HAW HAMBURG

MASTERTHESIS
Keno Weßels

Entwicklung eines Qualitätsmodells zur Bewertung von Softwarebibliotheken

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Keno Weßels

Entwicklung eines Qualitätsmodells zur Bewertung von Softwarebibliotheken

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang *Master of Science Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Bettina Buth
Zweitgutachter: Prof. Dr. Ulrike Steffens

Eingereicht am: 03.11.2020

Keno Weßels

Thema der Arbeit

Entwicklung eines Qualitätsmodells zur Bewertung von Softwarebibliotheken

Stichworte

Softwarequalitätspropagation, Softwarebibliotheken, Softwarequalität, Softwarequalitätsmodelle

Kurzzusammenfassung

Bei der Qualitätsbewertung von Software sollte die Qualität genutzter Softwarebibliotheken mit einbezogen werden. Die bekannten Ansätze dazu sind praxisorientiert und dienen zur Auswahl von jeweils geeignet erscheinenden Softwarebibliotheken. Anforderungen an ein allgemeines Qualitätsmodell für Softwarebibliotheken wurden bislang kaum untersucht. Diese Arbeit entwickelt auf der Basis des ISO-25010-Qualitätsmodells Ansätze für ein Softwarequalitätsmodell, das insbesondere auch die Propagation der Qualitätsmerkmale nachvollziehbar werden lässt.

Der Einfluss der Qualität von genutzten „externen“ Softwarebibliotheken auf ein Softwareprodukt kann grundsätzlich durch einen Graphen dargestellt werden, der die Propagationsstruktur abbildet. Die Qualität wird dabei aufgeschlüsselt auf verschiedene Merkmale in der Form eines Qualitätsvektors betrachtet. Dabei können die spezifischen Besonderheiten der jeweiligen Softwarequalitätsmerkmale auch durch jeweils passende Methoden zur Aggregation der propagierten Qualität dargestellt werden. Anhand der beispielhaft ausgewählten Qualitätsmerkmale *Wartbarkeit* und *Sicherheit* wird aufgezeigt, wie diese auf komplexe Weise propagieren. Dazu kommen auf der Basis eines Abhängigkeitsgraphen ein Propagationsgraph und parallel dazu ein Propagationsübergangsbaum zum Einsatz. Die Arbeit entwickelt beispielhafte Methoden zur Aggregation der lokalen und propagierten Qualität und bezieht strukturelle Probleme der Softwarequalitätspropagation wie Mehrfachabhängigkeiten und zirkuläre Abhängigkeiten ein. Die beiden Qualitätsmerkmale *Wartbarkeit* und *Sicherheit* erfordern zwei sehr unterschiedliche Aggregationsmethoden, und es zeigt sich, dass für unterschiedliche Softwarequalitätsmerkmale auch spezifisch angepasste Aggregationsmethoden entwickelt werden können und damit differenzierte Bewertungen der Qualität von Softwareprodukten möglich sind, die auch die propagierte Qualität von Softwarebibliotheken erfassen.

Keno Weßels

Title of Thesis

Development of a quality model for the evaluation of software libraries

Keywords

Software quality propagation, software libraries, software quality, software quality models

Abstract

Any assessment of software quality should take account of the quality of used software libraries. Existing approaches to such assessments are practice-oriented and designed to select software libraries that appear suitable in each case. To date, requirements for a general quality model for software libraries have hardly been explored. On the basis of the ISO-25010 model, this thesis develops approaches for a software quality model which captures in particular the propagation of quality characteristics.
In principle, the influence of the quality of used „external“ software libraries on a software product can be represented by a graph that portrays its propagation structure. To this end, the quality of the software library is broken down into different characteristics that are analysed in the form of a quality vector. In the quality vector, the specific propagation behaviour of each of the respective software quality characteristics can be represented by appropriate aggregation methods. Using *maintainability* and *security* as examples, the thesis illustrates how quality characteristics propagate in complex ways. For this purpose, the thesis uses a propagation graph based on a dependency graph and a propagation transition tree. On that basis, the thesis develops exemplary methods for aggregating local and propagated quality and considers structural problems of software quality propagation such as multiple dependencies and circular dependencies. The thesis concludes that the two quality characteristics *maintainability* and *security* require two very different aggregation methods, and that specifically adapted aggregation methods can also be developed for other software quality characteristics. This enables differentiated evaluations of the quality of software products that also capture the propagated quality of software libraries.

Inhaltsverzeichnis

Abbildungsverzeichnis **vii**

1 Einleitung **1**
1.1 Motivation 4
1.2 Ziele dieser Arbeit 5

2 Grundlagen der Softwarequalität von Softwarebibliotheken **7**
2.1 Analyse von Softwarebibliotheken 7
2.2 Qualitäts- und Softwarequalitätsbegriff 9
2.3 Softwarequalitätsmodelle 11
2.3.1 Basismodelle 11
2.3.2 Zugeschnittene Modelle und Open-Source-Software-Modelle 15
2.4 Das Abhängigkeitsmanagement von Eghan 17
2.5 Propagation/Vererbung von Softwarequalität durch einen Abhängigkeitsgraphen 18
2.5.1 Die strukturelle Abhängigkeit 19
2.5.2 Der Abhängigkeitsgraph 20
2.5.3 Direkte und transitive Abhängigkeiten 21
2.5.4 Mehrfachabhängigkeiten und zirkuläre Abhängigkeiten 22
2.6 Zusammenfassung: Ein zugeschnittenes Qualitätsmodell für Softwarebibliotheken 24

3 Vorgehensweise **27**
3.1 Zur Arbeit mit Graphen 27
3.2 Entwicklung eines Qualitätspropagationsprogramms 27
3.3 Entwicklung der Aggregationsmethoden 30

