vdf Lehrbuch

Informatik

Software-Prüfung

Eine Anleitung zum Test und zur Inspektion

6. Auflage

Karol Frühauf
Jochen Ludewig
Helmut Sandmayr

vdf

vdf Hochschulverlag AG
an der ETH Zürich

Karol Frühauf
Jochen Ludewig
Helmut Sandmayr

Software-Prüfung

Eine Anleitung zum Test und zur Inspektion

6. Auflage

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

ISBN 978-3-7281-3059-4 Printausgabe
ISBN 978-3-7281-3551-3 eBook
Doi-Nr. 10.3218/3551-3

1. Auflage 1988
2., durchgesehene Auflage 1991
3., durchgesehene Auflage 1997
4., durchgesehene Auflage 2000
5., überarbeitete Auflage 2004
6., überarbeitete und aktualisierte Auflage 2007

verlag@vdf.ethz.ch
www.vdf.ethz.ch

Vorwort zur ersten Auflage

Dieses Buch soll die wichtigsten Grundsätze der Software-Prüfung vermitteln. Wir betrachten dabei vor allem die *Aktivitäten* der Beteiligten, weniger die technischen Hilfsmittel. Es richtet sich an alle, die als Entwickler, Kunden oder Vorgesetzte mit der Prüfung und Qualitätssicherung von Software befasst sind.

Als Autoren streben wir keine wissenschaftliche Vollständigkeit an; unser Ziel ist es vielmehr, *wenige, aber praktikable* Möglichkeiten zu zeigen, wie man wirklich vorgehen kann – und nach dem Stand der Technik verfahren sollte. Wir wenden uns also *nicht* an erfahrene Testspezialisten, deren Vorkenntnisse über den Anspruch dieses Buchs hinausgehen. Vielmehr haben wir uns bemüht, das Elementare in leicht anwendbarer Form zusammenzustellen.

Nach den "wilden achtziger Jahren", in denen gerade im Software Engineering überall Unmögliches angeboten wurde, ist unser Anspruch sehr viel bescheidener: Aus unserer Kenntnis der Praxis wissen wir, dass auch die elementaren Regeln in vielen Betrieben und Organisationen missachtet werden. Wir wollen sie in Erinnerung rufen und in leicht handhabbarer Form zusammenstellen. Ihre konsequente Anwendung wird zwar keine Wunder bewirken, aber ein Zuwachs an Qualität (etwa 70 bis 90 % weniger Fehler in der ausgelieferten Software) und eine Erhöhung der Produktivität (etwa 10 bis 30 % niedrigere Kosten bei 5 bis 20 % kürzeren Entwicklungszeiten) ist allemal zu erwarten. Wo wäre es nicht lohnend, diese Verbesserung zu realisieren?

Nach der Einleitung befassen wir uns in Kapitel 2 mit dem gängigsten Ansatz, dem Test, d.h. der Prüfung eines Programms durch Ausführung auf einem Rechner. Kapitel 3 ist den Reviews gewidmet, also der Prüfung "mit den Augen", *ohne* Programmausführung. Im Kapitel 4 folgt ein kurzer Blick auf das Prüfen objektorientierter und logischer Programme. Das gleiche Kapitel befasst sich auch mit der Prüfung unter Zuhilfenahme von Werkzeugen. Das Kapitel 5 ist Management-Aspekten gewidmet. Im Kapitel 6 ist schließlich weiterführende Literatur zur Prüfung von Software und zur Qualitätssicherung zusammen gestellt.

Illustrationen

Die Bilder auf den vorderen Seiten des Buches sind unverkennbar von Wilhelm Busch. Die "Bugs" plagen Software-Entwickler genauso beharrlich wie Onkel Fritz.

Der Erste wird gefasst, aber die Erleichterung dauert nur so lange, bis sich der nächste zeigt: Ein mühsames und leidvolles Unterfangen ohne die richtigen (biologischen) Vertilgungsmittel. Ein solches zeigt Imre Sebestyén am Ende des Buches: Die weisen Elefanten beim Review.

