Die Zukunft des Software Engineer(ing)s

Karol Frühauf
INFOGEM AG, Postfach, CH-5401 Baden
Tel.: 056 222 65 32, Fax: 056 222 00 38
E-Mail: karol.fruehauf@infogem.ch

Zur Gegenwart und Zukunft des Informatikers wurde ich gebeten etwas zu sagen (besser zu schreiben). Zur Gegenwart ist wohl vieles gesagt (und schnell veraltet) und es wird wohl auch viel geschrieben (und schnell vergessen). Zur Gattung Informatiker will ich nichts sagen, sie hat zu viele Gesichter und Glieder. Ich bilde mir ein, etwas Kompetentes zum Personenkreis sagen zu können, der sich mit Software Engineering beschäftigt und ich weiss, dass ich nichts Kompetentes zur Zukunft sagen kann. Ich wende mich deshalb diesem Thema zu und verlasse mich auf die Gnade der vergesslichen Leser.

Drei Publikationen, die auf meinem Tisch liegen, greife ich (fast zufällig) aus und finde darin Stoff, der mich zum Denken anregen könnte. Die drei Aspekte sind das Wesen von Software, die Software kreierenden Wesen und das Wesentliche der Gestaltung von Produkten, die Software enthalten.

Software, das verkannte Wesen

Die März Ausgabe der Communications of the ACM ist der Zukunft gewidmet, ganz unbescheiden den nächsten 1000 Jahren. Im ersten Beitrag postuliert Armour (2001) zum Thema Software:

1. Software ist Wissen und zwar kein passives, wie z.B. das in einem Buch gespeicherte, sondern aktives Wissen; mit Software kann man was tun, man kann das Wissen anwenden.
2. Software wird das Zahlungsmittel der Zukunft sein.

Das wir in der Epoche der Wissensarbeit angelangt sind, dass wissen wir von Peter Drucker. Das wir Wissen als Zahlungsmittel einsetzen können, das klingt verblüffend aber auch verlockend.

Zahlungsmittel vereinfachen den Austausch von Gütern. Ich kann aus Äpfeln keinen Zaun bauen. Aber ich kann die Äpfel für Zahlungsmittel eintauschen und dafür Latten, Nägeln, etc. erstehen. Heute verwenden wir schnöde Geld als Zahlungsmittel. Wenn es nach Armour geht, wird es morgen die Software sein. Vermutlich erst übermorgen. Wir müssen zuerst noch einige Probleme lösen, wie dasjenige der Wechselkurse (Standardisierung der Arten der Zusammenarbeit von Software-Produkten, pardon, von Software-Bündeln), das Problem der Inflation (mehrfach vorhandene Funktionalität, Aufblähen von Programmen mit überflüssigem Code)

oder den Schutz der grossen Nennwerte (z.B. der Programme, die universelle Fräsmaschinen beim Schnitzen des Kerns von Atombomben steuern). Wenn wir sie alle gelöst haben, so um 2100 herum, kann ich z.B. für 100kg Äpfel ein 1000 MS Word 2095 Lizenzen bekommen und für jede Lizenz einen 10mm Nagel. Wenn es dann noch überhaupt Äpfel und Zäune gibt.

Von einem Zahlungsmittel erwarten wir, dass es fälschungssicher ist. Wird Software zum Zahlungsmittel, müssen wir dafür sorgen, dass sie von niemandem geändert werden kann, sobald sie in den Umlauf gebracht wurde, und auch dafür, dass seine Änderung nicht notwendig ist, d.h. dass das gekapselte Wissen immer richtig angewendet wird, wenn man die Software nutzt. Die Software wird nicht fehlerfrei sein müssen, aber wird nichts Falsches tun dürfen. Also jegliche Software wird eine vergleichbare Zuverlässigkeit haben müssen, wie die Software in den heutigen sicherheitskritischen Anwendungen. Solche Software kann nicht von Abenteurern, Überläufern, von jeglichem Software Engineering Wissen unbelasteten Enthusiasten erzeugt werden. Die Münzen werden auch nicht von Bibliothekaren geprägt, die Noten nicht von Psychologen gedruckt, seien sie noch so begeistert hierfür.