4 Probleme der Qualitätspropagation **32**
4.1 Ein Demonstrationsbeispiel 32

4.2 Strukturelle Probleme der Qualitätspropagation 34
4.3 Nicht-strukturelle Probleme der Qualitätspropagation 36

5 Die Propagation von Qualität 37
5.1 Das Bilden eines Propagationsgraphen 37
5.2 Qualitätspropagation und Qualitätsaggregation 44
5.2.1 Die Aggregation der lokalen und propagierten Qualität 45
5.2.2 Die Aggregation lokaler und transitiv propagierter Qualität 50
5.2.3 Die Aggregation lokaler und mehrfach propagierter Qualität 53
5.2.4 Der Umgang mit zirkulären Abhängigkeiten 55
5.2.5 Die Aggregationsmethoden im Zusammenspiel 57
5.3 Die Propagation der Sicherheit 59
5.4 Ergebnis 62

6 Diskussion und Ausblick 66
6.1 Konfigurierbarkeit 67
6.2 Genauigkeit 67
6.2.1 Softwarequalitätsmerkmale 67
6.2.2 Andere Aggregationsmethoden 69
6.2.3 Gewichtete Kanten 70
6.2.4 Knotendistanz 71
6.3 Übertragbarkeit 72

7 Fazit 74

Literaturverzeichnis 77

A Anhang 82
A.1 Abhängigkeitsgraph Laravel 82
A.2 Qualitätspropagationsprogramm - Beispielkonfiguration 83
A.3 Propagationsübergangsbaum 84

Glossar 85

Selbstständigkeitserklärung 86

Abbildungsverzeichnis

2.1 Ein Abhängigkeitsgraph – mit den Knoten A, B, C, und D 20
2.2 Direkte und transitive Abhängigkeit in Abhängigkeitsgraphen 22
2.3 Mehrfachabhängigkeit und zirkuläre Abhängigkeit in Abhängigkeitsgraphen 23

3.1 Beispielhafte Ausgabe des Qualitätspropagationsprogramms 29

4.1 Demonstrationsbeispiel – Abhängigkeitsgraph des Softwareprodukts *CalculatorCli* mit den genutzten Softwarebibliotheken 33
4.2 Demonstrationsbeispiel – Screenshot des Softwareprodukts *CalculatorCli* mit einigen Berechnungen 33

5.1 Das Bilden eines Propagationsgraphens 38
5.2 Propagationsgraph mit Zyklus 39
5.3 Beispiel 1 - Entfernen von Kanten um Zyklen zu verhindern 39
5.4 Beispiel 2 - Entfernen von Kanten um Zyklen zu verhindern 40
5.5 Propagationsgraph mit ineinander verschachtelten Zyklen 41
5.6 Das Bilden des Propagationsübergangsbaums (b) aus einem Propagationsgraph (a) mit zwei ineinandergreifende Zyklen 44
5.7 Qualitätspropagation ohne Kanten 45
5.8 Einfache Qualitätspropagation 46
5.9 Einfache Qualitätspropagation mit aggregierter Wartbarkeit 47
5.10 Mehrere Beispiele der Wartbarkeitspropagation 48
5.11 Propagationsgraph mit mehreren direkten Vorgängerknoten 48
5.12 Qualitätspropagation mit mehreren direkten Vorgängerknoten 50
5.13 Propagationsgraph mit einem transitiven Vorgängerknoten 51
5.14 Transitive Qualitätspropagation in zwei Graphen dargestellt 51
5.15 Beispiel der Qualitätspropagation mit transitiver Propagation 53
5.16 Qualitätspropagation mit Mehrfachpropagation 53
5.17 Beispiel der Qualitätspropagation mit Mehrfachpropagation 54

5.18 Beispiel der Qualitätspropagation bei zirklärer Propagation 55
5.19 Propagationsgraph (zirkuläre Propagation). Der entsprechende Propagationsübergangsbaum ist in Anhang A.3 zu sehen. 56
5.20 Demonstrationsbeispiel mit propagierter *Wartbarkeit*. Der entsprechende Propagationsübergangsbaum ist in Abbildung 5.21 zu sehen. 57
5.21 Demonstrationsbeispiel mit propagierter *Wartbarkeit* (Propagationsübergangsbaum) 58
5.22 Propagationsübergangsbaum des Demonstrationsbeispiels, berechnet mit einem Verhältnis von 1 : 1 lokaler zu propagierter Wartbarkeit 59
5.23 Propagationsgraph mit aggregierten Sicherheitsbewertungen 62
5.24 Demonstrationsbeispiel (*Wartbarkeit* und *Sicherheit* propagiert) 63

6.1 Schaubild Querpropagation 69
6.2 Propagationsgraph mit propagierter *Wartbarkeit* 71

A.1 Abhängigkeitsgraph Laravel Skeleton (erstellt mit Graph-Composer) . . . 82
A.2 Propagationsübergangsbaum (mehrfach zirkuläre Propagation) 84