Dank

Monika Peterhans danken wir ganz herzlich für ihren unermüdlichen Einsatz bei stets guter Laune. Die Handschrift im Original des einen Autors und die C-würdigen Verzeigerungen in den Korrekturen des anderen sorgten dafür, dass es ihr nie langweilig wurde.

Wir widmen dieses Buch unseren Frauen *Hanneke, Jutta und Lisbeth dafür,* dass sie die Prüfungen des Lebens mit uns wagen

Karol Frühauf, Jochen Ludewig und Helmut Sandmayr
Baden/Schweiz und Stuttgart, im August 1991

Zu den Neuauflagen

Die zweite bis vierte Auflage haben sich nur in ihrem äußeren Erscheinungsbild verändert.

In der fünften Auflage wurde das Buch gründlich überarbeitet.

Mit der Überarbeitung für diese sechste Auflage haben wir die wenigen Lücken gegenüber dem Lehrplan "Certified Software Tester – Foundation Level" geschlossen. Dieser Lehrplan wird vom International Software Testing Qualifications Board (ISTQB) herausgegeben und gepflegt. Der German Testing Board e.V. (GTB) und SAQ Swiss Testing Board (STB) sind daran aktiv beteiligt.

Wir sehen für den bisherigen Abschnitt "Test logischer Programme" im Kapitel 4 keinen Bedarf mehr und haben ihn darum gestrichen. Bitte geben Sie uns Bescheid, wenn sie anderer Meinung sind.

Wir danken *Ruedi Schild* für den sorgfältigen Vergleich, bei dem er auch den einen oder anderen Fehler im Text entdeckt hat.

Etwa gleichzeitig mit dieser sechsten Auflage erscheint (*nun endlich!*) das inhaltlich überlappende Lehrbuch des Software Engineerings (Ludewig, Lichter, 2006), das schon lange in der Pipeline steckte. Natürlich wurde in den die Prüfung betreffenden Kapiteln des neuen Buchs aus dem alten abgeschrieben; in sehr viel geringerem Maße gab es für diese Neuauflage auch einen Rückfluss.

Karol Frühauf, Jochen Ludewig und Helmut Sandmayr
Baden/Schweiz und Stuttgart, im August 2006

Die Autoren

Karol Frühauf und *Helmut Sandmayr* leiten die Beratungsfirma *INFOGEM AG* in Baden, Schweiz.

Jochen Ludewig ist ordentlicher Professor für Informatik an der Universität Stuttgart.

Inhalt

Verzeichnis der Abbildungen

Kapitel 1

Einleitung und Überblick

In diesem einleitenden Kapitel wird – nach einem Prolog über Prüfungen im Allgemeinen – die Software-Prüfung zunächst in den größeren Zusammenhang des Software Engineerings und des Software-Qualitätsmanagements gestellt. Dann werden die beiden wichtigsten Prüfverfahren, Test und Review, im Überblick und im Vergleich betrachtet. Schließlich werden einige Begriffe definiert, die in den folgenden Kapiteln immer wieder gebraucht werden.

1.1 Prüfungen

Non scholae sed vitae discimus
(Nicht für die Schule, für das Leben lernen wir!)

Prüfungen sind uns allen aus der Schulzeit in mehr oder minder unangenehmer Erinnerung. Welchen Nutzen, welche Wirkungen haben sie?

