Kurseinheit 9

Erstellung und Prüfung sicherheitsgerichteter Software

Qualitätssicherung sollte bereits ganz am Anfang der Software-Entwicklung beginnen und darf im Grunde erst dann beendet werden, wenn ein Software-System nicht mehr benutzt wird. Eine effektive und effiziente Qualitätssicherung muß sorgfältig geplant und gesteuert werden. Zur Projektbeschreibung gehört darum auch die Festlegung, was, wann, wie und von wem geprüft werden soll. Es ist selbstverständlich, daß automatisierbare Analysen auch automatisiert werden sollten. Was automatisiert werden kann, hängt aber auch von den angewandten Konstruktionsmethoden ab. Somit besteht eine gegenseitige Abhängigkeit von Konstruktion und Qualitätssicherung. Qualität läßt sich nicht im nachhinein in ein Produkt "hineinprüfen". Deshalb kann analytische Qualitätssicherung nur vorhandene Qualität feststellen. In Anbetracht dieser Anforderungen, Wechselwirkungen und Sachstände betrachten wir in diesem Kapitel sowohl Richtlinien zur Konstruktion sicherheitsgerichter Software als auch Methoden zu deren Prüfung. Letztere umfassen Verfahren zur Prüfung von Dokumentationen als auch Methoden zur manuellen und automatisierten Prüfung eigentlicher Programme.

9.1 Grundlegende Methoden der Software-Qualitätssicherung

Unter dem Begriff *Inspektion* fassen wir hier alle "manuellen" Prüfverfahren wie Entwurfs- und Codeinspektion [60], "(Structured) Walkthrough" [97] und Arbeitsbesprechnungen [189] zusammen. Das Prinzip dieser Verfahren ist n-Augen-Kontrolle, die davon ausgeht, daß "vier oder mehr Augen mehr als zwei sehen". Die Verfahren beruhen auf Gruppenbesprechungen zwischen Entwicklern und Inspektoren, in denen die Untersuchungsobjekte analysiert werden. Die Vorgehensweise besteht im Prinzip aus drei Schritten:

1. In einem Vorbereitungsgespräch erklären die Entwickler der Inspektorengruppe das Objekt im Großen. Die notwendigen Unterlagen werden verteilt. Bis zur nächsten Sitzung muß dann jeder Inspektor das Objekt untersuchen.

2. In den eigentlichen Inspektionssitzungen wird danach das Objekt im einzelnen durchgesprochen. Unklarheiten werden diskutiert und gegebenenfalls als Fehler identifiziert.

3. In Nachbereitungen werden alle Sitzungsergebnisse dokumentiert und die erforderlichen Änderungen veranlaßt und durchgeführt.

Inspektionsverfahren nutzen die Möglichkeiten menschlichen Denkens und Analysierens. Auch komplexe Zusammenhänge können erfaßt werden. Inspektionen sind prinzipiell für alle Software-Beschreibungen und zur Erhebung und Bewertung jeder Eigenschaft geeignet. Die für jede betrachtete Eigenschaft zu erhebenden Kenngrößen werden in Prüflisten aufgeführt.

Ein Nachteil von Inspektionsverfahren ist, daß ihr Erfolg wesentlich von den beteiligten Personen abhängt. So werden insbesondere an den Moderator große Anforderungen gestellt. Zusammensetzung und Arbeitsweise der Gruppe sind sehr wichtig für den Erfolg einer Inspektion. Die Gespräche dürfen nicht in hektischer oder angespannter Atmosphäre stattfinden und die Entwickler dürfen sich nicht in eine Verteidigungshaltung gedrängt fühlen. Sie dürfen nicht befürchten müssen, daß die Inspektion und ihre Ergebnisse zur Beurteilung ihrer Leistungen herangezogen werden. Die Erfahrung bei der Anwendung von Inspektionsverfahren zeigt, daß mit ihnen weit mehr erreicht werden kann, als vielfach angenommen wird.

Formale Verifikationstechniken werden immer häufiger als ein wichtiger Ansatz zur Erzielung verläßlicher Software akzeptiert, besonders in sicherheitskritischen Anwendungsgebieten. Sie benutzen mathematische Verfahren, um die Richtigkeit von Software streng zu verifizieren. Darin liegt aber auch ihr Hauptnachteil begründet: die Anwendung formaler Verifikationstechniken verlangt spezielle Fachkenntnisse und die üblicherweise ziemlich langen Programmkorrektheitsbeweise können Fehler enthalten, die lange Zeit unentdeckt bleiben können.

