# Kontextsensitive Qualitätsplanung für Software-Modelle

Hendrik Voigt und Gregor Engels
Universität Paderborn, Deutschland
Fachgebiet Informationssysteme
{hvoigt,engels}@uni-paderborn.de

**Abstract:** Der Goal Question Metric (GQM) Ansatz stellt eine allgemeine Qualitätsmanagementstrategie dar und berücksichtigt infolgedessen nicht die Besonderheiten von Software-Modellen. Wir haben eine kontextsensitive Qualitätsplanung für Software-Modelle entwickelt, die den GQM-Ansatz auf die Qualitätsplanung von Software-Modellen zuschneidet und um Konzepte und Aktivitäten erweitert. Dabei konzentrieren wir uns auf den Kontext eines Software-Modells als entscheidenden Einflussfaktor für die Dokumentation von Informationsbedürfnissen, Qualitätsverständnis und Messung. Unser Ansatz zur Qualitätsplanung besteht aus einem Metamodell zur Formulierung relevanter Inhalte und einem Prozess, der als Leitfaden bei der Planung dient.

## 1 Einleitung

Die modellbasierte Softwareentwicklung gewinnt durch den hohen Reifegrad und die weite Verbreitung der Unified Modeling Language (UML), praxiserprobter Modellierungsmethoden und unterstützender Werkzeuge zunehmend an Bedeutung. Sie stellt eine vielversprechende Alternative dar, um die Komplexität der zu implementierenden Softwaresysteme und den gleichzeitigen Kosten- und Termindruck zu beherrschen. Software-Modelle werden dabei beispielsweise zur Analyse der Anforderungsspezifikation, zur Konzeption der Architektur oder zum Entwurf des zu implementierenden Softwaresystems eingesetzt. Infolgedessen bilden Software-Modelle das Bindeglied zwischen den Wünschen des Kunden und dem fertigen Softwareprodukt und stellen zentrale Entwicklungsartefakte dar.

Die Stärke von Software-Modellen liegt in der Abstraktion. Sie vereinfachen komplexe Systeme und Probleme durch Reduktion und ermöglichen es einem Projektteam, durch schrittweise Verfeinerungen Lösungen systematisch zu entwickeln. Bei diesem Vorgehen sollten Qualitätsmängel bereits in der Modellierung erkannt werden, damit sich notwendige Korrekturen nur auf wenige Entwicklungsphasen und die entsprechenden Entwicklungsartefakte beziehen.

Um die Qualität von Software-Modellen kontrollieren zu können, muss ein Qualitätsmanagementsystem etabliert werden. Das Ziel dieses Qualitätsmanagementsystems besteht darin,

- die Qualitätsziele für Software-Modelle in einem modellbasierten Entwicklungsprojekt zu dokumentieren,

- den Erfüllungsgrad dieser Ziele zu messen sowie zu analysieren und
- die Analyse-Ergebnisse für Projektbeteiligte in maßgeschneiderten Qualitätsberichten aufzubereiten,
- damit die Projektbeteiligten Entscheidungen über das weitere Projektvorgehen auf einer fundierten Informationsbasis treffen können.

Das Qualitätsmanagement lässt sich auf vier wichtige Aktivitäten reduzieren (vgl. 1).

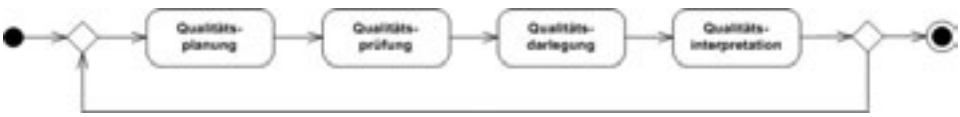


Abbildung 1: Aktivitäten im Qualitätsmanagement

- Qualitätsplanung: Festlegen der Qualitätsziele und der dafür ausschlaggebenden Charakteristiken eines Produkts, wie sie gemessen und bewertet werden und welche Ergebnisse erreicht werden sollen.
- Qualitätsprüfung: Durchführung der Prüfmaßnahmen zur Feststellung des erreichten Qualitäts-Ist-Zustandes.
- Qualitätsdarlegung: Berichten des Qualitäts-Ist-Zustandes im Vergleich mit den Qualitätszielen.
- Qualitätsinterpretation: Interpretation der Ergebnisse der Qualitätsdarlegung und Treffen der Entscheidung über das weitere Vorgehen (z.B. Qualitätsverbesserung des Produkts, Weiterentwicklung des Produkts oder Abbruch der Produktentwicklung).

In dem vorliegenden Papier konzentrieren wir uns auf die Aktivität *Qualitätsplanung* und stellen einen Ansatz für die Qualitätsplanung von Software-Modellen vor. Unserer Ansatz adaptiert wichtige Konzepte und Aktivitäten der Goal Question Metric (GQM) [BCR94] und schneidet diese auf Software-Modelle zu, indem Konzepte und Aktivitäten ergänzt werden.

Der Rest der Arbeit ist wie folgt aufgebaut. Kapitel 2 thematisiert wichtige Anforderungen an ein Qualitätsmanagementsystem für Software-Modelle. In Kapitel 3 werden relevante Arbeiten vorgestellt und ihr Nutzen für die Qualitätsplanung von Software-Modellen wird kurz diskutiert. Im vierten Kapitel wird das Grobkonzept unseres Ansatzes vorgestellt und der grundlegende Aufbau beschrieben. Das fünfte Kapitel stellt unseren Ansatz anhand eines einfachen, aber durchgängigen Beispiels detaillierter vor. Eine abschließende Zusammenfassung und ein Ausblick auf Erweiterungen und Modifikationen sowie nicht behandelte Problemstellungen werden in Kapitel 6 gegeben.

## 2 Anforderungen an ein Qualitätsmanagementsystem für Software-Modelle

Zuerst betrachten wir einige ausgewählte Standards, die sich auf Qualitätsmanagementsysteme und auf die Spezifikation sowie Evaluierung von Softwareproduktqualität beziehen. Software-Modelle abstrahieren von Softwareprodukten und stellen in einem modellbasierten Softwareentwicklungsprojekt Zwischenprodukte dar. Die Standards lassen sich sowohl auf Softwareprodukte als auch auf Software-Modelle anwenden. Diese folgenden Standards dienen uns dazu, Anforderungen an ein Qualitätsmanagementsystem für Software-Modelle ableiten zu können. Wir werden hier die wichtigsten top level-Anforderungen kurz vorstellen und falls nötig ihre Bedeutung für Software-Modelle darlegen.