- Anders als die expliziten, aber eher abstrakten Lehrpläne liefern Prüfungen eine implizite, aber sehr konkrete Definition der Leistungskriterien. Sie sagen damit den Schülern, welchen Stoff sie beherrschen sollten.
- Prüfungen zeigen kollektive und individuelle Schwächen und liefern den Lehrern eine Rückkopplung für den Unterricht.
- Direkt erhöhen sie die Leistungen der Prüflinge nur in Ausnahmefällen (beispielsweise, wenn eine Aufgabe einen Aha-Effekt auslöst).
- Prüfungen schaffen die Möglichkeit, die extrem guten und extrem schlechten Kandidaten zu erkennen; dann können die einen belohnt, die anderen gefördert oder in aussichtslosen Fällen ausgeschlossen werden. Durch eine Belohnung werden die Guten angespornt. Die Förderung der Schlechteren verringert die Gefahr, dass diese den Anschluss verlieren.
  Der Ausschluss als ultima ratio wirkt einerseits als Drohung, gibt andererseits den Lehrern ein Mittel, um die Lähmung der Arbeit durch einzelne Schüler zu verhindern.
- Die Erwartung einer Prüfung beeinflusst das Verhalten der Kandidaten: Sie bemühen sich um besseres Verständnis, damit sie gute Prüfresultate erzielen. Auf diesem Wege erhöht die Prüfung indirekt also auch die Leistung.
- Über die Prüfungsergebnisse werden auch die Lehrer – durch die Aufmerksamkeit der Schüler und ihrer Eltern – laufend geprüft, sodass ein grundsätzlicher Konflikt mit den Regeln des Unterrichts nicht unbemerkt bleiben wird.

Diese Beobachtungen lassen sich weitgehend auf das Thema dieses Buches, die Prüfung von Software, übertragen; dabei ist die Leistung der Schüler durch die Qualität der Software zu ersetzen.

- Prüfungen liefern – in Ergänzung zur oft unzureichenden Spezifikation der Anforderungen – implizit eine simple Definition der Qualitätskriterien. Sie lehren damit die Entwickler, welchen konkreten Anforderungen ihre Erzeugnisse genügen sollten.
- Prüfungen erhöhen die Qualität der Prüflinge nicht direkt. Sie zeigen die Schwächen der Prüflinge (d.h. der Dokumente, Programme, usw.) und liefern damit den Entwicklern eine Rückkopplung für die Arbeit an der Software insgesamt und für die Verbesserung der einzelnen Komponenten (auf Programmebene die Fehlerbehebung)
- Prüfungen schaffen die Möglichkeit, die extrem guten und extrem schlechten Prüflinge zu erkennen; dann können die einen als Vorbilder gezeigt, die anderen verbessert oder in aussichtslosen Fällen verworfen werden. Eine solche Ablehnung kann verhindern, dass ein ganzes System durch eine unzureichende Komponente unbrauchbar wird.
- Die Erwartung einer Prüfung beeinflusst das Verhalten der Entwickler: Sie bemühen sich, Software zu liefern, die gute Prüfresultate erzielt. Auf diesem Wege tragen die Prüfungen indirekt zur Qualität des Produkts bei.
- Die Einhaltung der Regeln zur Durchführung des Projekts, also die Qualität des Managements, sollte durch Audits regelmäßig überprüft werden.

Prüfen ist Vergleichen. Das Verhalten und die Beschaffenheit des Prüfgegenstands wird auf Abweichungen gegenüber den Vorgaben untersucht. Wenn es keine Vorgaben gibt, kann man Versuche machen im Sinne von “Was kann er, was kann er nicht?” oder eine Expertenmeinung über den Gegenstand einholen, wir sprechen in diesem Fall aber nicht von einer Prüfung.

In einer Prüfung werden die Teile des Prüfgegenstands identifiziert, die von den Vorgaben abweichen. Wir erhalten damit eine Aussage über den Grad, in dem der Prüfgegenstand die Anforderungen erfüllt, also über seine Qualität.

Dem, der Prüfungen nur für einen Zeitvertreib der Qualitätssicherer hält, sei gesagt: In der Entwicklung ist ohne Prüfungen keine wirksame Fortschrittskontrolle möglich.

Wenn ein Software-Produkt geprüft wurde, geht das Ergebnis an die Projektleitung. Denn es ist nicht Sache der Entwickler zu entscheiden, welche der vielen Fehler behoben werden sollen; das muss das Management bestimmen, unter Berücksichtigung der Projektsituation, speziell der Projektrisiken. Zudem entschärft die Einbindung des Managements die Feindschaft zwischen den Entwicklern und den Prüfern: Sie gedeiht prächtig, wenn sich die Prüfer über die schlechte Qualität der Prüflinge ärgern und ihre Resultate den Entwicklern über die Mauer werfen, garniert mit der unausgesprochenen Weisung “Bringt diesen Müll in Ordnung!”.

