum Beispiel durch Refactoring-Maßnahmen, welche die Codebasis kurzzeitig in die eine oder andere Richtung verändern können, ohne entsprechende Auswirkung auf die Funktionalität zu haben, dem folgend über die Zeit auch wieder aus.

Follow-up

Da dem eingangs erwähnten Artikel noch nicht genug statistisch relevantes Material für eine empirische Aussage zugrunde lag, haben Les Hatton und Michiel van Genuchten, diesmal erweitert um Diomidis Spinellis, 2017 "The long-term growth rate of evolving software: Empirical results and implications" (vgl. [Hat17]) veröffentlicht. Darin ermitteln sie die CAGR für über 404 Millionen Zeilen Code; wieder sowohl aus Open- als auch Closed-Source-Softwaresystemen.

Für Open-Source-Software wurde dabei auf öffentliche Repositories von Github, Launchpad und Mercurial mit insgesamt 124 GB Quellcode zurückgegriffen. Darin waren unter anderem Eclipse, Git, KDE, MongoDB, MySQL, Perl, PHP, Ruby, Spring und WINE enthalten. Die Daten zu Closed-Source-Software kamen überwiegend im Rahmen der Auswertung von existierenden Studien zusammen.

Mit diesem erweiterten Datensatz konnten die Autoren die jährliche Wachstumsrate des Quellcodes von Softwaresystemen auf 1,21 präzisieren. Ohne weitere Kenntnis über ein System kann man demzufolge davon ausgehen, dass *ein Softwaresystem den Umfang seines Quellcodes alle 42 Monate verdoppelt* (**siehe auch Abbildung 1**).

Interessant sind hier auch die in der Studie ausgewiesenen Unterschiede zwischen einzelnen Produkten und Branchen. Sun Solaris von Oracle und somit ein Closed-Source-System weist nur eine CAGR von 1,09 beziehungsweise eine Verdoppelung alle 8 Jahre auf, wohingegen der Linux Kernel (Open Source) auf eine CAGR von 1,32 kommt und sich somit schon nach 2,5 Jahren verdoppelt hat. Die Autoren konnten ansonsten keine Beziehung zwischen der CAGR und der Art der Software (Open vs. Closed Source) herstellen. Software im sicherheitskritischen Bereich wächst aber der Untersuchung zufolge mit einer CAGR eines Flugmanagementsystems von 1,10 nachvollziehbar langsamer als beispielsweise eine Navigationssoftware mit 1,23. Die DNS-Implementierung BIND ist mit 1,07 wiederum eines der stabilsten der untersuchten Systeme.

Anwendung

Software wächst – um wie viel genau ist individuell unterschiedlich. Im Schnitt kann man aber davon ausgehen, dass der Umfang an Quellcode, den man in der Wartungs- und Weiterentwicklungsphase eines Softwaresystems zu verwalten hat, pro Jahr um mehr als 20 Prozent zulegt. Darauf muss man sich als Entwicklungsleiter einer Softwareeinheit strukturell einstellen, sei es hinsichtlich Architektur, Infrastruktur, Mitarbeitern oder ganz allgemein mit Blick auf das Budget.

Bezogen auf eine aktuelle Microservices-Softwarearchitektur-Landschaft kann ich also davon ausgehen, nächstes Jahr entsprechend 20 Prozent mehr Microservices verwalten zu müssen. Funktioniert mein anfänglich vielleicht zunächst eher statischer Ansatz zum Service Discovery dann noch? Und was ist mit Logging und Tracing? Benötige ich jetzt eventuell ein Service Mesh? Im Wissen um das Wachstum meiner Software lässt sich solchen Fragestellungen der Überraschungseffekt nehmen.

Carola Lilienthal schreibt in [Lil17], dass in einem durchschnittlich komplexen Java-Softwaresystem *„pro 500.000 LOC die Kapazität von einem bis zwei Vollzeitentwicklern für die Wartung gebraucht wird“*. Eine Zahl, die wir aus unserer Praxis bestätigen können. Wächst eine Anwendung von 500.000 auf 1.000.000 LOC, sollte also auch die Entwicklungsmannschaft entsprechend auf zwei bis vier Vollzeitentwickler anwachsen.

Software besteht heute üblicherweise zum Großteil aus Fremdcode, wie wir in **Abbildung 2** am Beispiel der freien, webbasierten Projektmanagementsoftware Redmine exemplarisch aufzeigen.

Die eigentliche Anwendung ist hier mit 200.000 Codezeilen beziehungsweise 1 Prozent des Gesamtumfangs nur die Spitze des Eisbergs. 99 Prozent der Anwendung sind Fremdcode. Die Abhängigkeiten machen die Situation dabei besonders brisant: Löst man die rund 1.000 Abhängigkeiten von Kubernetes auf, so wächst allein diese Software schon auf stolze 35 Millionen Codezeilen an, wird also größer als das Betriebssystem.

