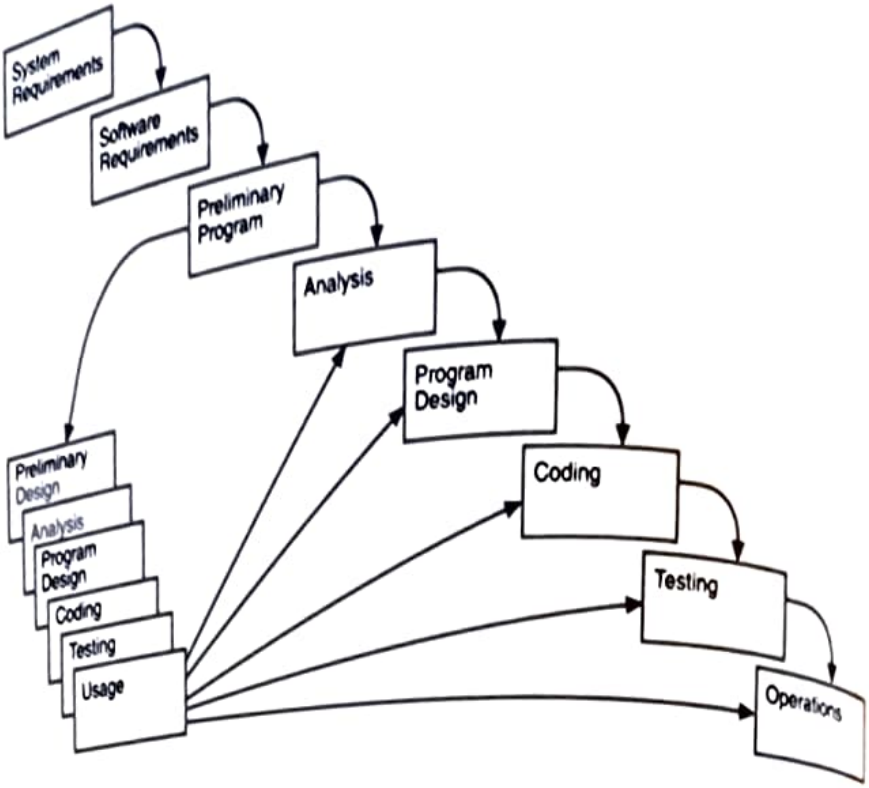

Abb. 9–5 *Attempt to do the job twice – the first result provides an early simulation of the final product (aus Royce, 1970)*

Wir führen nachfolgend zunächst die zentralen Begriffe Prototyp und Prototyping ein. Dann stellen wir die unterschiedlichen Arten von Prototypen und Prototyping vor. Dabei stützen wir uns weitgehend auf die Arbeiten von Floyd (1984), Kieback et al. (1992) und Lichter, Schneider-Hufschmidt, Züllighoven (1994).

9.4.1 Der Begriff des Prototyps

Im Maschinenbau ist es üblich, neue Geräte, Autos usw. zu erproben, bevor man sie in einer Großserie fertigt. Man baut, wenn die eigentliche Entwicklung weitgehend abgeschlossen ist, darum zunächst einen Prototyp, also ein einzelnes Exemplar, das – vor allem in den kritischen Merkmalen – dem geplanten Massenprodukt entspricht. Die Vorteile sind offensichtlich: Würde man gleich nach der Konstruktion die sehr teure Fertigungsanlage aufbauen, so müsste diese nach Erprobung der ersten Produkte und der folgenden Verbesserung des Entwurfs mit hohem Aufwand verändert werden. Würde die Konzeption nach der Erprobung ganz verworfen, so wäre die Anlage nahezu wertlos. Ein Prototyp gestattet es, die Fertigung nur dann und nur so aufzubauen, wenn sie und wie sie wirklich gebraucht wird.

Dieser Gedanke wurde im Software Engineering übernommen. Allerdings hat die immaterielle Natur der Software zur Folge, dass die Verhältnisse ganz anders

liegen als im Maschinenbau: Eine aufwändige Fertigung gibt es bei Software nicht. Wenn wir am Ende der Entwicklung einen voll funktionsfähigen Prototyp haben, ist der Rest der Arbeit trivial, wir müssen nur Kopien in der gewünschten Zahl anfertigen und diese verteilen oder verkaufen. Darum hat das Wort Prototyp bei uns eine ganz andere Bedeutung als beispielsweise im Autobau: Was wir herstellen, ist ein *Mock-up*, eine *Attrappe*. Das Wort »Attrappe« (von französisch attrape = Falle) lässt erkennen, dass es sich ursprünglich um einen Lockvogel handelt, der die echten Vögel in die Falle locken sollte. Wir alle kennen Attrappen aus den Möbelläden, wo in den Regalen Gegenstände stehen, die aus leichtem Kunststoff hergestellt sind und nur eine äußere Ähnlichkeit mit den Dingen haben, die sich die Kunden später ins Regal stellen werden. Software-Prototypen weisen typischerweise nur eine sehr rudimentäre Funktionalität auf. In den meisten Fällen zeigen sie die Oberfläche des (vermuteten) Zielsystems.

Kurz: Wir bezeichnen mit dem Wort »Prototyping« eine Vorgehensweise der Software-Entwicklung, bei der Attrappen, die wir Prototypen nennen, entworfen, konstruiert, bewertet und revidiert werden. Der Zweck des Software-Prototypings ist, die Anforderungen zu klären und damit eine lange und teure Entwicklung zu vermeiden, die am Ende ein Produkt liefert, das der Kunde so nicht haben will.

Das IEEE-Glossar liefert uns folgende Definitionen:

prototype — A preliminary type, form, or instance of a system that serves as a model for later stages or for the final, complete version of the system.

prototyping — A hardware and software development technique in which a preliminary version of part or all of the hardware or software is developed to permit user feedback, determine feasibility, or investigate timing or other issues in support of the development process.

rapid prototyping — A type of prototyping in which emphasis is placed on developing prototypes early in the development process to permit early feedback and analysis in support of the development process.