1 Einleitung

Im Jahr 2014 wurde unter der Bezeichnung Heartbleed ein schwerwiegender Sicherheitsfehler in der OpenSSL-Softwarebibliothek öffentlich bekannt gegeben, der es Angreifern ermöglichte, nicht für die Öffentlichkeit bestimmte Daten lesen zu können. Da diese Softwarebibliothek sehr populär war und ist, hatte dieser Programmfehler eine große Tragweite und erregte außerdem viel öffentliches Aufsehen[1]. Es waren viele Softwareprodukte betroffen, die sich von der OpenSSL-Softwarebibliothek abhängig gemacht hatten. Dieses Beispiel zeigt, dass bei der Bewertung der Qualität eines Softwareprodukts nicht nur die lokale, sondern unbedingt auch die „externe“ Qualität der genutzten Softwarebibliotheken beachtet werden muss. Dennoch wird in der Praxis noch wenig Wert auf die Qualitätsbewertung der genutzten Softwarebibliotheken gelegt. Zwar gibt es für die Analyse der Sicherheit von Softwarebibliotheken inzwischen einige Lösungen, wie beispielsweise die Github Security Alerts[2], aber anderen Aspekten der Qualität genutzter Softwarebibliotheken wird weniger Beachtung geschenkt. Das ist erstaunlich angesichts der Tatsache, dass Softwarebibliotheken in der modernen Softwareentwicklung eine bedeutende Rolle spielen. Bloch et al. [5] beschreiben Softwarebibliotheken als „eine organisierte Sammlung von Code mit zugehörigen Werkzeugen zur Unterstützung der Programmierung im Allgemeinen oder in bestimmten Bereichen, die in der Regel durch einen bestimmten Satz von Prinzipien und Konventionen vereint sind“. Im Unterschied zu Programmen sind Softwarebibliotheken keine eigenständig lauffähigen Einheiten. Sie enthalten Hilfsmodule, die von Programmen angefordert werden und sind für die Nutzung durch mehrere Personen und in verschiedenen Umgebungen vorgesehen. Aktuelle Software hängt meist von vielen Softwarebibliotheken ab. Gemäß Gerasimou et al. [15] werden Softwarebibliotheken genutzt, um höhere Modularität, weniger Wartungsaufwand und höhere Zuverlässigkeit zu erreichen. Softwarebibliotheken sind dadurch charakterisiert, dass sie üblicherweise nicht von dem Nutzer selbst entwickelt, sondern von Dritten bereitgestellt werden. So hat der Nutzer einer Softwarebibliothek, anders als bei selbstgeschriebener Software, keinen

[1] `https://www.zeit.de/digital/datenschutz/2014-04/openssl-faq` Zugriff am 19.10.2020
[2] `https://github.com/features/security#security-alert`

direkten Einfluss auf deren Entwicklung. Die vermehrte Nutzung von Softwarebibliotheken erhöht dadurch auch die Abhängigkeit von ihnen und damit auch von den für sie verantwortlichen Entwicklern.

Es hat verschiedene Vorteile, Quellcode durch die Verwendung von Softwarebibliotheken häufig wieder zu benutzen. Nach Mohagheghi et al. [25] kann die Wiederverwendung von Code die Produkteinführungszeit verkürzen, die Softwarequalität verbessern und die Gesamtproduktivität steigern. Mileva et al. [24] stellte schon im Jahr 2009 fest, dass die Mehrzahl aktueller Softwareprodukte stark von der Nutzung externer Softwarebibliotheken abhänge. Deshalb überrascht auch die Beobachtung von Wittern et al. [41] nicht, dass der Paketmanager *npm*[3] für die Laufzeitumgebung *NodeJs*[4] der Programmiersprache *JavaScript* eine stetig wachsende Anzahl an Softwarebibliotheken hat.

In diesem Zusammenhang wird die grundsätzliche These aufgestellt, dass die Softwarequalität einer genutzten Softwarebibliothek Einfluss auf die Qualität einer aktuell genutzten Software hat. Mit den gewünschten übernommenen Funktionen „erbt“ der Nutzer auch die „Qualität“ dieser Bibliotheken. Die Nutzung von Softwarebibliotheken wird deshalb durchaus auch kritisiert. Abdalkareem et al. [1] stellen anhand des von ihnen untersuchten „Ökosystems“ *npm* fest, dass Softwareprodukte in der Programmiersprache JavaScript häufig auf von ihnen als trivial bezeichneten Softwarebibliotheken aufbauen, die wiederum oft viele weitere direkte und transitive Abhängigkeiten haben. Diese Abhängigkeiten bringen mehr Wartungsarbeit für die Entwickler mit sich, wenn es um das Aktualisieren von Softwarebibliotheken oder das Lösen von Konflikten zwischen verschiedenen Softwarebibliotheken geht. Zudem wird es durch die große Anzahl von Softwarebibliotheken, von denen Softwareprodukte abhängen, für Entwickler schwieriger, sich eine Übersicht über die jeweiligen Softwarebibliotheken zu verschaffen, auf denen ihr besonderes Softwareprodukt basiert. Verschärft wird das Problem durch Softwarebibliotheken, von denen ein Softwareprodukt nur transitiv, also nicht direkt, sondern über eine andere Softwarebibliothek abhängig ist.

Trotz ihrer großen Bedeutung werden Softwarebibliotheken bislang in der Regel bei der Qualitätsanalyse kaum berücksichtigt. Es gibt zwar verschiedene Ansätze, die bei der Auswahl von Softwarebibliotheken helfen sollen. Zur Qualitätssicherung fehlt aber eine Möglichkeit zur Bewertung der in einem Softwareprodukt genutzten Softwarebibliotheken. Softwarebibliotheken verdienen aber besondere Beachtung, weil sie als Teil eines

[3] `https://www.npmjs.com`
[4] `https://nodejs.org`

Abhängigkeitsgraphen eines Softwareprodukts betrachtet werden können und somit ihre Qualität an ein anderes Softwareprodukt weitergeben. Gerade die Abhängigkeitsstrukturen werden aber bislang in der Literatur kaum thematisiert. Da Softwarebibliotheken wiederum andere Softwarebibliotheken nutzen können, können die Abhängigkeitsketten immer länger werden. In Anhang A.1 ist, um ein Beispiel zu nennen, der Abhängigkeitsgraph einer Basisapplikation (Skeleton) des PHP-Frameworks Laravel[5] zu sehen. Dieser besteht aus rund 60 Softwarebibliotheken, von denen die Basisapplikation abhängt. Nur 5 dieser Abhängigkeiten sind allerdings direkt, alle weiteren Softwarebibliotheken sind die Abhängigkeiten von anderen Softwarebibliotheken. Das zeigt, dass durch die Verwendung einer Softwarebibliothek oft eine Abhängigkeit von nicht nur dieser, sondern auch vielen weiteren Softwarebibliotheken entsteht. Wenn in der Praxis bei der Auswahl einer Softwarebibliothek überhaupt ihre Qualität betrachtet wird, was nach der Erfahrung des Autors oft nicht der Fall ist, so wird meist nur die Qualität dieser Softwarebibliothek auf der ersten Ebene bewertet. Um eine umfassende Qualitätsaussage treffen zu könnten, sollte allerdings auch die Qualität der direkten Abhängigkeiten der betrachteten Softwarebibliothek miteinbezogen werden, welche wiederum die Qualität ihrer direkten Abhängigkeiten miteinbeziehen sollten und so weiter. So müsste die Qualität durch den Abhängigkeitsgraphen weitergereicht werden, um eine Aussage über die Qualität eines Softwareprodukts machen zu können.

