von kerstin dittert

mehr zum thema:
www.agilealliance.org
www.b-agile.de

AGILE PROTOTYPEN: EVOLUTION OHNE DOGMA

Unter dem Eindruck sich rasch wandelnder Technologien und komplexer Produkt-Anforderungen setzen viele Projekte auf die Entwicklung von Prototypen. Bei einer iterativen Vorgehensweise scheint die Entwicklung evolutionärer Prototypen der Königsweg zu sein. Aber ist dieses Mittel immer die erste Wahl? Manche Projekte überfrachten einen einzigen Prototypen mit einer Vielzahl von Aufgaben. Agiles Vorgehen bedeutet vor allem Angemessenheit in der Wahl der Mittel. Dieser Artikel stellt unterschiedliche Prototyping-Ansätze gegenüber, diskutiert ihre Vor- und Nachteile und beschreibt mögliche Einsatzszenarios im Sinne eines agilen Vorgehens.

Was ist ein Prototyp?

Prototypen sind nicht nur in der Software-Industrie bekannt. Lange bevor das Software-Engineering diese Methode für sich entdeckte, arbeiteten etwa Architekten oder die Automobilindustrie mit Prototypen. Aber auch nicht-technische Domänen verwenden die Prototyp-Entwicklung, um Reaktionen auf zukünftige Produkte frühzeitig zu testen. So lässt beispielsweise die Lebensmittelindustrie neue Produkte häufig in verschiedenen Varianten von Testpersonen bewerten. Die Reaktionen der Testgruppen erlauben die schrittweise Annäherung an das optimale Endprodukt. Ein anderer populärer „Prototyper" war der Regisseur Alfred Hitchcock. Er drehte einzelne Filmsequenzen in verschiedenen Versionen und führte sie einem kleinen Publikum vor, um dessen Reaktionen zu testen. Kultfilme wie „Psycho" waren das Ergebnis dieser iterativen Vorgehensweise.

In der Softwareentwicklung sind Prototypen ebenfalls Teilschritte auf dem Weg zum Endprodukt. Nach [Pre94] ist ein Software-Prototyp ein System, das

- lauffähig ist und tatsächlich funktioniert,
- schnell und günstig erstellt werden kann.

Software-Prototypen vernachlässigen häufig die Umsetzung aller nicht-funktionalen Anforderungen (wie z. B. Wartbarkeit, Performance und Skalierbarkeit). Ausnahmen sind rein technische Prototypen, die sich *ausschließlich* mit derartigen Fragen beschäftigen (z. B. Performance-Vergleiche unterschiedlicher Lösungsansätze). Häufig ist die Architektur einer Anwendung noch nicht vollständig festgelegt, wenn die ersten Prototypen im Projekt implementiert werden.

Ziele eines Prototypen

Prototypen dienen stets dem Erkenntnisgewinn bezüglich fachlicher oder nicht-funktionaler Anforderungen. Ein Prototyp bietet gute Möglichkeiten zur Validierung und – beim Vorliegen einer Spezifikation – auch zur Verifikation von Anforderungen (**siehe Kasten 1**).

Mögliche Ziele eines Prototypen sind:

- die Lösung technischer Probleme (z. B. Verschlüsselung, Performance-Optimierung),
- die Evaluierung unterschiedlicher Technologien (z. B. Einsatz verschiedener Applikationsserver),
- die Spezifikation der fachlichen Anforderungen,
- die Validierung der fachlichen Anforderungen,
- die Festlegung der (grafischen) Benutzerschnittstelle,
- die Festlegung der Ablaufsteuerung,
- das Schaffen einer Diskussionsgrundlage für unterschiedliche Projektbeteiligte (z. B. Anwender, Programmierer, Management),
- das Sammeln von Erfahrung mit einer neuen Technologie.

Validierung von Anforderungen
- Es wird überprüft, ob die Anwendung für das Einsatzgebiet geeignet ist.
- Die fachlichen Aufgaben müssen erfüllt werden.
- Es ist keine formale Spezifikation erforderlich.