Die *symbolische Ausführung* wurde zur Analyse (komplexe) mathematische Algorithmen implementierender Programme entwickelt, für die sie sehr gute Möglichkeiten der Fehlerfindung bietet. Bei der symbolischen Ausführung wird ein Untersuchungsobjekt nicht mit konkreten Werten ausgeführt, sondern wie in der Schulalgebra mit symbolischen Bezeichnern schrittweise interpretiert. Dabei wird jeder zu berechnenden Größe ein Ausdruck zugewiesen, der aus den symbolischen Bezeichnern gebildet wird. Jeder Bedingung entspricht ein Prädikat, das ebenfalls aus den symbolischen Werten gebildet wird. Durch die symbolischen Werte können ganze Klassen möglicher Ausführungen durch eine einzige Interpretation ersetzt werden. Für die Analyse mathematischer und technisch-wissenschaftlicher, nicht zu umfangreicher Algorithmen ist symbolische Ausführung gut geeignet. In diesen Fällen erlaubt sie häufig, die Untersuchungsobjekte streng zu verifizieren.

Tests werden vor allem zur Prüfung von Programmen eingesetzt. Das Prinzip von Tests beruht auf Stichproben: ein Untersuchungsobjekt wird mit ausgewählten, möglichst repräsentativen Werten auf einem Rechner ausprobiert. Damit kann festgestellt werden, ob sich das Objekt in diesen speziellen Situationen wie erwartet verhält. Voraussetzung für Tests sind ausführbare Beschreibungen. Als alleiniges Mittel zur Identifizierung von Fehlern oder Mängeln sind Programmtests jedoch zu aufwendig. Sie sind im Entwicklungsprozeß erst viel zu spät einsetzbar und für einige Analyseaufgaben ungeeignet. Trotzdem sind Programmtests eine wertvolle

Analysetechnik, da sich mit ihr in recht effektiver Weise Fehler entdecken lassen und Fehlerfreiheit bzw. geforderte Qualität für ausgewählte Situationen gezeigt werden kann. Weil Tests wegen der großen oder gar unendlichen Anzahl abzudeckender Fälle meistens nicht erschöpfend sind, können sie jedoch allgemeine Korrektheit nicht nachweisen. Trotz dieses gravierenden Nachteils sind Programmtests die in der Praxis mit Abstand am weitesten verbreitete Prüftechnik. Es wird meist mit sehr viel Aufwand, aber oft unsystematisch getestet. Die Wirksamkeit des Verfahrens wird häufig überschätzt, denn Tests liefern zwar auf rationelle Weise sichere Aussagen über Objekte mit kleinen Wertebereichen — bei großen Wertebereichen vermitteln sie allerdings keine Sicherheit.

9.2 Qualitätssicherung der Dokumentation

Diesem Abschnitt liegt der Artikel [84] zu Grunde.

Die Darstellung eines Software-Systems umfaßt neben Konzepten, Entwürfen und eigentlichen Programmtexten als formale Objekte auch Dokumentationen, die zur informellen Beschreibung der vorgenannten in natürlicher Sprache dienen. Im Idealfall entstehen Dokumentationen parallel zur Entwicklung ihrer Objekte. Sie sind wie diese selbst der Qualitätssicherung unterworfen. Eine vollständige Dokumentation eines Produktes enthält jeweils für die verschiedenen Rollenträger wie Entwickler, Benutzer, Wartungstechniker usw. geeignete, spezifische Darstellungen. Sie kann beispielsweise aus Anforderungsdokumentation, Pflichtenheft, Entwurfsdokumentation, Programmlisten sowie Wartungs- und Anwendungsdokumentation (Benutzer- und Bedienerhandbuch) bestehen. Prüfergebnisse und Testprotokolle sollten ebenso in einer Produktdokumentation enthalten sein.

Zur Beschreibung der Qualität von Dokumenten sind die Eigenschaften Änderbarkeit, Aktualität, Eindeutigkeit, Kennzeichnung, Verständlichkeit, Vollständigkeit und Widerspruchsfreiheit als informelle Beschreibungsmittel relevant. Sie beziehen sich teilweise auf den Inhalt und zum Teil auf die Darstellungsform.

Die Prüfung von Dokumentationen entzieht sich weitgehend der Analyse mit formalen Methoden, da es noch keine zur Analyse natürlichsprachiger Texte anwendbare Techniken gibt. Hinzu kommt, daß in Dokumentationen oft eine Mischung formaler, tabellarischer und informeller Beschreibungen vorzufinden ist. Deshalb wird man im allgemeinen kaum eine andere Methode als Inspektion zur Prüfung von Dokumentationen einsetzen können.

