Modellqualität als Indikator für Softwarequalität: eine Taxonomie

Florian Fieber · Michaela Huhn
Bernhard Rumpe

Auch neuere Beispiele zur Entwicklung komplexer softwarebasierter Systeme zeigen, dass viele Projekte weiterhin mit erheblichen zeitlichen und finanziellen Risiken sowie dem Risiko des Scheiterns verbunden sind.

Deshalb ist es zwingend notwendig, dass eine strukturierte, frühzeitig einsetzende Qualitätssicherung (QS) in Kombination mit adäquater Meilensteinplanung in Softwareprojekten einen höheren Stellenwert einnimmt. Die Projektverantwortlichen erkennen zusehends den Wert von entwicklungsbegleitenden QS-Maßnahmen und stellen diesen entsprechend mehr Ressourcen zur Verfügung [39]. Neben konstruktiven Maßnahmen (wie dem Einsatz von Normen, Richtlinien und standardisierten Prozessen) stehen vor allem analytische Maßnahmen (z.B. statische Analyse, Review, dynamischer Test) zur Verfügung und werden immer erfolgreicher angewandt.

Die frühzeitige Planung und Durchführung von QS-Maßnahmen ist in der Regel nur auf Basis adäquater, formaler Dokumentation möglich. Dem manuellen Review bzw. der Inspektion solcher Dokumentationsartefakte zumeist weit überlegene Technik sind die automatisierte Analyse sowie Simulation, Testen und am besten die konstruktive Generierung des lauffähigen Codes aus diesen Dokumenten. Alle diese Techniken dienen der Effizienzsteigerung in der Entwicklung und gehen einher mit einer gleichzeitigen Qualitätssteigerung, da sie den Entwicklern manuelle Tätigkeiten abnehmen. Eine solche Automatisierung bedeutet aber, dass die im Entwicklungsprozess zum Einsatz kommenden Dokumentationsartefakte „maschinenlesbar" sein müssen. Jenseits von informellen Beschreibungen kommen daher im modellbasierten Softwareentwicklungsprozess verschiedenste Arten von Modellen zum Einsatz. Dabei ist sowohl nach Entwicklungstätigkeiten (früher: „Phasen") als auch nach den Einsatzformen der verwendeten Modelle zu unterscheiden.

Am umfassendsten unterstützt die UML [44, 45, 52, 53] mit ihren 13 Modellarten den Entwicklungsprozess. Leider sind die zugehörigen Werkzeuge im Allgemeinen nicht so ausgereift wie etwa spezialisierte Werkzeugketten für die Modellbildung von regelungstechnischen Teilaspekten um Matlab/Simulink [8], zur Modellierung von Testsammlungen mit TTCN [7] oder für bestimmte Zielsetzungen extra gebildete domänenspezifische Sprachen [16, 25, 26, 34]. Derzeit für die Softwareentwicklung wichtige Modellarten erlauben eine

- Struktur- und Schnittstellenbeschreibung,
- konstruktive Verhaltensbeschreibung, typischerweise durch Kombination von Zustand und Funktion,
- deskriptive Kommunikationsprotokolle und -mechanismen,
- Darstellung der logischen sowie physischen Verteilung,

DOI 10.1007/s00287-008-0279-4

Florian Fieber · Michaela Huhn · Bernhard Rumpe
Carl-Friedrich-Gauß-Fakultät,
Technische Universität Braunschweig,
Mühlenpfordtstr. 23, 38106 Braunschweig
E-Mail: {fieber, m.huhn, b.rumpe}@sse-tubs.de

[FHR08] F. Fieber, M. Huhn, B. Rumpe
Modellqualität als Indikator für Softwarequalität: eine Taxonomie
In: Informatik-Spektrum. Springer Verlag.
Band 31, Heft 5, Oktober 2008.
www.se-rwth.de/publications

Zusammenfassung
Komplexität, Anforderungsmanagement und Variantenvielfalt sind zentrale Herausforderungen bei der Entwicklung und Evolution heutiger softwaregesteuerter Systeme. Diesen wird zunehmend durch den Einsatz modellbasierter Entwicklungsmethoden begegnet. Dadurch wird das Modell zum zentralen Artefakt und die Erstellung und Nutzung von Modellen zu einer zentralen Tätigkeit in der Softwareentwicklung. Mit der Bedeutung der Modelle steigen auch die Ansprüche an ihre Qualität. Dieser Beitrag untersucht die Implikationen, die daraus entstehen, insbesondere werden sinnvolle Qualitätsmerkmale für softwarebeschreibende Modelle identifiziert und diskutiert.

- (meist informelle) Organisation und Strukturierung der Anforderungsbeschreibung,
- Modellierung von Aufgaben- und Prozessabläufen und die
- Datenmodellierung.

Neben konstruktiven Modellen, die primär zur Gewinnung von Codes dienen, werden Simulationsmodelle zur Gewinnung von Erkenntnissen mittels Durchführung von Abläufen und analytische Modelle für die Verifikation genutzt. Manche Modelle bieten auch die gleichzeitige Darstellung konstruktiver Anteile und zu prüfender Anteile. Konstruktiv sind etwa Klassen und Attribute im Klassendiagramm oder die Action an der Transition, zu Prüfzwecken eingesetzt zu werden bzw. zu verifizieren sind jedoch Kardinalitäten oder Nachbedingungen.

Zunächst scheint die Qualität der für all diese Zwecke im Einsatz befindlichen Modelle für das Ergebnis, einer qualitativ hochwertigen, korrekten, effizient und komfortabel nutzbaren, sicheren, robusten, rechtzeitig fertig gestellten und (meist auch) weiter entwickelbaren Software, nicht relevant zu sein! Dies ist zumindest die of geäußerte Meinung wenig erfahrener Softwareentwickler. Viele Erfahrungsberichte und Indikatoren zeigen jedoch, dass die Qualität der im Einsatz befindlichen Modelle mit der Qualität der entwickelten Software auf vielfältige Weise verbunden ist. Wurden Modelle bisher vor allem zu Zwecken der Visualisierung von Entscheidungen, Kommunikation mit Experten und Dokumentation des Systems eingesetzt, gewinnen sie mit dem zunehmenden Einsatz modellbasierter Softwareentwicklung stetig an Bedeutung [40] – die Modelle sind das wichtigste Artefakt der Entwicklung, ihre Qualität trägt somit entscheidend zum Erfolg des Projekts bei. Die QS-Maßnahmen allgemeiner Entwicklungsprozesse fordern daher, wie etwa in der Norm IEC61508 [19] beschrieben, zurecht eine adäquate Qualität der Artefakte bzw. Modelle im Softwareentwicklungsprozess ein. Modellqualität ist kein Selbstzweck, sondern abhängig von den Projektzielen zu identifizieren.

Erst durch die Identifikation geeigneter Qualitätsmerkmale lässt sich Qualität messbar und quantifizierbar machen. Bereits während der Entwicklung können damit Aussagen zur Qualität getroffen werden und gegebenenfalls Maßnahmen rechtzeitig geplant und eingeleitet werden. Für Softwaresysteme bzw. Codes wurden die unterschiedlichen Qualitätsmerkmale bereits mithilfe verschiedener Qualitätsmodelle definiert (z.B. das Qualitätsmodell der ISO 9126:2001 [32]). Die Merkmale dieser Qualitätsmodelle lassen sich jedoch nur bedingt auch auf Softwaremodelle anwenden. Bei der System- bzw. Softwaremodellierung gibt es noch kein anerkanntes und weithin eingesetztes Verfahren, um die Qualität der Modelle zu messen und zu verbessern. Die mangelnde Qualitätssicherung betrifft dabei alle Modelle der unterschiedlichen Phasen gleichermaßen, z.B. Modelle zur Analyse der Anforderungen (Modellierung der Anforderungen in einem Analysemodell) oder Modelle zum Entwurf des Systems (Transformation des Analysemodells in ein Designmodell).

Dieser Beitrag entwickelt eine Taxonomie der Qualitätsmerkmale für Softwaremodelle. Der Beitrag soll damit eine Grundlage für weitere empirische Untersuchungen und Werkzeugentwicklungen zur Modellqualität bilden. Zunächst werden die zugrunde liegenden Modell- und Qualitätsbegriffe eingeführt und eine Kategorisierung der Qualitätsarttribute für Modelle diskutiert. Dann wird ein Katalog konkreter Qualitätsattribute für Modelle vorgestellt und ihre Auswirkungen auf die Qualität der resultierenden Software diskutiert. Dieser Katalog ist strukturiert nach innerer Qualität und horizontalen sowie vertikalen Qualitätsanforderungen zwischen Modellen. Qualitätsanforderungen an Modelle implizieren auch Anforderungen an

die im Einsatz befindliche Modellierungsnotation, die anschließend eigenständig diskutiert werden. Abschließend wird über Techniken zur Qualitätsmessung und Bewertung diskutiert, die letztlich durch adäquate methodische Einbettung zu einer Qualitätssicherung von Modellen der verschiedenen Phasen bis hin zum laufenden System führen.

Der Modellbegriff

Der Begriff „Modell" wurde in der italienischen Renaissance als Fachbegriff der Architektur geprägt. Er bezeichnet ein Anschauungsobjekt, anhand dessen Auftraggeber Form und Gestaltung eines geplanten Gebäudes vorgeführt sowie konstruktive, architektonische Fragestellungen geklärt wurden. Bis zum Beginn des 20. Jahrhunderts etablierte sich der Modellbegriff als veranschaulichende Beschreibung komplexer Theorien in den Naturwissenschaften und als vereinfachendes Abbild oder Vorbild eines technischen Konstrukts in den Ingenieurswissenschaften. Eine heute allgemein anerkannte Definition des wissenschaftlichen Modellbegriffs geht zurück auf H. Stachowiak [49], der folgende Merkmale eines *Modells* als konstituierend herausstellt:

1. Abbildung: Modelle stehen immer in Bezug zu einem Original, das sie abbilden.
2. Verkürzung: Modelle erfassen i. Allg. nicht alle Attribute des durch sie repräsentierten Originals.
3. Pragmatismus: Modelle sind ihren Originalen nicht per se eindeutig zugeordnet, sondern ihre Ersetzungsfunktion erfüllen sie für bestimmte Subjekte unter bestimmten Einschränkungen (Zweck und Randbedingungen der Modellkonstruktion).