Der internationale Standard ISO 9001 legt Anforderungen an ein umfassendes Qualitätsmanagementsystem fest (vgl. [ISO01a]). Für die Dokumentation eines Qualitätsmanagementsystems wird gefordert, dass Qualitätsziele festgelegt und messbar sind. Qualitätsziele sollten der Philosophie des Total Quality Management (TQM) folgend während des gesamten Entwicklungsprozesses verfolgt werden, so dass Qualitätsziele und ihr Erfüllungsgrad auch während der Modellierung bestimmt werden sollten.

**Anforderung 1 (Qualitätsziele)** *Ein Qualitätsmanagementsystem für Software-Modelle sollte messbare Qualitätsziele festlegen.*

In ISO/IEC 9126-1 wird festgelegt, dass alle relevanten Qualitätscharakteristiken eines Softwareproduktes spezifiziert und evaluiert werden sollten (vgl. [ISO01b]). Qualitätscharakteristiken, die sich auf die externe Qualität und die Gebrauchsqualität des Softwareproduktes beziehen, lassen sich im Verlauf der Entwicklung nur im Falle von Prototypen direkt bestimmen. Jedoch werden diese Sichtweisen auf Software-Qualität von Qualitätscharakteristiken der internen Qualitäten beeinflusst (vgl. Quality model framework in [ISO01b]). Zwischenprodukte wie Software-Modelle repräsentieren interne Sichten auf das Softwareprodukt. Somit können Qualitätscharakteristiken von Software-Modellen der internen Qualität von Software zugeordnet werden und wirken sich auf externe Qualität und Gebrauchsqualität des Endproduktes aus.

**Anforderung 2 (Qualitätscharakteristiken)** *Ein Qualitätsmanagementsystem sollte Qualitätscharakteristiken für Software-Modelle spezifizieren und evaluieren.*

Die Bedeutung von Informationsbedürfnissen für die gezielte Qualitätsprüfung wird durch die ISO/IEC 15939 Norm herausgestellt (vgl. [ISO02]). Organisatorische Entscheidungen (z.B. verstärkter Einsatz von Systemarchitekten) oder fachliche Entscheidungen (z.B. Strategie zur Verbesserung von Software-Modellen) werden ausschließlich von Menschen und nicht von Systemen getroffen. Dafür benötigen die Entwickler eine fundierte Informationsbasis, die ihre Informationsbedürfnisse berücksichtigt und befriedigt. Die Entwickler treffen Entscheidungen über das weitere Projektvorgehen, indem sie Berichte bzgl. der Software-Modell-Qualität interpretieren und Konsequenzen ziehen. Diese Berichte bezeichnen wir in Anlehnung an [ISO02] als Informationsprodukte. Die Informationsprodukte müssen durch die Entwickler gelesen und verstanden werden.

**Anforderung 3 (Informationsbedürfnisse)** *Ein Qualitätsmanagementsystem für Software-Modelle sollte Informationsbedürfnisse der Entwickler dokumentieren.*

**Anforderung 4 (Informationsprodukte)** *Ein Qualitätsmanagementsystem für Software-Modelle sollte Informationsbedürfnisse durch Informationsprodukte befriedigen, die in der Sprache der Entwickler formuliert sind.*

Um die Entwickler mit Informationen bzgl. der internen Qualität von Softwareprodukten (hier: Software-Modelle) zu versorgen, schreibt [ISO01b] vor, dass Qualitätscharakteristiken auf Basis von validierten oder allgemein akzeptierten Metriken zu bestimmen sind. Das Measurement Information Model (MIM) in [ISO02] ergänzt diese Anforderung. Das MIM beschreibt generisch, wie relevante Charakteristiken quantifiziert und in Indikatoren überführt werden können, die eine Basis für Entscheidungsfindungen bieten.

Die Bestimmung eines Indikators erfolgt in 2 Schritten: Zuerst wird mittels Metriken die Quantifizierung einer Qualitätscharakteristik durchgeführt. Im Anschluss kombiniert eine Analyse durch Metriken generierte Zahlenwerte und vergleicht sie mit numerischen Grenzen oder Zielen, um Werte für Indikatoren zu produzieren. Hierbei muss beachtet werden, dass die Analyse im Vorfeld geplant sein muss und nicht erst nachdem die Metrikergebnisse bekannt sind.

**Anforderung 5 (Indikatoren)** *Ein Qualitätsmanagementsystem für Software-Modelle sollte Indikatoren und deren Bestimmung dokumentieren.*

Modellbasierte Softwareentwicklungsprojekte können grundverschieden sein. Beispielsweise variieren die angewandten Entwicklungsmethoden, die eingesetzten Rollen, die Anforderungen an das Endprodukt, die Abstraktionsgrade der Software-Modelle, die verwendeten Modellierungssprachen u.v.m.. Infolgedessen unterscheiden sich die Informationsbedürfnisse der Anwender eines Qualitätsmanagementsystems. Qualitätsplanungen für Software-Modelle können nach unserer Auffassung nur lokale Gültigkeit haben, z.B. für ein einmaliges Projekt, einen Projekttypen (Entwicklung von Webanwendungen, Datenbanken) oder unternehmensweit. Wir bezeichnen dieses Phänomen als Kontextsensitivität. Eine Qualitätsplanung für Software-Modelle muss die Unterschiede zwischen Software-Modellen respektieren.

Zudem bindet die Erstellung eines Qualitätsplans Ressourcen. Dadurch entstehende Kosten sollten minimiert werden. Kann ein Qualitätsplan als Ganzes oder können einzelne Teile wiederverwendet werden, so sollte dies möglichst einfach ersichtlich sein. Idealer Weise erkennt man am dokumentierten Kontext, bis zu welchem Grad eine Qualitätsplanung übertragen werden kann. Auf diese Weise muss man nicht alle Aspekte der Qualitätsplanung analysieren.

**Anforderung 6 (Kontext)** *Ein Qualitätsmanagementsystem sollte den Kontext von Software-Modellen dokumentieren.*

## 3 Verwandte Arbeiten

Wir gruppieren die relevanten Arbeiten in drei Literaturklassen und stellen sie in der folgenden Reihenfolge vor: Goal Question Metric (GQM), Qualitäts-Modelle und Metriken.