Bestehende Softwarequalitätsmodelle befassen sich nicht mit dem spezifischen Fall "Softwarebibliothek". Ausnahmen dazu bilden Ansätze für die Auswahl von Softwarebibliotheken, welche zwar auch eine Qualitätsbewertung vornehmen, dabei aber nur Teilaspekte der Qualität betrachten (siehe Kapitel 2.1 „Analyse von Softwarebibliotheken"). Eine weitere Ausnahme bildet der Semantic-Web-Ansatz zur Qualitätsbewertung von Eghan [11], der aber auch nur Teilaspekte der Softwarequalität in den Blick nimmt (siehe Kapitel 2.4 „Das Abhängigkeitsmanagement von Eghan"). All diese Ansätze berücksichtigen allerdings nicht die Weiterreichung („Propagation"[6]) von Softwarequalität durch den Abhängigkeitsgraphen.

Diese Arbeit untersucht die Anforderungen an ein Softwarequalitätsmodell für Softwarebibliotheken und entwickelt einen Anforderungskatalog. Dabei liegt der Fokus auf dem in der Literatur bisher vernachlässigten Thema der Propagation von Softwarequalität durch den Abhängigkeitsgraphen.

[5] `https://laravel.com/`

[6] Zur Verwendung des Begriffs der „Propagation" in dieser Arbeit in Abgrenzung zur „Vererbung" vergleiche Kapitel 2.5

Da sich das Vokabular für die Auseinandersetzung mit Softwarequalitätsmodellen für Softwarebibliotheken und der Propagation von Softwarequalität durch den Abhängigkeitsgraphen zumindest teilweise nicht aus der aktuellen Literatur herleiten lässt, neue Aspekte der Softwarequalität besprochen werden und sich dieses Vokabular zum Teil auch mit Begriffen aus anderen Domänen überschneidet, ist dieser Arbeit ein Glossar angehängt, dass diese Begrifflichkeiten zusammenfasst.

1.1 Motivation

Der Autor dieser Arbeit ist regelmäßig an der Entwicklung verschiedener Softwareprodukte beteiligt. Aus seiner Erfahrung im Entwicklungsprozess werden viele qualitätssichernde Maßnahmen eingeführt, die den Quellcode der Softwareprodukte analysieren, doch wird die Qualitätssicherung der Softwarebibliotheken, die in den Softwareprodukten genutzt werden, meist vernachlässigt. Da aber der Quellcode von Softwarebibliotheken Teil des Softwareprodukts ist, sollte auch ihre Qualität analysiert und beschrieben sein, damit sie in die Bewertung des Softwareprodukts einfließen kann - andernfalls können nur partielle Qualitätsaussagen getroffen werden. Diese Arbeit soll sich dieses Problems annehmen.

Als erste Schritte wurden zwei vorbereitende Arbeiten durchgeführt. Im Zuge der ersten Ausarbeitung [40] mit dem Titel „Analyse von PHP-Bibliotheken durch LibraryCheck“ wurde ein Prototyp entwickelt. Dieser untersucht mit verschiedenen Hilfswerkzeugen alle PHP-Bibliotheken, die in einem Softwareprodukt genutzt werden. Die Hilfswerkzeuge erstellen einige Quellcodemetriken zu den Softwarebibliotheken. LibraryCheck verarbeitet die Ergebnisse dieser Analysen allerdings nicht weiter, sondern zeigt sie dem Nutzer lediglich für jede genutzte Softwarebibliothek an.

Bei der Entwicklung der Problemstellung für diese Arbeit wurde auch schnell deutlich, dass für die Erstellung aussagekräftiger Bewertungen die von den Softwareanalysewerkzeugen gemessenen Metriken aggregiert werden müssen. Dafür wurden in einer weiteren Ausarbeitung mit dem Titel „Analyse von Metrik-Aggregation in den Softwareanalysewerkzeugen SonarCloud, Scrutinizer und Better Code Hub“ [39] verschiedene Softwareanalysewerkzeuge untersucht, um nachvollziehen zu können, wie diese gemessenen Metriken zu verwertbaren Informationen weiterverarbeiten beziehungsweise aggregieren. Dabei wurde unter anderem beobachtet, dass die Bewertung der Software bei den Werkzeugen in der Regel nicht vollständig vorgenommen wurde: Es fehlen häufig Qualitätsaspekte. Wenn diese doch vorhanden sind, werden sie zum Teil nicht aussagekräftig gemessen.

Außerdem werden die Softwarebibliotheken bei der Bewertung der Qualität des Softwareprojekts nicht berücksichtigt. Die Arbeit kommt schließlich zu der Schlussfolgerung, dass die folgenden Punkte beachtet werden müssen, wenn eine „vollständige Analyse des Softwareprojekts und eine allgemein aussagekräftige Bewertung“ gewährleistet werden soll:

- Die Analyse muss selbstverständlich alle relevanten Qualitätskriterien mit ausreichenden Metriken messen, sodass aus diesen eine Bewertung für das Qualitätskriterium aggregiert werden kann.
- Damit alle Teile des Softwareprodukts angemessen berücksichtigt werden können, müssen auch die genutzten Bibliotheken in die Bewertung mit aufgenommen werden. Das betrifft direkte und transitive Abhängigkeiten. Diese sind gleichfalls auf ihre Qualität zu analysieren.