Verifikation von Anforderungen
- Es wird überprüft, ob die Anforderungsspezifikation erfüllt ist.

die autorin

Kerstin Dittert (E-Mail: kerstin.dittert@oocon.de) ist Geschäftsführerin der OOcon Informatik GmbH. Sie unterstützt Unternehmen in allen Gebieten des objektorientierten Softwareentwicklungsprozesses sowie in Technologiefragen. Hierbei ist sie unter anderem als Softwarearchitektin, Analytikerin, Designerin und Projektleiterin tätig.

Jeder Prototyp muss mindestens ein klar definiertes Ziel haben, das allen Projektbeteiligten bekannt ist. Diese einfache Regel wird oftmals vom Projektmanagement nicht beherzigt. Mögliche Folgen können zum Beispiel die berüchtigten „goldenen Wasserhähne" und explodierende Aufwände bei geringem Projektfortschritt sein.

Kein einzelner Prototyp kann alle oben genannten Ziele gleichzeitig erreichen. Für unterschiedliche Einsatzgebiete eignen sich verschiedenartige Prototypen. Sie variieren bezüglich ihrer Lebenszeit, ihrer Ziele und der jeweils adressierten Projektbeteiligten.

Evolutionärer Prototyp

Voraussetzung für die Entwicklung eines evolutionären Prototypen ist zumindest ein rudimentäres Verständnis der fachlichen und nicht-funktionalen Anforderungen an das System. Es muss jedoch keine vollständige Spezifikation vorliegen (**siehe Abb. 1**). Der evolutionäre Prototyp zeichnet sich durch folgende Eigenschaften aus:

- Die Implementierung wird iterativ erweitert.
- Die Teile, die am besten verstanden werden, werden zuerst implementiert.
- Der Quellcode des Prototypen fließt *komplett* in das Produktivsystem ein. Die Prototypentwicklung geht in die Entwicklung des endgültigen Systems über.
- Die nicht-funktionalen Anforderungen und die Ziel-Architektur müssen von Beginn an zumindest rudimentär im Prototyp berücksichtigt werden.

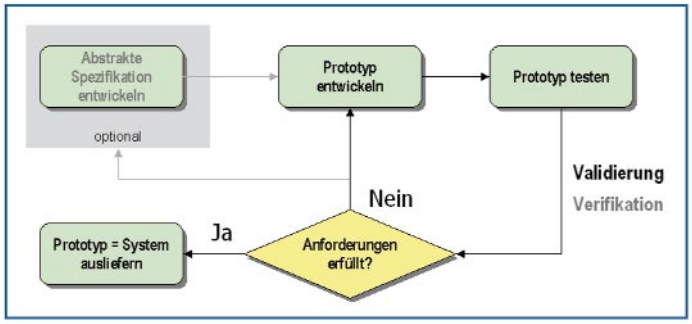

Abb. 1: Iterativer Softwareentwicklungsprozess mit Einsatz von evolutionären Prototypen (vgl. [Som95])

Ansonsten ist eine schrittweise Migration zum Produktivsystem unmöglich.

- Der Prototyp muss nach der letzten Iteration vollständig sein.

Die große Zahl von Änderungen macht den Quellcode im Laufe der Zeit möglicherweise unübersichtlich und schlecht erweiterbar. Regelmäßige Refaktorierungsphasen (siehe [Fow00]) sind deshalb unverzichtbar.

Wegwerf-Prototyp

Ein Wegwerf-Prototyp beantwortet eng begrenzte technische oder fachliche Fragestellungen. Er wird häufig auch als *revolutionärer Prototyp* oder *Rapid Prototype* bezeichnet. Er wird parallel zum Produktivsystem entwickelt (**siehe Abb. 2**). Ein Wegwerf-Prototyp besitzt typischerweise folgende Eigenschaften:

- Er löst ein technisches Teilproblem oder eine bestimmte fachliche Fragestellung.
- Es können Lösungsvarianten implementiert werden.
- Fragestellungen, die *am wenigsten* verstanden werden, werden *zuerst* implementiert. Aus diesem Grund werden bereits spezifizierte Anwendungsteile im Prototypen nicht umgesetzt.
- Der Prototyp ist *nicht vollständig* bezüglich der fachlichen und nicht-funktionalen Anforderungen an das Gesamtsystem. Er stellt stattdessen einen kleinen Ausschnitt des Systems dar.
- Er wird eher „quick-and-dirty" als perfekt umgesetzt.
- Die Fragestellung kann experimentell angegangen werden.
- Dokumentiert werden nur die Ergebnisse bezüglich des Prototypziels, eine formale Quellcode-Dokumentation unterbleibt.
- Die Implementierung geht *nicht* ohne weitere Überarbeitung in das Produktivsystem über.

Dennoch ist der Aufwand für einen solchen Prototypen keineswegs verschenkt. Vielfach lassen sich zentrale Komponenten aus dem Prototypen herauslösen und wiederverwenden.

Schnelle Lösungen führen manchmal zu Defiziten in der Qualität der Implementierung. Aus diesem Grund sollte der Code niemals ungeprüft komplett in das Produktivsystem übernommen werden. Stattdessen müssen wiederverwertbare Komponenten für das Zielsystem neu implementiert oder zumindest überarbeitet werden. Häufig kann eine Neu-Implementierung bei einem verbesserten Design sehr schnell umgesetzt werden. Dieses Phänomen beschreibt Kent Beck bereits in [Bec99].

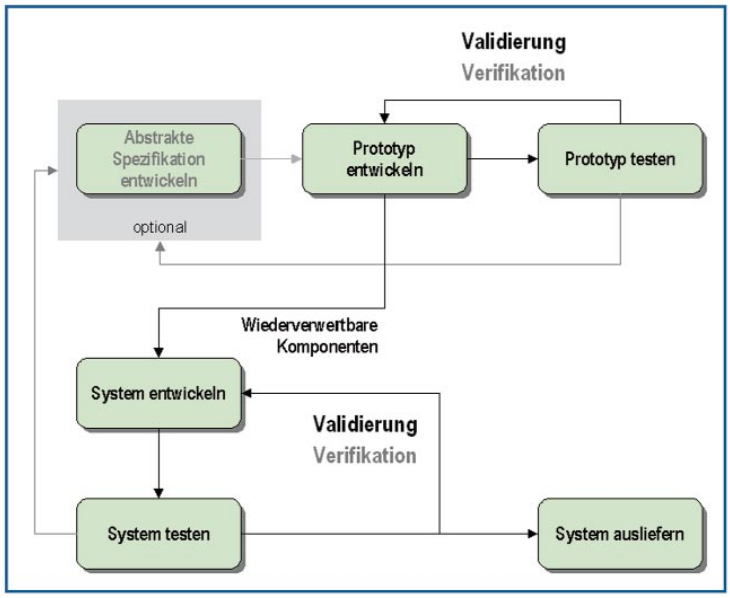

Abb. 2: Iterativer Softwareentwicklungsprozess mit Einsatz von Wegwerf-Prototypen (vgl. [Som95])

Benutzerschnittstellen-Prototyp

Benutzerschnittstellen-Prototypen dienen der Festlegung der (meist grafischen) Benutzerschnittstelle des Systems. Sie werden auch als *UI-Prototyp* oder *GUI-Prototyp* bezeichnet.

Die Schnittstelle zum Anwender hängt bei kommerziellen Informationssystemen sehr stark von den fachlichen Anforderungen ab. Häufig werden fachliche Details sogar erst beim Maskenentwurf festgelegt. Die Entwicklung des UI-Prototypen dient dann zur Unterstützung der Anforderungsanalyse, die auf Grund ihrer visuellen Ergebnisse die Kommunikation mit dem Anwender erleichtert.

Zur Umsetzung des Prototypen gibt es verschiedene Techniken:

- *Slide Show:* Bildschirmmasken werden den Anwendern in einer vorgegebenen Reihenfolge gezeigt (als Präsentation oder auf Papier). Hierdurch kann ein Ablauf simuliert werden.
- *Software-Prototyp:* Masken und Ablaufsteuerung werden als lauffähiges Programm zur Verfügung gestellt. Testdaten werden hierbei simuliert.
- *Wizard of Oz:* Ein Mensch simuliert den Computer. Dieses Verfahren ist für Frage/Antwort-Systeme oder natürlichsprachliche Systeme ohne grafische Benutzeroberfläche (GUI) geeignet.