Die Goal Question Metric (GQM) ist eine von Victor Basili geprägte Qualitätsmanagementstrategie (vgl. [BCR94]). Die Grundidee von GQM ist einfach zu vermitteln. GQM basiert auf der Annahme, *that for an organization to measure in a purposeful way it must first specify the goals for itself and its projects, then it must trace those goals to the data that are intended to define those goals operationally, and finally provide a framework for interpreting the data with respect to the stated goals.* Für die Anwendung von GQM bzw. darauf basierenden Ansätzen existieren bereits gute Leitfäden (vgl. [BDR97] und [PGF96]), die den Prozess und die zu erstellenden Dokumente detailliert beschreiben. Auf diese Weise lässt sich ein Qualitätsmanagementsystem für Software-Modelle individuell erstellen.

GQM ist ein sehr allgemeiner Ansatz, mit dem die Qualität von Produkten, Ressourcen und Prozessen charakterisiert, überwacht, evaluiert, vorausgesagt und kontrolliert werden kann. Es findet keine Spezialisierung auf Software-Modelle statt, so dass der eingeschränkte Kontext nicht beachtet wird. Zudem werden keine Qualitäts-Modelle wie in [ISO01b] eingesetzt, um die Gesamtqualität in Charakteristiken zu dekomponieren und auf diese Weise den Gegenstand der Messung zu detaillieren.

Dabei stellen Qualitäts-Modelle einen weit verbreitenden Ansatz dar, um Qualität nach dem Divide & Conquer Prinzip in mehrere Bestandteile zu gliedern und auf diese Weise zu präzisieren. Es existieren bereits mehrere Vorschläge für Qualitäts-Modelle, die den Begriff Modell-Qualität mit Hilfe von Qualitätscharakteristiken konkretisieren (vgl. [BCN92], [KLS95], [Moo98], [Sch98] und [Rei02]). Jedoch konnte sich bis dato keines von ihnen als de facto Standard etablieren. Dies hängt insbesondere damit zusammen, dass Qualität durch die Wahrnehmung von Menschen bestimmt und inhärent subjektiv ist. Begriffe und Definitionen, die Modell-Qualität beschreiben, hängen sowohl vom Modell und dessen Kontext selbst als auch von den Betrachtern ab.

Aus diesem Grund gibt es nicht ein allgemeingültiges Qualitäts-Modell. Innerhalb eines Qualitätsmanagementsystems repräsentiert ein Qualitäts-Modell ein spezifisches Qualitätsverständnis der Projektbeteiligten, das einen starken Bezug zum betrachteten Software-Modell aufweist. Qualitäts-Modelle unterstützen die Operationalisierung von Modell-Qualität. Qualitätscharakteristiken werden solange verfeinert, bis sie schließlich quantifizierbar sind. Die oben genannten Ansätze für Qualitäts-Modelle thematisieren z.T. auch Metriken. Allerdings sind die vorgeschlagenen Metriken mit Ausnahme der Arbeit von Reißing (vgl. [Rei02]) im Rahmen dieser Arbeit nicht erwähnenswert.

Stattdessen stellen wir eine Reihe von sehr speziellen Forschungsarbeiten vor, die sich auf einzelne Charakteristiken von Software-Modellen und Software-Diagrammen beschränken. Die Unified Modeling Language (UML) ist eine der dominierenden Sprachen in der modellbasierten Softwareentwicklung. Von den dreizehn in der UML-Superstructure beschriebenen Diagrammtypen haben UML-Klassendiagramme die höchste Verbreitung. Die folgenden Arbeiten befassen sich mit der Qualität von Software-Modellen, die als UML-

Klassendiagramme graphisch repräsentiert werden können.

Im Rahmen von Software-Qualität und Software-Modell-Qualität werden häufig Metriken eingesetzt. Eine Metrik bezeichnet im Allgemeinen eine Kennzahl und das Verfahren zur Messung dieser quantifizierbaren Größe. In der Literatur gibt es viele Vorschläge für Metriken.

Chidamber und Kemerer definieren in [CK94] sechs Software-Metriken, von denen drei auf UML-Klassendiagramme angewandt werden können. Sie beziehen sich auf die Komplexität von Klassen. McQuillan und Power haben in [MP06] mit Hilfe der Object Constraint Language (OCL) diese Metriken für UML formalisiert. Brito e Abreu nutzt ebenfalls OCL und hat eine ganze Reihe eigener Metriken für UML-Modelle in Rahmen von MOOD2 formalisiert (vgl. [Abr01]). Metriken wie Operations Inheritance Factor (OIF) oder Attributes Inheritance Factor (AIF) können den Gebrauch objekt-orientierter Mechanismen wie Vererbung quantifizieren. Dafür geeignet sind auch Metriken von Lorenz und Kidd [LK94]. Zudem schlagen die Autoren Metriken zur Bestimmung der Größe von Klassen vor. In [BPC05] werden basierend auf bestehenden Metriken zahlreiche neue Metriken für Klassen respektive Klassen-Diagramme vorgestellt. Einen guten Überblick über diese und weitere Metriken geben Reißing in [Rei02] und Genero et al. in [BPC05].

Als eine weitere Quelle für Metriken können grundsätzlich Experten befragt und Fachliteratur studiert werden. Das Buch von Fowler über Refactoring repräsentiert viel Erfahrungswissen bei der Implementierung von Software-Systemen (vgl. [FBB$^+$99]). Einige der darin vorgestellten Bad Smells lassen sich ebenfalls auf Klassendiagramme übertragen.

Das größte Problem besteht folglich nicht darin, irgendwelche Metriken zu finden, sondern gute Metriken zu finden, deren Werte analysiert werden können um Indikatoren zu bestimmen. Für viele Metriken gibt es keine Studien hinsichtlich ihrer Validität. Für eine ältere und viel untersuchte Metrik *Depth of Inheritance of a class (DIT)* widersprechen sich die Ergebnisse zum Teil. Eine Untersuchung von Daly et al. [DBM$^+$96] legt nahe, dass Systeme mit einer Vererbungstiefe von drei signifikant leichter zu warten sind im Vergleich mit Systemen, in denen keine Vererbung eingesetzt wurde. Dagegen besagt das Experiment von Harrison et al. [HCN00], dass Systeme ohne Vererbung leichter zu verstehen und warten sind als welche mit drei oder fünf Vererbungsebenen. An dieser Stelle möchten wir anmerken, dass Programmbibliotheken oder die Sprachdefinition der UML deutlich tiefere Vererbungshierarchien aufweisen.