Zusätzlich wird zwischen *deskriptiven* und *präskriptiven* Modellen unterschieden: Deskriptive Modelle (z.B. ein Stadtplan) beschreiben ein Original zum leichteren Verständnis, während präskriptive Modelle (z.B. ein Bebauungsplan) zur Erstellung eines Originals beitragen. Der primäre Fokus dieses Artikels liegt auf präskriptiven Modellen (z.B. Entwurfsmodellen) eines zu entwickelnden Softwaresystems. Diese dienen im Wesentlichen der schrittweisen Strukturierung und Formalisierung des Systems durch Verfeinerung [41]. Aufgrund der immateriellen Natur von Software ist allerdings auch der Code ein Modell des ausführbaren Systems, das natürlich Rückschlüsse auf die Qualität des Produkts, aber auch auf die Qualität anderer Artefakte zulässt (z.B. Konfiguration, Dokumentation).

Je nach Einsatzzweck eines Modells kann von unterschiedlichen Aspekten des Systems abstrahiert werden und damit *Sichten* bilden: Klassen- oder Zustandsdiagramme im Entwurf [36] sind präskriptive Modelle des zu erstellenden Softwaresystems. Daneben kommen aber auch deskriptive Modelle wie Umgebungsmodelle bei eingebetteten Systemen oder Ontologien im Zusammenhang mit Informationssystemen im Entwurf zum Einsatz. Bei der Anforderungsspezifikation wird aus der Beschreibung der Istsituation die Zielsituation abgeleitet, die mithilfe des zu entwickelnden Softwaresystems erreicht werden soll. In diesem Sinne ist die Anforderungsspezifikation ein *transientes Modell* [36]. Beim Nachweis der funktionalen Korrektheit werden deskriptive Entwurfs- und Systemmodelle *prognostisch* eingesetzt, wenn mit formaler Verifikation nachgewiesen wird, dass Anforderungen für jedes mögliche Verhalten im Modell erfüllt sind. Bei der Evolution von Software werden Architektur- und Entwurfsmodelle *explorativ* verwendet, d.h. geplante Veränderungen werden erst am Modell vorgenommen und evaluiert, bevor sie in einer Implementierung umgesetzt werden. Die Taxonomie der Modelle ergibt sich aus dem primären Zweck eines Modells: Ein Zustandsdiagramm als Verhaltensmodell im Entwurf ist präskriptiv, wenn es in die Dokumentation des implementierten Systems übernommen wird; wird es deskriptiv verwendet, wenn der Zustandsraum dieses Diagramms mit Model-Checking exploriert wird, um funktionale Anforderungen zu verifizieren, wird es prognostisch eingesetzt.

Eine andere Betrachtung des Modellierungszwecks findet sich im Umfeld der Referenzmodellierung von Informationssystemen [13]: Hier steht der *kreative Konstruktionsprozess* mit seinen Randbedingungen und subjektiven Einflüssen im Vordergrund. Diese Interpretation trägt dem immateriellen Charakter von Software und den kaum objektivierbaren Einflüssen im Softwareentwicklungsprozess Rechnung.

Der Fortschritt bei Modelltransformationen, Codegenerierung und formalen Methoden ermöglicht es, aus immer abstrakteren Modellen verfeinerte Modelle abzuleiten und somit automatisiert lauffähige Software zu erzeugen und zu

analysieren. Viele heute im Einsatz befindliche Modelltypen können automatisiert simuliert werden. So verstärkt sich der transiente Charakter von Modellen im Software-Engineering, und die Beziehungen zwischen Modell und dem modellierten System werden vielfältiger und komplexer.

Oft wird Modell auch als Synonym zu einer *Sicht* betrachtet. Eine Sicht beschreibt einen Teilaspekt einer, bzw. eine bestimmte Perspektive auf das Gesamtsystem (abgebildet durch die Softwarearchitektur) [11]. Aufgrund der Systemkomplexität gibt es kein Modell, das ein Softwaresystem als Ganzes und in allen Details beschreibt (außer bei sehr trivialen Systemen), sondern einzelne, geeignet komponierbare Modelle, die gemeinsam die Softwarearchitektur oder den Entwurf bilden. Dadurch entstehen in einem Entwicklungsprozess sich *horizontal* ergänzende Modelle, die in ihrer Kombination (d.h. Komposition) notwendige, konstruktionsrelevante Informationen über das gesamte System darstellen, sowie *vertikal* sich in ihrem Informationsgehalt jeweils ablösende Modellketten. Abbildung 1 zeigt eine von vielen verschiedenen, meist methodenabhängigen Kombinationsmöglichkeiten. Einzelne Elemente der Architektur können somit in mehreren Modellen vorkommen (bzw. referenziert werden). Modelle einer späteren Schicht der Modellkette beinhalten Informationen der früheren Schicht, angereichert um technische Details und Entwurfsentscheidungen. Sie bilden den Ausgangspunkt für weitere Entwicklungsschritte.

In der Research-Roadmap zur „Model-Driven Development of Complex Software“ [23] wurde angemahnt, dass neben der noch ausstehenden Konsolidierung und Verbesserung der Werkzeuglandschaft sowie der Standardisierung von Entwicklungsnotationen, die Qualitätssicherung von Modellen eine aktuelle Herausforderung an Wissenschaft und Industrie darstellt. Diese Qualitätssicherung muss auf einem empirischen Erfahrungsschatz zur Modellqualität beruhen, die geeignete Qualitätsziele und -attribute, Qualitätsmaße, sowie deren Messung und Interpretation umfasst. Dabei müssen gerade wegen der Durchgängigkeit vom Modell zum Original und den vielfältigen Zwecken der Modelle klar die verschiedenen Dimensionen der Qualität unterschieden werden: Soll anhand eines Modells die Qualität des Endprodukts Software beurteilt werden, oder ist das Modell und seine Beziehung zum Original auf seine Eignung für den angegebenen Zweck zu beurteilen und die Qualität der Software nur mittelbares Ziel. Außerdem sind die Einflüsse der einzelnen Qualitätsattribute der Modelle auf die Qualitätsattribute des Codes zu klären. So bedeutet etwa gute Lesbarkeit des Modells keineswegs, dass daraus generierter Code ebenfalls gut lesbar ist, während zwischen Modell- und Systemkorrektheit sehr wohl eine Korrelation bestehen dürfte.

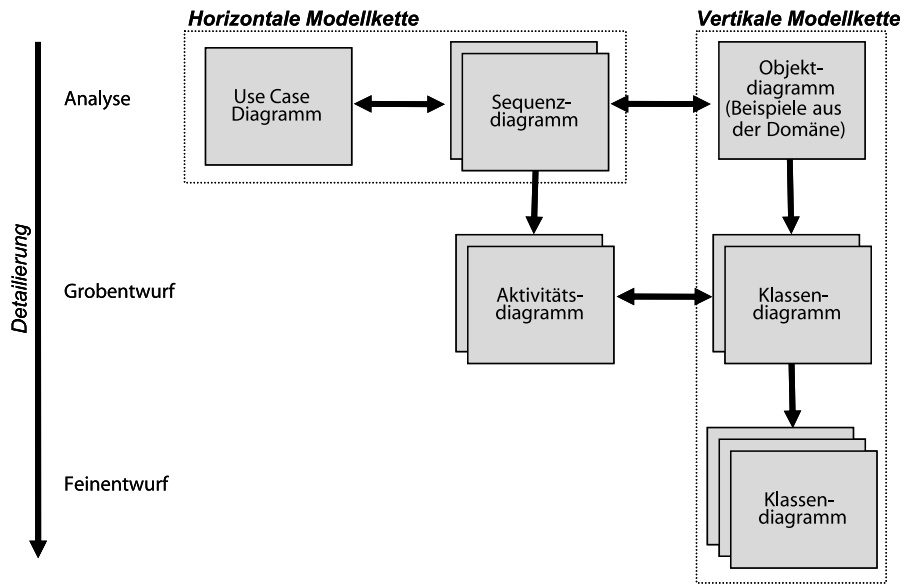

Abb. 1 Beispiele für eine Verwendung horizontaler und vertikaler Modellketten

Der Qualitätsbegriff

„Qualität" leitet sich von dem lateinischen Wort *qualitas* (Beschaffenheit) ab und wird in der Norm ISO/IEC 9126:2001 zur Softwareproduktqualität als „the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs" aus der ISO 8402 übernommen. Die Produktqualität wird von der Prozessqualität, d.h. der Güte des Entwicklungsprozesses, abgegrenzt. Die Norm unterscheidet zwischen interner, externer und Gebrauchsqualität:

- *Interne Qualität* ist die Gesamtheit der Gütemerkmale aus einer herstellungsorientierten Sicht. Sie umfasst die Qualitätsattribute an alle Zwischenprodukte der Entwicklung.
- *Externe Qualität* beinhaltet die technischen Gütemerkmale des Produkts aus einer externen Sicht. Die externe Qualität wird i. Allg. durch Messungen und Tests während der Ausführung der Software in einer definierten Umgebung bestimmt.
- *Gebrauchsqualität* beschreibt die Gütemerkmale aus Sicht des Benutzers, dabei ist nicht nur der Kontext der Benutzung zu berücksichtigen, sondern auch inwieweit der Benutzer die angestrebten Ziele mit der Software erreichen kann.

Wir beziehen dabei alle Qualitätsarten primär auf das Softwareprodukt. Darüber hinausgehend lässt sich die Modellqualität von präskriptiv verwendeten Entwurfsmodellen im Wesentlichen der internen Qualität zuordnen. Außerdem hängen transiente Modelle im Bereich der Anforderungserhebung und Modelle, aus denen automatisiert Code erzeugt wird, unmittelbar mit der externen und der Gebrauchsqualität zusammen.