Technisch oder fachlich?

Technische Prototypen ignorieren die fachlichen Systemanforderungen. Stattdessen konzentrieren sie sich ausschließlich auf die Umsetzung und Bewertung von Lösungsalternativen und die Überprüfung von Architektur-Anforderungen (Performance, Robustheit, Verfügbarkeit etc.). Sie identifizieren Schwachstellen und bilden das Fundament für Machbarkeitsstudien.

Fachliche Prototypen modellieren Teile der Benutzerschnittstelle oder setzen Teile der Fachlogik um. Hierbei werden technische Details außen vor gelassen.

Horizontal oder vertikal?

Ein *horizontaler Prototyp* bietet einen möglichst breiten Überblick über das Gesamtsystem, bezogen auf *eine* kon-

krete Fragestellung. Diese kann sowohl technisch als auch fachlich motiviert sein. Details werden hierbei vernachlässigt.

Architektur-Prototypen sind Beispiele für technische horizontale Prototypen. Sie betrachten einen minimalen fachlichen Ausschnitt der zu erstellenden Anwendung. Dafür enthalten sie jedoch alle wichtigen Systemkomponenten, wenn auch in rudimentärer Form. Dies reicht von der Benutzeroberfläche, über die Middleware bis hin zur Anbindung von Schnittstellen und persistenten Daten.

Ein fachlicher horizontaler Prototyp setzt hingegen einen Großteil der fachlichen Anforderungen um, ohne sich um technische Details zu kümmern. In stark datenzentrierten Anwendungen („Suchen-Anzeigen-Bearbeiten") entspricht ein derartiger Prototyp in der Regel dem Benutzerschnittstellen-Prototypen.

Vertikale Prototypen beleuchten spezielle Aspekte des Systems in aller Tiefe, ohne einen Anspruch auf Vollständigkeit zu besitzen. Technische vertikale Prototypen liefern häufig ausgefeilte Lösungen für schwierige Implementierungsprobleme. Ihre Komponenten lassen sich deshalb besonders gut in das Endprodukt integrieren.

Einsatzszenarien

Welcher Prototyp ist für welche Fragestellungen geeignet? Sollte der Prototyp evolutionär ausgebaut werden oder ist es besser die Ergebnisse in das Gesamtsystem zu integrieren? **Tabelle 1** bietet eine Übersicht über die häufigsten Ziele eines Prototypen und geeignete Prototyp-Arten. Mit Hilfe eines *technischen Prototypen* können kritische Einflussfaktoren bereits in einem frühen Projektstadium identifiziert werden. Im Umfeld sehr dynamischer Technologien, wie z. B. J2EE-Architekturen oder Web-Services, sind sie als Wegwerf-Prototypen ein wichtiges Mittel im Rahmen des Risiko-Managements. Sie bieten ein sicheres Fundament für die Technologieauswahl und helfen bei der Bewertung unterschiedlicher Implementierungsansätze.

Mit Ausnahme von Architektur-Prototypen und Machbarkeitsstudien sollten derartige Prototypen nicht evolutionär ausgebaut werden. Die Umsetzung der fachlichen und nicht-funktionalen Anforderungen überfrachten den Prototyp. Die Fokussierung auf die eigentlichen Ziele geht verloren und die Entwicklung wird langsam und schwerfällig.

Manche Projekte versuchen bei der Weiterentwicklung des evolutionären Prototypen gleichzeitig technische Lösungsalternativen zu evaluieren. Ein Beispiel hierfür ist ein J2EE-Projekt, das ein halbes Jahr am evolutionären Prototypen arbeitete. Der Prototyp umfasste bereits eine Vielzahl von Masken. Nun sollte dieser Prototyp die Frage beantworten, ob ein spezielles Persistenz-Framework einsetzbar sei. Auf Grund der bereits vorhandenen fachlichen Breite existierten sehr viele Modell-Klassen, welche die Masken mit Daten versorgten.