Geeignete Metriken sind für ein erfolgreiches Qualitätsmanagement von zentraler Bedeutung. Metriken müssen nützlich sein. Zuerst kosten sie Geld, weil sie definiert, gelesen und angewendet werden. Der Mehrwert bei der Verwendung von Metriken kann darin bestehen, dass Qualitätsprobleme frühzeitig erkannt und effizient behoben werden. Werden aber Charakteristiken eines Software-Modells gemessen, ohne das *warum* zu kennen, dann führen diese Metriken ausschließlich zu unnötigen Kosten. Metriken dürfen folglich nicht vorbehaltslos zur Qualitätsbewertung von Software-Modellen herangezogen werden. Es muss klar sein, wie die berechneten Kennzahlen zu interpretieren sind.

Die vorgestellte Literatur kann uns bei der Erfüllung unserer Anforderungen wie folgt helfen:

GQM beschreibt ein *praktikables top down-Vorgehen*, um Metriken auf Basis von Infor-

mationsbedürfnissen auszuwählen. Zudem wird in [BDR97] die Idee skizziert, den Kontext einer Organisation bzw. eines Projektes in die Qualitätsplanung miteinzubeziehen. Allerdings muss insbesondere der letztere Aspekt für die Qualitätsplanung von Software-Modellen stark angepasst werden.

Qualitäts-Modelle eignen sich für die *Dokumentation von Qualitätscharakteristiken* und repräsentieren das Qualitätsverständnis der Projektbeteiligten. Qualitäts-Modelle können dabei helfen, Informationsbedürfnisse zu detaillieren und die Bestimmung der Indikatoren vorzubereiten. Unserer Ansicht nach stellen Qualitäts-Modelle ein wichtiges Bindeglied zwischen den Informationsbedürfnissen und den Indikatoren dar.

Die *Dokumentation von Metriken* kann sich an den bereits existierenden Metriken orientieren. Handelt es sich bei dem betrachteten Software-Modell beispielsweise um ein UML-Modell, dann bietet sich die Definition der Metriken mittels OCL an (vgl. [Abr01] und [MP06]).

*Analysen und Indikatoren* lassen sich nur partiell auf Basis der Literatur aufstellen. Hier spielt das Expertenwissen von Projektverantwortlichen und Qualitätsmanagern eine wichtige Rolle, um fehlende Grenz- und Zielwerte für Mess-Ergebnisse auszugleichen.

Die Basis für *Informationsprodukte* wird durch die Gesamtheit aller Dokumentationen, der Ergebnisse der Messungen und Analysen gelegt. Wir sind primär an der Sammlung und Verknüpfung von relevanten Informationen interessiert, um Qualitätsinterpretationen zu fördern. Für die Visualisierung von Mess-Ergebnissen existieren bereits Arbeiten (vgl. [TLTC05]). Über die Schaffung einer umfassenden Informationsbasis für Software-Modell-Qualität konnten wir bis dato keine interessanten Arbeiten finden.

*Qualitätsziele* werden indirekt in allen drei Literaturklassen (GQM, Qualitäts-Modelle und Metriken) thematisiert. Qualitätsziele fließen in Ziele und Fragen in GQM, Qualitätscharakteristiken in Qualitäts-Modellen und in Grenz- und Zielwerte für Indikatoren, den zu Grunde liegenden Metriken sowie Analysen mit ein. Die Summe all dieser Konzepte ermöglicht die Berücksichtigung von Qualitätszielen in der Qualitätsplanung von Software-Modellen.

Inwiefern die unterschiedlichen Arbeiten uns bei der Erfüllung unserer Anforderungen helfen, ist in Tabelle 1 zusammengefasst. Diese Anforderungsmatrix trifft kein wertendes Urteil über die vorgestellten Arbeiten. Aus diesem Grund haben wir explizit darauf verzichten, die Ausprägungen bei der Unterstützung zu differenzieren.

| **Anforderung** | | **Literatur zu** | | |
|---|---|---|---|---|
| **Nr.** | **Kurzbeschreibung** | **GQM** | **Qualitäts-Modell** | **Metrik** |
| 1 | Qualitätsziele | X | X | X |
| 2 | Qualitäts-Charakteristiken | | X | X |
| 3 | Informationsbedürfnisse | X | | |
| 4 | Informationsprodukte | | | |
| 5 | Indikatoren | X | | X |
| 6 | Kontext | | | |

Tabelle 1: Anforderungsmatrix

# 4 Eigener Ansatz

GQM ist ein sehr genereller Ansatz, um effektive Messungen zu identifizieren. Allerdings berücksichtigt GQM bedingt durch seine Positionierung keine Besonderheiten von Software-Modellen. Deshalb spezialisieren wir GQM für die Qualitätsplanung von Software-Modellen, indem wir geeignete Konzepte und Aktivitäten hinzufügen. Das Ergebnis unseres Qualitätsmanagementansatzes bezeichnen wir als Modell-Qualitäts-Plan (MQP). Zur Erstellung eines MQPs definieren wir ein MQP-Metamodell und einen MQP-Prozess. Das MQP-Metamodell beschreibt relevante Konzepte sowie deren Zusammenhänge und spezifiziert die möglichen Inhalte eines MQPs. Der MQP-Prozess spezifiziert das Vorgehensmodell zur Erstellung eines MQPs und dient als Leitfaden bei der Qualitätsplanung.

Der MQP-Prozess ist ein inkrementeller und iterativer Prozess. Er weist einige Spezialisierungen im Vergleich zu GQM auf. Unser Prozess beginnt mit der Dokumentation des Kontextes des betrachteten Software-Modells. Auf diese Weise stellen wir die Besonderheiten des Software-Modells in seinem Entwicklungskontext fest. Im nächsten Schritt werden Informationsbedürfnisse dokumentiert. Dies erfolgt analog zu GQM mittels Zielen und Fragen. Allerdings hat die Identifikation von Zielen und Fragen im GQM-Ansatz einen starken ad-hoc Charakter. Deshalb führen wir eine Systematisierung ein und nutzen die Kontextdokumentation zum Ableiten von Zielen. Nachdem die Informationsbedürfnisse festgelegt sind, werden sie mit Qualitätscharakteristiken in Beziehung gesetzt. Die Definition von Qualitätscharakteristiken und deren Abhängigkeiten untereinander bilden das Qualitäts-Modell. Messungen und Analysen werden basierend auf dem Qualitäts-Modell definiert. Abschließend findet die Planung der Darlegung der Software-Modell-Qualität statt. Hierbei wird eine Auswahl der wichtigsten Informationen getroffen.

