Warum ist Software-Entwicklung schwierig?

Software ist in den vergangenen Jahrzehnten zu einem wichtigen Faktor geworden, ohne den in den hochentwickelten Ländern fast gar nichts mehr geht. Die Entwicklung von Software wird jedoch bis heute nur ungenügend beherrscht. Software-Entwicklung ist also offenbar etwas Schwieriges. Dieser Beitrag zeigt Gründe auf, warum dies so ist.

VON MARTIN GLINZ, PETER BAUMANN UND HELMUT SCHAUER

Computer durchdringen alle Lebensbereiche – und mit ihnen die Software, welche diese Computer steuert. Wirtschaft und Gesellschaft sind abhängig geworden von Software. Und die Abhängigkeit nimmt zu. Viele Selbstverständlichkeiten des Alltagslebens sind ohne Computer und deren Software nicht mehr möglich.

In krassem Gegensatz zu dieser Abhängigkeit steht die Tatsache, daß weltweit die Erstellung von Software nur ungenügend beherrscht wird. Termin- und Kostenüberschreitungen bei Software-Projekten sind die Regel; Software, die Fehler enthält oder sich nicht entsprechend den Vorstellungen und Bedürfnissen der Benutzenden verhält, gehört zum Alltag. Immer wieder kommt es auch vor, dass ganze Projekte scheitern. Eines der spektakulärsten Beispiele aus jüngerer Zeit ist das Scheitern des CONFIRM-Projekts: Im Auftrag grosser amerikanischer Hotel- und Mietwagenunternehmen sollte ein neues, umfassendes Reservierungssystem entwickelt werden. Nach dreieinhalb Jahren Entwicklungszeit und Investitionen von rund 125 Millionen Dollar wurde das Projekt im Juli 1992 eingestellt, als erkannt wurde, dass die gestellten Anforderungen mit dem gewählten Lösungsansatz nicht erfüllbar waren.

Dabei erscheint Programmieren auf den ersten Blick gar nicht so schwierig. Was ist ein Programm denn eigentlich mehr als das Zusammensetzen einfacher Befehle zu einer geeigneten Folge von Anweisungen an einen Computer? Warum also ist Software-Entwicklung schwierig?

Eine einfache Antwort auf diese Frage haben wir nicht – es gibt wohl auch keine. Wir wollen in diesem Beitrag Faktoren aufzeigen, die dazu beitragen, dass Software-Entwicklung schwierig ist, und ausserdem Ursachen dafür aufdecken, dass die Schwierigkeiten so oft unterschätzt werden.

Hierzu müssen wir mehr über Software wissen: Was ist Software überhaupt? Wozu dient sie? Welchen Gesetzmässigkeiten unterliegt sie? Die besonderen Eigenschaften von Software geben uns Aufschluss über die Schwierigkeiten im Umgang mit ihr.

Was ist Software?

Unter Software verstehen wir Programme, Verfahren, zugehörige Dokumentation und Daten, die mit dem Betrieb eines Computersystems zu tun haben.

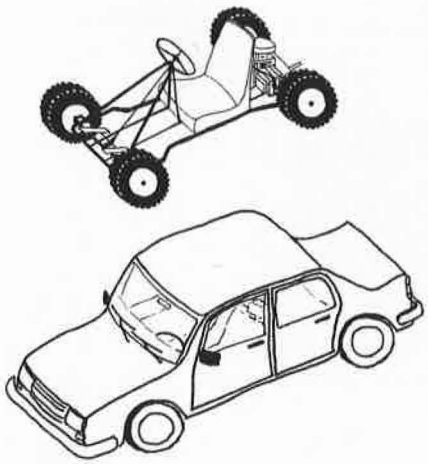
Diese Definition erlaubt uns, drei wichtige Feststellungen über Software zu machen:

Software umfasst erheblich mehr als nur Programme. Der Aufwand für das Schreiben der Programme beträgt in der Regel nur 10 bis 20 Prozent des Gesamtaufwands für die Entwicklung von Software. Da die Programme jedoch der entscheidende Bestandteil von Software sind (ohne Programme geht nichts), wird bei der Schätzung des Aufwands für die Entwicklung von Software allzuoft nur auf die Programme abgestellt. Dadurch wird der tatsächliche Aufwand um ein Vielfaches unterschätzt.