Qualitätsmodelle

Zur Differenzierung der Qualitätsanforderungen werden sogenannte „Qualitätsmodelle" verwendet. Qualitätsmodelle setzen sich aus – häufig hierarchisch strukturierten – Qualitätsattributen zusammen. Den Qualitätsattributen werden Techniken zur Qualitätsbewertung zugeordnet. Eine Qualitätsvorgabe gibt an, zu welchem Grad die Erfüllung des Qualitätsattributs für das zu bewertende Artefakt gefordert wird. Zur Erstellung eines Qualitätsmodells werden entweder generische Qualitätsmodelle den spezifischen Projekterfordernissen angepasst oder aber ein spezifisches Qualitätsmodell gemäß eines definierten Vorgehensmodells entwickelt:

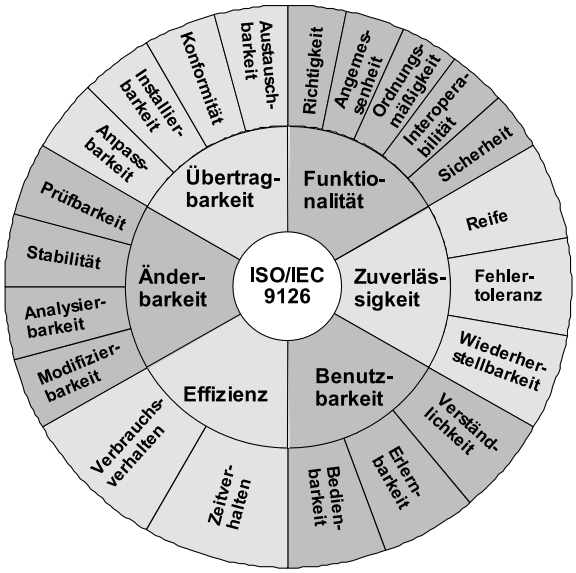

Abb. 2 ISE/IEC 9126 Qualitätsmodell

Die ISO/IEC 9126:2001 bietet das in Abb. 2 dargestellte hierarchische, generische Qualitätsmodell für die interne und externe Qualität sowie ein einfaches für die Gebrauchsqualität. Die darin vorgeschlagenen Qualitätsattribute müssen dann gemäß der Norm im Kontext der aktuellen Phase im Lebenszyklus und im Sinne der jeweiligen Stakeholder verfeinert, ergänzt oder verworfen werden. In der Entwicklung sind die externen und die Gebrauchsattribute in interne Qualitätsattribute zu transformieren, um im Prozess zielgerichtet auf die Qualitätsanforderungen an das Produkt zuzusteuern. Methodisch ähnliche Merkmale haben eine Reihe von Ansätzen wie etwa [31].

Wie bereits angesprochen, gibt der Modellierungszweck die Ebene einer Qualitätsanforderung vor, z.B. setzt Wiederverwendbarkeit als Qualitätsziel die langfristige Perspektive auf einen evolutionären Lebenszyklus voraus. Diese Dimension ist naturgegebener Maßen bestimmend bei der Ableitung von Bewertungstechniken: Während für Qualitätsziele für klar abgegrenzte Zwecke Metriken oder analytische Verfahren etabliert sind, wird bei Qualitätsanforderungen an das Endprodukt und seine Evolution oft mit Heuristiken, Best Practices und qualitativen Verfahren gearbeitet.

Alternativ wurden systematische Vorgehensmodelle zur Entwicklung spezifischer Qualitätsmodelle

vorgeschlagen wie der Goal-Question-Metric-Ansatz (GQM) [3, 10]. Bei GQM werden zuerst Qualitätsziele (goals) festgelegt, wobei sich ein Qualitätsziel aus dem zu bewertenden Objekt, den Qualitätsattributen für dieses Objekt sowie der Perspektive eines Stakeholders zusammensetzt. Im zweiten Schritt werden dann Fragestellungen (questions) abgeleitet, die von den Qualitätszielen zu Bewertungen für die Qualitätsattribute der zu bewertenden Objekte führen. Auf der dritten, quantitativen Ebene stehen Metriken (metrics). Zur Evaluation eines Qualitätsziels werden dann die den abgeleiteten Fragestellungen zugeordneten Metriken ausgewertet und gewichtet kumuliert.

Als Weiterentwicklung und Spezialisierung des GQM-Ansatzes zur Bewertung von Softwarearchitekturen können szenarienbasierte Architekturevaluationsmethoden [18, 20] verstanden werden: Wie GQM unterstützen sie die Bildung eines spezifischen Qualitätsmodells durch detaillierte Richtlinien und Vorlagen für ein standardisiertes, systematisches Vorgehen. Um ein gemeinsames Verständnis der Stakeholder von den Qualitätsanforderungen schon in der frühen Phase des Architekturentwurfs zu gewährleisten, werden Szenarien spezifiziert, mit denen einerseits komplexe Situationen – etwa zur geplanten Evolution eines Systems – beschrieben werden können, andererseits der diffuse Raum der Möglichkeiten eingeschränkt wird.

Generische Qualitätsmodelle haben den Vorteil, dass sie eine strukturierte Ausgangsbasis mit einer oft umfangreichen Liste von zu berücksichtigenden Qualitätsattributen anbieten. Allerdings müssen – wie schon Rombach in [5] ausführt – in praktisch allen Fällen projekt- und phasenspezifische Anpassungen vorgenommen werden. Insbesondere die Ableitung aussagekräftiger Bewertungstechniken für Zwischenprodukte der Entwicklung aus den Qualitätsattributen wird selten gelöst. GQM und szenarienbasierte Ansätze tragen der Vielschichtigkeit von Qualitätsanforderungen Rechnung, indem komplexere Qualitätsziele oder Szenarien spezifiziert werden. Zusätzlich erhält der Qualitätsmanager Techniken an die Hand, um Metriken für die Qualitätsziele abzuleiten. Daher sind GQM und szenarienbasierte Ansätze sicher im ersten Anlauf als aufwändiger zu betrachten, allerdings führen sie zu passgenaueren Qualitätsmodellen, und sie stellen die Frage nach dem Zusammenhang zwischen Qualitätszielen und Maßen für die Zwischenprodukte der Entwicklung methodisch in den Mittelpunkt.

Qualitätsattribute in der Softwareentwicklung

Als Zwischenprodukte der Entwicklung werden Modelle implizit fast immer zur Bewertung der Softwarequalität herangezogen. Explizit werden Aspekte der Modellqualität allerdings nur in Teilbereichen behandelt. Im Folgenden wird ein Überblick zu bekannten Ansätzen in der Literatur gegeben.

Modellqualität in Informationsmodellen

Informationsmodelle dienen der Strukturierung von semantischen Räumen bei der Konzeption komplexer Informationssysteme, insbesondere für Anwendungen in Betrieben und Behörden.

- In den Arbeiten zu Grundsätzen ordnungsgemäßer Modellierung [15] werden als übergeordnete Qualitätsattribute im Rahmen der Informationsmodellierung Richtigkeit, Relevanz, Wirtschaftlichkeit, Klarheit, Vergleichbarkeit und systematischer Aufbau gefordert. Diese Attribute werden dann verfeinert und müssen vor der Anwendung projektspezifisch angepasst werden.
- Schütte entwickelt diese Grundsätze in [48] weiter und fordert Konstruktionsadäquanz, Sprachadäquanz sowie Klarheit, Vergleichbarkeit und systematischen Aufbau.

Die Herausforderung bei diesen Modellen besteht darin, dass das Original, die strukturierte Begriffswelt der Anwendungsdomäne, rein gedanklich und nur in den seltensten Fällen standardisiert ist. Die Qualitätsattribute leiten sich daher aus Anforderungen an die Abbildung und dem Zweck des Modells als Kommunikationsgrundlage der Projektbeteiligten ab. Sie sind unabhängig von dem gewählten Modellierungsparadigma und der -notation. Aber – wie Becker und Schütte herausstellen – sind sie in Teilen nicht objektiv bewertbar: So ergibt sich die semantische Richtigkeit des Informationsmodells (aus softwaretechnischer Sicht: die ursprüngliche Anforderungs- und Umgebungsspezifikation) „aus dem Diskurs der Gutwilligen und Sachkundigen".

Qualität der Softwarearchitektur

Eine Softwarearchitektur beschreibt die wesentlichen Strukturen eines Systems, die Softwarekom-

ponenten, ihre sichtbaren Eigenschaften und Beziehungen sowie die bestimmenden Prinzipien des Entwurfs und der Evolution des Systems [42, 54]. Produktorientierte Qualitätsziele sind daher einer Architektur inhärent und werden in einer Reihe von Ansätzen diskutiert:

- [4] definiert als Qualitätsmerkmale für eine Architektur: Verfügbarkeit, Änderbarkeit, Performance, Sicherheit, Testbarkeit und Benutzbarkeit.
- [35] definiert als Qualitätskriterien: Testbarkeit, Wartbarkeit, Erweiterbarkeit und Portierbarkeit.
- [11] definiert als nicht-funktionale Eigenschaften einer Softwarearchitektur: Änderbarkeit (mit den Aspekten Wartbarkeit, Erweiterbarkeit, Restrukturierbarkeit und Portierbarkeit), Interoperabilität, Effizienz, Zuverlässigkeit (mit den Aspekten Fehlertoleranz und Robustheit), Testbarkeit und Wiederverwendbarkeit.
- [12] definiert als Qualitätsmerkmale: Performance, Maintainability, Flexibility, Reliability, Security und Safety.

Weitere Merkmale aus der Literatur sind Vollständigkeit, unterteilt in syntaktische und semantische Vollständigkeit [24] sowie Skalierbarkeit, Portabilität [4]. Naturgemäß werden an die Architektur nachhaltige Qualitätsanforderungen gestellt, die in der frühen Phase des Architekturentwurfs nur ungenau und aufwändig bewertet werden können. Daher dominieren in diesem Bereich qualitative Bewertungsmethoden wie etwa ATAM (Architecture Trade-Off Analysis Method) [4]. Quantitative Verfahren versuchen, die Bewertungen für verschiedene Qualitätsattribute über Qualitätsraten zusammenzuführen [4, 20]. Für einzelne Qualitätsattribute wie Sicherheit oder Performanz existieren quantitative Bewertungsverfahren, die aus der Architektur Analysemodelle ableiten, die dann mathematisch oder simulativ ausgewertet werden (s. [29] für einen Überblick).

Qualität von Entwurfsmodellen

Insbesondere zur Qualität objektorientierter Entwürfe gibt es eine Vielzahl von Arbeiten:

- [6] definiert ein hierarchisches Qualitätsmodell mit den übergeordneten Qualitätsattributen Wiederverwendbarkeit, Flexibilität, Verständlichkeit, Funktionalität, Erweiterbarkeit und Effektivität. Auf der zweiten Ebene haben Bansiya und Davis jedem Qualitätsattribut mehrere Entwurfsmerkmale zugeordnet, die dann mit Metriken bewertet werden.
- Ein Ansatz zur Entwurfsqualität findet sich in [51]. Dort werden die Qualitätsmerkmale Vollständigkeit (completeness), Inhärenz (inherence), Übersichtlichkeit (clarity), Konsistenz (consistence), Orthogonalität (orthogonality) und Allgemeingültigkeit (generality) als Eigenschaften innerhalb und zwischen Modellen definiert.
- Ausführlich diskutiert Reißing in [43] Qualitätsattribute für objektorientierte Entwürfe. Sein differenziertes, generisches Qualitätsmodell konzentriert sich auf die herstellungsorientierten Attribute Wartbarkeit, Wiederverwendung, Wiederverwendbarkeit, Brauchbarkeit, Testbarkeit und Prüfbarkeit, für die er dann auf Basis seines Entwurfsmetamodells Designkriterien und Metriken entwickelt.