In diesem Zusammenhang soll keine vollständige Betrachtung aller Qualitätsaspekte durchgeführt werden. Das besondere Augenmerk muss an dieser Stelle vielmehr auf die Vererbung beziehungsweise Propagation von Qualität durch den Abhängigkeitsgraphen gelegt werden, da dieser Aspekt in der aktuellen wissenschaftlichen Literatur noch nicht ausgiebig behandelt wird, aber offensichtlich wichtig ist für die Qualitätsbewertung von Software. Aus diesem Grund untersucht die vorliegende Arbeit die damit einhergehenden Probleme und erarbeitet Konzepte und Methoden wie die Propagation von Qualität behandelt werden kann.

1.2 Ziele dieser Arbeit

Im Grundsatz geht die vorliegende Arbeit davon aus, dass die Softwarequalität einer bereits bestehenden Softwarebibliothek durch die Einbindung in eine neue Software an diese weitergegeben wird. Diese Form der Qualitätspropagations wird in dieser Arbeit näher untersucht. Es wird gezeigt, wie propagierte Qualität (genutzte Softwarebibliotheken) und lokale Qualität („eigenes“ Softwareprodukt) aggregiert werden kann um eine umfangreichere und aussagekräftigere Qualitätsbewertung zu machen. Dabei wird ein Ansatz entwickelt, der auf dem Abhängigkeitsgraphen aufbaut.

Das Ergebnis dieser Arbeit soll Ansätze zur Entwicklung eines neuen Qualitätsmodells für Softwarebibliotheken ergründen, das die Weitergabe – also Propagation – der Soft-

warequalität in ihren Abhängigkeitsstrukturen beachtet und so eine umfassendere Qualitätsbewertung ermöglicht.

Dabei soll diese Arbeit die folgenden Fragen beantworten:

- Kann die Weitergabe von Softwarequalität über den Weg einer statischen Betrachtung der Abhängigkeiten in der Form von Abhängigkeitsgraphen, nachverfolgt werden?
- Wie können die strukturellen Besonderheiten eines Abhängigkeitsgraphen, also Mehrfachabhängigkeit und Zyklen, in der Qualitätspropagation behandelt werden?
- Welchen Einfluss auf die Propagation hat die Diversität der Softwarequalitätsmerkmale?

Diese Arbeit soll also einen Grundstein für die Forschung zur Softwarequalitätsbewertung legen, die auch die nicht-lokale Softwarequalität beachtet. Dabei wird hier nicht der Anspruch erhoben, ein vollständiges Modell zu entwickeln oder alle Qualitätsmerkmale zu überprüfen. Noch wird versucht, alle Aspekte genau zu untersuchen. Vielmehr überprüft die Arbeit die grundsätzliche Plausibilität eines Ansatzes zur Qualitätsbewertung von Softwarebibliotheken, der die Abhängigkeitsstrukturen umfasst, ohne dabei die Komplexität des Problems einer umfassenden Bewertung von Softwarequalität aus dem Auge zu verlieren.

2 Grundlagen der Softwarequalität von Softwarebibliotheken

Um sich einer Problemlösung zu nähern, ist es zunächst notwendig, zu betrachten, welche Ansätze der Qualitätsanalyse von Softwarebibliotheken in der Literatur beschrieben werden, bevor der Qualitäts- und Softwarequalitätsbegriff genauer untersucht wird und basierend darauf verschiedene Softwarequalitätsmodelle betrachtet werden können. Auf dieser Grundlage wird die konkrete Frage der Vererbung – Propagation – von Softwarequalität durch einen Abhängigkeitsgraphen in den Blick genommen.

2.1 Analyse von Softwarebibliotheken

In der Literatur gibt es verschiedene Ansätze zur Analyse und Bewertung von Softwarebibliotheken, von denen einige hier vorgestellt und bewertet werden. Sie orientieren sich in der Regel nicht an komplexen Softwarequalitätsmodellen, sondern betrachten den Qualitätsaspekt meist nur zum Zweck der Hilfestellung bei der Auswahl einer Softwarebibliothek. Dabei werden allerdings in der Regel nur Teilaspekte der Softwarequalität in den Blick genommen und nur unterschiedlich komplexe Analysen durchgeführt. Diese kann man im Vergleich zu den abstrakteren Softwarequalitätsmodellen auch als eine etwas praxisorientiertere Art der Qualitätsbewertung bezeichnen. Da sich diese Ansätze explizit mit Softwarebibliotheken auseinandersetzen, werden im Folgenden einige dieser Analyseansätze vorgestellt.

Bei der Auswahl von Softwarebibliotheken geht es natürlich zunächst immer darum, eine für das Problem beziehungsweise die Aufgabenstellung passende Softwarebibliothek auszuwählen. Mit diesem Aspekt beschäftigt sich zum Beispiel Thung [35] in seiner Forschung. Dieser Ansatz ist eher nutzungsorientiert, und das ist in Hinsicht auf die Qualitätsbewertung, die hier problematisiert wird, weniger relevant.