Software ist ein immaterielles technisches Produkt. Man kann Software nicht anfassen. Die Konsequenzen dieser trivial scheinenden Feststellung sind vielfältig:

- Geistige Güter werden in unserer Gesellschaft in der Regel als *weniger wertvoll* eingestuft als mit dem gleichen Aufwand erstellte materielle Güter: Der Wert von Software wird *unterschätzt.*
- Im Gegensatz zu materiellen Produkten gibt es für Software keine natürlichen Grenzen in

«Software umfasst erheblich mehr als nur Programme»



So wie ein Auto wesentlich mehr ist als nur ein fahrbarer Untersatz, so umfasst Software wesentlich mehr als nur Programme.

Dr. Martin Glinz ist ausserordentlicher Professor, Dr. Peter Baumann ist Assistenzprofessor, und Dr. Helmut Schauer ist ordentlicher Professor am Institut für Informatik der Universität Zürich.

Form von Materialeigenschaften oder Naturgesetzen. Es gibt einzig die theoretische Grenze der Berechenbarkeit, aber diese wirkt sich in der Praxis kaum aus. Während ein Brückenbauingenieur bei seinen Konstruktionen Rücksicht nehmen muss auf die Eigenschaften der verwendeten Materialien und auf die Gesetze der Statik und Schwingungsdynamik, ist ein Software-Entwickler grundsätzlich in der Wahl seiner Konstruktionen frei. Software findet ihre praktischen Grenzen nur in der Begrenztheit des Könnens ihrer Entwickler. Da diese Grenzen schwierig zu erkennen sind und Menschen ausserdem dazu neigen, die Begrenztheit ihres Könnens zu verdrängen, werden bei Software häufiger und leichtsinniger (zu) schwierige Vorhaben angegangen als in anderen technischen Disziplinen.

• Bei der Entwicklung materieller Produkte sind viele *Fehler* leicht als solche erkennbar. Wenn beim Bauen das Dach statt des Kellers in die Baugrube gesetzt wird, so ist dies unschwer als Fehler zu erkennen. Ein Fehler ähnlicher Schwere in Software ist nur durch sorgfältige und aufwendige Prüfmassnahmen zuverlässig erkennbar oder verhütbar.

• Gleiches gilt für die Beurteilung des *Entwicklungsstands:* Behauptet ein Bauunternehmen, der Bau sei praktisch fertig, obwohl erst der Rohbau steht, so ist leicht feststellbar, dass diese Behauptung nicht stimmt. Die Beurteilung des Fertigstellungsgrads von Software ist ungleich schwieriger und aufwendiger.

• Software ist scheinbar sehr flexibel und leicht zu ändern. Es gibt ja kein Material, das dabei umgeformt oder gar neu produziert werden muss. Kleinst-Software ist tatsächlich leicht änderbar. Mit zunehmender Grösse der Software hat jede Änderung jedoch so viele Effekte und Konsequenzen, die alle bedacht sein müssen, dass die tatsächlichen Aufwendungen für Software-Änderungen oft höher sind als diejenigen für vergleichbare materielle Produkte.

Software ist Bestandteil eines Computersystems. Solche programmgesteuerten Computersysteme haben eine sehr unangenehme Eigenschaft: Kleinste Veränderungen in Programmen oder Daten (im Extremfall genügt schon die Setzung eines Punkts anstelle eines Kommas) können massive Veränderungen im Verhalten des Systems bewirken. Es ist daher ungleich schwieriger und aufwendiger, das wunschgemässe Funktionieren von Software mit einer gegebenen Wahrscheinlichkeit zu gewährleisten, als dies bei Produkten der klassischen technischen Disziplinen der Fall ist.

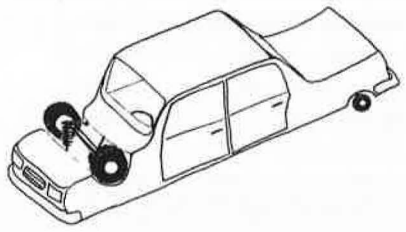
Eine Milchmädchenrechnung

«Ein Mann braucht zum Bau einer 2 m langen Brücke 0,5 Tage.
Wie lange brauchen 100 Leute für den Bau einer 2 km langen Brücke?

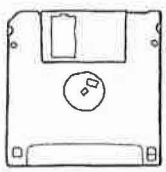
Rechne.»