Produktdokumentationen haben die Aufgabe, zu dokumentierende Objekte in natürlicher Sprache derart zu beschreiben, daß die angesprochenen Rollenträger die für sie relevanten Informationen erhalten. Daraufhin sind entsprechende Prüfungen auszurichten. Die Dokumenteninspektion beinhaltet mithin zwei Prüfaufgaben, und zwar (1) ob Dokumente die dargestellten Objekte (Konzepte, Entwürfe, Programme) korrekt beschreiben und (2) ob Dokumentationen ihren Anforderungen genügen.

Daraus resultiert ein Problem mit *n:m* Beziehungen zwischen n Rollenträgern, für die Software geschrieben wird bzw. die mit ihr umgehen müssen, und m Darstellungen dieser Software. Eine Dokumentation muß der Aufgabe gerecht werden, jedes Objekt auf die n Rollenträger abzubilden. Im Zuge der Inspektion von Software-

Dokumentationen müssen die Inspekteure also die Korrektheit dieser Abbildung von Informationen und Aktivitäten, Funktionen und Daten sowie Programmen auf die Aufgaben der Rollenträger prüfen.

Die Effektivität einer Inspektion hängt wesentlich vom Interesse der Beteiligten ab. Im Idealfall werden Dokumentationen von darauf spezialisierten technischen Dokumentatoren verfaßt. Entwickler sind daran interessiert, daß ihre Produkte richtig dargestellt werden. Benutzer betrachten Produkte eher von den Aufgabenstellungen her.

Aus alledem folgt, daß sich die Inspektion von Dokumenten wesentlich von der Inspektion formaler Beschreibungen von Konzepten, Entwürfen oder Programmen unterscheidet. Während dort die Richtigkeit einer entwickelten Beschreibung, möglicherweise unter Berücksichtigung anderer formaler Beschreibungen, im Vordergrund steht, geht es hier um die korrekte Darstellung einer formalen Darstellung in einer aufgabenorientierten "Fachsprache". Die Organisation einer Inspektion hat diesen Aufgaben gerecht zu werden, indem sie die Analyse der untersuchten Dokumentationen aus der formalen Sicht der Entwickler und aus der informellen Sicht der rollenorientierten Entwickler erzwingt.

Die Bewertung von Dokumenten soll nun exemplarisch anhand der beiden Eigenschaften Korrektheit und Komplexität diskutiert werden. Für Produktdokumentationen wird sicherlich gefordert, daß sie korrekt sind. Dokumente sind als informelle Beschreibungen von Konzepten, Entwürfen und Programmen unbrauchbar, wenn sie das jeweilige Objekt nicht wiedergeben. Die Korrektheit eines Dokumentes kann in Merkmalen wie Eindeutigkeit, Vollständigkeit oder Widerspruchsfreiheit zum Ausdruck kommen. Die Komplexität von Dokumenten sollte stets niedrig sein. Man kann zwischen der Komplexität der Struktur und der Komplexität des Inhaltes eines Dokumentes, also der Problemstellung, unterscheiden. In der Bewertung drückt sich niedrige Komplexität dadurch aus, daß ein Dokument leicht verständlich und nachvollziehbar sowie gut lesbar und strukturiert ist.

Produktdokumente können mit Hilfe sogenannter Texteindrucksbeurteilungen bewertet werden. Hierzu gehört das Hamburger Modell [137, 182, 188], nach dem Dokumente nach den vier faktorenanalytisch ermittelten Dimensionen Einfachheit, Gliederung-Ordnung, Kürze-Prägnanz und zusätzliche Stimulanz eingeschätzt werden. Jede dieser vier Dimensionen setzt sich aus mehreren Einzelfaktoren wie "einfache Darstellung", "anschaulich" oder "konkret" zusammen. Eine Bewertung wird anhand einer fünfstufigen Skala durch speziell geschulte Beobachter vorgenommen. Die Schätzurteile mehrerer Beurteiler werden nach Dimensionen zu Mittelwerten zusammengefaßt.

9.3 Qualitätssicherung von Programmen

Diesem Abschnitt liegt der Artikel [84] zu Grunde.

Die Qualitätssicherung von Programmen besteht aus der des Entwurfs, und hier insbesondere der Modularisierung, und jener der Implementierung. Der Entwurf hat entscheidenden Einfluß darauf, welche Schwierigkeiten bei der Fehlerfindung, ihrer Behebung und anderen Änderungen in späteren Entwicklungsschritten auftre-

ten. Modularisierung, d.h. Zerlegung eines Systems in Teile, ist eine anerkannte Methode zur Beherrschung von Komplexität. Sie wirkt sich auf Programmierung und Qualitätssicherung von Programmen in folgender Weise positiv aus:

- modularisierte Programme sind übersichtlicher und leichter verständlich,
- sie lassen sich leichter ändern, anpassen und erweitern,
- bei ihrer Programmierung werden erfahrungsgemäß weniger Fehler gemacht,
- sie sind leichter zu testen und
- Fehler sind einfacher zu finden und zu beheben.