Bei der Entwurfsqualität dominieren Arbeiten zu Bewertungsverfahren: Metriken für objektorientierte Entwürfe werden inzwischen bereits von einer Reihe von Werkzeugen unterstützt [21]. Standardisierte oder projektspezifisch angepasste Modellierungsrichtlinien oder Checklisten zur Qualitätssicherung sind auch in der industriellen Praxis weit verbreitet [1, 30]. Neuere Arbeiten beziehen auch Domänenwissen und softwaretechnische Expertise (Best Practices) in die Qualitätsbewertung ein, indem etwa die Verwendung von Mustern, aber auch Anti-Mustern („Bad Smells"), analysiert wird. Briand et al. [9] weisen empirisch für einzelne Qualitätsattribute nach, dass die mit Metriken festgestellten Qualitätsmaße auf Entwurfsebene signifikant für korrespondierende Qualitätsmerkmale der Implementierung sind.

Softwarequalität

Neben der oben diskutierten ISO/IEC 9126 zur Softwareproduktqualität wurden eine Reihe weiterer generischer Qualitätsmodelle entwickelt, beispielsweise:

- Der IEEE Standard 1061 gibt die gleichen Qualitätsattribute wie die ISO/IEC9126 auf der obersten Ebene vor, unterscheidet aber auf der zweiten Ebene andere Qualitätskriterien. Der Schwerpunkt

der IEEE 1061 liegt auf der Bereitstellung von Metriken.
- Hewlett-Packard entwickelte 1987 ein ähnliches Qualitätsmodell, es unterteilt sich in die Merkmale Functionality, Usability, Reliability, Performance und Supportability, wobei diese ebenfalls weitere Untermerkmale besitzen [2].
- Bereits eines der ersten Qualitätsmodelle von McCall [37] betrachtet die Qualitätsattribute unter den Perspektiven Anwendung, Entwicklung und Evolution.

Bei der Softwarequalität werden Modelle nur indirekt adressiert, insoweit sie als Dokumentation des Produkts Rückschlüsse auf dessen Qualität zulassen.

Dieser Überblick über existierende Qualitätsattribute entlang des Softwareentwicklungsprozesses zeigt, dass der Zweck des Modells auch den Schwerpunkt der Qualitätsbetrachtungen bestimmt: Bei Informationsmodellen und Entwurfsmodellen steht der Nutzen des Modells für die folgenden Entwicklungsschritte im Vordergrund: Zum Teil werden sogar die konstituierenden Merkmale eines Modells zu Qualitätsattributen, wenn etwa Konstruktionsadäquanz, Verständlichkeit oder Effektivität gefordert wird. Auch Qualitätsattribute wie systematischer Aufbau, Einhaltung von Standards oder Übereinstimmung mit bewährten Entwurfsprinzipien zielen unmittelbar auf die Erfüllung der präskriptiven Aufgaben der Modelle im Entwicklungsprozess, aus der sich dann die Qualität der entwickelnden Software mittelbar ergibt.

Bei Architekturmodellen und Softwareproduktqualität dagegen adressieren die Qualitätsattribute in erster Linie die Qualitäten des Endprodukts Software, die prognostiziert oder exploriert wird. Ein Modell tritt „nur“ als adäquater Träger dieser Informationen in Erscheinung und wird selbst keiner Qualitätsbetrachtung unterzogen.

Arten von Qualitätsanforderungen an Modelle

Nachdem Modellqualität als eigenständiges Ziel erkannt ist, stellt sich nun die Frage, welche qualitativen Anforderungen an ein Modell zu stellen sind, um dem Gesamtziel, der qualitativ hochwertigen Software, zu genügen. Dabei können grundsätzlich Kategorien von Qualitätsanforderungen unterschieden werden:

Wir bezeichnen Qualitätsattribute des Modells als „innere Qualität“, wenn diese alleine für sich festgestellt werden können (die „innere Qualität“ ist somit kein Synonym zur „internen Qualität“ der ISO/IEC 9126:2001, alle hier definierten Qualitätsattribute zählen zur internen Qualität). Dazu gehören etwa ein übersichtliches Layout der Darstellung, zum Beispiel, dass Startzustände eines Statechart-Links oben angeordnet sind und dergleichen mehr.

Demgegenüber stehen Qualitätsattribute, die ein Modell mit einem anderen Betrachtungsgegenstand in Beziehung setzen. Zu diesen „relativen“ oder „äußeren Qualitätsanforderungen“ gehören Beziehungen zu den (informellen) Anforderungen der Nutzer, wie etwa Korrektheit des Modells oder in Bezug auf den darzustellenden Systemausschnitt auch die Vollständigkeit. Grundsätzlich sind diese Beziehungen auch zu dem im Entwicklungsprozess nachfolgenden, zu realisierenden System relevant. Allerdings wird etwa die korrekte und vollständige Realisierung des Systems primär als Qualitätsattribut des Systems und nicht des Modells betrachtet. Während also relative Qualitätsanforderungen formal als Relation zwischen Entwicklungsartefakten zu sehen sind, ist ihre praktische Auswirkung zumeist als Anforderung an die Qualität des im Entwicklungsprozess nachfolgenden Artefakts zu sehen. Diese zeitlich orientierte Sichtweise ist auch für Werkzeugketten bzw. die Ketten an Modellen einzusetzen, wenn in einem mehrstufigen Prozess Modelle aus Anforderungen verfeinert, diese in eine Architektur umgesetzt, zum Design detailliert, technische Aspekte hinzugefügt und schließlich zur Implementierung gebracht werden.

Dieser vertikalen Modellkette mit ihren Beziehungen steht in großen Softwaresystemen eine horizontale Zerlegung von Modellen in kleine, handhabbare Einheiten gegenüber. Dazu gehört etwa die Bildung mehrerer Datenmodelle der einzelnen Softwarekomponenten durch mehrere, nur an „Klebestellen“ zusammen zu setzende Klassendiagramme, oder die hierarchische Zerlegung von Architekturen. Diese horizontale Zerlegung erfordert weitere qualitative Anforderungen zwischen Modellen, wie etwa die Konsistenz der einzelnen Modelle. Demgegenüber ist die Vollständigkeit solcher Einzelmodelle im Allgemeinen kein Kriterium. Diese Form der im Sinn der Entwicklungsebene horizontalen Beziehungen zwischen Modellen kann verfeinert werden in homogene Beziehungen, etwa

zwischen Teilklassendiagrammen, und inhomogene Beziehungen, etwa zwischen Statecharts und Sequenzdiagrammen, die das Verhalten derselben Komponenten aus unterschiedlicher Sicht beschreiben. Darüber hinaus bestehen bei einer hierarchisch angelegten Notation, wie etwa den in [14] eingeführten Architekturmodellen, hierarchische Beziehungen zwischen denen, die im Entwicklungsprozess oft, wenn auch nicht zwingend, eine zeitliche Reihenfolge intendieren. Architektur kann zum Beispiel Top-Down oder Bottom-Up erstellt werden und wird zumeist evolutionär auf allen Ebenen parallel weiter entwickelt.

Es lässt sich zusammenfassend feststellen, dass Qualitätsanforderungen für Modelle in folgende Kategorien eingeteilt werden können, die die nachfolgende Struktur des Artikels bestimmen:

- Innere Qualität: Qualitätsanforderungen an ein einzelnes Modell.
- Äußere Qualität: Qualität mit Bezug auf andere Entwicklungsartefakte.
  - Horizontale Beziehungen: Qualität mit Bezug auf Nachbarmodelle.
    - Homogene Beziehung (zum selben Modelltyp).
    - Inhomogene Beziehung (zu anderem Modelltyp).
    - Hierarchische Komposition aus demselben Modelltypen.
  - Vertikale Beziehungen: Qualität mit Bezug auf den Vorgänger/Nachfolger im Entwicklungsprozess.
  - Externe Beziehungen: Qualität mit Bezug auf weitere Artefakte, bspw. Anforderungen, Systemumgebung.
  - Bezug zum Endprodukt, dem Softwaresystem.

Die eingeführte Kategorisierung von Qualitätsanforderungen für Modelle erlaubt uns nun, diese detaillierter zu diskutieren.

Innere Qualität

Darstellung

Die Darstellung ist für jede Art von Modell, insbesondere aber für ein grafisches Modell, ein für die Wahrnehmung und Akzeptanz durch den Benutzer entscheidendes Qualitätsmerkmal. Ein wichtiger Einsatzzweck von Modellen ist die Dokumentation und Kommunikation der getroffenen Entscheidungen – je besser die Darstellung, desto besser ist die Wahrnehmung und umso höher die Verständlichkeit des Systems. Zwar können mit einem Modell komplexe Sachverhalte verständlicher dargestellt werden, jedoch ist der Mensch in seinen Möglichkeiten der kognitiven Wahrnehmung begrenzt – ein überladenes, unübersichtliches Modell wird nicht verstanden, nicht akzeptiert, gegebenenfalls ignoriert und führt letztlich zu Fehlern in den auf die Modellbildung folgenden Aktivitäten. So ist zum Beispiel die Anzahl der Elemente begrenzt, die ein Mensch gleichzeitig erfassen kann.

Die Ästhetik eines Modells beeinflusst das Empfinden des Nutzers. Ein Modell, das durch den Benutzer als „ansprechend“ oder „schön“ betrachtet wird, weist eine höhere (subjektive) Gebrauchstauglichkeit auf. Obwohl die Ästhetik nicht vollständig objektivierbar ist, so lassen sich doch mithilfe allgemeingültiger Gestaltungsrichtlinien ästhetisch anspruchsvolle Modelle entwickeln.

Mithilfe des Modells sollen unterschiedliche Aspekte verdeutlicht werden. Durch eine hohe Verständlichkeit bzw. eine gute Übersichtlichkeit des Modells lassen sich die Aspekte schneller und einfacher begreifen. Enthält bspw. ein Klassenmodell zu viele unterschiedliche Klassen, so fällt es dem Benutzer schwer, die wichtigen Aspekte des Modells schnell zu begreifen. Ebenso führt ein „chaotisches“ Modell zu Verwirrung beim Benutzer. Durch ein übersichtliches und strukturiertes Modell (d.h. ein Modell mit einem guten Layout) können leicht die wichtigen Aspekte hervorgehoben werden.