Nichtlinearer Anstieg des Aufwands mit zunehmender Produktgrösse und Quantensprünge in diesem Anstieg sind Phänomene, die nicht nur in der Informatik auftreten.

«Man kann Software nicht anfassen»



Diese mechanische Konstruktion ist offensichtlich falsch.



Wie erkennen wir aber Fehler in der hier gespeicherten Software-Konstruktion?

Wozu dient Software?

Software dient dazu, ein Problem zu lösen oder zu dessen Lösung beizutragen, indem menschliche oder technische Arbeitsvorgänge automatisiert oder unterstützt werden. Aus dieser Zweckbestimmung folgen drei wesentliche Eigenschaften von Software:

• Wenn ein Problem von seiner Natur her komplex und schwierig ist, so ist die Entwicklung der Software zur Lösung dieses Problems in der Regel nicht weniger komplex und schwierig, selbst wenn das Problem eine einfache Lösung haben sollte (was selten der Fall ist). Dies liegt daran, dass ein Problem erst dann gelöst werden kann, wenn die Problemstellung genau verstanden worden ist. Je schwieriger die Probleme werden, desto schwieriger wird folglich auch die Entwicklung von Software.

• Bei der Entwicklung von Software sind immer zwei Schwierigkeiten gleichzeitig zu bewältigen:
1. Das Problem ist im Kontext seines Sachgebiets zu verstehen und befriedigend zu lösen.
2. Die Problemlösung muss auf adäquate Software-Strukturen abgebildet werden.

• Problemlösungen schaffen neue Realitäten und wecken neue Bedürfnisse. Software ist daher nicht einfach ein Abbild der Realität und bisheriger manueller Problemlösungen. Sie konstruiert und verändert die Realität. Beispielsweise ermöglicht Logistik-Software die Fertigung von Gütern nach dem «Just in time»-Prinzip (Zulieferteile werden genau dann angeliefert, wenn sie in der Produktion benötigt werden). Dies schafft streng termingebundenen Lastwagenverkehr als neue Realität und softwaregestützte Optimierung solchen Verkehrs als neues Bedürfnis.

Tabelle 1: Klein-Gross-Gegensätze in der Software-Entwicklung

klein	gross
Programme von 1 bis etwa 300 Zeilen Länge	längere Programme
Für den Eigengebrauch	Für den Gebrauch durch Dritte
Vage Zielsetzung genügt, das Produkt ist seine eigene Spezifikation	Genaue Zielbestimmung, d.h. die Spezifikation von Anforderungen, erforderlich
Ein Schritt vom Problem zur Lösung genügt: die Lösung wird direkt programmiert	Viele Schritte vom Problem zur Lösung erforderlich: Spezifikation der Anforderungen, Konzept der Lösung, Entwurf der Teile, Programmieren der Teile, Zusammensetzen der Teile
Validierung (Feststellen, ob die Software sich den Erwartungen entsprechend verhält) und nötige Korrekturen finden am Endprodukt statt	Auf jeden Entwicklungsschritt muss ein Prüfschritt folgen, sonst wird das Risiko, dass das Endergebnis unbrauchbar ist, viel zu gross
Eine Person entwickelt: Keine Koordination mehrerer beteiligter Personen erforderlich, keine Kommunikationsbedürfnisse	Mehrere Personen entwickeln gemeinsam: Koordination und Kommunikation notwendig
Komplexität des Problems in der Regel klein, Strukturieren der Software und Behalten der Übersicht nicht schwierig	Komplexität des Problems grösser bis sehr gross, explizite Massnahmen zur Strukturierung und Modularisierung erforderlich
Software besteht aus wenigen Komponenten	Software besteht aus vielen Komponenten, die spezielle Massnahmen zur Komponentenverwaltung erfordern
In der Regel wird keine Dokumentation erstellt	Dokumentation dringend erforderlich, damit Software wirtschaftlichen betrieben und gewartet werden kann
Keine Planung und Projektorganisation erforderlich	Planung und Projektorganisation zwingend erforderlich für eine zielgerichtete, wirtschaftliche Entwicklung

Welche Gesetzmässigkeiten gelten für Software?