Um über die Nutzbarkeit hinaus auch die Qualität einer Softwarebibliothek bei der Auswahl zu berücksichtigen, schlagen Mileva et al. [24] beispielweise einen Ansatz zur Auswahl der Version einer Softwarebibliothek nach dem Grad der Popularität vor. Man solle sich immer für die meist genutzte Version einer Softwarebibliothek entscheiden, auch wenn diese nicht die neuste Version darstelle. Dieser Ansatz ist einfach umzusetzen, und das macht ihn für den Nutzer grundsätzlich interessant. Er basiert allerdings auf der Annahme, dass die populärste Version einer Softwarebibliothek gleichzeitig immer auch die qualitativ beste sei. Wie Eghan [11] zu Recht kritisiert, kann aber eine Entscheidung nicht allein auf der Grundlage von Popularität basieren, da diese kein Garant für die Qualität sein kann. Es gibt viele andere mögliche Gründe, warum eine Software populär sein könnte. Auch die populärste Version einer Softwarebibliothek könnte beispielsweise bekannte Sicherheitslücken enthalten, die in einer aktuelleren Version schon behoben worden sind. Entscheidungen über den Einsatz von Softwarebibliotheken müssen sich also auf eine komplexere Analyse der Softwarebibliotheken stützen.

In dieser Hinsicht bieten Raemaekers et al. [29] Ansatzpunkte. Sie betrachten die Möglichkeit, die Stabilität einer Softwarebibliothek zu bestimmen, indem ihr Verlauf analysiert wird. Im Gegensatz zu dem Ansatz von Mileva et al. werden dazu vier für die Stabilität relevante Quellcode-Metriken präsentiert. Mit diesen Metriken soll die Stabilität einer Softwarebibliothek bewertbar sein. Eine höhere Stabilität einer Softwarebibliothek bedeutet demnach weniger Wartungsaufwand für den Nutzer, da dieser sein Softwareprodukt nicht bei jedem Update der Softwarebibliothek neu anpassen muss.

Positiv an diesem Ansatz von Raemaekers et al. ist, dass Quellcode-Metriken genutzt werden und nicht wie bei dem Ansatz von Mileva et al. nur von außen feststellbare Metriken. Raemaekers et al. geben allerdings selbst und mit Recht zu bedenken, dass es fraglich sei, ob diese rein auf historischen Daten basierende Bewertung auch auf die zukünftige Stabilität einer Softwarebibliothek übertragbar sei.

Eine umfangreichere Analyse zur Auswahl einer passenden Softwarebibliothek findet sich bei López de la Mora und Nadi [27, 26]. Ihr Ansatz basiert auf acht verschiedenen Qualitätsmerkmalen, die zum Teil auf Produkt- und zum Teil auf Prozessmetriken basieren. Sie nutzen neben der *Popularität* zusätzlich noch die *Release-Häufigkeit*, *Zeiten für die Beantwortung und Schließung von Tickets*, *Aktualität*, *Migration*, *Fehleranfälligkeit*, *Leistung* und *Sicherheit*. Die vorgeschlagene Analyse ist zwar umfangreicher und komplexer, allerdings werden zur Bestimmung einzelner Qualitätsmerkmale Metriken genutzt, die nicht auf der Grundlage des Quellcodes beruhen. Sie werden vielmehr aus verschiedenen

Quellen gesammelt und gemeinsam verwendet. Dabei werden beispielsweise Bugreports analysiert, um daraus die Leistung oder *Sicherheit* zu bestimmen. So ist diese Form der Analyse zwar nicht auf quelloffene Bibliotheken angewiesen, aber die Ergebnisse sind weniger aussagekräftig. Es könnte beispielsweise sein, dass aus der Nutzung einer Softwarebibliothek mit geringerer Popularität weniger Bugreports mit Bezug auf Sicherheitslücken resultieren. Die Analyse des Quellcodes könnte aber gegebenenfalls zur Auffindung potenzieller Sicherheitslücken führen. Es ist aber positiv anzumerken, dass diese Analyse auch Metriken verwendet, mit denen die Qualität der Community abgebildet werden soll.

Die praxisorientierten Ansätze haben gemeinsam, dass sie nicht auf einem allgemeinen Qualitätskonzept aufbauen. Es handelt sich vielmehr um theoretisch anspruchslose Ad hoc-Ansätze zur pragmatischen Lösung der Frage, welche Softwarebibliothek sich für ein vorliegendes Softwareprodukt eignet. Dabei können die Auswirkungen, die die Softwarebibliothek auf die lokale Software hat, aber nur lückenhaft oder gar nicht erfasst werden. Eine komplexere Berücksichtigung der Vielzahl von Qualitätsmerkmalen, in denen sie die lokale Software beeinflussen, ist auf dieser Grundlage nicht möglich.

Gleichzeitig zeigen diese praxisorientierten Ansätze zur Hilfe bei der Auswahl von Softwarebibliotheken, dass die Aspekte der Stabilität einer Softwarebibliothek und auch die Qualität der Community rund um die Softwarebibliothek besondere Aufmerksamkeit erhalten. Ein Modell zur Bewertung von Softwarebibliotheken sollte diese auch entsprechend berücksichtigen.

2.2 Qualitäts- und Softwarequalitätsbegriff

Um die Merkmale und Formen beschreiben und bewerten zu können, durch die sich die Einbindung einer Softwarebibliothek auf die lokale Software auswirkt, ist ein komplexeres Qualitätsmodell erforderlich. Dazu lassen sich Ansätze aus der informatikwissenschaftlichen Forschung zur Messung der Qualität von Software heranziehen.

Ganz allgemein betrachtet ist „Qualität“ ein komplexes und facettenreiches Konstrukt und Produkt eines konsensualen Prozesses. Vor diesem Hintergrund wird beispielsweise in der wirtschaftswissenschaftlichen Literatur zum Qualitätsbegriff die Position vertreten, dass es nur einen subjektiven Begriff von „Qualität“ geben kann und es mangels einer allgemeingültigen Definition häufig zu Missverständnissen kommt [14]. In seinem Buch

„Quality is Free“ entwickelt Crosby [8] einen Begriff von Qualität, der diese lediglich als die Erfüllung von Anforderungen versteht. Da Anforderungen und Erwartungen allerdings von einem subjektiven Blickwinkel abhängen, sieht er Qualität als nicht allgemeingültig messbar an. Ähnlich definiert Deming [9] aus ingenieurwissenschaftlicher Perspektive Qualität als den Grad, zu dem die Erwartungen der Anwender erfüllt sind. Allgemein gefasst kann Qualität demnach grundsätzlich in Abstufungen als Grad der Erfüllung von Anforderungen oder Erwartungen dargestellt werden.