Muss ich den Fremdcode also in die Wachstumsbetrachtung mit aufnehmen? Unserer Einschätzung nach nur bedingt. Natürlich will auch dieser Code verwaltet und insbesondere aktuell gehalten werden. Für die Personalplanung reicht aber der Blick auf den eigenen Code. Die Statistik des Redmine-Repositories unter [REDM] zeigt entsprechend, dass deren Codebasis zu rund drei Viertel von einem Entwickler gewartet wird. Das restliche Viertel steuern eine Handvoll weitere Entwickler bei. Überschlagsweise können wir hier also 1,5 Vollzeitentwickler für die 200.000 LOC ansetzen, was sich mit der in [Lil17] genannten Zahl deckt,

Literatur & Links

[And11] M. Andreessen, Why Software Is Eating the World, siehe: **https://a16z.com/2016/08/20/why-software-is-eating-the-world/**

[Cha18] O. Chang, H. Lipson, Self-replicating Neural Networks, in: Neural Network Quine, siehe: **https://arxiv.org/abs/1803.05859**

[FOR13] You Will Not Beat The Market, Virtually No One Does, siehe: **https://www.forbes.com/sites/mitchelltuchman/2013/10/25/you-will-not-beat-the-market-virtually-no-one-does/**

[Gen12] M. van Genuchten, L. Hatton, Compound Annual Growth Rate for Software, in: IEEE Software, vol. 29, no. 4, pp. 19-21, 2012, siehe: **http://doi.ieeecomputersociety.org/10.1109/MS.2012.79**

[Hat17] L. Hatton, D. Spinellis, M. van Genuchten, The long-term growth rate of evolving software: Empirical results and implications, in: J Softw Evol Proc, vol. 29, no. 5, May 2017, e1847, siehe: **https://doi.org/10.1002/smr.1847**

[Lil17] C. Lilienthal, Langlebige Software-Architekturen, dpunkt.verlag GmbH, 2017

[MITTR] T. Simonite, Moore's Law Is Dead. Now What?, MIT Technology, 13.3.2016, siehe: **https://www.technologyreview.com/s/601441/moores-law-is-dead-now-what/**

[TNYT] The New York Times, Software Progress Beats Moore's Law, siehe: **https://bits.blogs.nytimes.com/2011/03/07/software-progress-beats-moores-law/**

[REDM] Statistik des Redmine-Repositories, siehe: **https://www.redmine.org/projects/redmine/repository/statistics**

[Wiki] CAGR, siehe: **https://de.wikipedia.org/wiki/Wachstumsrate#Jährliche_Wachstumsrate_(Compound_Annual_Growth_Rate)**

[Wir95] N. Wirth, A plea for lean software (Wirthsches Gesetz), in: IEEE Computer, Feb. 1995, siehe: **https://ieeexplore.ieee.org/document/348001**