Werden gewisse Erfahrungen immer wieder gemacht, so lässt sich vermuten, dass eine Gesetzmässigkeit vorliegt. Solche Gesetzmässigkeiten kennen wir auch für Software. Im Gegensatz zu den Gesetzen z.B. der Physik sind die Software-Gesetzmässigkeiten nur ansatzweise quantitativ gefasst und statistisch abgesichert. Sie bewegen sich also noch eher in der Gegend von Bauernregeln. Dennoch ist die Kenntnis solcher Regeln wesentlich für die Frage nach der Schwierigkeit der Entwicklung von Software. Die folgenden Gesetzmässigkeiten scheinen uns in diesem Zusammenhang besonders wichtig:

Die Entwicklung von Klein-Software unterscheidet sich fundamental von der Entwicklung grösserer Software (Tabelle 1). Dabei stellt letzteres den Normalfall dar.

Viele Probleme existieren für Klein-Software gar nicht. Die Erfahrung, dass die Entwicklung kleiner Programme recht einfach ist, ist der Hauptgrund für den Trugschluss vieler Leute, dass Software-Entwicklung generell etwas Einfaches sein müsse.

Der *Aufwand* für die Erstellung von Software *steigt mit wachsender Problemgrösse überproportional* an. Dies liegt unter anderem daran, dass der Aufwand zur Beherrschung der wachsenden Komplexität und der Kommunikationsaufwand überproportional wachsen. Das Wachstum einzelner Faktoren ist dabei nicht kontinuierlich, sondern weist Quantensprünge auf (Bild 1).

Software ist einer Evolution unterworfen. Die Umwelt und die Bedürfnisse der Benutzenden verändern sich ständig. Software, die sich in Gebrauch befindet, bleibt daher ohne ständige Anpassungen und Erweiterungen nicht gebrauchstauglich. Ferner werden im Betrieb auch immer wieder Fehler entdeckt, die behoben werden müssen. Solche Entwicklungsarbeiten an in Betrieb befindlicher Software werden «Wartung» genannt.

Wird ein Haus ständig erweitert und umgebaut, so wird es immer grösser und unüber-

Wachstum und Quantensprünge beim Kommunikationsbedarf

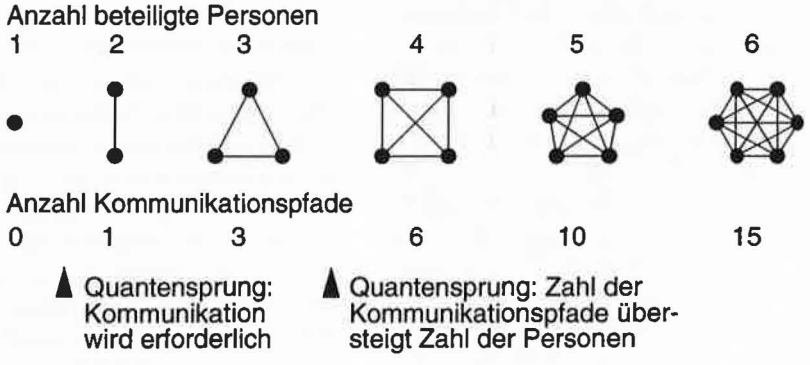


Bild 1

Das Geheimnisprinzip

Für jedes Teilproblem werden alle Modellierungs- und Entwurfsentscheidungen zu einer Einheit (einem Modul) zusammengefasst, und zwar so, dass ausserhalb des Moduls nur bekannt ist, *was* im Modul geschieht, aber nicht *wie* es geschieht.

sichtlicher. Mit jedem weiteren An- oder Umbau wird es schwieriger, den Bau so auszuführen, dass nicht Teile des Gebäudes dabei einstürzen oder danach ihren Verwendungszweck nicht mehr erfüllen. Gleiches gilt für Software. Die erforderlichen Anpassungen und Erweiterungen bringen es mit sich, dass Umfang und innere Unordnung von in Gebrauch befindlicher Software ständig zunehmen. Die Wartung einer Software wird daher mit zunehmendem Alter immer schwieriger und teurer.

Die Software-Evolution geht bisweilen so schnell, dass ein Software-Produkt schon während seiner Entwicklung modifiziert werden muss, weil ein Teil der zu Beginn festgelegten Anforderungen infolge von Änderungen im Umfeld und in den Bedürfnissen nicht mehr stimmt.