Das MQP-Metamodell spezifiziert relevante Konzepte und deren Zusammenhänge. Dabei dienen UML-Klassendiagramme als Beschreibungsmittel. Das MQP-Metamodell strukturiert sich in fünf Pakete: Kontext-Metamodell, Informationsbedürfnis-Metamodell, Qualitäts-Metamodell, Mess-Metamodell und Präsentations-Metamodell. Jedes Paket enthält fachlich zusammengefasste Konzepte und deren Beziehungen. Auf Basis des MQP-Metamodells verknüpfen wir Konzepte aus mehreren Ansätzen (GQM, Qualitäts-Modelle, Mess-Theorie) und ergänzen das Metamodell zusätzlich um Konzepte zur Kontextdokumentation.

Für jedes der Pakete des Metamodells gibt es eine Menge an Aktivitäten, die ausgeführt werden sollen, um einen MQP zu erstellen. Anhand eines kleinen Ausschnitts des Kontext-Metamodells konkretisieren wir den Aufbau des MQP-Metamodells exemplarisch und zeigen das Zusammenspiel mit dem MQP-Prozess auf. Dafür fasst der folgende Abschnitt einige Aspekte eines modellbasierten Softwareentwicklungsprozesses zusammen:

Ein Softwareentwicklungsprozess spezifiziert ein Vorgehen bei der Softwareentwicklung. Dieser Prozess gliedert sich in unterschiedliche Phasen, die inhaltlich aufeinander aufbauen. Typische Phasen sind durch Anforderungsspezifikation, Analyse, Entwurf, Implementierung und Test gegeben. In einer Phase werden Softwareentwicklungsartefakte produziert. Software-Modelle sind solche Softwareentwicklungsartefakte, die sich ebenfalls diesen Phasen zuordnen lassen. Software-Modelle basieren auf einer Modellierungsspra-

che. Beispiele für Modellierungssprachen sind die UML oder das Entity-Relationship-Modell (ERM). Der resultierende Ausschnitt des Kontext-Metamodells ist Abbildung 2 zu entnehmen.

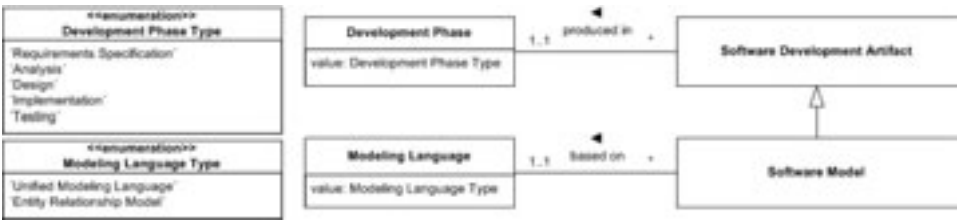


Abbildung 2: Ausschnitt Kontext-Metamodell

Die zu diesem Ausschnitt des Kontext-Metamodells entsprechenden MQP-Prozessschritte sind *Dokumentation der Entwicklungsphase des Software-Modells* und *Dokumentation der Modellierungssprache des Software-Modells*.

In dem vorliegenden Papier können wir das MQP-Metamodell und den MQP-Prozess aufgrund von Seitenrestriktionen nicht vollständig darstellen. Stattdessen stellen wir die Grundlagen vor. Der nächste Abschnitt enthält ein kurzes Beispiel. Es zeigt, wie man ausgehend von einem gegebenen Kontext systematisch eine Metrik aufstellen kann. Auf wichtige Ergänzungen zum MQP-Metamodell und MQP-Prozess gehen wir an den jeweiligen Stellen ein.

# 5 Beispiel

## 5.1 Dokumentation des Kontextes

Der Kontext beschreibt das Umfeld, in dem ein Software-Modell betrachtet werden sollte. Die Dokumentation des Kontextes ermöglicht die Berücksichtigung von Besonderheiten des betrachteten Software-Modells für die Qualitätsplanung. Die Dokumentation des Kontextes beschränkt sich in diesem Beispiel auf

- die Entwicklungsphase und die verwendete Modellierungssprache.

Die *Entwicklungsphase* gibt Aufschluss darüber, wie implementierungsnah das Software-Modell ist. Handelt es sich beispielsweise um ein Entwurfsdiagramm, dann können bereits mehr Qualitätsanforderungen an das Software-Modell gestellt werden, die auch für Quell-Code gelten. Für ein Modell des Problembereichs hätte diese Qualitätsanforderung dagegen keine Gültigkeit.

Der Kontext *Modellierungssprache* kann beispielsweise zur Überprüfung der Syntax herangezogen werden. Diesen Aspekt möchten wir hier jedoch nicht weiter verfolgen. In

diesem Beispiel benötigen wir die verwendete Modellierungssprache, um auf eine geeignete Formalisierung der Metriken zu schließen. Metriken sollten möglichst direkt auf dem Software-Modell operieren können. Transformationen des Software-Modells in eine andere semantische Domäne sind aufwändig. Falls es möglich ist, sollten diese Transformationen vermieden werden. In unserem Beispiel soll es sich bei dem betrachteten Software-Modell um ein UML-Entwurfsdiagramm handeln (vgl. Abbildung 3).

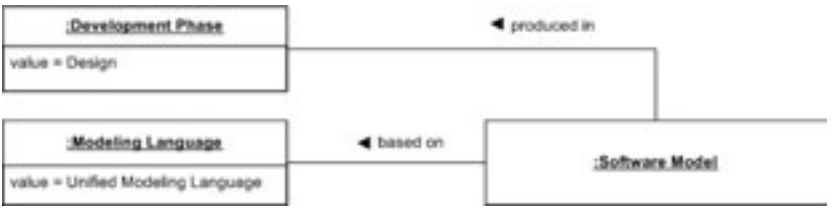


Abbildung 3: Beispiel für den Kontext eines Software-Modells

### 5.2 Dokumentation der Informationsbedürfnisse

Basierend auf der Dokumentation des Kontextes können wir Informationsbedürfnisse systematisch ableiten. Informationsbedürfnisse werden festgelegt, damit sich die Qualitätsplanung auf das Wesentliche beschränkt. Sie werden dokumentiert, indem

- Prüf-Ziele und Fragen identifiziert und dokumentiert werden.

