Informatik-Spektrum 20: 3–4 (1997)

Editorial

Evolutionäre und kooperative Software-Entwicklung

Die wachsende Vielfalt und Komplexität von Software-Anwendungen haben dazu geführt, daß sich auch die Vorgehensweisen und Methoden zur Lösung software-technischer Aufgaben stark diversifiziert haben. Bereits 1980 hat M. M. Lehman unterschiedliche Problemstellungen bei der Software-Entwicklung klassifiziert und darauf abgestellte Vorgehensweisen gefordert. Unter anderem ging es ihm darum, den Unterschied zwischen Systemen mit vorwiegend technischen Aufgabenstellungen und sogenannten sozial eingebetteten Systemen deutlich zu machen. Die letztgenannten Systeme spielen für ihn eine besondere Rolle, weil ihre Anforderungen von den (oft nicht genügend klar artikulierten) Bedürfnissen ihrer Benutzer abhängen und sich ihr Einsatz unmittelbar auf deren Arbeits- und Verhaltensweisen auswirkt. Das führt unter anderem zu instabilen Anforderungen, zu Rückkopplungen und zu beträchtlichen Veränderungen an bestehenden Software-Systemen. Lehman spricht in diesem Zusammenhang von *Software-Evolution* und fordert für diese Art von Systemen eine spezifisch „evolutionäre" Vorgehensweise.

„Im Kleinen" - d. h. bezogen auf die noch zu erreichende korrekte Lösung einer Programmieraufgabe – besteht eine seit langem praktizierte Spielart des „evolutionären" Vorgehens im sogenannten „trial and error"-Verfahren: Entwickle eine (Programm-) Variante, erprobe sie im Test, prüfe die Resultate, akzeptiere sie oder entwickle eine neue Variante usw.

Lehman extrapoliert diesen Ansatz auf die Software-Entwicklung „im Großen" und deren Einbettung in den sozialen Kontext. So bietet sich ein evolutionäres Vorgehen zum Beispiel an, wenn zu Beginn eines Software-Projekts noch nicht bzw. nicht genau bekannt ist, wie die angestrebte Software beschaffen sein soll. Häufig sind die Zielvorstellungen, der geforderte Funktionsumfang und die Art der Einbettung der Software in die Anwendungsumgebung nur unscharf formuliert. Evolutionär zu entwickeln bedeutet unter anderem, bewußt mit (noch) unvollständigen Anforderungen zu beginnen, im Laufe des Entwicklungsprozesses diese Defizite zu erkennen, explizit zu machen und die erkannten Lücken schrittweise so weit wie möglich zu füllen.

Verkürzt können wir sagen: *Evolutionär* ist ein *Prozeß* immer dann, wenn dieser etwas Neues hervorbringt, dabei auf schon Bekanntem aufbaut, Modifikationen vornimmt und wenn sich das entstandene Neue in seiner vorhandenen Umgebung bewähren muß, woraus ggf. neue Modifikationen und Anpassungen resultieren. Bei einem solchen Prozeß ist in der Regel dessen Resultat zu Beginn noch nicht bekannt, es kann oft nicht einmal genau spezifiziert werden. Software-Evolution wird damit zum Abbild eines kontinuierlichen Lernprozesses, der von vagen Zielvorstellungen bis zu einer (im Erfolgsfall) akzeptierten Systemlösung führt.

Wie für jede Form der Software-Entwicklung stellt sich die Frage nach einem geeigneten *methodischen Ansatz*. Generell läßt sich der Prozeß der Software-Entwicklung in die folgenden drei Abschnitte gliedern:

- von der Problembeschreibung bis zur Software-Spezifikation (auch Problemanalyse, Systemanalyse, Requirements Engineering bzw. frühe Phasen der Software-Entwicklung genannt),
- die Software-Entwicklung im engeren (technischen) Sinn,
- die Integration der Software in den Anwendungszusammenhang.

Diese Einteilung gilt ebenso für die evolutionäre Software-Entwicklung, allerdings wird hier der Schwerpunkt anders gelegt. Steht beim klassischen Ansatz in der Regel die Software-Entwicklung im engeren Sinn im Mittelpunkt, so werden beim evolutionären Ansatz alle drei Abschnitte möglichst gleichgewichtig und im Zusammenhang betrachtet. Daraus resultiert eine natürliche Tendenz zu einer *zyklischen* Vorgehensweise.