Bei der Implementierung werden die Datenstrukturen und Algorithmen jedes Moduls festgelegt. Die Strukturierte Programmierung hat sich als grundlegendes Konzept der Implementierung durchgesetzt. Sie geht zum einen auf Boehm und Jacopini, die in [25] zeigten, daß die drei Kontrollstrukturen Anweisungsfolge, Auswahl und Wiederholung zum Schreiben sequentieller Programme ausreichen, und zum anderen auf Dijkstra zurück, der sich in [38] gegen die Verwendung von Sprunganweisungen aussprach. Das Konzept hat große Bedeutung hinsichtlich der Prüfbarkeit von Programmen, weil diese durch schlecht strukturierten Kontrollfluß wesentlich erschwert wird.

Aufgabe der Qualitätssicherung von Programmen ist festzustellen, ob letztere in sich richtig, d.h. widerspruchsfrei und vollständig, sind — interne Konsistenz — und ob sie das tun, was in ihren vorherigen Beschreibungen festgelegt wurde — externe Konsistenz. Die Prüfung von Entwürfen hat zu klären, ob die Schnittstellen zwischen Modulen vollständig und widerspruchsfrei sind sowie ob gewählte Zerlegungen und Modulspezifikationen plausibel und den Implementierungskonzepten angemessen sind. Außerdem muß gezeigt werden, daß das tatsächliche Zusammenwirken von Systemteilen diesen Konzepten entspricht. Zur Prüfung von Implementierungen zählen Aufbau-, Kontrollfluß- und Datenflußanalyse. Die Aufbauanalyse stellt beispielsweise fest, ob die verwendeten Objekte deklariert sind und die deklarierten Objekte verwendet werden. Zusätzlich kann sie prüfen, ob die Objekte in der deklarierten Form verwendet werden. Mit der Kontrollflußanalyse soll z.B. festgestellt werden, ob alle Anweisungen erreichbar sind und ob es Endlosschleifen geben kann. Aufgabe der Datenflußanalyse kann es schließlich sein zu prüfen, ob Variablen vor ihrer Verwendung Werte zugewiesen werden und ob sie nach Wertzuweisungen verwendet werden. Neben diesen formalen Prüfungen muß festgestellt werden, ob die gewählten Algorithmen und Datenstrukturen angemessen und die Berechnungen und Entscheidungen richtig sind. Zur Aufdeckung in Programmen enthaltener Fehler lassen sich die im folgenden betrachteten Methoden anwenden.

9.3.1 Inspektion von Programmen

Inspektionstechniken lassen sich für alle Prüfungen eines Programmes verwenden. Welche Fragestellungen für Prüfungen von Entwurf und Implementierung hinsichtlich formaler Kriterien mittels Inspektion relevant sind, hängt davon ab, welche Fragen bereits mit Hilfe des verwendeten Übersetzers und möglicherweise mit Hilfe

zusätzlicher Werkzeuge zur statischen Analyse beantwortet wurden. Es gibt Programmiersprachen, die den Entwurf dadurch unterstützen, daß sie Sprachmittel zur Beschreibung von Modulen und ihrer Beziehungen untereinander bereitstellen. Somit kann ein entsprechender Übersetzer die Modulschnittstellen prüfen. Die (statische) Analyse von Kontroll- und Datenfluß ist ebenfalls Aufgabe von Übersetzern, die allerdings die wenigsten wahrnehmen, weshalb hierfür oft zusätzliche Werkzeuge eingesetzt werden.

Die Verwendung von Inspektionstechniken für Prüfungen hinsichtlich formaler Kriterien kann nur ein Notbehelf sein, wenn geeignete Werkzeuge zur statischen Analyse fehlen. Die eigentliche Aufgabe der Inspektion von Programmen ist die inhaltliche Prüfung ihrer Entwürfe und Implementierungen. Ob eine Zerlegung plausibel und dem Konzept angemessen ist, kann ebensowenig automatisch beantwortet werden wie die Frage, ob spezifizierte Algorithmen und Datenstrukturen angemessen implementiert sind. Bekannte Methoden zur Programminspektion sind Fagans "Entwurfs- und Codeinspektion" [60] und "Strukturiertes Nachvollziehen" [162]. Beide Methoden werden in den ursprünglichen Formen zur Prüfung von Implementierungen verwendet.