Die Schwierigkeit, in benachbarten wissenschaftlichen Feldern einen konkreten und abstrakten Begriff von Qualität zu entwickeln, spiegelt sich auch in der informatikwissenschaftlichen Literatur: Die Qualität von Software ist nicht absolut zu definieren. Dies gilt nicht nur für den Spezialfall von Softwarebibliotheken, sondern allgemein gefasst für jegliche Software. Dennoch wird der Versuch unternommen, bewertende Aussagen zur Softwarequalität zu treffen. Wagner [37] unterscheidet zwischen zwei Arten von Qualität in Bezug auf die Softwareentwicklung: Prozessqualität und Produktqualität. Im Rahmen dieser Arbeit kann die Prozessqualität hier nicht weiter untersucht werden. Vor dem Hintergrund des Vorhabens, die Qualität von Softwarebibliotheken zu beschreiben, liegt der Fokus hier vielmehr auf der Produktqualität.

Produktqualität von Software wird von einem aktuellen ISO-25010-Standard [18] beschrieben als „Grad, in dem das System die erklärten und implizierten Bedürfnisse seiner verschiedenen Interessengruppen befriedigt und somit einen Wert schafft“. Auch diese Definition bleibt vage, eröffnet aber doch in dieser groben Definition die Möglichkeit, eventuelle Bedürfnisse einer oder mehrerer Interessengruppen zu beschreiben und dadurch einen Kanon von Qualitätskriterien zu entwickeln, die transparent präsentiert werden können. Da der Qualitätsbegriff und so auch der Softwarequalitätsbegriff also nicht generell zu definieren sind und immer von dem Betrachter und der Domäne abhängen, kann auch kein Softwarequalitätsmodell entwickelt werden, das grundsätzlich allgemeingültig ist. Das Bedürfnis nach Hilfestellungen zur Bewertung von Softwarequalität besteht aber fort. Auch deshalb führt die ISO-25010 abseits dieser grundsätzlichen Erwägungen zur Subjektivität von Qualität detaillierte Kriterien zur Qualitätsbewertung ein und eröffnet damit zugleich die Frage nach der Gewichtung einzelner Kriterien. Immerhin wird hiermit die individuelle Entscheidung des Nutzers ermöglicht, sich diesen Kriterien anzuschließen und unter diesen Bedingungen zu einer Bewertung zu kommen.

Ein möglichst aussagekräftiges Softwarequalitätsmodell sollte also viele Merkmale der Softwarequalität transparent beachten und bewerten, damit die verschiedenen Blickwin-

kel aller Stakeholder beachtet werden und für die konkreten Bedürfnisse im Einzelfall angepasst werden können. Diese können sich dann auf die für sie wichtigen Qualitätsmerkmale konzentrieren.

Der Darstellung von Softwarequalität in einem strukturierten Rahmen dienen Softwarequalitätsmodelle. Laut dem ISO-Standard 9126-1 [19] ist ein solches Modell „die Menge der Merkmale und die Beziehungen zwischen ihnen, die die Grundlage für die Festlegung von Qualitätsanforderungen und die Bewertung bilden“. Über die Jahre sind verschiedene derartige Softwarequalitätsmodelle entstanden, die helfen sollen, Softwarequalität besser zu beschreiben. Miguel et al. [28] teilen sie auf in „Basismodelle“ und „zugeschnittene Modelle“. Erstere beschreiben Modelle, „deren Ziel die vollständige und umfassende Produktevaluierung ist“. Unter zugeschnittenen Softwarequalitätsmodellen hingegen verstehen sie solche, die einer spezifischen Domäne der Softwareentwicklung angepasst sind. Open-Source-Software-Qualitätsmodelle wiederum bestimmen sie als eine wichtige Kategorie dieser zugeschnittenen Modelle.

2.3 Softwarequalitätsmodelle

Zur Bewertung von Software gibt es neben den oben beschriebenen praxisorientierten Ansätzen auch theoriebasierte Ansätze. Im Folgenden werden die theoretisch orientierten Ansätze der Softwarequalitätsmodelle betrachtet. Hier unterscheidet die aktuelle wissenschaftliche Literatur zwischen Basismodellen und zugeschnittenen Modellen. Erstere sind grundlegende Softwarequalitätsmodelle, die kein spezifisches Einsatzgebiet haben. Darüber hinaus gehende, auf Domänen zugeschnittene, Softwarequalitätsmodelle sind dagegen für besondere Einsatzgebiete entwickelt. Auch diese sind hier von Interesse und werden in diesem Zusammenhang besonders als Softwarequalitätsmodelle für das spezifische Einsatzgebiet der Open Source Software (OSS) näher betrachtet. Diese Softwarequalitätsmodelle sind allerdings nicht speziell für Softwarebibliotheken entwickelt worden.

2.3.1 Basismodelle

Basismodelle zur Analyse von Softwarequalität sind laut Miguel et al. [28] und Wagner [37] hierarchisch strukturiert. Sie können an jede Art von Softwareprodukt angepasst werden und sind auf Bewertung und Verbesserung ausgerichtet. Sie sind außerdem nicht

auf eine bestimmte Domäne zugeschnitten und bilden so eine breite Basisdefinition der Softwarequalität.