IEEE Std 610.12 (1990)

Hinter dem Prototyping stehen folgende Überlegungen und Absichten:

Wenn wir einen gewünschten Gegenstand beschreiben, halten wir uns, wenn es geht, an Vorbilder, wir sagen etwa: »Ich hätte gern einen Koffer, wie er im Schaufenster steht, aber ein bisschen größer und mit einem speziellen Fach für meinen Laptop.« Wir gehen also von einem Muster aus und korrigieren nur einzelne Attribute. Wenn wir kein Muster haben, weil es nicht einmal einen *ähnlichen* Gegenstand gibt, werden wir sehr wahrscheinlich keine sinnvollen Anforderungen formulieren, zumindest werden wir wichtige Anforderungen vergessen. Im Software Engineering sind wir oft, fast regelmäßig, in der Situation, dass es bislang kein ähnliches System gibt. Wir schaffen es selbst in Form des Prototyps, der einen Teil der Anforderungen realisiert. Er ermöglicht uns nicht nur die Überprüfung dieser Anforderungen, sondern dient gleichzeitig als

Bezugspunkt zur Formulierung neuer und abweichender Anforderungen. Beson-
ders schwierig ist erfahrungsgemäß die abstrakte Beschreibung einer Bedien-
oberfläche. Darum haben die meisten Prototypen hier ihren Schwerpunkt.
- In den meisten Fällen lässt sich eine Aufgabe auf sehr unterschiedliche Weisen lösen. Die Erfahrungen bei der Realisierung des Prototyps geben Hinweise, ob ein bestimmter Lösungsansatz geeignet ist. Natürlich ist die Aussagekraft dazu beschränkt, weil der Prototyp die gewünschte Funktionalität nicht oder nur sehr rudimentär zeigt.

Daraus lassen sich folgende allgemeine Aussagen zu Software-Prototypen ableiten:

- *Ein Prototyp ist lauffähig.*
  Eine reine Bildschirmmaske, die mit einem Grafikeditor erstellt wurde, ist kein Prototyp.
- *Ein Prototyp realisiert ausgewählte Aspekte des Zielsystems.*
  Welche Aspekte in einem Prototyp umgesetzt werden, richtet sich nach den speziellen Fragen, die zu klären sind. Wichtig ist, dass die Aspekte, die ein Prototyp realisieren soll, *vor* seiner Entwicklung festgelegt werden, weil diese die Konstruktion maßgeblich beeinflussen. Außerdem bestimmen sie, welche Fragen von einem Prototyp beantwortet werden können, und damit auch, welche nicht an ihn gestellt werden sollten. Zumeist, insbesondere bei interaktiven Anwendungen, werden sich diese Fragen auf die Bedienoberfläche beziehen. Allerdings ist das nicht zwingend. Ein Prototyp kann sehr wohl modellieren, wie eine Anwendung mit einer Datenbank zusammenarbeitet, während von der Darstellung der späteren Oberfläche weitgehend abgesehen wird.
- *Ein Prototyp wird von Klienten geprüft und detailliert bewertet.*
  Zeigt sich dabei, dass der Prototyp stark vom gewünschten System abweicht, so wird er modifiziert, bis die Klienten im Wesentlichen einverstanden sind.

Ein Prototyp, den der Klient akzeptiert hat, ist somit ein Bestandteil der Anforderungsspezifikation.

9.4.2 Prototypentwicklung

Prototypen sollten immer dann entwickelt werden, wenn wichtige Anforderungen fehlen oder nur vage und unvollständig formuliert werden können. Ein Prototyp kann helfen, diese Situation zu verbessern. Damit ein Prototyp zielgerichtet entwickelt und bewertet werden kann, müssen die folgenden Fragen im Vorfeld beantwortet werden:

- Welche Aufgabe, welchen Zweck hat der Prototyp, wie lauten also die offenen Fragen?

- Welche Personengruppen sind an der Entwicklung und insbesondere an der Bewertung des Prototyps beteiligt?
- Wie lange darf die Prototypentwicklung dauern und welche Kosten dürfen entstehen?

Auf dieser Basis wird der Prototyp entwickelt. Dazu werden spezielle Entwicklungsumgebungen eingesetzt, die es erlauben, den Prototyp möglichst schnell und kostengünstig zu realisieren. Anschließend beginnt der Zyklus, in dem der Prototyp benutzt, bewertet und modifiziert wird. Abbildung 9–6 zeigt den Prototyp-Entwicklungsprozess (angelehnt an Lichter, 1993).

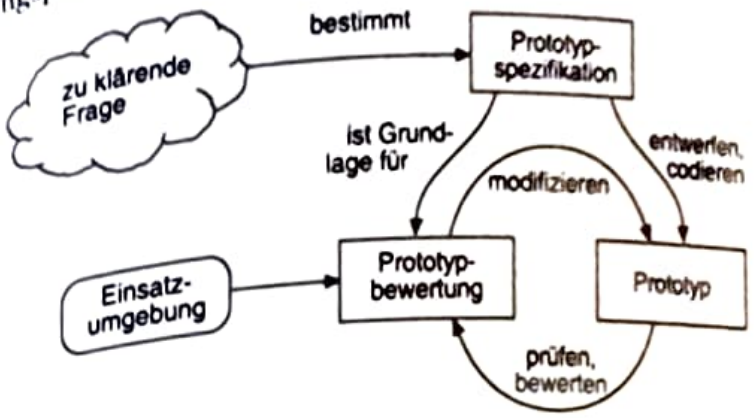

Abb. 9–6 *Allgemeine Vorgehensweise beim Prototyping*

Der Zyklus rechts unten in der Grafik wird so lange durchlaufen, bis der untersuchte Aspekt zufriedenstellend im Prototyp modelliert ist. Der Prototyp, der zuerst ein deskriptives Modell und damit ein Abbild der ursprünglich vermuteten Anforderungen war, wird durch den Zyklus von Prüfung und Modifikation zu einem präskriptiven Modell, einer Vorgabe für das Zielsystem. Wir haben also ein explorativ Modell vor uns (Abschnitt 1.3.4).

9.4.3 Spezielle Prototypen und verwandte Begriffe

Das Gebiet des Prototypings ist durch eine verwirrende Vielfalt von Begriffen gekennzeichnet, die sich nicht in ein übersichtliches Schema bringen lassen. Wir erläutern nachfolgend einige gängige Begriffe. Es ist nicht zu vermeiden, dass wir dabei im Zweifel unserem Verständnis folgen, ohne dafür Literatur-Autoritäten anführen oder gar einen Konsens der Fachleute präsentieren zu können.