- Der Begriff Entwurf in "Entwurfs- und Codeinspektion" steht für Programmentwurf im Sinne von Pseudocode. Die Entwurfs- und Codeinspektion konzentriert sich im wesentlichen auf Prüfungen nach formalen Kriterien und kompensiert auf diese Art Schwächen verwendeter Sprachen und Werkzeuge (einschließlich Übersetzer). Es ist aber prinzipiell möglich, diese Methode an gewünschte Prüfungen anzupassen, indem man die Prüflisten geeignet ändert.
- Beim strukturierten Nachvollziehen werden Programme "am Schreibtisch im Kopf" ausgeführt. Dazu werden Testfälle ausgewählt, mit denen die Programme dann im Rahmen von Prüfsitzungen durch Verfolgung der Kontroll- und Datenflüsse anhand der Quelltexte durchgespielt werden.

Da Inspektionsmethoden wesentlich auf den Fähigkeiten des menschlichen Gehirns beruhen, lassen sie sich nicht automatisieren. Man kann sie aber sehr wohl durch Werkzeuge unterstützen. So können viele Fragen, die bei Inspektionen auftreten, wie z.B. die, wo bestimmte Funktionen verwendet werden, durch Werkzeuge zur Quelltextanalyse beantwortet werden. Auch Ergebnisse aus anderen Prüfungen können im Rahmen von Programminspektionen herangezogen werden.

9.3.2 Verifikation von Programmen

Die Aufgabe der Programmverifikation ist zu prüfen, ob die Implementierung eines Programmes mit seiner Spezifikation verträglich ist. Dabei werden die Algorithmen, d.h. Kontroll- und Datenfluß, geprüft. Zur Verifikation muß ein Programm mit zusätzlichen Annotationen in Form von Zusicherungen versehen werden, die die Programmzustände beschreiben. Vorbedingungen geben den Anfangszustand und Nachbedingungen den Endzustand an. Zu zeigen ist, daß sich für jeden Pfad durch das Programm die Nachbedingungen aus den Vorbedingungen ableiten lassen. Da es wegen Schleifen eine unüberschaubare Vielzahl von Pfaden durch ein Programm geben kann, sind für jede Schleife von den Schleifendurchläufen unabhängige Zustände

zu finden. Diese Zustände werden durch induktive Zusicherungen (Schleifeninvarianten) beschrieben. Die Methode geht auf Floyds Idee der Schleifeninvarianten [65] und Hoares axiomatische Beschreibung von Programmkonstrukten [93] zurück und besteht aus drei Schritten:

1. Erzeugung von Verifikationsbedingungen oder Theoremen. Man kann dazu von den Vorbedingungen ausgehen und den Programmablauf vorwärts verfolgen oder umgekehrt von den Nachbedingungen rückwärts. Für jede Anweisung auf einem Pfad durch ein Programm muß festgestellt werden, wie die Bedingungen durch die Anweisung transformiert werden. Dazu benötigt man für jeden Anweisungstyp eine Transformationsregel, die die Anweisungssemantik beschreibt. Wenn ein ganzer Pfad abgearbeitet ist, hat man somit eine Menge abgeleiteter Bedingungen, die Verifikationsbedingungen. Letztere sind mathematische Ausdrücke, meist in Prädikatenlogik erster Stufe. Daß aus jeder abgeleiteten Nach- oder Vorbedingung die spezifizierte Nach- oder Vorbedingung folgt, sind nun die zu beweisenden Theoreme.

2. Beweis der Theoreme. Wenn ein Theorem bewiesen werden kann, zeigt dies, daß der betrachtete Programmpfad korrekt implementiert ist, d.h. daß er der Programmspezifikation entspricht. Ist es nicht möglich, das Theorem zu beweisen, so kann dies verschiedene Ursachen haben. Es kann daran liegen, daß das Programm falsch implementiert ist. Es kann aber auch sein, daß die Spezifikation falsch ist, oder es kann bedeuten, daß dem Verifizierer der Beweis nur nicht gelungen ist.

3. Wenn alle Theoreme bewiesen sind, ist gezeigt, daß das Programm, sofern es von einem Anfangs- zu einem Endzustand kommt, richtig arbeitet. Dies wird als "partielle Korrektheit" bezeichnet. Zur vollständigen Korrektheit muß nun noch gezeigt werden, daß das Programm einen Endzustand erreicht, d.h. anhält.

Solch ein formaler Beweis ist, selbst bei kleinen und kleinsten Programmen, sehr mühsam und fehleranfällig, wenn er manuell geführt wird. Deshalb werden unterstützende Werkzeuge benötigt. Ein Verifikationssystem besteht aus mehreren Werkzeugen. Zuerst untersucht ein Parser Spezifikation und Programm auf syntaktische Fehler. Die Ausgabe des Parsers ist Eingabe für einen Verifikationsbedingungsgenerator, der eine Menge logischer Ausdrücke erzeugt. Im nächsten Schritt müssen alle Verifikationsbedingungen bewiesen werden. Dazu braucht man einen Theorembeweiser. Gelingt es diesem nicht, ein Theorem zu beweisen, so muß der Bediener eingreifen.