Was also macht Software-Entwicklung schwierig? Zusammenfassend stellen wir fest: Software ist ein technisches Produkt. Ihre Entwicklung ist keinesfalls weniger schwierig als diejenige eines herkömmlichen technischen Produkts mit vergleichbarer Komplexität. Die Schwierigkeiten werden durch drei Faktoren verstärkt:

- Software ist immateriell
- es sind immer gleichzeitig ein Sachproblem zu lösen und die dazu passende Software zu erstellen
- der Wert von Software und die Schwierigkeit ihrer Erstellung werden unterschätzt.

Dass Software-Entwicklung eine schwierige und anspruchsvolle Aufgabe ist, wurde erstmals in den 60er Jahren erkannt, als die Computer leistungsfähig und zuverlässig genug wurden, um grosse Software auf ihnen ablaufen zu lassen. Grosse Software-Projekte scheiterten oder konnten nur mit grösster Mühe zum Erfolg gebracht werden. Das Wort «Software-Krise» machte die Runde. In dieser Situation postulierten einige der damals bekanntesten Informatiker *Software Engineering*, das Anwenden der Prinzipien des Ingenieurwesens auf Software, als Ausweg aus der Krise.

Software Engineering ist inzwischen zu einem eigenständigen Fachgebiet in der Informatik geworden. Es befasst sich mit der Frage, wie Software mit einem gegebenen Leistungsprofil und einem gegebenen Qualitätsstandard möglichst kostengünstig entwickelt und fortentwickelt (gewartet) werden kann.

Hierzu versucht man, den Entwicklungsprozess zu verstehen und daraus Verfahren zur Vereinfachung oder Beschleunigung von Entwicklungsschritten abzuleiten. Dabei sind durchaus Erfolge zu verzeichnen. Wir wissen heute über Software und ihren Entwicklungsprozess wesentlich mehr als vor 25 Jahren. Als Beispiel für eine fundamentale Erkenntnis sei hier das Geheimnisprinzip (*information hiding*, siehe Kasten oben) genannt.

Die Software-Krise ist uns allerdings treu geblieben – wahrscheinlich, weil es gar keine Krise ist, sondern nur der sichtbare Ausdruck der Tatsache, wie schwierig Software-Entwicklung tatsächlich ist. Auch werden Fortschritte dadurch kompensiert, dass Software für immer grössere und schwierigere Probleme eingesetzt wird.

Trotz aller Fortschritte sind wir von einem umfassenden und tiefen Verständnis des Software-Entwicklungsprozesses noch weit entfernt. Die meiste Forschungsarbeit konzentriert sich daher auf Teilgebiete, von denen man weiss oder stark vermutet, dass sie einen signifikanten Einfluss auf den Gesamtprozess haben. Die Praxis behilft sich mit Verfahren, die sich näherungsweise bewährt haben. Deren Qualität und Wirksamkeit liegt in der Regel erheblich unter dem nach heutigem Wissen erreichbaren Stand.

Dementsprechend wichtig sind Forschung und Ausbildung in Software Engineering. Die Leute, welche Software Engineering an Hochschulen betreiben, müssen daher einerseits in ihrer Forschungsarbeit ein besseres Verständnis des Entwicklungsprozesses oder von Teilen davon erreichen. Andererseits müssen sie durch Ausbildung, Weiterbildung, Beratung sowie durch praktische Erprobung ihrer Forschungsergebnisse helfen, den Stand der Praxis zu verbessern.

Am Institut für Informatik der Universität Zürich konzentrieren wir uns auf die folgenden Teilgebiete des Software Engineerings:

- Methoden und Sprachen für verbesserte Anforderungsspezifikationen
- Mathematische Methoden zur Verhinderung und Erkennung von Fehlern in Software
- Verstärkter Einbezug der Benutzer in die Entwicklung von Software.

Daneben bieten wir auch Beratung und Fortbildung für Software-Entwickler und Software-Manager an.

Glossar

Software Programme, Verfahren, zugehörige Dokumentation und Daten, die mit dem Betrieb eines Computersystems zu tun haben.

Software Engineering Das systematische, disziplinierte und quantitativ fassbare Vorgehen bei Entwicklung, Betrieb und Wartung von Software, das heisst, die Anwendung der Prinzipien des Ingenieurwesens auf Software.

Software-Entwicklung Die Umsetzung der Bedürfnisse eines Benutzers in Software. Umfasst Spezifikation der Anforderungen, Konzept der Lösung, Entwurf und Programmierung der Komponenten, Zusammensetzung der Komponenten sowie Überprüfung des Entwickelten nach jedem Schritt.