Präzision

Die Präzision beschreibt, inwieweit das Modell die für den aktuellen Zweck relevanten Merkmale der Umgebung, der Anforderungen oder des zu realisierenden Systems widerspiegelt. Präzision adressiert die Güte der Verkürzung, die bei der Modellbildung stattfindet: In einem präzisen Modell sind die ausgelassenen Attribute des Originals für die aktuelle Entwicklungsphase unbedeutend. Präzision ist nicht zu verwechseln mit Detailreichtum oder Formalität: So lässt sich Interaktion auf der Anwendungsebene einfach und präzise als synchrone Kommunikation modellieren, sofern die darunter liegende Kommunikationsarchitektur entsprechende Dienste

anbietet. Von den möglicherweise komplexen technischen Details kann im Entwurfsmodell der Anwendung abstrahiert werden. Andernfalls wurden für den Entwurf wesentliche Verhaltenseigenschaften durch die Modellierung unzulässig vereinfacht.

Universalität

Die Universalität des Modells gibt an, inwieweit sich das Modell auf die für die aktuelle Entwicklungsphase notwendigen Festlegungen fokussiert, oder aber darüber hinaus Eigenschaften hinzugefügt werden, die eine unnötig komplexe oder eingeschränkte und damit letztlich teure Lösung induzieren, oder die spätere Weiterentwicklung und Wiederverwendung erschweren. Ein wichtiger Schritt zur Verbesserung der Universalität von Modellen ist Model-Driven-Development, bei dem bewusst zwischen plattformunabhängigem, fachlichen Entwurf und technischen, plattformabhängigen Modellen getrennt wird.

Einfachheit

Ein wichtiges Qualitätsattribut ist die Einfachheit: Ein Modell sollte nicht komplexer sein, als die dargestellte Materie es verlangt. Einfachheit in der Darstellung kann oft durch Reformulierung und Restrukturierung semantikerhaltend (also ohne den Informationsgehalt zu verändern) erreicht werden. Deshalb betrachten wir die Einfachheit des Modells als eine Anforderung an die innere Qualität eines Modells, ohne den Bezug zu den darzustellenden Inhalten zu fordern.

Da im Modell die für einen gewissen Darstellungskontext unwichtigen Details weggelassen werden können, um von einem komplexen Zusammenhang zu abstrahieren und diesen verständlich zu machen, ist eine zweite Interpretation des Qualitätsattributs Einfachheit mit dem Zweck des Modells und dem darzustellenden Original bzw. System gekoppelt. Zu beachten ist, dass ein einfaches Modell – wie oben beschrieben – gleichzeitig präzise sein kann, d.h. dass Einfachheit nicht bedeutet, dass Präzision aufgegeben werden muss.

Semantische Adäquatheit

Durch geeignete Interpretation ist es möglich, denselben Sachverhalt in verschiedenen Modellierungssprachen darzustellen. So kann etwa eine Kombination aus Klassen- und Objektdiagramm eingesetzt werden, um ein Zustandsmodell zu beschreiben. Das Entwurfsmuster „State“ schlägt zur Implementierung eines Zustandsautomaten genau das vor. Adäquat modelliert ist dies aber nicht. Genau so ist es oft unsinnig, zustandslose Komponenten mit Statecharts zu modellieren oder Entwurfsmuster unangemessen einzusetzen. Die semantische Adäquatheit des Modells für den beabsichtigten Zweck spielt dementsprechend für die Qualität eines Modells eine wesentliche Rolle.

Konsistenz

Die Konsistenz gibt an, in welchem Maß das Modell in sich konsistent ist. Eine Sammlung von meist syntaktischen Konsistenzbedingungen legt zum Beispiel fest, dass Variablen vor ihrer Nutzung einzuführen sind und ihre Typisierung passen muss. Konsistenzbedingungen, die innerhalb eines Modells geprüft werden können, heißen auch Intramodellkonsistenz. Für die Wiederverwendbarkeit und Modularität eines Modells ist es günstig, möglichst viele Konsistenzbedingungen innerhalb des Modells zu klären. Konsistenzbedingungen, die über mehrere Modelle hinweg zu klären sind, werden Intermodellkonsistenz genannt. Sie treten etwa dann auf, wenn die Strukturmodelle Attribute, Speicherplätze oder Schnittstellen beinhalten, auf die in den Verhaltensmodellen Bezug genommen wird.

Konzeptionelle Integrität/Uniformität

Um eine einheitliche, gleichförmige und systematisch realisierbare Software zu erreichen, sollte versucht werden, wiederholt die gleichen Regeln, Muster und Prinzipien in kohärenter Form anzuwenden. In diesem Zusammenhang spricht man auch von konzeptioneller Integrität oder Uniformität („Gleiches wird gleich gelöst“). Ähnlich wie die Konsistenz ist die konzeptionelle Integrität aber oft auch modellübergreifend zu verstehen, da Einheitlichkeit über alle Dokumente eines Projekts erreicht werden sollte.

Konformität

Die Konformität ist ein Maß für die Verwendung von Standards und Normen im Modell. Konformität wird in Bezug auf genormte und standardisierte Methoden, Muster, Prinzipien oder Sprachen festgelegt.

Die bereits diskutierte konzeptionelle Integrität kann eine Konsequenz der Konformität sein, wenn diese einen genügend engen Rahmen und Freiheiten vorgibt. Allerdings sind Normen und Standards gerade so gewählt, dass eher viele Freiheiten gelassen werden und für eine konzeptuelle Integrität zusätzliche, meist projektspezifische Festlegungen zu treffen sind.

Sprachspezifische, semantische Qualitätsanforderungen

Weitere Qualitätsanforderungen können festgelegt werden, wenn die Modellierungssprache spezifische, meist semantisch zu verifizierende Kriterien anbietet. Dazu gehört zum Beispiel die Vollständigkeit eines Statechart/Automaten, mit der gesichert wird, dass für die modellierte Komponente in jedem Zustand zu jeder möglichen Eingabenachricht eine Reaktion existiert. Petrinetze bieten mit ihrer ausgefeilten Theorie zu Lebendigkeit etc. eine Sammlung an semantischen Qualitätsanforderungen, die allein auf Basis des Modells entschieden werden können.

Äußere Qualität (Horizontale Beziehungen)

Viele Qualitätsanforderungen gelten als Relationen zwischen Modellen. Gemäß unserer Einteilung werden in diesem Abschnitt Qualitätsanforderungen definiert, die zwischen Modellen innerhalb einer Entwicklungsphase relevant sind. Diese Modelle ergänzen sich in ihren Inhalten zu einer Gesamtinformation über das zu realisierende System, decken jedoch nur einzelne Phasen im Entwicklungsprozess ab.

Konsistenz, konzeptionelle Integrität, sprachspezifische, semantische Qualitätsanforderungen

Die bereits oben genannten, innerhalb eines Modells relevanten Qualitätsattribute Konsistenz, konzeptionelle Integrität und viele sprachspezifische Qualitätsanforderungen werden erst bei der Betrachtung der Beziehungen zwischen Modellen formulierbar. Als zusätzliche Komplexitätsstufe entsteht hier die Definition von Schnittstellen, die Sichtbarkeiten beschreibt, und es geschieht das Management (import, export, hiding) und Durchreichen von benannten Elementen dieser Schnittstellen.

Vollständigkeit nach unten

Im nachfolgenden Abschnitt wird die „Vollständigkeit nach oben“ als die vollständige Umsetzung der in der vorhergehenden Phase der Modellkette vorhandenen Information eingefordert. Die „Vollständigkeit nach unten“ unterscheidet sich demgegenüber, weil sie fordert, dass ein Modell alle notwendigen Informationen enthält, um die Artefakte des nächsten Glieds der Modellkette abzuleiten oder zu generieren. Dies ist eine semantische Qualitätsanforderung, die nur die Modelle einer Schicht der Modellkette betrifft und zu deren Prüfung kein Bezug zu den zum Prüfzeitpunkt noch nicht vorhandenen implementierungsnäheren Modellen sein kann und darf. Dementsprechend wird diese Form der Vollständigkeit als horizontale Qualitätsanforderung betrachtet.

Kohäsion

Kohäsion bedeutet, dass ein inhaltlich oder technisch zusammenhängender Aspekt in einem eng zusammenhängenden Teil des Modells behandelt wird. Kohäsion ist ein Maß dessen, wie viele Modellteile zu verstehen sind, um einen Aspekt durchdrungen zu haben bzw. wie verteilt sich eine Änderung auswirkt. Je kohäsiver eine Systemmodellierung einen Aspekt darstellt, umso einfacher ist dieser im Systemkontext zu verstehen. Der Begriff „Aspekt“ wird zwar in diesem Kontext nicht im Sinne der Aspektorientierung verstanden, aber es zeigt sich, dass eine konsequente, auf Aspekte hin orientierte Modellierung des Systems durch geeignete Organisation der Architektur und durch Konzentration technischer Aspekte in entsprechende Module die Kohäsion der Modellierung stark erhöhen kann, auch ohne aspektorientierte Modellierung einzusetzen [50]. Die Kohäsion ist somit eng mit der Frage der Kopplung bzw. Kapselung der verschiedenen Aspekte verbunden.

Modularisierung

Das Pendant zur Kohäsion ist die Modularisierung inhaltlich oder technisch unabhängiger Aspekte. Modularisierung ist die Grundlage dafür, dass unabhängige Teile auch unabhängig voneinander entwickelt, erweitert und wieder verwendet werden können und ein wesentliches Mittel zur Beherrschung der Komplexität. Als Modellqualität verstehen wir unter Modularisierung die Forderung, dass ein Modellelement oder Teilmodell auch nur

einen Aspekt behandelt. Unter Modularisierung fällt beispielsweise die Anforderung, dass eine Klasse eine einzige Verantwortlichkeit haben sollte, oder dass bei einer zustandsbasierten Modellierung die Betriebsmodi des Systems die Strukturierung auf den oberen Hierarchieebenen determinieren.

Redundanzfreiheit

Eine Qualitätsanforderung zwischen Modellen kann die möglichst redundanzfreie Darstellung von Informationen sein. Redundanzfreiheit ist für die evolutionäre Weiterentwicklung wichtig, weil in einer Anpassungsphase redundante Darstellungen desselben Sachverhalts nicht notwendigerweise konsistent modifiziert werden. Dies kann schnell zu Inkonsistenzen führen. Völlige Redundanzfreiheit wird aber in einer Welt heterogener Modellierungssprachen, die Sichten in einzelne Artefakte zerlegen und per Namen zu einem Gesamtmodell kombinieren, nicht zu erlangen sein.