Mit der wachsenden Verbreitung der Software steigen auch die Kosten ihrer Unzuverlässigkeit, die z.B. 1996 beim (durch Software verursachten) Verlust der Ariane V etwa 600 Mio. € betrugen. Ganz unspektakulär, aber durch die große Zahl der Betroffenen nicht weniger teuer sind die nagenden Fehler in einem Textverarbeitungsprogramm. Gelänge es, diese Fehler vor der Verteilung der Software zu entdecken und zu beheben, so blieben vielen Millionen Menschen Leerlauf und Frustration erspart, und der volkswirtschaftliche Nutzen wäre beträchtlich.

Auch für den Hersteller der Software können die Fehler fatale Folgen haben, wenn er für die Fehlerfolgen haften muss. Bei Textverarbeitungssoftware und ähnlichen

Systemen ist das heute noch kein Thema, aber bei den “eingebetteten” Systemen ist die Lage für die Hersteller deutlich riskanter.

1.2 Software Engineering und Software-Prüfung

Programme werden aus vielerlei Gründen geschrieben: Beileibe nicht nur von Software-Entwicklern, die als Angestellte oder als Kleinunternehmer direkt oder indirekt vom Verkauf der Software leben, sondern auch von Laboranten, die Messgeräte abfragen müssen, von Schallplattensammlern, die Ordnung in das musikalische Chaos zu bringen versuchen, von Schülern und Studenten, die ihre Hausaufgaben machen oder Prüfungsfragen beantworten, von spielenden Kindern und spielenden Erwachsenen und von vielen anderen.

Vor allem dort, wo aus gutem Grunde nur von *Programmen*, nicht von *Software* gesprochen wird (vgl. Kapitel 1.8), besteht oft nicht der Wunsch oder die Möglichkeit, systematische Prüfungen durchzuführen. Es ist weder unsere Aufgabe noch unsere Absicht, darüber eine Wertung abzugeben. Wir empfehlen die Prüfung nicht mit quasimoralischen Argumenten (im Sinne eines Ehrenkodex oder einer Benimmregel, “Kartoffeln nicht mit dem Messer zu schneiden”), sondern weil sie nahezu überall, wo Software wirklich gebraucht wird und Kosten erzeugt, dazu führt, dass die geprüfte Software weniger Probleme verursacht und damit letztlich geringere Kosten.

Die Prüfung von Software ist – wie das ganze Software Engineering – kein Selbstzweck, sondern wirtschaftlich geboten; darum ist es sinnvoll, sich zu Beginn über einige wirtschaftliche Aspekte klar zu werden. Dazu sollen hier in Anlehnung an Frühauf, Ludewig, Sandmayr (2002) einige zentrale Aussagen kurz präsentiert werden. Für eine gründliche Einführung sei auf die umfangreiche Literatur des Software Engineerings hingewiesen (vgl. Kapitel 6.1).

1.2.1 Spezielle Eigenschaften von Software

Schon der Begriff *Software Engineering* ist Programm: Software soll als technisches Produkt (des Zusammenwirkens eines versierten Teams) und nicht als künstlerisches Produkt (eines mehr oder weniger begnadeten Einzelkämpfers) behandelt werden. Dabei kann durchaus auf die Erfahrungen anderer Disziplinen zurückgegriffen werden. In einigen Punkten sind aber Besonderheiten zu beachten.

Das wichtigste Merkmal von Software ist ihr immaterieller Charakter. Daher kennen wir weder Fertigungs- oder Transportprobleme noch Abnutzungserscheinungen; die *Produktion* beschränkt sich auf das, was in anderen Bereichen *Entwicklung* heißt. Das gilt auch für die *Wartung*, denn eine Wiederherstellung des *ursprünglichen* Zustands ist ja in keinem Falle gewünscht. Wenn wir also anderen Ingenieuren bei der Prüfung über die Schulter sehen wollen, so müssen wir in die Entwicklung und Konstruktion gehen, nicht in die Fertigung.