Daneben gibt es weitere Querbezüge: Orientiert sich die Software-Entwicklung an einem festen Ziel, verfolgt dieses jedoch durch schrittweises Erweitern zunächst unvollkommener Teillösungen, so wird dieses Vorgehen *inkrementell* genannt. Unfertige Versuchsanordnungen oder Teilsysteme, die am Beginn einer solchen Entwicklung stehen, werden oft als *Prototypen* bezeichnet. Wählt man *Prototyping* als Vorgehensweise, so kann entweder das spätere Produkt inkrementell aus einem oder mehreren Prototypen weiterentwickelt werden oder man setzt nach einer (Wegwerf-) Prototypphase neu auf. Beides sind Formen evolutionärer Entwicklung.

Solche und ähnliche Herangehensweisen sind um so eher angezeigt, je weniger Erfahrungen mit vorangegangenen Projekten in dem betreffenden Anwendungsgebiet bestehen, je weiter die Anwendung in neue Bereiche vorstößt und je komplexer der Sachverhalt ist, der durch Software unterstützt werden soll. Evolutionäres Vorgehen wird damit zum möglichen Mittel zur
(a) schrittweisen Wissensakquisition,
(b) Behandlung dynamisch sich ändernder Anforderungen,
(c) Komplexitätsbewältigung.

Es versteht sich von selbst, daß ein solcher Ansatz nicht allein die Software-Entwickler (und ihre Führungskräfte) betrifft. Da Software-Entwicklung in der Regel eine Dienstleistung ist, ist sie auf ein bestimmtes Maß an Kommunikation zwischen Bestellern und Dienstleistenden angewiesen. Spielen Wissensakquisition und Komplexitätsbewältigung eine besondere Rolle, so ist diese oft nur in enger Zusammenarbeit der Entwickler mit den künftigen Betreibern und Benutzern der Software zu leisten. Daher sollte das Thema der evolutionären Software-Entwicklung nicht von Fragen der *Kooperation* (sowohl der Entwickler untereinander als auch zwischen Entwicklern und Anwendern) getrennt werden. Der Sinn eines bewußten kooperati-

ven Vorgehens besteht hauptsächlich darin, die damit verbundenen Synergieeffekte zu nutzen. Jüngere Tendenzen zur stärkeren Dezentralisierung und Diversifizierung von Software-Entwicklung, „outsourcing“ und Verlagerung von Teilen in wirtschaftlich attraktive Standorte verstärken die Notwendigkeit zur Kooperation und verschaffen dem Thema erneute Aktualität.

Kooperation ist aber nicht immer unproblematisch. So können etwa Mißverständnisse aufgrund unterschiedlicher Sichten bzw. Interessen der Beteiligten auftreten. Benutzer-Partizipation kann sich auf den Arbeitsprozeß der Entwickler verzögernd oder gar störend auswirken und damit zu Terminüberschreitungen und Kostenerhöhungen führen. Ähnliches gilt für die Verteilung der Arbeit auf große (und womöglich geographisch getrennte) Entwicklergruppen. Durch verstärkte Kooperation oder „evolutionäre“ Entwicklungszyklen bewirkte Ineffizienzen müssen durch eine erhöhte Qualität des entstehenden Produkts gerechtfertigt werden. Dies ist aufgrund der zu erwartenden Synergieeffekte nicht unrealistisch, doch sind Kosten-Nutzen-Betrachtungen oft problematisch oder kaum möglich. Evolutionäre und kooperative Systementwicklung muß sich also auch einer *Qualitätsbetrachtung* stellen.