Kontrollierte Redundanz

Andererseits ist die kontrollierte Modellierung redundanter Informationen ein wesentliches Mittel zur Qualitätssicherung. So können verschiedene Sichten auf einen Sachverhalt dazu genutzt werden, sowohl generativ/konstruktiv das lauffähige System zu entwickeln, als auch Testfälle dafür zu generieren [46, 47]. Beispielsweise können Statecharts für die Codegenerierung und Sequenzdiagramme für Testfälle eingesetzt werden. In diesem Fall ist eine kontrollierte Redundanz erwünscht. Sie lässt sich sogar messen in Form einer Testfallüberdeckung für den generierten Code bzw. die Zustände und Transitionen des Statecharts.

Andere Formen kontrollierter Redundanz werden genutzt, um Informationen für verschiedene Entwickler- und Testgruppen aufzubereiten und in deren spezifisch nutzbarer Sicht darzustellen oder Anforderungen aus verschiedenen Blickwinkeln zu betrachten und in Reviews und Inspektionen die Konsistenz als qualitätssichernde Maßnahme manuell zu sichern.

Äußere Qualität (Vertikale Beziehungen)

Die Qualitätsanforderungen eines Modells können auch im Bezug auf andere Entwicklungsartefakte definiert werden, die in der Entwicklungsgeschichte Vorgänger bzw. Nachfolger des Modells sind. Frühe Analysemodelle haben als Vorgänger typischerweise nur die informellen Anforderungen, die etwa in einem Lastenheft dokumentiert sein können. In der vertikalen Modellkette folgt nach dem Feindesign schließlich der Code. Jedes Modell einer Modellkette ersetzt das vorhergehende, indem es die dort vorhandenen Informationen in eine implementierungsnähere Form umsetzt und durch in Entwurfsentscheidungen getroffenen Details ergänzt. Wesentliche in dieser Modellkette zu identifizierende Qualitätsanforderungen sind:

Korrektheit

Korrektheit bedeutet, dass das Modell die Anforderungen (gemäß in der Modellkette vorhergehenden Artefakte) präzise abbildet, d.h. dass alle Strukturen, Funktionen und Ausnahmen des Systems abgebildet sind. Diese Form der Korrektheit wird auch als semantische Korrektheit verstanden, gelegentlich aber auch als „phasenübergreifende Konsistenz“ bezeichnet. Ihr gegenüber steht die oben diskutierte Konsistenz als syntaktische Korrektheit des Modells innerhalb der Regeln einer Modellierungssprache. Sie verlangt, dass das Modell die Syntax der eingesetzten Sprache (z.B. UML) korrekt verwendet. Oft wird hierfür auch der Begriff „statische Semantik“ verwendet. Demgegenüber beinhaltet die (semantische) Korrektheit sowohl funktionale Korrektheit als auch die korrekte Abbildung nichtfunktionaler Anforderungen. Zwischen syntaktisch überprüfbarer Konsistenz und semantischer Korrektheit kann nicht immer eindeutig unterschieden werden. Beispielsweise ist Terminierung zunächst eine Anforderung, also eine Korrektheitseigenschaft. Durch geschickte Wahl der Modellierungssprache können evtl. aber hinreichende, syntaktisch prüfbare Konsistenzbedingungen gefunden werden, sodass sich eine Korrektheitseigenschaft des Modells durch eine statisch prüfbare Konsistenzeigenschaft ersetzen lässt.

Korrektheit nach unten heißt, alle Entscheidungen einer Modellierungsebene werden in der nachfolgenden Schicht korrekt umgesetzt. Während die Abbildung funktionaler Anforderungen relativ gut verstanden ist, ist das Zerlegen und Umbauen nichtfunktionaler Anforderungen bis zur Implementierung und die Absicherung ihrer Korrektheit noch weitgehend Forschungsgegenstand. Insbesondere fehlen auch noch formalere Darstellungsmittel für nichtfunktionale Anforderungen in

implementierungsnäheren Artefakten. So können etwa zeitliche Vorgaben im Feindesign oder Code zumeist nicht wiedergegeben werden und entziehen sich damit zumeist einer Prüfung auf Korrektheit.

Vollständigkeit nach oben

Verwandt mit der korrekten Abbildung der Anforderungen im Modell ist die Frage nach der Vollständigkeit der Abbildung aller Anforderungen. Da implementierungsnähere Modelle im Allgemeinen eine andere Sicht des Systems darstellen und einen wesentlich höheren Detaillierungsgrad beinhalten, werden zum Beispiel die Informationen eines Anforderungsmodells zumeist in mehrere Entwurfsmsodelle aufgeteilt. Umgekehrt stammen die in einem Entwurfsmodell zusammengefügten Informationen ggf. aus mehreren Anforderungsmodellen. Daher ist die Vollständigkeit der Abbildung aller Modellinformationen zwischen zwei Ebenen keine binäre Beziehung, sondern erfordert zumeist die Betrachtung der Gesamtheit aller beteiligten Artefakte der beiden Ebenen. Praktische Erfahrungen zeigen, dass von der Architektur über den Grob- und Feinentwurf zur Implementierung häufig eine relativ schmale, wenn nicht eine 1:1-Beziehung zwischen den realisierenden Artefakten und den realisierten Artefakten der Modellkette besteht. Demgegenüber ist die Verwebung von der Anforderungsbeschreibung zur ersten Spezifikation/Architektur sehr breit. Das heißt, hier ist die Vollständigkeit nach oben nur durch eine weitgehend globalisierte Betrachtung zu erkennen.

Verfolgbarkeit

Verfolgbarkeit ist ein Maß für die Nachvollziehbarkeit, welche Informationen von einem Modell an welchen Stellen zur Modellbildung in einer nachfolgenden Entwicklungsaktivität eingesetzt wurden. Verfolgbarkeit ist damit eine phasenübergreifende Beziehung zwischen Modellen. Mittels der Verfolgbarkeit lassen sich notwendige Änderungen zwischen den Artefakten verfolgen und somit Aussagen bzgl. der Auswirkungen und des Aufwands von Änderungen treffen. Beispielsweise kann bestimmt werden, welche Auswirkungen Änderungen im Modell für die nachfolgenden Artefakte bedeuten. Verfolgbarkeit ist auch eine Grundlage für die Messung der als eigenständiges, aber verwandtes Qualitätsattribut identifizierte „Vollständigkeit nach oben“.

Änderbarkeit

Änderbarkeit von Modellen ist in der heutigen, schnelllebigen Welt wichtig, um eine evolutionäre Weiterentwicklung des Systems zur Anpassung an neue Geschäftsziele und sich daraus ergebende Anforderungen und neue Technologien zu ermöglichen. Bei softwareintensiven Produkten spielt die Änderbarkeit der Software vor allem dann eine Rolle, wenn eine weiterentwickelbare Produktlinie langfristig den Geschäftserfolg sichert. Änderbarkeit von Software bedeutet immer auch Änderbarkeit der Modelle, die diese Software beschreiben. Änderbarkeit lässt sich in die Aspekte Wartbarkeit, Erweiterbarkeit und Wiederverwendbarkeit unterteilen.

Die Wartbarkeit ist der Aufwand, der betrieben werden muss, um im Modell Fehler zu beheben, sowie Spezifikationsänderungen und -anpassungen zu modellieren. Die Erweiterbarkeit ist ein Maß für die Leichtigkeit, mit dem das System erweitert werden kann. Unsere Erfahrung zeigt, dass anders als bei vielen Qualitätsattributen, die Erweiterbarkeit des Systems nicht notwendigerweise mit der Erweiterbarkeit der beschreibenden Modelle korreliert. Diese Korrelation ist nur dann gegeben, wenn das System aus den Modellen generiert wird. Wiederverwendbarkeit ist die Eigenschaft des Modells, für andere Systeme oder andere Varianten des Systems in einer Produktlinie wiederverwendet werden zu können. Durch Wiederverwendbarkeit sind im neuen System weniger Fehler zu erwarten, da das ursprüngliche Modell bereits qualitätsgesichert wurde. Dadurch lassen sich deutliche Aufwände für die Modellierung und Qualitätssicherung einsparen.

Weitere Qualitätsmerkmale

Der bisher diskutierte Katalog von Qualitätsmerkmalen erhebt nicht den Anspruch auf Vollständigkeit, sondern soll und muss an projekt- und unternehmensspezifische Anforderungen angepasst werden. Weitere Kriterien können etwa die Transformierbarkeit oder die Prüfbarkeit sein. Transformierbarkeit des Modells einer Sprache in ein implementierungsnäheres Modell einer anderen Sprache ist zunächst eine Eigenschaft der Sprache, kann aber als Modelleigenschaft verstanden werden, wenn die Transformation auf einem bestimmten Sprachstil beruht. Oft können Transformatoren nicht alle Sprachelemente bzw. alle Kombinationen von Sprachelementen übersetzen.

Wie bei den Qualitätsmodellen schon ausgeführt, können nicht alle Qualitätsmerkmale gleich gut erfüllt werden, sondern es sind Priorisierungen, etwa über Gewichtungen, notwendig. Teilweise widersprechen sich Qualitätsmerkmale auch. Ebenso sind die einzelnen Merkmale nicht notwendigerweise gegeneinander abgrenzbar, sondern können sich bedingen oder ergänzen.

Qualitätsanforderungen an eine Modellierungsnotation

Eine Reihe von Anforderungen an einzelne Modelle bzw. an miteinander in horizontaler oder vertikaler Beziehung stehenden Modellen steht in direktem Zusammenhang mit den Sprachelementen, die eine Modellierungssprache überhaupt anbietet und den werkzeugbasierten oder manuellen Techniken, die für die Sprache zur Verfügung stehen.

Dinge, die sich nicht ausdrücken lassen, können auch nicht analysiert oder entwickelt werden. Während für Programmiersprachen mittlerweile ein gutes Verständnis darüber herrscht, welche Konzepte eine Sprache im Kern und für eine Modularisierung braucht sowie welche Konzepte in eine Bibliothek, ein Framework oder in vorgefertigte Bausteine auszulagern sind, ist dies für Modellierungssprachen immer noch nicht gesichert. Eine Betrachtung der UML zeigt, dass diese ein reiches Spektrum an Einzelnotationen für unterschiedliche Phasen und Sichten zur Verfügung stellt. Jedoch werden teilweise auch dieselben Notationen für unterschiedliche Zwecke eingesetzt. Allen voran kann das Klassendiagramm in der Analyse Gegenstände der realen Welt darstellen oder in der Implementierung Klassen des realisierten Systems. Häufiger werden Klassen auch zur statischen (!) Modellierung von Vorgängen eingesetzt. Solche „Objektifizierung" findet bspw. statt, wenn Zustandsautomaten mit dem bekannten State-Muster als Klassen repräsentiert werden oder Vorgänge durch „Sachbearbeiter" [17] oder Webservices als zustandslose Vorgangsobjekte umgesetzt werden.