Nach der Art, wie die Prototypen im Entwicklungsprozess eingesetzt werden, unterscheiden wir in Anlehnung an Kieback et al. (1992) die folgenden Konzepte:

- Ein *Demonstrationsprototyp* zeigt die prinzipiellen Einsatzmöglichkeiten, die mögliche Handhabung des künftigen Systems. Demonstrationsprototypen helfen besonders in der Start- oder Akquisitionsphase eines Projekts, Ent-

scheidungen vorzubereiten und eine Vision des zukünftigen Systems zu entwickeln. Sie sind daher »Wegwerfprodukte«, die schnell und einfach gebaut werden. (Natürlich werden sie nicht wirklich weggeworfen, sie werden nur nicht Teil des Zielsystems.) Ihre fachliche Detailtreue und ihr softwaretechnischer Standard spielen keine Rolle. Demonstrationsprototypen vermitteln dem Klienten einen ersten Eindruck davon, wie ihr System aussehen könnte. Sie sind noch ausreichend weit vom Zielsystem entfernt, um daran auch die Einschränkungen eines Prototyps zu zeigen und damit falschen Erwartungen vorzubeugen.

- *Funktionale Prototypen* modellieren in der Regel Ausschnitte der Bedienoberfläche und Teile der Funktionalität. Diese Prototypen werden parallel zur Analyse des Anwendungsbereichs erstellt und unterstützen die Anforderungsanalyse. Sie können in ihrer Architektur bereits dem Zielsystem entsprechen. Falls die Fragen an den Prototyp inhomogen und offen sind, ist es zweckmäßiger, mehrere kleine Prototypen für einzelne Fragen zu konstruieren, als einen einzigen sehr umfangreichen Prototyp zu bauen.
- Ein *Labormuster* modelliert einen technischen Aspekt des Zielsystems. Dieser Aspekt kann sich je nach der zu klärenden Frage auf die Architektur oder auf die Funktionalität beziehen. Labormuster sind für die Entwickler ein Experimentalsystem und eine Form von Machbarkeitsstudie. Sie müssen daher nicht zwingend in das Zielsystem eingehen.
- Ein *Pilotsystem* ist ein Prototyp, dessen Funktionalität und Qualität mindestens für einen vorübergehenden echten Einsatz ausreichen. Er realisiert einen abgeschlossenen Teil des Zielsystems und wird schrittweise ausgebaut. Auch seine komfortable und sichere Bedienbarkeit und ein Mindestmaß an Benutzungsdokumentation unterscheiden ihn qualitativ von anderen Prototypen.

Man beachte den großen Unterschied zwischen einem Wegwerfprototyp und Prototypen, die später Teil des Produkts werden: Nur was nicht ins Produkt gelangt, darf nach dem Prinzip »quick and dirty« entwickelt werden. Darum ist die Entscheidung für oder gegen Wegwerfen zu Beginn der Prototypentwicklung zu treffen.

9.4.4 Die Taxonomie von Floyd

Floyd (1984) klassifiziert Prototyping nach dem angestrebten Ziel in drei Arten:

Exploratives Prototyping
Der Prototyp wird mit dem Ziel erstellt, die Analyse zu unterstützen und zu ergänzen. Dazu werden ggf. alternative Prototypen konstruiert. Als Prototyparten kommen meist Demonstrationsprototypen und funktionale Prototypen in Frage. Exploratives Prototyping eignet sich am ehesten zur Integration in ein konventionelles Vorgehensmodell. In der Literatur findet man für diese Art des Prototypings häufig die Bezeichnung *Rapid Prototyping* (Abschnitt 9.5.1).

- *Experimentelles Prototyping*
  Die Betonung bei dieser Form des Prototypings liegt auf der technischen Umsetzung eines Entwicklungsziels. Einerseits sollen die Benutzer im Experiment ihre Vorstellungen vom Zielsystem weiter detaillieren, andererseits können die Entwickler dadurch besser einschätzen, ob das geplante System realisierbar und zweckmäßig ist. Die Kommunikation zwischen Benutzern und Entwicklern über technische und software-ergonomische Fragen steht dabei im Vordergrund: Neben funktionalen Prototypen werden auch Labormuster zur Klärung technischer Fragen konstruiert.
- *Evolutionäres Prototyping*
  Prototyping wird nicht nur als Hilfsmittel innerhalb eines einzelnen Entwicklungsprojektes eingesetzt, sondern kann auch ein Prozess sein, um ein System den sich rasch verändernden Randbedingungen anzupassen. Damit verliert die Software-Entwicklung den Charakter eines abgeschlossenen Projekts und wird zu einem evolutionären Prozess, der die Anwendung ständig begleitet. Wenn kurze Entwicklungszyklen angestrebt werden, ist es sinnvoll, den Unterschied zwischen Prototyp und Zielsystem aufzuheben und Pilotsysteme zu entwickeln. Der Begriff der evolutionären Entwicklung (Abschnitt 9.5.2) ist mit diesem evolutionären Prototyping eng verwandt.

Es sollte klar sein, dass sich die genannten Kategorien überlappen, dass es also keine scharfe Abgrenzung zwischen ihnen gibt.

9.5 Nichtlineare Vorgehensmodelle

In diesem Abschnitt werden das Rapid Prototyping und einige andere nichtlineare Vorgehensmodelle näher betrachtet. Das Spiralmodell wird als Metamodell separat in Abschnitt 9.6 behandelt. Abbildung 9–7 zeigt die Gemeinsamkeiten und Unterschiede der fünf betrachteten Ansätze.