Gestern Abend hat mich ein Freund angerufen. Er ist von Haus aus Bildhauer, einer mit akademischen Weihen. Seit einigen Monaten ist er in einem Internet-Haus als Art Director engagiert und gestaltet nicht nur den Auftritt, sondern auch die Software (damit sie auch designt und nicht nur codiert ist). Ganz stolz berichtete er mir gestern, dass er das Problem der Mehrsprachigkeit gelöst hat. Meine spontane Reaktion, dass er nicht der erste ist, der dieses Problem gelöst hatte, trübte ein wenig seine Freude aber knickte zum Glück nicht sein Selbstvertrauen.

Würde es mein Freund in 2100 tun, würde er die Inflation anheizen, das Zahlungsmittel würde an Wert verlieren, weil er ein bereits gelöstes Problem löste, d.h. er hat kein neues Wissen gekapselt. Einem Bildhauer kann man es nachsehen. Was soll man aber von der Umgebung halten, die es zulässt (Informatiker) oder womöglich sogar fordert (Manager)?

Wollen wir auch in Zukunft eine feste Währung haben, müssen wir wohl oder übel dafür sorgen, dass das, was wir schon heute über Software Engineering wissen, halbwegs konsequent in den Software-Prägeanstalten angewendet wird und auch dafür, dass wir die Vorgehensweisen, Methoden und Werkzeuge gründlich weiterentwickeln, damit wir kein zu teueres Zahlungsmittel haben bzw. zu teuer für das Zahlungsmittel Software zahlen müssen. Themen wie Integrierbarkeit, Integrität, Korrektheit, Portabilität, Testbarkeit, Zuverlässigkeit rücken in den Vordergrund; Benutzerfreundlichkeit, Effizienz, Flexibilität, Wartbarkeit und Wiederverwendbarkeit dürfen keine Frage mehr sein, sie müssen garantiert sein. Gemerkt? Die Qualität der Software wird entscheidend sein.

Daher ist es kaum überraschend, dass die in den letzten Jahren aufgekommenen „mageren Vorgehensweisen“ (light methodologies), allesamt kompromisslos sind, was die Qualität betrifft. In drei Bereichen ist also Handlungsbedarf:

1. die Forschung auf dem Gebiet des Software Engineerings muss uns bessere Grundlagen liefern für Verfahren zum kostenarmen Bau zuverlässiger;
2. das Management in den Software-Unternehmen muss d die Qualität der Produkte (wieder) in den Fokus rücken und entsprechende Konsequenzen für ihr Verhalten ableiten;
3. die Software-Entwickler müssen das nötige Handwerk beherrschen lernen, um Software, die den Anforderungen eines Zahlungsmittels genügt, entwickeln zu können.

Qualifikation zum Software Engineer

Die Nachricht in IEEE CS (2001) kündigt an, dass IEEE CS demnächst beginnen wird Software Engineers zu adeln. Ähnlich wie es die Software Division der American Society for Quality (ASQ) seit einigen Jahren bereits mit Software Quality Engineers tut, beginnt nun auch IEEE CS, anhand einer bestandenen Prüfung, ein Zertifikat für Software Engineers mit einer Gültigkeitsdauer von drei Jahren zu erteilen. Diese Zertifikate sind nicht als Führerscheine zu verstehen, es sind keine Zulassungen, einen bestimmten Beruf auszuüben, sondern die Anerkennung für das Beherrschen des für das Ausüben des Berufs Software Quality Engineer bzw. Software Engineer benötigten Basiswissens.

Die Zertifikate sind besonders in angelsächsischen Ländern beliebt. Schon früher konnte man sich als Software-Projektleiter zertifizieren lassen, seit ein paar Jahren kann man auch das Zertifikat Software-Tester erwerben. Dass es auch Hersteller abhängige Zertifikate für sehr teueres Geld gibt, sei nur am Rande erwähnt. Diese haben schon eher den Charakter des Führerscheins, einer Zulassung, die Kunden des Herstellers (vielleicht) auf die Erfolgsstrasse zu führen. Am anderen Rande stehen die zertifizierten Auditoren und Assessoren, die zur Prüfung von Vorgehensweisen in Unternehmen zugelassen sind, um diesen dann ein Zertifikat oder eine glaubwürdige Bewertung ausstellen zu können.

Ich akzeptiere ohne Murren, dass es für spezifische, vor allem für aus ethischen oder technischen Gründen heikle Tätigkeiten eine Zulassungsprüfung gibt.
Ich akzeptiere auch ohne Murren, dass keine Mittelschule, keine Fachhochschule und auch keine Universität die letzten Details des Betriebssystems eines Herstellers den Studenten beibringt.
Ich habe jedoch sehr viel Mühe zu akzeptieren, dass Themen wie Projektmanagement und Testen und Design und Requirements Engineering und Konfigurationsmanagement etc. nicht Bestandteil des Curriculums der Ausbildung an diesen Ausbildungsinstituten sein sollten, so dass das Abschlussdiplom „das Beherrschen des für das Ausüben des Berufs Software Engineer benötigten Basiswissens“ nicht bescheinigen kann.