Speziell für Modellierungssprachen ist aber die Frage nach den richtigen, in der Sprache eingesetzten Konzepten und ihrer Semantik nicht gelöst. Gegenüber Programmiersprachen ist vor allem das Konzept der gewollten, kontrollierten Unterspezifikation nur wenig beherrscht. Oft wird sogar die Möglichkeit zur Unterspezifikation mit der Ungenauigkeit der Bedeutung der Ausdrucksmittel einer Sprache verwechselt und Formalität mit eindeutiger, ausführbarer und damit nicht zur Unterspezifikation fähiger Modellierungssprachen gleichgesetzt.

Viele der wesentlichsten Qualitätsanforderungen für eine Modellierungssprache lassen sich aus den Anforderungen für einzelne Modelle sowie den horizontalen und vertikalen Beziehungen zwischen Modellen ableiten. Dazu gehören etwa die Möglichkeit zur Vollständigkeit nach unten und oben, zur redundanzfreien bzw. kontrolliert redundanten Darstellung, zur Sicherung der Konsistenz, Kohärenz, Korrektheit, Verfolgbarkeit von Entscheidungen, etc. Zusätzlicher Beachtung bedürfen lediglich folgende Qualitätsattribute:

Formalisierungsgrad

Eine Modellierungssprache kann vollständig formalisiert sein, halbformal oder völlig informell nutzbar sein. Typischerweise steigt der Grad der notwendigen Formalität einer Sprache in den späten Phasen eines Projekts an. Es ist also in Ordnung, Anforderungen in informellem Text, spezielle Algorithmen in Pseudo-Code oder Protokolle durch informelle Abläufe zu erklären und deren Strukturierung durch Use-Cases- oder Pseudo-Architekturbilder vorzunehmen. In der vertikalen Modellkette sollte aber der Grad der Formalisierung der Sprache nur ansteigen – nie kleiner werden.

Nur bei adäquater Formalisierung können Analyse, Simulation und Generierung genutzt sowie Tracing von Modellbeziehungen und Anforderungen etabliert werden.

Adäquatheit für die Anwendungsdomäne

Sprachen wie die UML und ihre Teilsprachen sind dazu gedacht, allgemeingültig zu sein. Die aufkommenden Domain Specific Languages (DSLs) [16] werden häufig ebenfalls zur Softwareentwicklung herangezogen. Sie bilden eine Anwendungsdomäne, die wesentlich spezifischer durch zur Verfügung stehende Sprachkonstrukte ist, ab. Dementsprechend ist die Adäquatheit der Modellierungssprache für die Anwendungsdomäne wichtig, da sonst umständliche Modellbildungen oder nicht darstellbare Sachverhalte die Folge sind.

Teilaspekte sind hierbei die Verständlichkeit der Sprache für den Domänenexperten und den SE-Entwickler, die Kompaktheit der Modellierungssprache für eine Anwendungsdomäne, die Angemessenheit der Modellierungselemente in

Bezug auf Komplexität, Parametrisierbarkeit, etc. sowie die Fähigkeit zur Repräsentation von Domänenwissen, Mustern oder Rahmenbedingungen für den Einsatz. Abhängig von der Domäne müssen ggf. sowohl funktionale als auch nichtfunktionale Eigenschaften dargestellt werden können.

Bewertung von Qualitätsanforderungen an Modelle

Während die Qualitätsanforderungen an eine Sprache typischerweise nicht innerhalb eines Projekts, sondern a priori entschieden werden, ist es dringend geboten, die Qualitätsanforderungen an einzelne Modelle effektiv bewerten zu können. Diese „Meta-"Qualitätsanforderung gilt gleichzeitig für die gewählte Modellierungsnotation und die zur Verfügung stehende Werkzeuginfrastruktur. Ideal wären quantifizierte, automatisierte Verfahren zur Qualitätsbewertung, die aber bisher nur für einzelne Qualitätsanforderungen in der Praxis umgesetzt werden: Für die Qualitätsanforderung Darstellung bieten Ansätze wie [30] dem Entwickler die Möglichkeit, Modellierungsregeln individuell festzulegen und automatisiert zu überprüfen. Für die Überprüfung der Korrektheit für funktionale Anforderungen bieten sich formale Verifikationsmethoden wie Model-Checking oder Theorembeweisen an, die aber ein hinreichend formales Entwurfsmodell und formalisierte Anforderungen voraussetzen. Auch für laufzeitrelevante, nichtfunktionale Anforderungen wie Performanz, Echtzeitverhalten oder Energieverbrauch gibt es inzwischen Werkzeugunterstützung für die Modellanalyse [28], die über eine Modelltransformation an die Entwurfsmodellierung angebunden werden können [27]. Viele Qualitätskriterien für Modelle sind jedoch sehr schlecht direkt messbar und benötigen eine Ersatzmessung, eine relative Messung über die Entwicklungszeit oder qualitative Befragungen, um Rückschlüsse auf das eigentliche Qualitätskriterium zu erhalten. Dazu gehören insbesondere herstellungsorientierte Qualitätsanforderungen mit einer sehr langfristigen Perspektive wie Wartbarkeit, Flexibilität oder Wiederverwendbarkeit.

Jedoch hängt der Automatisierungsgrad von Messkriterien sehr stark von der Form der Sprachkonzepte ab. So sind etwa Erfüllbarkeit von Constraints oder die Vollständigkeit von Automaten mit Vorbedingungen mit Formeln der Logik erster Stufe grundsätzlich nicht entscheidbar. Der Nachteil einer Restriktion der Sprache und damit des Modellierungskomforts kann durch die automatisierte Prüfbarkeit bzw. konstruktive Umsetzbarkeit auch nichtfunktionaler Anforderungen mehr als aufgewogen sein, wie das Werkzeug Alloy [33] oder synchrone Sprachen [38] zeigen.

Über die genannten Qualitätskriterien hinaus gibt es für Modellierungssprachen eine Anzahl weiterer, oft nur projektspezifisch einzusetzender Kriterien. Als allgemein gültiges Metakriterium kann die für eine Notation zur Verfügung stehende Werkzeugunterstützung gesehen werden. Komfortable beziehungsweise effektive Werkzeugunterstützung ist ggf. für verschiedenste Aufgaben notwendig. Dazu gehören Editor, geeignete Versionskontrollmechanismen, die auch parallele Änderungen verschmelzen können, Analysatoren einfacher und komplexer Kontextbedingungen, die Analyse in Bezug auf spezielle (meist nichtfunktionale) Anforderungen, ggf. Techniken zur Validierung und Verifikation bis hin zu Infrastrukturen zu Simulation und Codegenerierung. Diese Werkzeuge müssen sowohl in einer Modellkette als auch in einer heterogenen Sammlung horizontal in Beziehung stehender Modelle kompatibel sein. Diese Metaqualitätsanforderungen haben nicht direkt mit den Modellen zu tun und sind deshalb in diesem Artikel nicht weiter vertieft. Es ist jedoch evident, dass die Qualität von Modellen sich erst dann auf die Qualität der letztendlich zu entwickelnden Software überträgt, wenn die qualitativ hochwertigen Modelle durch Werkzeuge adäquat umgesetzt werden. Dementsprechend ist eine adäquate Werkzeuginfrastruktur essenziell für die sinnvolle Nutzung von Modellen im Softwareentwicklungsprozess.

Zusammenfassung

Dieser Artikel gibt einen Überblick über den derzeitigen Stand zur Frage, was Modellqualität ist und wie viel wir über die Beziehung zwischen Modellqualität und Softwarequalität wissen. Allgemein anerkannt ist, dass adäquate Modellbildung und Analyse der Fähigkeiten des Modells sowie die Extrapolation dieser Eigenschaften auf das zu bildende Softwaresystem einen deutlichen Qualitätsvorteil und Planungssicherheit liefern können.

Weit unklarer ist die Antwort auf die Frage, wie viel Modellbildung unter Nutzung welcher Modellierungssprachen für welche Arten von Projekten adäquat ist. Diese Frage wird in der Praxis

gerade intensiv diskutiert, während sich die Wissenschaft einer empirischen oder analytischen Klärung dieser Frage nur zaghaft nähert. Eine empirische Klärung erscheint aufgrund der äußeren Rahmenbedingungen (viele Einflusskriterien und dadurch unvergleichbare komplexe Softwareentwicklungsprojekte) schwer und aufgrund der immer noch hohen Veränderlichkeit der im Einsatz befindlichen Sprachen auch von keinem dauerhaften Wert.

Dieser Artikel verzichtet aufgrund der vielen höchst unterschiedlichen Modelltypen und ihrer Einsatzformen notwendigerweise auf eine detaillierte und vollständige Auflistung von potenziellen Qualitätskriterien. Viele davon wären auch nur sehr spezifisch für bestimmte Notationen zugänglich. So sind in grafischen Modellen die zentralen Elemente an den Lesebeginn zu legen, der sich jedoch z.B. bei Automaten links oben und bei Klassendiagrammen leicht oberhalb der Mitte befindet (weil mehr Subklassen unterhalb als Oberklassen darüber liegen). Viele weitere solcher Kriterien sind für jede Sprache einzeln zu identifizieren. Gegebenenfalls ändert sich mit der Expertise der Nutzer auch die Ästhetik. All diese Dinge sind empirisch nur wenig bis nicht untersucht.

Dennoch werden weiter Untersuchungen zur Modellqualität vorgenommen und insbesondere allgemeine, parametrisierbare Messwerkzeuge geschaffen, die individuell einstellbar verschiedene Messungen vornehmen können. Wie schon DeMarco ausführte, verändert die Messung das Verhalten des Projektmitglieds. Deshalb sind Messungen vorsichtig und zielgerichtet in ein Entwicklungsprojekt zu integrieren.

Obwohl wir, wissenschaftlich gesehen, nur Indizien haben, welche Modellqualitätskriterien SW-Qualität implizieren und welche zu mehr Effektivität und/oder Prozessqualität beitragen, halten wir derzeit die verstärkte und zielgerichtete Messung zumindest für größere und geschäfts- oder sicherheitskritische Softwaresysteme für gerechtfertigt. Einzelne Anwendungsdomänen, wie etwa die Flugzeug- und die Automobilindustrie [21] haben damit bereits größere Erfolge erzielen können.