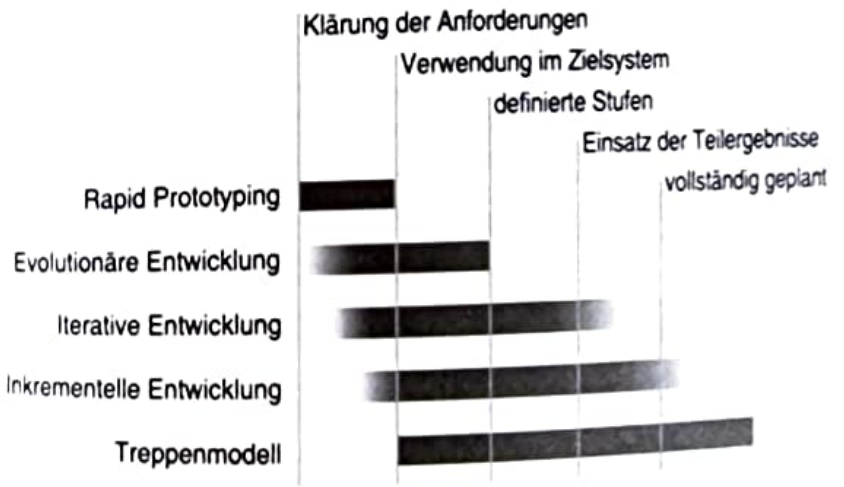

Abb. 9–7 *Merkmale der nichtlinearen Vorgehensmodelle*

Wie man sieht, gibt es zwischen den beiden Extremen (Rapid Prototyping und Treppenmodell) keine Überlappung. Es ist darum sicher nicht sinnvoll, für alle Ansätze dasselbe Wort »Prototyping« zu verwenden, wie es immer wieder geschieht.

Häufig werden Begriffe wie Prototyping, evolutionäre oder inkrementelle Entwicklung missbraucht, um ein planloses Vorgehen (Aktivitäten werden in irgendeiner Reihenfolge durchgeführt) mit einer wohlklingenden Bezeichnung zu kaschieren und zu rechtfertigen. Das ist natürlich Etikettenschwindel; jedes Vorgehensmodell erfordert Planung und Kontrolle. Goldberg und Rubin (1995) vergleichen das chaotische Vorgehen mit der Taktik, wie man ein Puzzle zusammenbauen kann: mal hier ein wenig, mal dort ein wenig. Sie sprechen in einem solchen Fall von einem opportunistischen Vorgehensmodell.

9.5.1 Rapid Prototyping

Wenn im Software Engineering von *Rapid Prototyping* die Rede ist, dann ist die Entwicklung von Software-Attrappen gemeint, die dazu dienen, Anforderungen zu klären, die sich abstrakt, also durch Text oder formale Modelle, kaum klären lassen. Ein Klient wünscht vielleicht, viele Messdaten zugleich auf dem Bildschirm zu sehen; ein Prototyp, der ihm die unübersichtliche Datenflut demonstriert, wird ihn wahrscheinlich veranlassen, Alternativen in Erwägung zu ziehen. In der Terminologie von Lehman (Abschnitt 9.3.1) sorgen wir dafür, dass (mindestens) ein Innovationszyklus durchlaufen ist, bevor wir großen Aufwand in eine Implementierung stecken.

Zwei populäre Missverständnisse müssen vermieden werden: Ein Prototyp ersetzt nicht die Spezifikation, und ein Prototyp darf auch nicht in beliebig schlechter Qualität realisiert sein.

Die Spezifikation ist auch mit einem Prototyp unentbehrlich, weil man ja auch den Prototyp nicht ins Blaue hinein entwickeln kann; man braucht Vorgaben, die nur in der Spezifikation zu finden sind. Diese Vorgaben können natürlich dort weniger detailliert sein, wo der Prototyp die Klärung bringt. Zum anderen ist der Prototyp nur teilweise Vorbild für das Produkt. All die Merkmale, die nicht demonstriert werden, in der Regel fast die ganze Funktionalität, deckt der Prototyp nicht ab. Schließlich ist nicht immer klar, wo die Grenze zwischen Vorbild und Versuchsaufbau liegt: Die Spezifikation muss präzise definieren, welche Merkmale des Prototyps ins Zielsystem übernommen werden sollen.

Ein Prototyp ist dort nützlich, wo die Anforderungen nicht durch Beobachtungen und Fragen erhoben werden können. Wir müssen also damit rechnen, dass die Klienten mit der ersten Version nicht einverstanden sind, sondern Änderungswünsche haben. Diese sollten im Prototyp umgesetzt werden; andernfalls führt das Prototyping nur zu einer Aussage, was die Klienten *nicht* wünschen. Die Änderung setzt aber eine minimale Wartbarkeit des Prototyps voraus. Er muss

weder portabel noch perfekt kommentiert sein, aber er muss sauber genug strukturiert sein, dass notwendige Änderungen mit geringem Aufwand möglich sind.

Der Prototyp ist Teil der Anforderungsspezifikation, er wird nicht Teil des Produkts, andernfalls handelt es sich um eine evolutionäre Entwicklung (siehe unten). In der Praxis wird diese Regel sehr oft außer Kraft gesetzt, die Entwickler werden von ignoranten Vorgesetzten praktisch gezwungen, den Prototyp zum Produkt auszubauen. Wer diese Gefahr sieht und keine Möglichkeit hat, sie zu bannen, sollte sich nicht auf das Rapid Prototyping einlassen. Wer zu einem Spaziergang aufbricht, ist einfach nicht dafür gerüstet, ein paar Stunden später in der Steilwand zu klettern.

9.5.2 Evolutionäre Entwicklung

Mit dem Begriff der Evolution verbinden die meisten Menschen den Namen des Naturforschers Charles Robert Darwin (1809 bis 1882), der mit seiner Theorie der Evolution (*On the Origin of Species by Means of Natural Selection*, 1859) ein neues Weltbild schuf.

Wenn im Software Engineering von Evolution die Rede ist, meint man aber in der Regel nicht die Evolution einer Art, sondern die eines Individuums, nämlich eines Software-Systems. Die *evolutionäre Entwicklung* (*iterative enhancement*) ist dem Prototyping in manchen Punkten ähnlich, unterscheidet sich aber in anderen Punkten deutlich. Auch hier ist die Ausgangssituation, dass die Anforderungsanalyse keine für die Entwicklung ausreichenden Resultate liefert. Anders als das Prototyping zielt diese Strategie aber darauf ab, durch Zyklen aus Erprobung und Verbesserung das Produkt so oft zu verändern und zu erweitern, bis es sich als brauchbar erweist. In Anlehnung an Züllighoven (2005) definieren wir:

Evolutionäre Software-Entwicklung — Vorgehensweise, die eine Evolution der Software unter dem Einfluss ihrer praktischen Erprobung einschließt. Neue und veränderte Anforderungen werden dadurch berücksichtigt, dass die Software in sequenziellen Evolutionsstufen entwickelt wird.