[BDR97] schlägt ein Muster zum Aufstellen von Prüf-Zielen vor. Das Template listet fünf Dimensionen auf: Gegenstand, Zweck, Qualitätsschwerpunkt, Sichtweise und Kontext der Analyse. Alle Dimensionen sind auch für unseren Ansatz sinnvoll. Wir haben bereits den Gegenstand der Prüfung und den Kontext hinreichend spezifiziert. Jetzt widmen wir uns dem Zweck und dem Qualitätsschwerpunkt. Die Sichtweise bei der Prüfung ist ebenfalls relevant, wird hier jedoch nicht weiter thematisiert. In unserer Qualitätsplanung sind nur zwei Prüfzwecke vorgesehen:

1. *Evaluation* zielt auf den Vergleich und die Bewertung der Qualität eines Software-Modells ab.
2. *Charakterisierung* bestimmt den aktuellen Zustand eines Software-Modells. Dies ist dann notwendig, wenn noch keine Erfahrungswerte für die Evaluation vorliegen.

Der Qualitätsschwerpunkt referenziert bereits Qualitätscharakteristiken des Qualitäts-Modells, das in der nächsten Aktivität überarbeitet und weiter verfeinert wird. Fragen beziehen sich auf die Prüf-Ziele und helfen dabei, Informationsbedürfnisse weiter zu präzisieren. Abbildung 4 modelliert die Zusammenhänge.

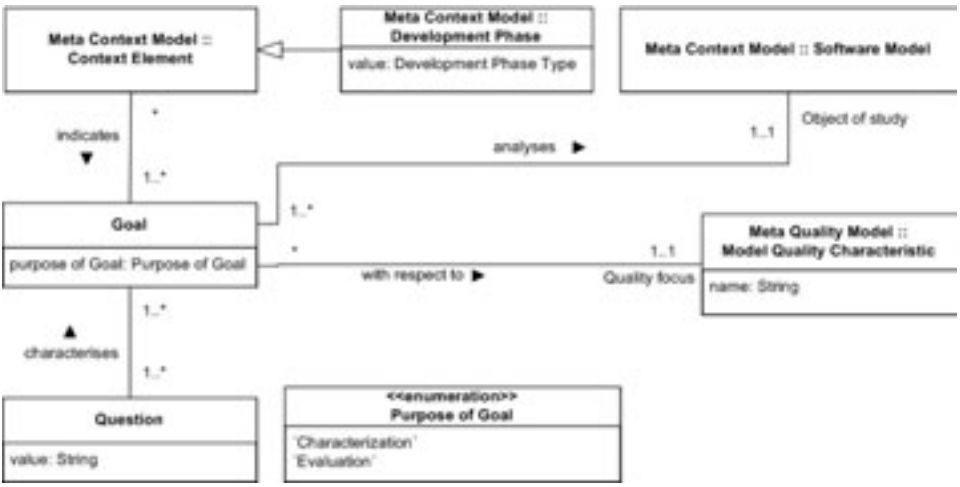


Abbildung 4: Ausschnitt Meta-Informationsbedürfnis-Modell

In unserem laufenden Beispiel handelt es sich um ein Software-Modell in der Entwurfsphase. Der Entwurf eines Softwaresystems soll unter anderem die Wartbarkeit des zu implementierenden Softwaresystems fördern (vgl. [Oes01]). Nach Fowler beeinflussen zu große Klassen in einem Softwaresystem die Wartbarkeit negativ und er schlägt zur Behebung Refactorings vor. Damit in der Implementierung dieser Aufwand erst gar nicht entsteht, sollten Entwurfsmodelle diese Anforderung bereits berücksichtigen. Wir können aus dem Wissen, dass es sich um ein Entwurfsmodell handelt, ein Prüf-Ziel und eine damit zusammenhängende Frage ableiten. Das Prüf-Ziel besteht darin, dass das betrachtete Software-Modell hinsichtlich seiner *Wartbarkeit evaluiert* werden soll. Dieses Prüf-Ziel wird durch die Frage *Enthält das Software-Modell große Klassen?* verfeinert.


Abbildung 5: Beispiel für ein Informationsbedürfnis

### 5.3 Dokumentation des Qualitätsverständnisses

Basierend auf den Prüf-Zielen und Fragen können wir das Qualitäts-Modell systematisch ableiten. Das Qualitäts-Modell beschreibt das grundlegende Qualitätsverständnis mittels Qualitätscharakteristiken. Qualitäts-Modelle werden dokumentiert, indem

- Qualitätscharakteristiken bezeichnet, Synonyme angegeben sowie definiert und in Beziehung miteinander gesetzt werden, so dass ersichtlich ist, wie sich die Qualitätscharakteristiken untereinander beeinflussen.

Das dadurch entstehende Qualitäts-Modell für das betrachtete Software-Modell muss mindestens die Granularität der Informationsbedürfnisse aufweisen, damit keine Details verloren gehen. Im Qualitäts-Modell können Fragen des Informationsbedürfnis-Modells mittels Qualitätscharakteristiken gruppiert werden. Für die unterste Ebene des Qualitäts-Modells fordern wir, dass sie gemessen werden kann. Die Elemente dieser Ebene bezeichnen wir als Qualitätsattribute, die eine spezielle Form von Charakteristiken darstellen. Der entsprechende Ausschnitt des Paketes Qualitäts-Metamodell ist in Abbildung 6 enthalten. Abbildung 7 zeigt die Instanziierung für das laufende Beispiel mit einer Definition für *Wartbarkeit* und dem Qualitätsattribut *Große Klasse*.