Auch im 21-en Jahrhundert sind Berufsverbände geeignete Gremien, das „benötigte Basiswissen“ zu definieren (vgl. SWEBOK – Software Engineering Book of Knowledge) oder gar normieren. Das Beherrschen dieses Wissens herbeiführen, das sollten unsere Ausbildungsstätten. Ein Lichtblick in der Schweiz: Mit der Unterstützung der SoftNet–Initiative des Bundes versuchen die Fachhochschulen den Unterricht nach SWEBOK auszurichten und den Unterrichtsstoff gemeinsam zu erarbeiten. Möge ihr Streben erfolgreich sein. Und eine weitere Verbreitung finden. Zum Beispiel bis in die Ausbildung der Wirtschaftsinformatiker mit eidgenössischem Ausweis. Damit sie das ewige Konzipieren verlernen, dafür lernen dürfen, was Design ist und was Anforderungen sind.

Gestaltung der Software

Norman (1999) plädiert in seinem Buch für die Abkehr vom universellen Rechner, wie der heutige PC; er kann Vieles, aber nichts richtig bzw. nichts kann mit ihm einfach bewerkstelligt werden. Er sollte durch dedizierte „Informations-Haushaltgeräte“ ersetzt werden, durch Geräte, die bestimmte Tätigkeiten des Menschen ideal unterstützen. „*People should learn the task,*

not the technology. They should be able to take the tool to the task, not as today, where we must take the task to the tool."

Meine Frau wollte nicht abseits stehen und wie alle um sie herum auch mit E-Mail kommunizieren können. Unser Sohn hat sich seiner Mutter erbarmt und ihr einen PC mit installierter Software geschenkt. Nach längerem Üben habe ich es geschafft, alle Einstellungen vorzunehmen, damit sie mit ihm mailen kann. Ihre Bedürfnisse sind: Ein Mail schreiben, erhaltene Mails abholen und diese beantworten. Um etwas davon tun zu können, muss sie

a) den Rechner einschalten
b) warten bis das System hochgefahren ist
c) Dreifingerübung machen
d) Verbindung über Modem herstellen
e) Mail-Programm starten

Bis auf den ersten Schritt alles Dinge, die von ihrer Warte aus gesehen, völlig überflüssig sind, mit dem was sie tun will gar nichts zu tun haben.

Wenn ich das Auto starte, bekomme ich angezeigt, ob der Öldruck o.k. ist, wie viel Liter Treibstoff ich noch habe, wie spät es ist, welche Temperatur draussen ist, welche als gewünschte Innentemperatur eingestellt ist, welche Station ich am Radio eingestellt habe, etc. Information, die mich zum und beim Autofahren mehr oder weniger interessiert oder interessieren sollte. Ich muss hierzu nur einen einzigen Schlüssel drehen, kein Warten auf das Hochfahren der (vielen) Rechner, keine Dreifingerübung(en), kein Starten des autolokalen Netzwerks und schon gar nicht irgendeines Anzeigeprogramms. Mein Auto ist ein solches „Informations-Haushaltgerät", die Rechner sind für den Bediener (nicht nur im wörtlichen Sinne) unsichtbar, ich bediene nur mein Werkzeug zum Fahren, das Auto.

Um diese Geräte der Zukunft, um unsere „rechnerlose Zukunft" zu bilden, schlägt Norman (1999) eine Vorgehensweise vor, die um den Menschen, den zukünftigen Benutzer kreist. Die Prinzipien dieser Vorgehensweise sind:

a) Beobachte die Benutzer, lerne, was sie tun.
b) Untersuche den Markt, lerne über ihn so viel nur möglich ist.
c) Formuliere und validiere die Bedürfnisse der Benutzer, des Marktes.
d) Bilde, zusammen mit den Benutzern, Attrappen und Prototypen des zukünftigen Produkts.
e) Schreibe die Gebrauchsanweisung des zukünftigen Produkts.
f) Designe das Produkt anhand der Gebrauchsanweisung, der Attrappe und des Prototyps.
g) Prüfe das Design und ändere es kontinuierlich.