Das Resultat zeigt typischerweise die gewünschte Funktionalität, ist aber strukturell in schlechtem Zustand. Denn die Architektur wurde für den ersten Versuch entworfen, sie ist für das Endprodukt weniger gut geeignet; zudem führen Änderungen zu einer Korrosion der Strukturen. Darum gibt es gute Gründe, das Resultat der evolutionären Entwicklung nur kurze Zeit zu verwenden und bald durch eine Neuimplementierung zu ersetzen. Brooks (1975) hat das ausgedrückt durch die Regel: »Plan to throw one away, you have to do it anyway.« (»Planen Sie ein, dass Sie die erste Fassung bald wegwerfen; es wird Ihnen gar nichts anderes übrig bleiben.«)

In der Praxis ist die evolutionäre Entwicklung, unter welcher Bezeichnung auch immer, sehr populär. Der wichtigste Grund für ihre Beliebtheit ist allerdings von zweifelhafter Güte: »Wir entwickeln evolutionär« kann in vielen Fällen über-

setzt werden in »Wir haben keine Lust oder sind nicht in der Lage, das System zu spezifizieren. Also probieren wir einfach etwas aus.« Natürlich ist das nicht sinnvoll. Es gibt aber Situationen, in denen keine andere Möglichkeit gegeben ist. Wenn die Klienten nicht in der Lage sind, Anforderungen zu nennen, oder wenn völlig unklar ist, welchen Effekt der Rechnereinsatz hat, dann ist die evolutionäre Entwicklung das gebotene Vorgehensmodell. Wenn beispielsweise ein System durch optische und akustische Maßnahmen die Zahl der Wildunfälle vermindern soll, dann kann man weder die Rehe nach ihren Anforderungen fragen noch mit irgendeinem mathematischen Modell die Effekte des Systems zuverlässig voraussagen. Man muss es (auf der Basis von Hypothesen) realisieren, installieren und dann so lange modifizieren, bis es befriedigend funktioniert oder als untauglicher Versuch verworfen wird.

Die evolutionäre Entwicklung ist besonders populär, wenn objektorientiert modelliert und programmiert wird, sie ist darum auch ein Kernpunkt aller agilen Prozesse (siehe Abschnitt 10.6). In diesen wird die Tendenz einer strukturellen Korrosion auf Grund häufiger Änderungen durch das *Refactoring* (siehe Abschnitt 23.3), quasi das Aufräumen der Strukturen, bekämpft.

9.5.3 Iterative Software-Entwicklung

Je größer ein Projekt ist, desto schwieriger wird es, zu Beginn eine realistische Planung vorzugeben. Zudem ist kaum vorherzusehen, welche Anforderungen die Klienten haben, wenn das Projekt abgeschlossen ist – weil die Entwicklung lange dauert und weil das System die typischen Merkmale eines E-Programms hat (Abschnitt 9.3.1).

Die iterative Entwicklung wirkt dem entgegen, weil sie ein großes Projekt in eine Folge kleiner Projekte gliedert. In das Endprodukt gehen also alle Erfahrungen aus dem ersten Durchgang ein und auch neuere Entwicklungen (auf dem Markt, in der Technik).

Wir definieren die iterative Software-Entwicklung folgendermaßen:

Iterative Software-Entwicklung — Software wird in mehreren geplanten und kontrolliert durchgeführten Iterationsschritten entwickelt. Ziel dabei ist, dass in jedem Iterationsschritt – beginnend bei der zweiten Iteration – das vorhandene System auf der Basis der im Einsatz erkannten Mängel korrigiert und verbessert wird. Bei jedem Iterationsschritt werden die charakteristischen Tätigkeiten Analysieren, Entwerfen, Codieren und Testen durchgeführt.

Vom Wortsinn her (lateinisch iteratio: Wiederholung) wäre es logisch, bei zwei Durchgängen vom ersten Durchgang, gefolgt von *einer* Iteration, zu reden. Es ist aber heute allgemein üblich, in diesem Falle einfach von *zwei* Iterationen zu sprechen. Wir passen uns diesem Sprachgebrauch an.

Hinter diesem Vorgehensmodell stehen zwei Erkenntnisse:

- Auch mit einer sehr aufwändigen Analyse ist es in vielen Fällen nicht möglich, im ersten Wurf ein den tatsächlichen Anforderungen genügendes System zu erstellen. Dies soll die Abbildung 9–8 illustrieren.
- Die Existenz und der Einsatz eines Systems verändern die Anforderungen an das System (wie in Abschnitt 9.3.2 beschrieben).

In Cockburn (1993) wird der Grundgedanke der iterativen Entwicklung, der auf McMenamin (1992) zurückgeht, sehr knapp und treffend formuliert:

We get things wrong before we get them right.
We make things badly before we make them well.

In jeder Iteration werden die Tätigkeiten Analysieren, Entwerfen, Codieren und Testen ausgeführt, und das resultierende System wird erprobt. Das Ergebnis des Iterationsschritts n ist die Basis für den darauffolgenden Iterationsschritt n+1. In jedem Schritt wird das vorhandene System auf Basis der gewonnenen Erfahrungen und der gefundenen Mängel systematisch überarbeitet mit dem Ziel, Fehler zu beheben und Verbesserungen einzuarbeiten.

Ob ein iteratives Vorgehen sinnvoll ist oder nicht, hängt von den Randbedingungen des Projekts ab. Soll beispielsweise ein System in einem Anwendungsfeld entwickelt werden, in dem noch keine Erfahrungen vorliegen, dann sollten Iterationsschritte eingeplant werden, da mit hoher Wahrscheinlichkeit die erste Lösung nicht alle Anforderungen erfüllt und überarbeitet werden muss.