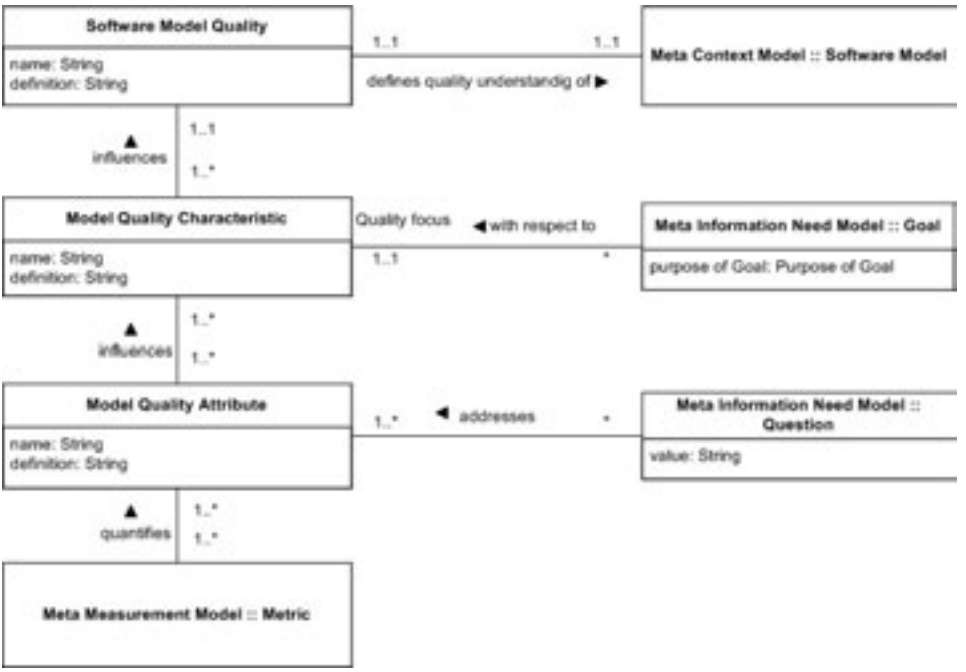


Abbildung 6: Ausschnitt Qualitäts-Metamodell


Abbildung 7: Beispiel für ein Qualitäts-Modell eines Software-Modells

### 5.4 Dokumentation der Messungen und Analysen

Metriken quantifizieren Qualitätsattribute. Eine Metrik beschreibt im Allgemeinen eine Kennzahl und das Verfahren zur Messung dieser quantifizierbaren Größe. Wir werden das Mess-Metamodell hier nicht weiter vorstellen. Um einen ersten Eindruck wichtiger Bestandteile bei der Dokumentation einer Metrik zu erhalten, kann das Beispiel in Tabelle 2 verwendet werden.

Für das laufende Beispiel stellt sich die Frage, wie das Qualitätsattribut *Große Klasse* gemessen werden kann. Abbildung 6 stellt bereits den Zusammenhang im MQP-Metamodell her. Für die Quantifizierung dieses Attributs konnten wir allerdings in der Literatur keine geeignete Metrik finden. Fowler hat zwar den Begriff *Große Klasse* in [FBB+99] geprägt. Jedoch schlägt er keine fixen Kriterien für die Identifizierung vor. Aus diesem Grund stellen wir die untere Metrik zur Diskussion (vgl. Tabelle 2). Die *Metrik NLC* zählt die Anzahl großer Klassen. Eine große Klasse liegt unserer Meinung dann vor, wenn das Verhältnis der Anzahl der eigenen Operationen und Attribute drei mal größer als der Durchschnitt für das ganze UML-Entwurfsmodell ist. Dieser Grenzwert leitet sich aus unserer Erfahrung ab und dient als Vorschlag. In einem anderen Softwareentwicklungsprojekt müsste dieser Wert noch validiert und bestätigt bzw. korrigiert werden. Als Ergänzung zu dieser Metrik bieten sich noch andere statistische Standardverfahren an. Z.B. lässt sich das beschriebene Verhältnis zur Berechnung der Differenz zwischen den maximalen und minimalen Wert über alle Klassen verwenden.

Weil es sich in unserem Beispiel um ein UML-Modell handelt und OCL die Anfragesprache für UML-Modelle darstellt, ist die *Metrik NLC* mittels OCL formalisiert. NOA() und AOA() sind Hilfskonstrukte und erhöhen die Lesbarkeit der unteren OCL-Anfrage. Zudem ermöglicht uns dieser Kunstgriff die Anfrage stark zu verkürzen. Die weiteren Details möchten wir hier nicht weiter aufführen. NOA() bestimmt die absolute Anzahl an verfügbaren Attributen und Operationen einer Klasse. AOA() berechnet die durchschnittliche Anzahl an verfügbaren Attributen und Operationen in einem Modell.

Im Anschluss kann die *Kennzahl NLC* wie folgt analysiert werden. Wenn der Wert echt

größer 0 ist, dann sollte der Entwurf besser ausbalanciert werden. Je höher der Wert im Einzelfall ausfällt, desto mehr Aufwand ist zu erwarten.

Abschließend wird das Präsentations-Modell definiert. Es baut auf den Vorarbeiten auf und setzt Informationsprodukte mit diesen in Beziehung. Mit dem Präsentations-Modell wird der Zweck verfolgt, Informationsbedürfnisse durch Informationsprodukte zu befriedigen.

Wir möchten an diesem Punkt das Beispiel abschließen und gehen auf die Dokumentation der Informationsprodukte für das laufende Beispiel nicht weiter ein. Wir haben unser Ziel erreicht und eine Metrik für UML-Entwurfsmodelle aufgestellt, die einen klaren Bezug zum Kontext hat.

| **Metric** | |
|---|---|
| **Name (Acronym)** | Number of Large Classes (NLC) |
| **Informal definition (English)** | Total number of Large Classes in the model. A class is a Large Class if the total number of its operations and its attributes is more than three times higher than the average for all Classes. |
| **Formal definition (OCL)** | `context Model::NLC(): Integer`<br>`post: result = AllClasses() -> iterate (`<br>`elem:Class; acc: Integer = 0 \|`<br>`if ((elem.NOA() / self.AOA()) > 3)`<br>`then acc + 1`<br>`endif` |
| **Type of measurement** | Objective |
| **Output** | Measure (NLC) |
| **Scale (Type of scale)** | Integers from zero to infinity (Ratio) |

Tabelle 2: Definition einer Metrik

## 6 Zusammenfassung und Ausblick

Wir haben einen Ansatz für die Qualitätsplanung von Software-Modellen entwickelt, der den Goal Question Metric (GQM) Ansatz spezialisiert. Unser Ansatz besteht aus einem Metamodell zur Formulierung relevanter Inhalte und einem Prozess, der als Leitfaden bei der Planung dient. Das Ergebnis der Qualitätsplanung bezeichnen wir als Modell-Qualitäts-Plan (MQP), das zugrunde liegende Metamodell als MQP-Metamodell und den Prozess als MQP-Prozess.

Im Rahmen dieser Arbeit haben wir uns darauf konzentriert, die grundlegende Idee und den Aufbau unseres Ansatzes vorzustellen. Auf einige zentrale Konzepte unseres MQP-Metamodells und ergänzende Aktivitäten des MQP-Prozesses konnten wir anhand eines

durchgängigen Beispiels kurz eingehen.

In unserem Ansatz stellt der Kontext eines Software-Modells einen entscheidenden Einflussfaktor für die Qualitätsplanung dar. Zukünftige Forschungsaktivitäten werden sich auf die Dokumentation des Kontextes konzentrieren und die Auswirkungen auf Informationsbedürfnisse, Qualitäts-Modell, Metriken und Analysen untersuchen.