9.3.3 Symbolische Ausführung von Programmen

Ausgangspunkt der symbolischen Programmausführung ist Quellcode. Die Methode vollzieht Kontrollflüsse nach und prüft mithin Implementierungen. Ein Programm wird interpretiert, indem ausgehend von symbolischen Bezeichnern für die Eingabevariablen jede Anweisung auf einem Pfad durch das Programm ausgewertet wird, d.h. indem, wie in Tabelle 9.1 an einem ganz einfachen Beispiel gezeigt wird, mathematische Ausdrücke gebildet werden.

Anweisungsfolge	Werte von x und y	
	x	y
Anfangswert	a	b
$y{:=}x{+}y$	a	$a{+}b$
$x{:=}y{-}x$	b	$a{+}b$
$y{:=}y{-}x$	b	a

Tabelle 9.1: Einfaches Beispiel zur symbolischen Programmausführung

Mit der symbolischen Programmausführung läßt sich aus einem komplexen mathematischen Programm ableiten, welche Formel es tatsächlich berechnet. Diese abgeleitete Formel wird dann mit der spezifizierten verglichen. Die Methode ist auch anwendbar, um Pfadprädikate auszurechnen, die für die Bestimmung von Testfällen benutzt werden sollen. Symbolische Programmausführung kann für kleinere Programmstücke von Hand durchgeführt werden. Im allgemeinen sind jedoch Werkzeuge erforderlich. Diese arbeiten interaktiv mit dem Bediener zusammen, um so das Problem nicht vom Werkzeug entscheidbarer Verzweigungen zu lösen.

9.3.4 Test von Programmen

Das Testen eines Programmes erfordert die Übersetzung des Quellcodes in ein Maschinenprogramm, das auf einem Rechner mit ausgesuchten Eingabedaten ausgeführt werden kann. Ziel ist festzustellen, ob sich das Programm erwartungsgemäß verhält und die erwarteten Ergebnisse produziert. Fehlerlokalisierung und -behebung gehören nach dieser Definition nicht zum Test.

Beim Testen von Programmen kann man Testphasen unterscheiden, die den Entwicklungsschritten entsprechen. Jeder Entwicklungsschritt liefert ein Ergebnis, das die Vorgabe zu einer Testphase darstellt. Unterschieden werden sollte zumindest zwischen:

Modultest Für jedes Modul wird analysiert, ob es sich bei Ausführung seiner Beschreibung entsprechend verhält, d.h. seine Implementierung wird gegen seine Spezifikation geprüft.

Integrationstest Module werden schrittweise zu immer größeren Systemteilen zusammengefügt. Dabei wird geprüft, ob die Module so zusammenwirken, wie es die Entwürfe vorsehen.

Systemtest Es wird geprüft, ob ein System seiner Spezifikation entsprechend arbeitet.

Jede dieser Testphasen besteht ihrerseits aus den drei Schritten (1) Vorbereitung der Testläufe, (2) deren Ausführung und (3) ihrer Auswertung.

Hauptprobleme bei der *Vorbereitung* sind die Bestimmung von Testfällen und die Erzeugung von Testdaten. Dieser Schritt ist deshalb wichtig, weil es im allgemeinen nicht möglich ist, ein Programm vollständig zu testen. Bei der Auswahl von Testfällen lassen sich drei Ansätze unterscheiden: Generierung von Zufallswerten und Testfallbestimmung aus der Spezifikation oder aus dem Quelltext. Zufallswerte

können zwar automatisch erzeugt werden, jedoch ist diese Methode im allgemeinen keine befriedigende Lösung, da auch bei sehr vielen Testläufen häufig nur wenige Programmteile ausgeführt werden [113].Testfallableitung aus Spezifikationen ist bisher nur manuell möglich, für die Ableitung aus Quellcode lassen sich Werkzeuge zur symbolischen Programmausführung nutzen. Die Generierung von Testdaten läßt sich durch Testspezifikationen unterstützen. Dieser Ansatz erscheint vielversprechend, denn er reduziert nicht nur die Schreibarbeit bei der Testdatengenerierung, sondern unterstützt auch die anderen beiden Testschritte. Allerdings muß man die ausgewählten Testfälle spezifizieren.