Ein weiteres wichtiges Merkmal ist die fehlende Stetigkeit von Programmen. Sie verändern anders als Straßen und Kabel ihre Eigenschaften nicht stetig, wenn man irgendwelche Parameter verändert, sondern sprunghaft, und zwar prinzipiell in unbeschränktem Maße schon bei Änderung eines einzigen Bits. Fehler des Systems treten bei irgendeiner Kombination der Variablenwerte sprungartig auf, zeichnen sich also nicht wie Verschleißeffekte langsam ab. Da schon relativ kleine Programme leicht

10^{20} verschiedene Datenkombinationen als Eingabe erhalten können, kann eine praktische Erprobung (ein Test) nur einen verschwindend kleinen Teil der real auftretenden Situationen prüfen und nicht zu einer Korrektheitsaussage führen.

Schließlich ist uns Software als Produkt des Geistes offenbar wesentlich näher als ein materieller Gegenstand, den wir mit Werkzeugen bearbeitet haben. Diese Nähe hindert uns daran, selbstkritisch seine Schwächen zu erkennen. Prüfungen müssen daher so angelegt sein, dass sie durch diese Hemmung nicht sabotiert werden können.

1.2.2 Der Software-Lebenslauf

Der Software-Entwicklungsprozess wird in der Regel in einzelne Schritte unterteilt. Nach jedem Entwicklungsschritt, meist *Phase* genannt, wird eine Positionsbestimmung durchgeführt und entschieden, ob und wie der nächste Schritt gewagt werden soll. Diese Entscheidungspunkte werden *Meilensteine* genannt. Die bekannten Modelle des Software-Entwicklungsprozesses verwenden diese beiden Begriffe mit unterschiedlicher Interpretation.

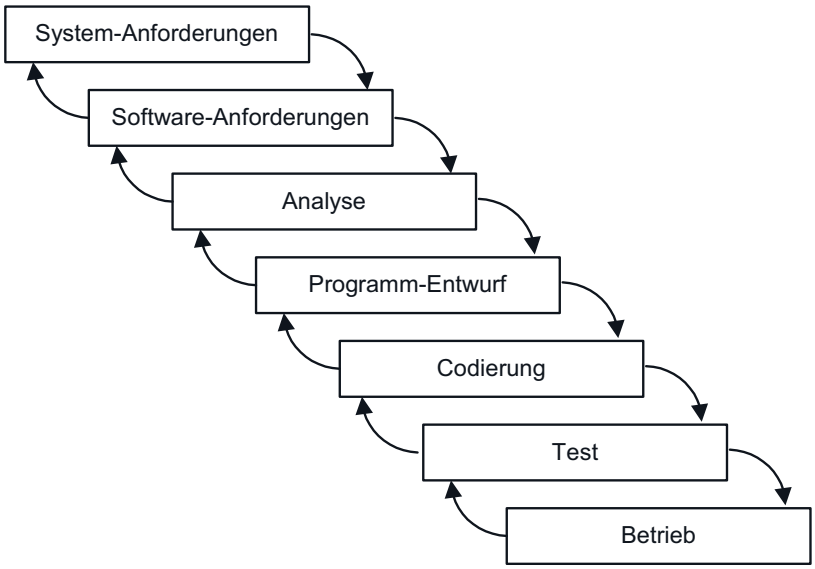
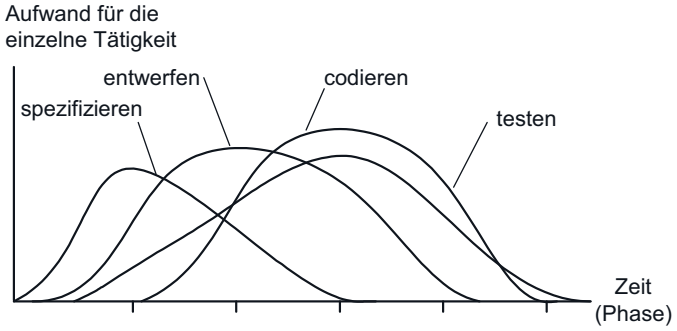