Für den Iterationsschritt zur Framework-Evaluierung wurden 90% des geplanten Aufwands allein für die Anpassung der Modell-Klassen verbraucht. Am Ende der Iteration war das Framework zwar integriert worden, für Lasttests usw. war jedoch keine Zeit mehr. Es gab keinerlei qualitative Aussagen bezüglich der Performance, dem Transaktionsverhalten und der Stabilität. Mehr als ein kurzes „Es funktioniert" war als Antwort leider nicht drin!

Ein solches Vorgehen führt am Ziel vorbei und lässt die Aufwände explodieren. Es fördert Fehlentscheidungen, da die Evaluierung *einer* Technologie bereits soviel Zeit und Geld gekostet hat. Die Entscheidung *gegen* den Untersuchungsgegenstand fällt schwer, da vordergründig keiner der Projektbeteiligten einen Vorteil davon hätte: Das Realisierungsteam möchte nicht noch einmal eine Vielzahl von Klassen anpassen und die Projektleitung kann die neuen Aufwände ihrem Auftraggeber schwerlich plausibel machen.

Projekte, die agil vorgehen, klären technische Detailfragen *außerhalb* eines evolutionären Prototypen. Schnelle Lösungen und Abkürzungen sind in der Implementierung erlaubt. Das gewährleistet eine rasche und kostengünstige Evaluierung. Funktionierende Komponenten werden einer Refaktorierung unterzogen und zu *einem passenden Zeitpunkt* in die Entwicklung des Zielsystems integriert. Dies entkoppelt die Entwicklung des Produktivsystems von der Klärung technischer Details.

Lauffähige *Benutzerschnittstellen-Prototypen* beeindrucken den Auftraggeber immer besonders und erlauben ein bestmöglichstes Feedback der Anwender. Nicht immer müssen derartige Prototypen jedoch horizontal ausgebaut werden. Sie sollten das Kosten/Nutzen-Verhältnis sorgfältig abwägen, da der Realisierungsaufwand meist beträchtlich ist.

Die Auswahl geeigneter Tools zur Erstellung von GUI-Prototypen ist schwierig. Häufiges Manko sind fehlende Komponenten zur formatierten Eingabe, zur einfachen Testdatenanbindung etc. Vielfach enthalten nur 4GL-Werkzeuge eine umfassende Bibliothek komplexer GUI-Elemente. Hiermit können komfortable und optisch ansprechende Masken zwar schnell erstellt werden, aber die vom Werkzeug vorgegebene System- und Schichtenarchitektur ist für größere Projekte häufig ungeeignet.

Prototyp-Arten	Fachlich	Technisch
Horizontal	fachliche Spezifikation	E Validierung der Architektur
E	Benutzerschnittstelle	
Vertikal	Validierung ausgewählter Aspekte der Fachlogik	Style-Guide der Benutzerschnittstelle
	Festlegung der Ablaufsteuerung und des Interaktionsstil der Anwendung (vgl. [Sta02])	Validierung nicht-funktionaler Anforderungen (Performance, Transaktionsverhalten etc.)
		E Machbarkeitsstudien
		Technologieauswahl
		Bewertung von Lösungsalternativen

Alle Prototypen können als Wegwerf-Prototyp umgesetzt werden, wobei einzelne Komponenten später wieder verwendet werden können. Prototypen mit der Kennzeichnung E eignen sich darüber hinaus für eine evolutionäre Entwicklung.

Besitzt das Zielsystem eine Java-Swing-Oberfläche und enthalten die Masken sehr viele Tabellen und Baumansichten, so ist ein evolutionärer Ausbau des GUI-Prototypen ebenfalls oft ungünstig. Bei diesen komplexen GUI-Elementen ist die Versorgung mit Testdaten kompliziert und es wird sehr viel Aufwand in die Erstellung von Masken gesteckt, die sich noch häufig ändern werden. Mit einem *Slide-Show*-Prototypen oder einer *Quick-and-Dirty*-Lösung eines 4GL-Tools kann häufig effektiver vorgegangen werden.