Das Problem bei der *Ausführung* von Testläufen ist, die Umgebungen der Testobjekte zu simulieren. Bei Modul- und Integrationstests sind diese Umgebungen nicht oder nur unvollständig vorhanden. Selbst wenn einzelne Teile bereits vorhanden sind, ist es oft besser, diese Teile auch zu simulieren, um aus der Umgebung eines Testobjektes stammende Probleme vom Test fernzuhalten. Die Umgebung eines Testobjektes, ein Testbett, besteht aus einem Treiber und Ersatzfunktionen. Der Treiber ruft das Testobjekt auf, versorgt es mit Parametern und übernimmt die berechneten Ergebnisse. Die Ersatzfunktionen simulieren nicht vorhandene Funktionen sowie Dateizugriffe. Der manuelle Aufbau eines Testbettes für jedes Testobjekt erfordert sehr viel Aufwand. Andererseits ist aber auch die Nutzung mancher Testwerkzeuge recht aufwendig. Eine mögliche Lösung dieses Problems besteht in der Verwendung einer Testspezifikation, die die Schnittstelle eines Testobjektes zu seiner Umgebung beschreibt.

Bei der *Auswertung* von Testläufen müssen die Testergebnisse sowie die Wirksamkeit vorhergehender Testläufe untersucht werden. Durch die Auswertung von Testläufen soll festgestellt werden, ob Fehler aufgedeckt wurden. Ein Fehler kann ein falsches (Zwischen-) Ergebnis oder ein falscher Programmdurchlauf sein. Neben der Fehleranalyse muß abgeschätzt werden, welche Programmsituationen getestet wurden. Testläufe werden manuell ausgewertet, was jedoch erheblich durch Werkzeuge unterstützt werden kann. Insbesondere sind solche Werkzeuge hilfreich, die aus den während Testläufen gesammelten Informationen spezielle Berichte erzeugen, welche z.B. über beim Test durchlaufene Programmteile, über erreichte Abdeckung von Kontrollfluß und Datenverwendung oder über Abweichungen zwischen spezifizierten und tatsächlich berechneten Werten Auskunft geben.

9.4 Industrielle Prüfung der Software von Prozeßautomatisierungssystemen

Dieser Abschnitt basiert auf der Arbeit [129].

Im Prozeß der Software-Entwicklung sind grundsätzlich zwei Aktivitäten zu unterscheiden: Entwurf und Entwicklung einerseits und Prüfung andererseits. Im Rahmen der Entwicklung werden die geforderten Funktionen und Eigenschaften eines Systems zunächst entworfen (Gestaltung), dann beschrieben (Dokumentation) und schließlich realisiert (Programme). Durch die Prüfung werden die Ergebnisse der Entwicklung überprüft und validiert. Einer der Gründe für die nach wie vor mangelnde Zuverlässigkeit von Software liegt in der nur wenig ausgebauten Rechnerun-

terstützung zur Qualitätssicherung und insbesondere zur Prüfung von Software. In diesem Abschnitt werden Fragen der automatischen Prüfung behandelt und an Beispielen der industriellen Prüfung von Funktionen eines Prozeßleitsystems erläutert. Weitere Entwicklungen auf dem Gebiet automatischer Prüfungen werden aufgezeigt.

9.4.1 Grundlagen der Prüfung von Software

Prüfarten

Software besteht aus Dokumentation und Programmen (Software-Funktionen). Daher sind zwei Prüfarten zu unterscheiden:

1. Prüfung der Dokumentation einschließlich der Programme (Quellcode) mit den darin enthaltenen Kommentaren,

2. Prüfung der Software-Funktionen im Entwicklungs- und auf dem Zielsystem.

In der Literatur werden zu beiden Prüfarten mehrere verschiedene Methoden beschrieben [77, 161, 163, 181, 199, 201, 217]. Eine Standardisierung bezüglich Prüfplan, Prüfdokumentation und bestimmter Prüfverfahren wird in immer stärkerem Maße vorangetrieben [2, 5, 4].

Prüfungen im Phasenmodell

Die in der Software-Entwicklung anwendbaren Prüfungen lassen sich wie in Bild 9.1 gezeigt in einem sogenannten Phasenmodell darstellen. Der linke abfallende Zweig zeigt das Top-down-Verfahren von der obersten Ebene der Pflichtenhefte (Anforderungsspezifikation) über den Entwurf (Entwurfsspezifikation) bis zu Programmkomponenten (Modulspezifikation) und dem eigentlichen Modulcode mit Kommentaren. Der rechte aufsteigende Zweig entspricht dem Bottom-up-Verfahren, das eine schrittweise Integration von Modulen über Teilsysteme bis zu einem Gesamtsystem vorsieht.