Aufgrund des hohen Detaillierungsgrades unseres Ansatzes ist der initiale Erstellungsaufwand für einen MQP erheblich. Trotzdem sind wir davon überzeugt, dass eine auf dieser Basis durchgeführte Qualitätsbewertung langfristig kosteneffektiv sein wird. In der Einbeziehung des Kontextes steckt viel Potential. Die Wiederverwendung eines MQPs als Ganzem oder einzelner Teile sollte idealer Weise an der Dokumentation des Kontextes erkannt werden. Zudem lässt sich die Erstellung eines MQPs effizienter gestalten, indem eine Reihe von Informationsbedürfnissen systematisch aus dem Kontext abgeleitet werden können. Diese Beobachtung sollte sich auch auf die Beschreibung von Metriken auswirken, damit ihr Anwendungsfeld differenziert wird und leichter ersichtlich ist, unter welchen Umständen welche Metriken sinnvoll eingesetzt werden können.

Das Beispiel in Kapitel 5 vermittelt eine Vorstellung der stark vernetzten Strukturen eines MQPs. Auf dieser Basis findet die Darlegung der Ergebnisse in einem Informationsprodukt statt. Diese Informationsprodukte sind im Präsentations-Metamodell enthalten, auf das wir allerdings hier nicht eingehen konnten.

Eine einjährige studentische Projektgruppe von 11 Studierenden mit dem Titel *Model Cockpit* arbeitet zur Zeit an der Universität Paderborn an einer Werkzeugunterstützung und setzt die theoretischen Arbeiten prototypisch um. In Rahmen dieser Projektgruppe entsteht ein Eclipse-Plugin, mit dem MQPs erstellt und auf Software-Modelle angewandt werden können.

## Literatur


[Abr01] Fernando Brito e Abreu. Using OCL to formalize object oriented metrics definitions. Bericht, FCT/UNL and INESC, Portugal, 2001.

[BCN92] Carlo Batini, Stefano Ceri und Shamkant B. Navathe. *Conceptual Database Design: An Entity-Relationship Approach*. Benjamin/Cummings, 1992.

[BCR94] V. Basili, G. Caldiera und H.D. Rombach. The Goal Question Metric Approach. *Encyclopedia of Software Engineering*, Seiten pp. 528–532, 1994.

[BDR97] Lionel C. Briand, Christiane M. Differding und H. Dieter Rombach. Practical Guidelines for Measurement-Based Process Improvement. *Special issue of International Journal of Software Engineering & Knowledge Engineering*, 2, 1997.

[BPC05] Marcela Genero Bocco, Mario Piattini und Coral Calero. A Survey of Metrics for UML Class Diagrams. *Journal of Object Technology*, 4(9):59–92, 2005.

[CK94] Shyam R. Chidamber und Chris F. Kemerer. A Metrics Suite for Object Oriented Design. *IEEE Trans. Software Eng.*, 20(6):476–493, 1994.

[DBM+96] J. Daly, A. Brooks, J. Miller, M. Roper und M. Wood. An empirical study evaluating depth of inheritance on the maintainability of object-oriented software, 1996.

[FBB+99] Martin Fowler, Kent Beck, John Brant, William Opdyke und Don Roberts. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, June 1999.

[HCN00] Rachel Harrison, Steve Counsell und Reuben V. Nithi. Experimental assessment of the effect of inheritance on the maintainability of object-oriented systems. *Journal of Systems and Software*, 52(2-3):173–179, 2000.

[ISO01a] ISO. International Organization for Standardization (ISO) 9001:2001: Quality Management Systems Requirements, 2001.

[ISO01b] ISO/IEC. International Organization for Standardization (ISO) and International Electrotechnical Commision (IEC) 9126:2001: Software engineering Product quality Part 1: Quality model, 2001.

[ISO02] ISO/IEC. International Organization for Standardization (ISO) and International Electrotechnical Commision (IEC) 15939:2002: Software engineering Software measurement process, 2002.

[KLS95] John Krogstie, Odd Ivar Lindland und Guttorm Sindre. Defining quality aspects for conceptual models. In Eckhard D. Falkenberg und Wolfgang Hesse, Hrsg., *ISCO*, Jgg. 26 of *IFIP Conference Proceedings*, Seiten 216–231. Chapman & Hall, 1995.

[LK94] Mark Lorenz und Jeff Kidd. *Object-Oriented Software Metrics: A Practical Guide*. Prentice Hall, Englewood Cliffs, New Jersey, 1994.

[Moo98] Daniel L. Moody. Metrics for Evaluating the Quality of Entity Relationship Models. In Tok Wang Ling, Sudha Ram und Mong-Li Lee, Hrsg., *Conceptual Modeling - ER '98, 17th International Conference on Conceptual Modeling, Singapore, November 16-19, 1998, Proceedings*, Jgg. 1507 of *Lecture Notes in Computer Science*, Seiten 211–225. Springer, 1998.

[MP06] Jacqueline A. McQuillan und James F. Power. A Definition of the Chidamber and Kemerer Metrics suite for UML. Bericht, Department of Computer Science, National University of Ireland, Maynooth, Co. Kildare, Ireland, 2006.

[Oes01] Bernd Oestereich. *Objektorientierte Softwareentwicklung mit der Unified Modeling Language*. Oldenbourg, 2001.

[PGF96] Robert E. Park, Wolfhart B. Goethert und William A. Florac. *Goal-Driven Software Measurement A Guidebook*. Software Engineering Institute Carnegie Mellon University, Pittsburgh, PA 15213, August 1996.

[Rei02] R. Reißing. *Bewertung der Qualität objektorientierter Entwürfe [Assessment of the Quality of Object-Oriented Designs]*. Dissertation, Universität Stuttgart, Fakultät Informatik, 2002.

[Sch98] Reinhard Schütte. Vergleich alternativer Ansätze zur Bewertung der Informationsmodellqualität. *Informationssystem-Architekturen*, Heft 2, 1998.

[TLTC05] Maurice Termeer, Christian F. J. Lange, Alexandru Telea und Michel R. V. Chaudron. Visual Exploration of Combined Architectural and Metric Information. In Stéphane Ducasse, Michele Lanza, Andrian Marcus, Jonathan I. Maletic und Margaret-Anne D. Storey, Hrsg., *VISSOFT*, Seiten 21–26. IEEE Computer Society, 2005.