Die Klärung fachlicher Fragen an Hand der Benutzeroberfläche sowie die Festlegung des Interaktionsstils kann so von der technischen Umsetzung der Masken entkoppelt werden. Während das Analyseteam zum x-ten Male die Masken mit den Anwendern abstimmt, kann das Entwicklungsteam bereits festgelegte Masken umsetzen, ohne sich mit Vorarbeiten für die nächste Diskussionsrunde zu beschäftigen.

Randbedingungen und Stolpersteine

Prototyping entspricht der *Bottom-Up*-Vorgehensweise. Das Gesamtsystem entsteht allmählich aus vielen Einzelergebnissen. Die schrittweise Verfeinerung lässt Spielraum für Kreativität und erlaubt ein flexibles Reagieren auf neue Anforderungen in einem dynamischen Umfeld. Die Methode ist deshalb unverzichtbarer Bestandteil der meisten agilen Prozesse.

Die große Flexibilität ist die Stärke dieses Ansatz. Sie kann jedoch bei unzureichender Projektsteuerung dazu führen, dass das Entwicklungsteam sich verzettelt. Hierin liegt die eigentliche Gefahr, insbesondere beim evolutionären Prototyping.

Klare und begrenzte Ziele sind unverzichtbar. Eine Vermischung fachlicher und technischer Aufgaben führt zu schwerfälligen Prototypen, in denen sich die Teilteams gegenseitig behindern. Die Entwicklung des Zielsystems kann sogar vollständig zum Stillstand kommen, wenn die Klärung komplexer Implementierungsfragen nicht in gesonderte Wegwerf-Prototypen verlagert wird.

Die Vorteile eines Wegwerf-Prototypen liegen *nicht* auf der Hand: Für dieses Vorgehen muss die Projektleitung aktives Marketing bei fast allen Projektbeteiligten betreiben. Für den Auftraggeber klingt schon die Bezeichnung „Wegwerf-Prototyp" nach dem sprichwörtlich aus dem Fenster hinaus geworfenem Geld. Wozu Ressourcen und Geld verschwenden, wenn die Ergebnisse doch in der großen Tonne landen? Das Wort „Wegwerf-Prototyp" sollte in Ihren Management-Präsentationen nicht auftauchen! Vermeiden Sie negative Assoziationen und sprechen Sie besser von „Realisierungsstudien" oder „Technischen Prototypen". Machen Sie klar, dass die Ergebnisse weiterverwendet werden – und zwar in modifizierter und verbesserter Form.

Eine besondere Gefahr liegt darin, dass Auftraggeber und Management nicht selten einen Prototypen mit dem Endprodukt verwechseln. Insbesondere lauffähige, horizontale GUI-Prototypen bergen diese Gefahr. Je positiver der Prototyp angenommen wird, desto größer ist die Wahrscheinlichkeit der Fehleinschätzung.

Hier ein Beispiel: Ein Projekt präsentierte seinen horizontalen GUI-Prototypen äußerst erfolgreich, Anwender und Management waren begeistert. Die Euphorie des Projektleiters lies jedoch schnell nach. Er wurde gefragt, was er denn in der langen Restlaufzeit des Projekts machen wolle, das Produkt sei doch schon so gut wie fertig! Die anschließende Diskussion war schwierig, für den Auftraggeber war (zumindest optisch) ja schon alles da: das bisschen Performance und Stabilität, und dann noch schnell die Anbindung an Applikationsserver und Datenbank...

Dieser Misere sollten Sie vorbeugen, indem Sie durch klare Informationen über den Leistungsumfang des Prototypen die Erwartungshaltung begrenzen. Eine andere Alternative ist eine Beschränkung des GUI-Prototypen auf ausgewählte Anwendungsfälle. Die restlichen Masken, denen gleichartige Abläufe zu Grunde liegen, können dann beispielsweise als Papier-Prototyp erstellt werden.