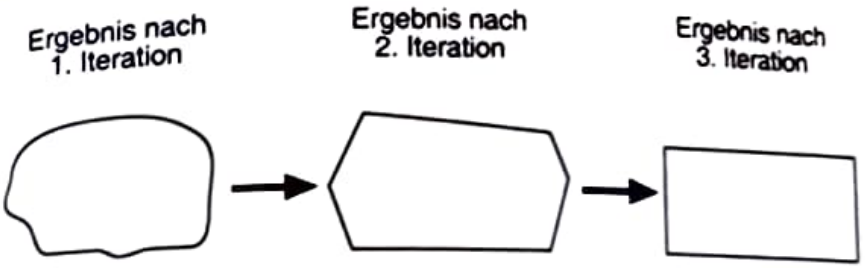

Abb. 9–8 Annäherung durch iterative Entwicklung

Eine schwierige Frage in diesem Zusammenhang ist, wie viele Iterationen notwendig sein werden. Dies kann nicht generell beantwortet werden. Wichtig ist jedoch, dass die explizite Durchführung von Iterationsschritten allen Projektbeteiligten bekannt sein muss, insbesondere auch dem Klienten, und dass dies in der Vertragsgestaltung und der Projektfinanzierung angemessen berücksichtigt wird.

Beispielsweise hat sich gezeigt, dass große studentische Projekte mit einem Gesamtaufwand von einigen Entwicklerjahren erfolgreicher laufen, wenn sie in mindestens zwei Schritte gegliedert sind: In etwa 60 % der Zeit, die für das Projekt insgesamt zur Verfügung steht, wird ein System mit beschränkter Funktionalität realisiert. Der zweite Durchgang wird erst im Detail geplant, wenn sich her-

ausgestellt hat, wie lange der erste Durchgang wirklich gebraucht hat und was der Kunde zum Zwischenresultat sagt.

Die Teilergebnisse, also die Dokumente, werden bei der iterativen Entwicklung immer wieder bearbeitet. Das setzt natürlich eine entsprechende Organisation und Werkzeugunterstützung voraus. Wie an vielen Stellen des Software Engineerings ist auch hier die Konfigurationsverwaltung gefordert (siehe Kap. 21).

9.5.4 Inkrementelle Software-Entwicklung

Bei der inkrementellen Entwicklung wird das zu entwickelnde System nicht in einem Zug konstruiert, sondern in einer Reihe von aufeinander aufbauenden, jeweils funktional erweiterten Ausbaustufen. Jede Ausbaustufe wird in einem eigenen Projekt erstellt, in der Regel auch ausgeliefert und eingesetzt.

Inkrementelle Software-Entwicklung — Das zu entwickelnde System bleibt in seinem Gesamtumfang offen; es wird in Ausbaustufen realisiert. Die erste Stufe ist das Kernsystem. Jede Ausbaustufe erweitert das vorhandene System und wird in einem eigenen Projekt erstellt. Mit der Bereitstellung einer Erweiterung ist in aller Regel auch (wie bei der iterativen Entwicklung) eine Verbesserung der alten Komponenten verbunden.

Bei dieser Vorgehensweise wird also explizit darauf verzichtet, das komplette System in einem einzigen Projekt herzustellen, sondern das System wird schrittweise realisiert, wobei es typischerweise keinen definierten Endzustand gibt. Betriebssysteme wie Linux, Windows oder Mac OS werden inkrementell entwickelt. Nach einigen Jahren ist die Architektur des Systems zerrüttet, überholt. Dann wird, bei Wiederverwendung vieler Komponenten, ein völlig neues System mit neuer Architektur geschaffen.

Die Begriffe »iterativ« und »inkrementell« werden oft verwechselt oder als Synonyme verwendet. Betrachten wir beispielsweise die IEEE-Definition der inkrementellen Software-Entwicklung:

Incremental development — A software development technique in which requirements definition, design, implementation, and testing occur in an overlapping, iterative (rather than sequential) manner, resulting in incremental completion of the overall software product.

IEEE Std 610.12 (1990)

Das entspricht weitgehend unserer Definition der iterativen Entwicklung; diese unterscheidet sich aber von der inkrementellen Entwicklung dadurch, dass mit ihr ein bestimmtes Ziel verfolgt und von Iteration zu Iteration besser erreicht wird. Bei der inkrementellen Entwicklung dagegen wird das Ziel mit jedem Zyklus weiter gesteckt, vor allem durch funktionale Erweiterungen des Systems.

Zu Beginn einer inkrementellen Entwicklung muss sichergestellt sein, dass in der ersten Ausbaustufe, dem *Kernsystem*, ein zentraler, funktional nutzbringend einsetzbarer Ausschnitt des gesamten Systems realisiert ist. Handelt es sich um

den Nachfolger einer älteren Linie, so muss außerdem die Aufwärtskompatibilität garantiert sein, was erhebliche Probleme schafft und meist nur durch spezielle Bausteine erreicht werden kann, die die neuen auf die alten Schnittstellen abbilden, also das alte System emulieren.

Nachdem das Kernsystem realisiert ist, kann das System im Anwendungsbereich eingesetzt werden. Die Inkremente werden anschließend nacheinander entwickelt, in das vorhandene System integriert, ausgeliefert, benutzt und bewertet. Abbildung 9–9 zeigt schematisch die Vorgehensweise bei der inkrementellen Entwicklung.

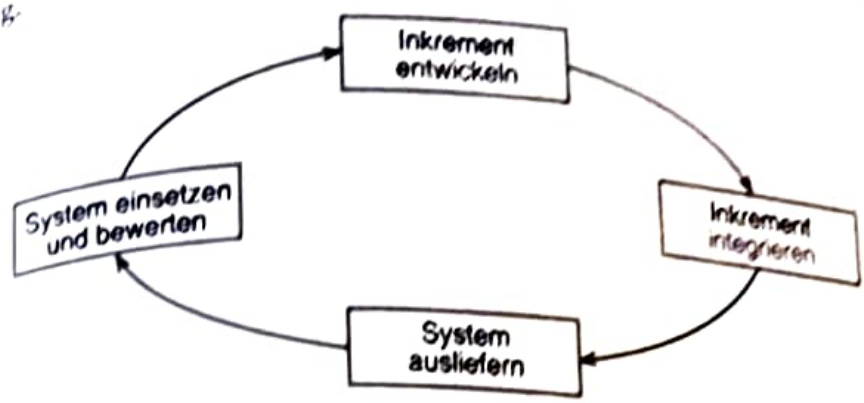

Abb. 9-9 Vorgehensweise beim inkrementellen Entwickeln

Die inkrementelle Vorgehensweise hat folgende Vorteile:

- Bereits sehr früh wird das System (Kernsystem oder Ausbaustufe des Systems) eingesetzt und kann von den Anwendern bewertet werden. Schwachstellen und Probleme fachlicher oder auch technischer Art werden dabei erfasst und können behoben werden.
- Die Entwicklungszeiten der einzelnen Ausbaustufen sind in Relation zur Gesamtentwicklungszeit kurz. Werden Fehler entdeckt, können sie kostengünstig korrigiert werden.