In den Beiträgen dieses Themenheftes werden einige der genannten Aspekte der evolutionären und der kooperativen Software-Entwicklung aufgegriffen:

Bei dem Beitrag von Ch. Dahme und A. Raeithel steht die Frage im Mittelpunkt, wie der Übergang von einer gegebenen Problembeschreibung bis zur Software-Spezifikation einer methodischen Kontrolle unterworfen werden kann, ohne die Einordnung in den Anwendungszusammenhang zu verlieren. Software-Entwicklung wird als Teil eines Transformationsprozesses verstanden, der menschliche Tätigkeiten in eine maschinelle Form überführt. Wie läßt sich eine solche Transformation in einer (relativ) leicht kommunizierbaren und nachvollziehbaren Weise finden? Dazu wird ein in der Softwaretechnik noch relativ wenig bekannter, tätigkeitstheoretischer Ansatz vorgestellt. Dieser Ansatz liefert einen Rahmen sowie ein methodisches Rüstzeug und Hilfsmittel für die kooperative und evolutionäre Software-Entwicklung.

In dem Beitrag von Ch. Floyd et al. wird aus Entwicklersicht über Erfahrungen mit der evolutionären Vorgehensweise berichtet. Dazu wird das an der TU Berlin entwickelte zyklische Vorgehensmodell STEPS als Methodenrahmen benutzt und seine Anwendung wird beispielhaft an einem aktuellen Projekt diskutiert.

W. Hesse geht der Frage nach, inwieweit die heute bekannten und praktizierten Prozeßmodelle für den evolutionären Ansatz geeignet bzw. damit verträglich sind. Dazu werden die Vorgehensmodelle einiger bekannter neuerer Analyse- und Entwurfsmethoden auf ihre „evolutionären“ Anteile und Einflüsse hin untersucht und miteinander verglichen. Im zweiten Teil schlägt der Autor selbst ein Modell für die evolutionäre, objektorientierte Software-Entwicklung vor, das deren zyklischen Charakter besonders betont.

B. Kuhnt konzentriert sich auf Probleme der Kooperation bei der Software-Entwicklung und stellt ein Vorgehen - genannt systemische Beratung - zur Überwindung solcher Probleme vor. Dabei wird ein Schwerpunkt auf die Bereiche Projektetablierung, Projektbegleitung und Projektbewertung gelegt.

G. Pomberger und R. Weinreich untersuchen die Auswirkungen von Prototyp-orientierter Software-Entwicklung auf qualitative Merkmale (wie Benutzerfreundlichkeit, Wartungsfreundlichkeit, Korrektheit, Zuverlässigkeit und Prozeßqualität) sowie auf quantitative Merkmale (wie Entwicklungszeit und -aufwand, Betriebskosten, Software-Lebenszykluskosten). Es wird also sowohl die Qualität des Software-Produkts als auch des Software-Entwicklungsprozesses betrachtet. Dabei überprüfen die Autoren die Tauglichkeit des prototypischen Ansatzes anhand der Entwicklung von mehreren Software-Produkten unter kommerziellen Bedingungen.

Neben einer insgesamt positiven Einschätzung machen Pomberger und Weinreich auch auf die oben genannten Risiken, insbesondere bei der Spezifikation (u. a. bezüglich des Zeitaufwands und der methodischen Kontrolle) aufmerksam. Andere Autoren (z. B. Dahme/Raeithel und Kuhnt) weisen darauf hin, wie solche Ineffizienzen durch methodische Unterstützung der Kooperation gemildert werden können.

Schließlich sollte evolutionäres Vorgehen nicht als vage definierter, womöglich zufälliger oder gar chaotischer Prozeß mißverstanden werden. Für ihn gilt, wie es K. Nygaard für jede Software-Entwicklung formulierte: „To program is to understand“. Das heißt, Software-Entwicklung ist untrennbar mit Prozessen der *Wissensaneignung und -umsetzung* verbunden. Im Gegensatz zu den traditionellen Vorgehensweisen steht im evolutionären Fall der Wissenserwerb nicht als ein geschlossener Block am Projektanfang, sondern ist auf den gesamten Entwicklungsprozeß verteilt.

In ihrer täglichen Praxis werden Software-Entwickler, -Anwender und -Manager immer häufiger mit Anforderungen, Projekten und Fragestellungen der geschilderten Art konfrontiert. Daher scheint es uns geboten, sich mit diesem Thema zu beschäftigen, neue Begriffe, Vorgehensweisen und die daran geknüpften Vorstellungen zu formulieren, zu diskutieren, mögliche Lehren für die Praxis daraus zu ziehen und ggf. in die künftige Ausbildung einfließen zu lassen. Dazu soll dieses Heft beitragen.

Christian Dahme Wolfgang Hesse