Prototyping im Team

Evolutionäre Prototypen lassen sich umso besser umsetzen, je kleiner und qualifizierter das Team ist. Ein Team mit vielen Anfängern bringt nicht die nötige Erfahrung mit, um eine effiziente Architektur auch ohne vollständige Spezifikation auf- und umsetzen.

Wegwerf-Prototypen werden auf Grund ihres begrenzten Umfangs meist von Ein- oder Zwei-Personen-Teams erstellt. Aber nicht jeder Entwickler ist für die Umsetzung eines Wegwerf-Protopyen geeignet. Das „Wegwerfen" der Arbeitsergebnisse kann auf manche demotivierend wirken. Andere empfinden den steten Neuanfang als Bereicherung, da sie sich nicht um die Konsistenz einer großen Anzahl von Komponenten und Klassen kümmern müssen.

Hedwig Keller beschreibt in [Kel00] unterschiedlichen Aufgabenorientierungen von Softwareentwicklern. Danach sind die „Ideenschleudern" und der „Prototyper" hervorragend für die Mitarbeit an Wegwerf-Prototypen geeignet. „Detaillisten" können für knifflige technische Detailfragen eingesetzt werden. „Sammler" sind hingegen für alle Arten von Prototypen ungeeignet, da sie sich immer wieder von Teilen ihrer Arbeit trennen müssen. Die gelungene Zuordnung der Entwickler zu speziellen Realisierungsaufgaben wird damit zu einem der Schlüsselfaktoren für einen erfolgreichen Prototypen.

Evolution ohne Dogma

In vielen Fällen kann ein iterativer Softwareentwicklungsprozess effektiv durch die Entwicklung eines evolutionären Prototypen umgesetzt werden. Fast immer muss der Prototyp jedoch noch durch kleinere technische Wegwerf-Prototypen ergänzt werden.

Nicht jeder kleinste Entwicklungsschritt muss in allen Details dokumentiert werden. Das macht ein Verwerfen und Verbessern nur unnötig teuer. Der Quellcode, das Design und die Architektur werden immer dann dokumentiert, wenn es für das Entwicklungsteam hilfreich ist. Zu Beginn der Realisierung muss jedoch vereinbart werden, welchen Dokumentationsgrad das System bei der Auslieferung haben soll. Eine Dokumentation „auf den letzten Drücker" ist ebenso unsinnig wie die stoische Dokumentation jeder getXY()-Methode. Stattdessen sollen Entwicklungsteams stabile Komponenten nach und nach im Projektverlauf dokumentieren.

Die scheinbar offensichtliche Gleichung „iterativ = evolutionär" ist falsch. Agile Projekte setzen auf die Kombination verschiedenartiger Prototypen, die ihrer jeweiligen Aufgabe am besten gerecht werden. Sie schaffen dem Realisierungsteam ein Umfeld, in dem *Experimentieren*, *Verwerfen* und *Verbessern* erlaubt sind. ■

Literatur & Links

[Bäu96] D. Bäumer, W.R. Bischofberger, H. Lichter, H. Züllighoven, Objektorientiertes Prototyping – Konzepte, Werkzeuge, Erfahrungen, Proc. Softwaretechnik 1996

[Bec99] K. Beck, Extreme programming explained, Addison-Wesley 1999

[Ber] L. Bernstein, Importance of Software Prototyping (siehe http://www.dacs.dtic.mil/awareness/newsletters/technews2-1/prototyping.html)

[Fow00] M. Fowler, Refactoring, Addison-Wesley 2000

[Kel00] H. Kellner, Projekte konfliktfrei führen, Hanser 2000

[Opf01] S. Opferkuch, Prototyping / Extreme Programming (siehe http://studweb.stud.uni-stuttgart.de/studweb/users/inf/inf24723/hase01/Ausarbeitung.html)

[Pre94] J. Preece, Y. Rogers, H. Sharp, D. Benyon, Human-computer interaction, Addison-Wesley 1994

[Som95] I. Sommerville, Software Engineering, Addison-Wesley 1995

[Sta02] G. Starke, Effektive Software-Architekturen, Hanser 2002