Wir wiederholen daher die von zahlreichen Indikatoren unterstützte Hypothese, dass zwischen Modellqualität und Softwarequalität ein deutlicher Zusammenhang existiert. Welche Qualitätseigenschaften sich in welcher Form von Modell zum Produkt übertragen und welche dabei besonders relevant sind, muss allerdings eine größere empirische Wissensbasis noch präzisieren. Diese zu erstellen, wird in den nächsten Jahren eine wesentliche Aufgabe der Wissenschaft in Zusammenarbeit mit der modellbasiert entwickelnden Industrie sein.

Parallel dazu ist der Werkzeugbau, also die automatisierbare Messung und Bewertung von Qualitätsattributen sowie die Entwicklung geeigneter Ersatzmessungen von nicht direkt messbaren Qualitätskriterien voran zu bringen.

Danksagung

Wir danken Ruth Breu und Andy Schürr sehr herzlich für ihre konstruktiven Anmerkungen.

Literatur

1. Ambler SW (2003) The elements of UML style. Cambridge University Press, Cambridge
2. Balzert H (1998) Lehrbuch der Software-Technik – Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung. Spektrum Akademischer Verlag, Heidelberg
3. Basili VR, Rombach DH (1988) The tame project: Towards improvement-oriented software environments. IEEE Trans Softw Eng 10(11):758–773
4. Bass L, Clements P, Kazman R (2003) Software architecture in practice, 2nd edn. Addison-Wesley, Boston
5. Basili V, Caldiera G, Rombach D (1994) Goal/Question/Metric Paradigm. In: Marciniak JJ (ed) Encyclopedia of Software Engineering, Volume I. John Wiley & Sons, New York, S 528–532
6. Bansiya J, Davis CG (2002) A hierarchical model for object-oriented design quality assessment. IEEE Trans Softw Eng 28(1):4–17
7. Baker P, Dai Z, Grabowski J, Haugen O, Schieferdecker I, Williams C (2007) Model-driven Testing. Using the UML testing profile. Springer, Berlin
8. Beucher O (2006) MATLAB und Simulink. Pearson Studium, München
9. Briand LC, Morasca S, Basili VR (1999) Defining and validating measures for object-based high-level design. IEEE Trans Softw Eng 25(5):722–743
10. Briand LC, Morasca S, Basili VR (2002) An operational process for goal-driven definition of measures. IEEE Trans Softw Eng 28(12):1106–1123
11. Buschmann F, Meunier R, Rohnert H, Sommerlad P, Stal M (1998) Patternorientierte Softwarearchitektur. Addison-Wesley, München
12. Bosch J (2000) Design & use of software architectures. Addison-Wesley, Oxford
13. vom Brocke J (2003) Referenzmodellierung. Gestaltung und Verteilung von Konstruktionsprozessen. Logos, Berlin
14. Broy M, Rumpe B (2007) Modulare hierarchische Modellierung als Grundlage der Software- und Systementwicklung. Informatik-Spektrum 30(1):3–18
15. Becker J, Rosemann M, Schütte R (1995) Grundsätze ordnungsgemäßer Modellierung. Wirtschaftsinformatik 37:435–445
16. Czarnecki K, Eisenecker U (2002) Generative Programmierung. Addison-Wesley, München
17. Denert E, Siedersleben J (1992) Software Engineering. Methodische Projektabwicklung. Springer, Berlin
18. Dobrica L, Niemelä E (2002) A survey on software architecture analysis methods. IEEE Trans Softw Eng 28(7):638–653
19. EN 61508 (2000) Funktionale Sicherheit sicherheitsbezogener elektrischer/elektronischer/programmierbarer elektronischer Systeme. Teil 1–7, IEC 61508
20. Florentz B, Huhn M (2007) Architecture potential analysis. A closer look inside architecture evaluation. J Softw 2(4):43–56
21. Farkas T, Leicher A, Röbig H, Born M, Klein T, Zander-Nowicka J (2006) Werkzeugübergreifende Konsistenzsicherung von Artefakten bei der Entwicklung softwarebasierter Systeme im Automobil. In: Hochberger C, Liskowsky R (Hrsg) INFORMATIK 2006, 2.–6. Oktober 2006, Dresden. Lecture Notes in Informatics (LNI), GI-Edition
22. Frank U, Rumpe B (2006) Qualität konzeptueller Modelle. Workshop Modellierung 2006, 21.–24. März 2006, Innsbruck

23. France R, Rumpe B (2007) Model-driven development of complex software: A research roadmap. In: Future of Software Engineering 2007 at ICSE, 26.–27. Mai 2006, Minneapolis, S 37–54
24. Frankel D (2003) Model driven architecture. Wiley Publishing, Indianapolis
25. Grönniger H, Krahn H, Rumpe B, Schindler M (2006) Integration von Modellen in einen codebasierten Softwareentwicklungsprozess. In: Mayr HC, Breu R (Hrsg) Modellierung 2006, 22.–24. März 2006, Innsbruck. Lecture Notes in Informatics (LNI), GI-Edition
26. Grönniger H, Krahn H, Rumpe B, Schindler M, Völkel S (2006) MontiCore 1.0 – Ein Framework zur Erstellung und Verarbeitung domänenspezifischer Sprachen. Informatik-Bericht 2006-04, Technische Universität Braunschweig
27. Hagner M, Huhn M (2007) Modellierung und Analyse von Zeitanforderungen basierend auf der UML. In: 5. GI-Workshop Automotive Software Engineering, GI-Jahrestagung 2007, 27. September 2007, Bremen
28. Henia R, Hamann A, Jersak M, Racu R, Richter K, Ernst R (2005) System level performance analysis the SymTA/S approach. IEE Proc Comp Digit Tech 152(2):148–166
29. Harel D, Rumpe B (2004) Meaningful modeling: What's the semantics of „Semantics"? Computer 37(10):64–72
30. Huhn M, Braam J-C, Harms M, Horstmann M, Mutz M (2005) Structured hardware/software development for enhanced quality and safety in automotive systems. Conference on Automation, Assistance and Embedded Real Time Platforms for Transportation (AAET 2005) 16.–17. Februar 2005, Braunschweig, S 489–512
31. IEEE 1061-1998 (1998) IEEE standard for a software quality metrics methodology
32. ISO/IEC 9126 (2001) Software engineering – product quality. International Standarization Organization
33. Jackson D (2002) Alloy: A lightweight object modelling notation. ACM Trans Softw Eng Methodol (TOSEM'02) 11(2):256–290
34. Krahn H, Rumpe B, Völkel S (2007) Roles in software development using domain specific modeling languages. In: Proceedings of the 6th OOPSLA Workshop on Domain-Specific Modeling (DSM' 06), 22. Oktober 2006, Portland, Oregon, USA
35. Ludewig J, Lichter H (2007) Software Engineering. dpunkt.verlag, Heidelberg
36. Ludewig J (2002) Modelle im Software Engineering – eine Einführung und Kritik. Modellierung 2002, 25.–27. März 2002, Tutzing. Lecture Notes in Informatics (LNI), GI-Edition, S 7–22
37. McCall J, Richards P, Walters G (1977) Factors in software quality, vol. 1–3. US Rome Air Development Center Report, NTIS AD/A-049 014
38. Potop D, Edwards S, Berry G (2007) Compiling Esterel. Springer
39. Pol M, Koomen T, Spillner A (2000) Management und Optimierung des Testprozesses. dpunkt.verlag, Heidelberg
40. Petrasch R, Meimberg O (2006) Modellgetriebene Software-Entwicklung – Eine praxisorientierte Einführung. dpunkt.verlag, Heidelberg
41. Rumbaugh J, Blaha M, Premerlani W, Eddy F, Lorensen W (1991) Object-Oriented Modelling and Design. Prentice Hall
42. Reussner R, Hasselbrink W (2006) Handbuch der Softwarearchitektur. d.punkt.verlag, Heidelberg
43. Reißing R (2002) Bewertung der Qualität objektorientierter Entwürfe, Dissertation, Universität Stuttgart
44. Rumpe B (2004) Agile Modellierung mit UML – Codegenerierung, Testfälle, Refactoring. Springer, Heidelberg
45. Rumpe B (2004) Modellierung mit UML – Sprache, Konzepte und Methodik. Springer, Heidelberg
46. Rumpe B (2004) Agile modeling with the UML. In: Wirsing M, Knapp A, Balsamo S (eds) Radical innovations of software and systems engineering in the future. 9th International Workshop, RISSEF 2002, 7.–11. Oktober 2002, Venice, Italy. LNCS 2941
47. Rumpe B (2006) Agile test-based modeling. In: Proceedings of the 2006 International Conference on Software Engineering Research & Practice. SERP'2006, 26.–29. Juni 2006. CSREA Press, USA
48. Schütte R, Rotthowe T (1998) The guidelines of modelling as an approach to enhance the quality of information models. In: Conceptual Modeling – ER '98, 17th International ER-Conference, 3.–6. November 1997, S 240–254
49. Stachowiak H (1973) Allgemeine Modelltheorie. Springer, Wien
50. Steinmann F (2005) Domain models are aspect free. In: MoDELS 2005, LNCS 3713. Springer, Berlin, S 171–185
51. Teeuw WB, v.d. Berg H (1997) On the quality of conceptual models. In: Proceedings of the ER'97 Workshop on Behavioral Models and Design Transformations: Issues and Opportunities in Conceptual Modeling, 6.–7. November 1997, Los Angeles, California
52. OMG Unified Modeling Language (OMG UML) (2007) Superstructure, V2.1.2
53. OMG Unified Modeling Language (OMG UML) (2007) Infrastructure, V2.1.2
54. Vogel O, Arnold I (2005) Software-Architektur. Spektrum Akademischer Verlag

Florian Fieber arbeitet als Berater und Entwicklungsleiter bei qme Software, Berlin. Er beschäftigt sich schwerpunktmäßig mit Softwarequalitätsmanagement und -sicherung, insbesondere bei modellbasierten Entwicklungsprojekten.

Dr. Michaela Huhn entwickelt Methoden und Werkzeuge für den verbesserten Einsatz pragmatischer und formaler modellbasierter Techniken für die effiziente Entwicklung eingebetteter Systeme. Angewendet werden diese Techniken dabei vor allem in den Bereichen Bahn, Automobil und mobile Systeme.

Prof. Dr. Bernhard Rumpe leitet das Institut für Software Systems Engineering an der Technischen Universität Braunschweig. Dort hat er ein Forschungslabor zur modellbasierten Softwareentwicklung eingerichtet, das Modellierungssprachen (DSLs) vor allem für evolutionäre Softwareentwicklung, Generierung und Konfiguration von Softwarearchitekturen und zur Qualitätssicherung eingebetteter Software einsetzbar macht. Bernhard Rumpe ist Autor mehrerer Bücher und Herausgeber des Journal for Software and Systems Modelling (SoSyM).