Die inkrementelle Software-Entwicklung ist besonders attraktiv, wenn einerseits die Zeit drängt, ein Produkt abzuliefern, anderseits periphere Details des Produkts noch nicht geklärt sind, womöglich noch nicht geklärt werden können. Wer eine Software zur Verwaltung von Digitalfotos auf den Markt bringen will, wird zunächst ein Kernsystem entwickeln, das die Speicherung und Wiedergabe von Fotos erlaubt. Welche weiteren Komponenten gewünscht werden, zeigt sich erst, wenn dieses Kernsystem fertig ist. Sind inzwischen viele Kameras auf dem Markt, die neben Fotos auch Video-Sequenzen aufzeichnen können, so gibt es Bedarf für eine entsprechende Erweiterung.

9.5.5 Das Treppenmodell

Ein weiteres Vorgehensmodell, bei dem zunächst nur ein Teilprodukt entsteht, ist das *Treppenmodell*. Es ist der inkrementellen Entwicklung sehr ähnlich, unterscheidet sich aber durch die überlappende Bearbeitung der Ausbaustufen.

Software-Entwicklung nach dem Treppenmodell — Das zu entwickelnde System wird in definierten Ausbaustufen realisiert und ausgeliefert. Die erste Stufe ist das Kernsystem. Jede weitere Ausbaustufe erweitert das vorhandene System um Leistungen und Merkmale, die überwiegend bereits zu Beginn des Gesamtprojekts geplant worden sind.

Beim Treppenmodell liegt das Problem nicht in unklaren Anforderungen oder in Planungsproblemen, sondern in einem Konflikt zwischen Termindruck und angestrebtem Funktionsumfang. Das Produkt muss bald auf den Markt, oder der Kunde drängt, das System möglichst rasch einführen zu können. Andererseits kann der gewünschte Umfang unmöglich in kurzer Zeit realisiert werden. Der Ausweg ist ein Kompromiss: Ein zentraler Teil des Systems, der minimale Funktionalität bietet, wird in kurzer Zeit fertiggestellt. Weitere Teile folgen in Abständen von einigen Monaten. Auf diese Weise wird nicht nur der Konflikt zwischen Termin und Umfang entschärft, es besteht auch die Möglichkeit, die Entwickler nach dem Eimerketten-Prinzip arbeiten zu lassen: Nachdem die Planer und Architekten ein Teilprodukt an die Implementierer übergeben haben, wenden sie sich der nächsten Ausbaustufe zu. Das Gleiche geschieht bei der Übergabe von den Implementierern an die Integrierer und Tester. Auf diese Weise können spezialisierte Entwickler praktisch kontinuierlich beschäftigt werden (Abb. 9–10).

Das Treppenmodell wird also immer dann angewendet, wenn die Einführung auf dem Markt oder die Installation beim Kunden drängt, wenn also die »Time to Market« so kurz wie möglich sein soll. Voraussetzung ist, dass die Anforderungen klar sind. Dann muss eine präzise Planung dafür sorgen, dass alle Teilprojekte pünktlich abgeschlossen werden, sodass kein Leerlauf entsteht. Die Anforderungen an die Qualität vor allem der ersten Teilsysteme sind wie bei der inkrementellen Entwicklung hoch.

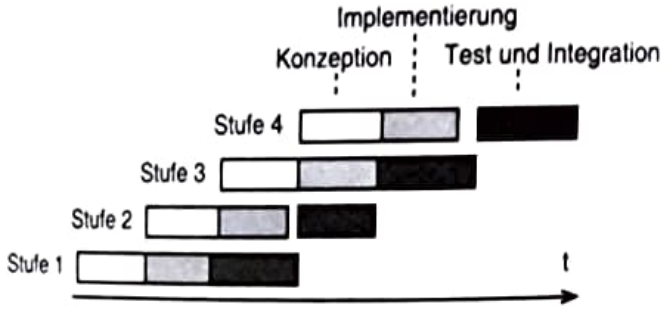

Abb. 9–10 *Das Treppenmodell mit überlappender Bearbeitung*

Das Treppenmodell kann natürlich nur dort eingesetzt werden, wo das zu entwickelnde System auch sinnvoll in ein Kernsystem und darauf basierende Ausbaustufen unterteilt werden kann. Dies ist beispielsweise bei einem System für die Kontoführung im Internet möglich: Das Kernsystem realisiert den technischen Anschluss und die sichere und geschützte Übertragung der Daten. Weiterhin erlaubt es, dass der Benutzer einfache Aktionen an einem Girokonto durchführen kann, beispielsweise kann er den Kontostand abfragen und Überweisungen veranlassen. Die zweite Ausbaustufe erweitert das Kernsystem um die Möglichkeit, Daueraufträge einzurichten, Formulare anzufordern und mit der Bank zu korrespondieren. Die nächste Ausbaustufe erlaubt die Führung eines Wertpapierdepots. Abbildung 9–11 zeigt schematisch die Situation, bei der die Ausbaustufen 1 bis 3 abgeschlossen sind. Stufe 4 steht kurz vor dem Abschluss, Stufe 5 wurde begonnen.

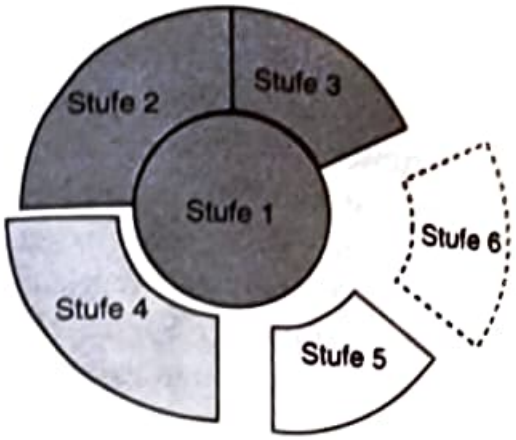

Abb. 9–11 Momentaufnahme der Entwicklung nach dem Treppenmodell

Bei einer Kraftwerksteuerung oder einem Navigationssystem für Flugzeuge ist diese Art der Systementwicklung nicht möglich, da nur das komplette System eingesetzt werden kann.