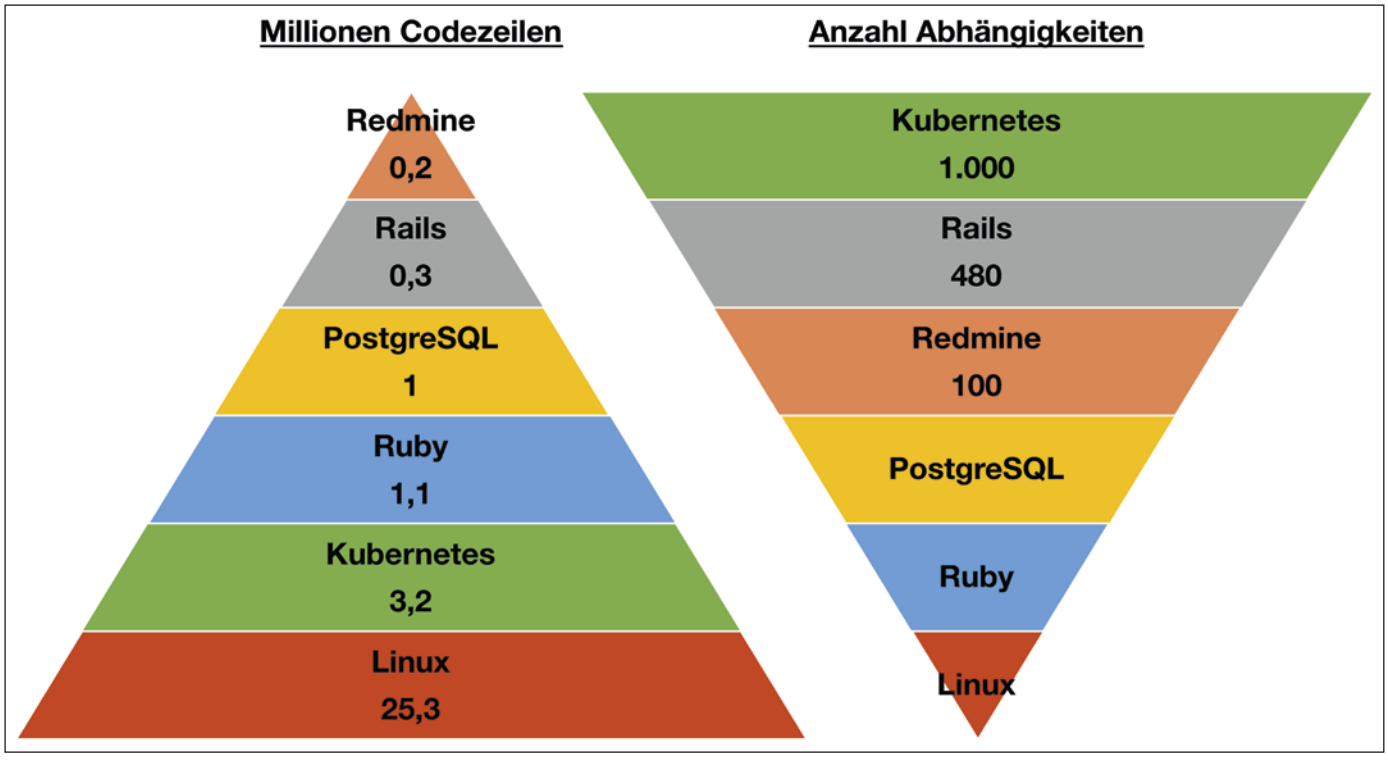

Abb. 2: Umfang und Abhängigkeiten eines exemplarischen Software-Stacks

auch wenn der Fremdcode, auf dem Redmine aufsetzt, um ein Vielfaches größer ist.
Aus der Finanzwelt weiß man, dass niemand den Markt schlägt (vgl. FOR13]), man sein Geld also beispielsweise besser in einen passiv gemanagten Indexfonds anlegt, anstatt es einem aktiv gemanagten Fond anzuvertrauen. Auf die Softwareentwicklung umgemünzt, sollte beispielsweise eine Roadmap, welche eine Verdoppelung der Software im kommenden Jahr voraussetzt, sehr kritisch hinterfragt werden. Was genau werden wir anders machen, um soviel besser als der „Markt“ zu sein?
Während der initialen Implementierung oder Entstehung eines Softwaresystems sind sicherlich auch noch größere Wachstumsraten üblich. Die Datenbasis der Studien fußt auf reifen Systemen, die sich in der Weiterentwicklungsphase befinden. Dementsprechend sollte die CAGR für Systeme, die sich in der initialen Entstehungsphase befinden, wenn überhaupt, nur bedingt zum Einsatz kommen.

Ausblick

Moore‘s Law nähert sich einem physikalischen Ende (vgl. [MITTR]). Wie aber beispielsweise auch in dem weiter oben erwähnten Artikel der New York Times (vgl. [TNYT]) formuliert wurde, hat Software keine solchen Begrenzungen:
“Software […] is pure abstraction – a building material without material constraints.”
Somit ist ein Ende des Wachstums von Software nicht absehbar. Die hier vorgestellte Wachstumsrate erfasst dabei nur, wie ein einzelnes System wächst. Die Menge an Software insgesamt wird noch wesentlich stärker wachsen: “Software is eating the world” (vgl. [And11]). Da Software sich absehbar auch noch selbst vervielfältigt (vgl. [Cha18]), dürften wir vor einer Art „kambrischen Explosion“ der Software stehen.

Fazit

Software wächst zwar langsamer als Hardware, aber sie verdoppelt ihren Umfang im Schnitt doch auch alle 42 Monate. Dies konnten die Autoren der Studie (vgl. [Hat17]) empirisch nachweisen. Damit haben Beteiligte der Softwareentwicklung mit der CAGR für Software eine Faustformel an der Hand, die eine ähnliche Qualität wie das mooresche Gesetz für Hardware aufweist.
Wie wir gezeigt haben, kann das Wissen um diese Wachstumsrate helfen, die Softwarearchitektur und -infrastruktur entsprechend auszurichten. Dem Management kann es bei der Budget- und Personalplanung helfen. Außerdem bietet sich die Kennzahl als Einstiegspunkt für Diskussionen bei individuellen Abweichungen, wie zum Beispiel überzogenen Roadmap-Erwartungen, an. ||

Der Autor

Michael Schwarze
(michael.schwarze@schwarze-consulting.de) entwickelt seit dem VC 20 mit Begeisterung Software. Mit seiner Firma Schwarze Consulting GmbH hilft er heute Unternehmen im Zuge der Digitalisierung, besser Software zu entwickeln und die Softwareentwicklung besser zu managen.