Eine Prüfung ist in jeder Phase durchzuführen. Im Bild 9.1 sind folgende Prüfarten dargestellt: Pflichtenheftrevision, Entwurfsrevision, Code-Inspektion, Modul-, Integrations- und Systemtest. Die Prüfungen auf den unteren Ebenen werden auch als Verifikation bezeichnet. Die Vorgabe hierfür ist immer die entsprechende Spezifikation. Die Prüfung auf der obersten Ebene wird auch Validierung genannt. Als Vorgabe dienen hier die in den Pflichtenheften beschriebenen Anforderungen. Diese Prüfart wird hier auch mit Software-Typprüfung bezeichnet.

Aktivitäten einer Prüfung

Es gibt drei Bereiche von Prüfaktivitäten: Vorbereitung, Durchführung und Auswertung einer Prüfung. Im folgenden wird nur die dynamische Prüfung der Software-Funktionen betrachtet.

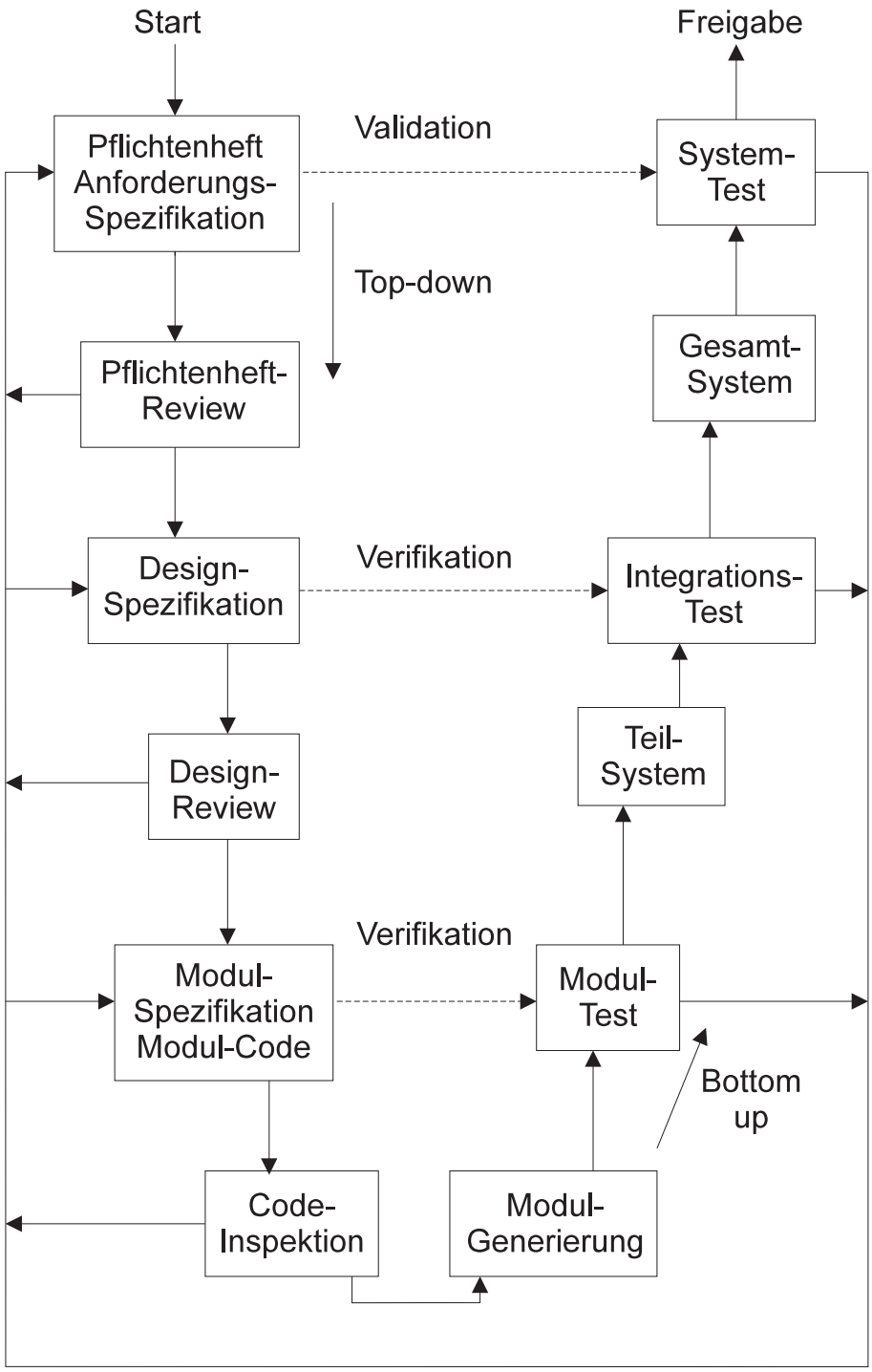

Bild 9.1: Phasenmodell mit Prüfungen