Abbildung 1-1: Das Wasserfall-Modell (Royce 1970)

Das meistzitierte Modell, das *Wasserfall-Modell* von Royce und Boehm, zeigt die Abbildung 1-1 mit der ursprünglichen Terminologie von 1970. Die einzelnen Entwicklungsschritte sind in diesem Modell als Tätigkeiten zu interpretieren. Da es nur bei den einfachsten Programmen möglich ist, die Tätigkeiten streng sequentiell anzuordnen, sind in diesem Modell Zyklen vorgesehen, die aber mit dem Konzept des Meilensteins unverträglich sind.

Wir verwenden durchgehend den Software-Lebenslauf in der Interpretation als *Kostenmodell*. In Abbildung 1-2 sind einige typische *Tätigkeiten* dargestellt (die Aufwandsverteilung ist qualitativ und erhebt keinen Anspruch auf quantitative Genauigkeit). Aus der Darstellung ist offensichtlich, dass in einem bestimmten Zeitabschnitt

verschiedene Tätigkeiten ausgeführt werden; von diesen Tätigkeiten dominiert eine und gibt üblicherweise der Phase den Namen.



Tätigkeit	Arbeitsergebnis	M1	M2	M3	M4	M5
spezifizieren	Anforderungsdokument	V1	V2		V3	
entwerfen	Architekturbeschreibung		V1	V2	V3	
entwerfen	Entwurfsbeschreibung			V1	V2	
codieren	Quellcode				V1	V2
testen	Testvorschrift, -bericht			V1	V2	V3
schreiben	Benutzerhandbuch		V1			V2
		Versionen (Vi) der Arbeitsergebnisse an den Meilensteinen (Mj)				

Abbildung 1-2: Tätigkeiten und Phasen – das Kostenmodell (Sandmayr 1991)

Die Phase ist definiert durch ihre *Arbeitsergebnisse*; dazu gehören Dokumente, die in der Phase neu erarbeitet wurden, und Dokumente aus früheren Phasen, die überarbeitet wurden und in neuer Version vorliegen (Vi in der Abbildung 1-2). Die Überarbeitung kann durch Fehler in einem Dokument, aber auch durch geänderte Bedürfnisse des Auftraggebers oder durch neue Erkenntnisse der Entwickler bei der weiteren Ausarbeitung begründet sein.

Das Ende der Phase ist durch einen Meilenstein markiert, an dem geprüfte und bewertete Arbeitsergebnisse vorliegen. In den Normen werden häufig Reviews gefordert, an denen Arbeitsergebnisse formell freigegeben werden. Diese so genannten *Freigabe-Reviews* finden in Form einer Sitzung statt, in der unter anderem kontrolliert wird, ob die Arbeitsergebnisse mit den zugehörigen Prüfberichten vorliegen. Das bedeutet keineswegs, dass am letzten Tag der Phase mit einem Kraftakt alle Arbeitsergebnisse einer fachlichen Prüfung in Form von technischen Reviews unterzogen werden.

Schon in den achtziger Jahren wurden verschiedene weniger rigide Vorgehensmodelle vorgeschlagen und eingesetzt, z.B. das so genannte Prototyping, bei dem ein Ausschnitt des Zielsystems rasch entwickelt und überprüft wird, um auf diese Weise die Anforderungen zu klären. Inzwischen sind weitaus radikalere Modelle populär geworden, vor allem die so genannten leichten oder agilen Prozesse, bei denen auf eine umfassende Spezifikation im traditionellen Sinne ganz verzichtet wird. Dazwischen positioniert sind iterative Modelle, z.B. der iterative Rational Unified Process (Kruchten, 2000), dem das oben geschilderte Kostenmodell in ausgefeilter Form zu Grunde liegt.