9.6 Das Spiralmodell

Auf die Diskussionen über das Wasserfallmodell reagierte Boehm einige Jahre später mit einem neuen Vorschlag, der als *Spiral Model* bekannt wurde (Boehm, 1988). Dabei handelt es sich eigentlich nicht um ein konkretes, sondern um ein generisches Vorgehensmodell, das eine Anleitung zur konkreten Ausprägung eines Vorgehensmodells liefert.

In Abbildung 9–12 ist die Darstellung, die den Namen des Modells begründet, mit unserer Übersetzung wiedergegeben. Viele Diskussionen haben aber gezeigt, dass die Grafik das Verständnis nicht erleichtert, sondern eher erschwert. Wir erklären das Modell darum zunächst, ohne auf die Abbildung Bezug zu nehmen.

Fundamental für das Spiralmodell ist der Begriff des *Risikos*. Ein Vorgehensmodell sollte sicherstellen, dass Risiken erkannt und möglichst früh im Projekt bekämpft werden. Daraus leitet Boehm folgendes einfache Rezept für das Vorgehen ab:

Wiederhole den folgenden Zyklus bis zum erfolgreichen Abschluss oder bis zum Scheitern des Projekts:

1. **Suche alle Risiken, von denen das Projekt bedroht ist. Wenn es keine Risiken mehr gibt, ist es erfolgreich abgeschlossen.**
2. **Bewerte die erkannten Risiken, um das größte Risiko zu identifizieren.**
3. **Suche einen Weg, um das größte Risiko zu beseitigen, und gehe diesen Weg. Wenn sich das größte Risiko nicht beseitigen lässt, ist das Projekt gescheitert.**

Dieses risikogetriebene Vorgehen hat zwei wichtige Vorteile:

- Falls das Problem unlösbar ist, stellt sich dies in der Regel rasch heraus. Denn sehr wahrscheinlich scheitert die Lösung am größten Risiko. Nehmen wir beispielsweise an, dass eine Entwicklung durch extreme Speicherknappheit gekennzeichnet ist. Bei einem Vorgehen nach dem Wasserfallmodell werden alle Teile implementiert, getestet und integriert. Dann endlich stellt sich heraus, dass der Speicher nicht ausreicht. Bei Anwendung des Spiralmodells wird dieses Risiko, wenn es als das größte identifiziert worden ist, als erstes bekämpft. Vielleicht kann durch Analysen, Vergleiche mit älteren Systemen oder durch partielle Implementierung der Speicherbedarf ausreichend genau ermittelt werden. Das Projekt wird in diesem Falle abgebrochen, bevor großer Aufwand erbracht wurde.
- Ist das größte Risiko entschärft, so kommt das Projekt bald in ruhiges Fahrwasser, denn die Beteiligten wissen, dass nur noch kleinere Risiken folgen. Sollten in einem der ersten Zyklen noch Fragen offen geblieben sein, so steht zur Klärung die ganze verbliebene Projektdauer zur Verfügung. Im Beispiel oben könnte etwa untersucht werden, ob eine sehr umfangreiche Komponente reduziert werden kann oder ganz entbehrlich ist.

Boehm schränkt die Art der Risiken nicht ein; es können also auch Risiken außerhalb des Entwicklungsprozesses berücksichtigt werden, z. B. die Gefahr, dass ein wichtiger Mitarbeiter die Firma verlässt, bevor das Projekt abgeschlossen ist, oder die Möglichkeit, dass es am Markt für das entwickelte Produkt keine Nachfrage gibt. Das Spektrum der Gegenmaßnahmen ist entsprechend breit. In vielen Fällen ist es notwendig, die offenen Fragen durch Analysen, Simulationen oder Prototypen zu klären. Viele Exegeten des Spiralmodells haben daraus die falsche Annahme abgeleitet, dass die Prototypen das wesentliche Merkmal des Modells seien. Boehm selbst bestätigt aber, dass die Orientierung an den Risiken der entscheidende Punkt ist.

Das Spiralmodell wurde oben als generisches Modell bezeichnet, weil darin alle eigentlichen Vorgehensmodelle enthalten sind. Bei einem Routine-Projekt

sind die Risiken der Reihe nach ein Scheitern der Spezifikation, des Architekturentwurfs, der Implementierung, der Integration und der Inbetriebnahme: Wir haben das Wasserfallmodell vor uns. Sind die Risiken anders gelagert, so entstehen andere Modelle, z. B. die iterative oder die evolutionäre Entwicklung oder auch ganz neue Modelle.

Abb. 9-12 *Spiralmodell nach B. W. Boehm*

Der durch die Schritte 1 bis 3 beschriebene Zyklus ist in der Grafik (Abb. 9-12) durch einen Umlauf in der Schnecke dargestellt. Das Projekt beginnt im Zentrum. Die Quadranten mit den Nummern 1 und 4 tragen nicht viel zur Aussage bei, wesentlich sind die Quadranten 2 und 3.

Die Beschriftung ist irreführend, weil sie suggeriert, dass es im Spiralmodell eine feste, vorgegebene Planung gäbe. Tatsächlich erstreckt sich die Planung aber nur jeweils über einen Zyklus. Dass ein Projekt nach dem Spiralmodell nicht vollständig geplant werden kann, ist gerade eines der Probleme.

In der Praxis wird weit öfter vom Spiralmodell gesprochen, als es wirklich angewendet wird; viele Menschen verwechseln es mit dem iterativen Vorgehen. Eine strenge Anwendung wäre auch nur in wenigen Fällen möglich, denn sie setzt große Flexibilität sowohl beim Software-Hersteller (Personalbedarf) als auch beim Klienten (Liefertermin, Preis) voraus. Aber die Grundidee, sich an den Risiken zu orientieren, sollte jeder im Kopf haben, der ein Projekt plant und durchführt.