Das Vorgehen setzt voraus, dass Mitarbeiter gewisser Qualifikation vorhanden sind. Benötigt werden:

1. Verhaltens-Beobachter
2. Verhaltens-Designer
3. Modellbauer
4. Benutzertester
5. Graphik- und Industrie-Designer
6. Technische Autoren

Mit der Ausnahme des Graphik- und Industrie-Designs ist mir keine Ausbildungsstätte in der Schweiz bekannt, in der eines dieser Fertigkeiten erlernt werden könnte. Neuerdings gibt es wohl die Möglichkeit der Ausbildung zum Web-Designer; sie konzentriert sich jedoch auf das Erlernen von Hilfsmitteln und Notationen, die Prinzipien benutzerfreundlichen Designs und die Vorgehensweisen stehen schüchtern im Hintergrund.

Die Gestaltung der Produkte der Zukunft erfordert in noch stärkerem Masse von den Software-Entwicklern, die Kluft zwischen dem Computer und dem Anwendungsgebiet überbrücken zu können, den Rechner im Anwendungswerkzeug verschwinden lassen zu können. Auf diese Aufgabe müssen sie vorbereitet werden. Wennes die Schulen noch nicht tun, dann müssen die Berufsverbände in die Bresche springen. Wenn die es auch nicht nicht können, dann müssen die einzelnen Firmen für diese Ausbildung sorgen, wollen sie wirklich innovativ sein und neue Märkte erschliessen oder gar schaffen.

Software Engineer, das soziale Wesen

Noch ein Aspekt zum Schluss. Software Engineering ist zuerst und vor allem anderen Kommunikation. Kommunikation unter Menschen, mit den Benutzern, Managern, anderen Entwicklern, Unterlieferanten. Kommunikation in unterschiedlichen Begriffswelten, die des Anwendungsgebiets, der Zahlen, der eingesetzten Technologie und, hoffentlich von Zeit zu Zeit auch, in der des Software Engineerings. Und auch noch im Slang des Projekts, mit der Namenswelt des Produkts, das (weiter)entwickelt wird. Schliesslich Kommunikation in der natürlichen Sprache, welche das alles zusammenhält.

Verschiedene Formen kommen hinzu. Kommuniziert wird sowohl schriftlich als auch verbal. Schriftlich über E-Mail anders als im Brief und beim Verfassen eines Dokuments wieder ganz anders. Diese Vielfalt überfordert die meisten. Hier setzt zum Beispiel das Extreme Programming (Beck 2000) an: Es wird nur eine, die verbale Kommunikation forciert und gefördert.

Ob verbal oder schriftlich, sich auszudrücken muss gelernt werden. Dies muss zu einem Schwerpunkt der Ausbildung in Software Engineering werden. Genauso wie das Arbeiten in Teams, mit der erforderlichen Arbeitsteilung, dem Üben der Konsensfähigkeit, Verlässlichkeit und Respektieren aller anderen selbstauferlegten Spielregeln.

Schlussfolgerungen

In der etwas ferner liegenden Zukunft könnte sich die ökonomische Rolle der Software grundlegend ändern. Der Verbreitungsgrad von Software Engineering wird markant erhöht werden müssen und dies wird nur gelingen, wenn die Umsetzung mit effizienten Mitteln unterstützt wird. Dieser Herausforderung wird man nur mit einer in Inhalten und Formen stark veränderten Ausbildung auf allen Ebenen gerecht.

Nur weil jeder Informatik braucht, ist nicht jeder Informatiker.
Nur weil einer eine Programmiersprache beherrscht, ist er noch kein Software Entwickler.
Weil manch einer Software entwickelt, ist Software Engineering nur mancherorts zu Hause.
Dies gilt es zu ändern. In nicht allzu ferner Zukunft.

Literatur

Armour (2001)
Armour, Ph.G.: The Business of Software.
Communications of the ACM, Vo. 44, No. 3, March 2001, pp 13-14.

Beck (2000)
Beck, K.: Extreme Programming Explained - Embrace Change.
Addison-Wesley, Boston etc., 2000, ISBN 0-201-61641-6.

IEEE CS (2001)
Society to Offer Software Engineering Professional Certification.
IEEE Computer, Vol. 34, No. 3, March 2001, pp 82-83.

Norman (1999)
Norman, D.A.: The Invisible Computer.
The MIT Press, Cambridge, Ma, 1999 (2nd printing), ISBN 0-262-14065-9.

SWEBOK
http://www.swebok.org