Natürlich sind Prüfungen bei jedem Vorgehen notwendig, und ihr Nutzen ist umso größer, je früher sie durchgeführt werden. Und natürlich kommt keine Prüfung ohne Bezug aus. Der Verzicht auf dokumentierte Anforderungen ist in jedem Falle ein Wagnis, das man nicht leichtfertig eingehen sollte.

1.2.3 Software-Nutzen und -Kosten

Der *Nutzen* eines Software-Produkts ist bestimmt durch die Übereinstimmung zwischen Produkt und tatsächlichen Anforderungen sowie durch zusätzliche Leistungen und andere Eigenschaften, die nicht gefordert waren, aber als vorteilhaft empfunden werden.

Aus der Sicht des Herstellers setzen sich die *Kosten* eines Software-Produkts aus *Herstellungskosten* und *Qualitätskosten* zusammen. Alle Aufwendungen für das Erbringen der geforderten Leistung, das *Erzeugen der Qualität*, zählen zu den Herstellungskosten. Die Qualitätskosten umfassen alle (zusätzlichen) Aufwendungen für das Verhüten, Erkennen, Lokalisieren und Beheben von Fehlern sowie die eventuellen Folgekosten aus Fehlern, die erst im Betrieb auftreten.

Bis zur Auslieferung der Software gibt es keine Folge- und Garantiekosten (grau hinterlegt in Abbildung 1-3), d.h. die Fehlerkosten bestehen nur aus den Fehlerbehebungskosten. Natürlich wird man umso mehr in Prävention und Prüfung investieren, je höher das Risiko durch Folgekosten und Garantieleistungen ist.

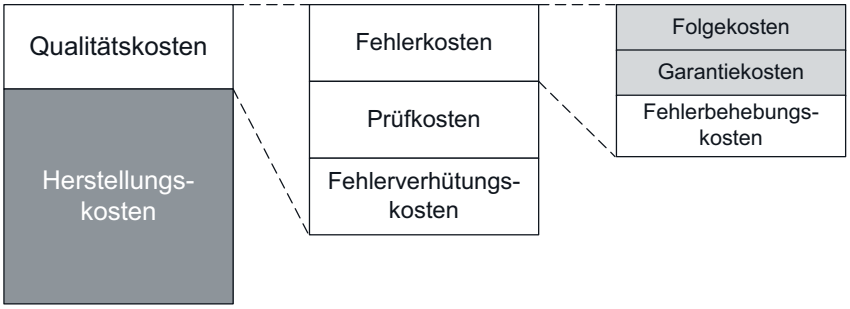


Abbildung 1-3: Aufteilung der Software-Kosten

Die Abgrenzung zwischen den Kostenarten ist nicht eindeutig und muss von jedem Unternehmen nach seinen Bedürfnissen definiert werden. Bei der Abgrenzung sind die folgenden Fragen *praktikabel* zu beantworten:

1. Sind konstruktive Maßnahmen als Fehlerverhütung zu betrachten (*Fehlerverhütungskosten* als Teil der Qualitätskosten) oder als Beitrag zu den Herstellungskosten? Zählen Ausbildungskosten als Personal- oder als Fehlerverhütungskosten?
2. Welche Tätigkeiten zur Fehlererkennung sind unter *Prüfkosten* zu verbuchen? Ist ein Compilerlauf als eine Prüfung anzusehen oder als Entwicklungstätigkeit? Sind die vom Programmierer durchgeführten und nicht dokumentierten Laufversuche als Prüfung zu werten?
3. Ab wann soll ein entdeckter Mangel als Fehler verbucht werden? Verursacht die Behebung eines Syntax-Fehlers *Fehlerbehebungskosten*? Und die Korrektur von Tippfehlern in einem Dokument?
4. Sind Kulanzkosten als Werbeausgaben oder als *Folgekosten* zu betrachten?