Bereits in den Jahren 1977 und 1978 veröffentlichen McCall [23] und Boehm [6] jeweils ein Softwarequalitätsmodell. Beide ähneln sich und werden über einen Qualitätsbaum definiert. Dabei steht die Eigenschaft „Qualität“ an der Spitze, und diese unterteilt sich in mehrere Kategorien. Der 1991 erschienene Standard ISO-9126 [19] beschreibt ebenfalls ein Softwarequalitätsmodell und basiert laut Fenton und Biemann [12] im Grunde auf dem McCall-Modell. Im Jahr 2011 wurde der ISO-9126 Standard durch den aktuellen, hier bereits mehrfach angesprochenen Standard ISO-25010 [18] ersetzt. Der neuere Standard ist zwar eine Weiterentwicklung des Vorgängers, aber im Grundsatz hat sich nicht viel verändert. Das neue Modell betrachtet im Gegensatz zum ISO-9126 Vorgänger-Standard als neue Merkmale die Aspekte *Sicherheit* und *Kompatibilität*. Während die *Sicherheit* im ISO-9126 Standard ein Untermerkmal der Funktionalität ist, bildet sie im ISO-25010-Standard ein eigenes Hauptmerkmal.

Das ISO-25010 Modell ist in einer hierarchischen Struktur aufgebaut, die die „Qualität“ in „Merkmale“ unterteilt, die aus „Untermerkmalen“ und wiederum aus „Unter-Untermerkmalen“ bestehen können. Wagner [37] beschreibt dieses Konzept als eine Taxonomie, die das komplexe Konzept der Qualität von Softwareprodukten in kleinere, hoffentlich handhabbare Teile zerlege. Die Idee dahinter ist, dass die Partition ein Niveau erreicht, auf dem die Teile messbar werden und zur Bewertung der Qualität des Softwareprodukts verwendet werden können. Die acht Hauptmerkmale sind:

1. Functional suitability (Funktionale Eignung)
2. Performance efficiency (Leistungseffizienz)
3. Compatibilty (Kompatibilität)
4. Usability (Benutzerfreundlichkeit)
5. Reliability (Zuverlässigkeit)
6. Security (Sicherheit)
7. Maintainability (Wartbarkeit)
8. Portability (Portabilität)

Laut Wagner geht der ISO-25010-Standard davon aus, dass unter Verwendung dieser hierarchischen Struktur ein Qualitätsmodell aufgebaut werden kann. Dennoch betone der Standard, dass nicht alle Merkmale in jeder Art von Software relevant seien. Er gebe jedoch auch keine Hilfestellung, wie das Qualitätsmodell angepasst werden könne.

Nach dem ISO-25010-Standard teilt sich die Softwarequalität also in verschiedene Softwarequalitätsmerkmale. Diese Arbeit untersucht die Propagation der Softwarequalitätsmerkmale im Kontext der Nutzung von Softwarebibliotheken auf Grundlage des ISO-Basismodells. Die Qualität eines Softwareprodukts wird dabei nicht in einer einzelnen Bewertung aggregiert, sondern wird in dieser Arbeit als ein Softwarequalitätsvektor ($Q(v)$) verstanden, der sich aus den Qualitätsbewertungen der einzelnen Softwarequalitätsmerkmale zusammensetzt. So kann die „Gesamtqualität“ einzelner Softwareprodukte zwar nicht so einfach verglichen werden, der Betrachter kann dafür aber eine differenziertere und auf seine Ansprüche angepasste Einschätzung der Qualität erlangen.

Im Rahmen dieser Arbeit kann dabei nicht näher betrachtet werden, wie ein solcher Qualitätsvektor für die lokale Softwarequalität ($Q_{lokal}(v)$) ermittelt wird. Die lokale Softwarequalität bezeichnet die Softwarequalität eines Softwareprodukts, die die propagierte Softwarequalität nicht einbezieht. Mit der Frage wie innerhalb eines Knotens also einer Softwarebibliothek oder eines Softwareprodukts, aggregiert werden kann, hat sich eine vorhergegangene Ausarbeitung (siehe [39]) beschäftigt. Die vorliegende Arbeit beschränkt sich stattdessen darauf, auf der Grundlage bereits bestehender Softwarequalitätsbewertungen zu arbeiten.

In der Praxis könnte es aufgrund der Diversität der Softwarequalitätsmerkmale sinnvoll sein, unterschiedliche Skalen für die verschiedenen Softwarequalitätsmerkmale zu nutzen. Der Einfachheit halber wird in dieser Arbeit aber von einer für alle Softwarequalitätsmerkmale geltende Skala von $0,00$ bis $10,00$, ausgegangen. Wobei jeweils $0,00$ die schlechteste und $10,00$ die beste Bewertung darstellt. Das heißt, jede Softwarebibliothek beziehungsweise jeder Knoten hat bereits eine Bewertung aller zu betrachtenden Softwarequalitätsmerkmale auf dieser Skala.

Im Folgenden werden die Softwarequalitätsmerkmale der ISO-25010 näher erläutert.

Die *Funktionale Eignung* wird laut Rodríguez et al. [30] in dem ISO-25010-Standard definiert als der Grad, in dem ein Produkt oder System Funktionen bereitstellt, die die angegebenen oder impliziten Anforderungen erfüllen. Die Bewertung der funktionalen Eignung wird also laut Rodríguez et al. als der Grad verstanden, in dem ein Produkt

oder System den in der Produktanforderungsspezifikation beschriebenen funktionalen Anforderungen entspricht, da es laut den Autoren nicht möglich ist, die impliziten Anforderungen des Nutzungskontextes zu kennen.

Laut ISO-25010 [18] bezeichnet die *Leistungseffizienz* die „Leistung im Verhältnis zur Menge der unter den angegebenen Bedingungen verwendeten Ressourcen“. Laut Wagner [37] bildet das Merkmal vereinfacht ab, wie gut das Produkt auf Benutzeranfragen reagiert und wie effizient es in seiner Ausführung ist.

Heutige Softwareprodukte arbeiten laut Wagner [37] selten in einer isolierten Umgebung, und daher definiert die *Kompatibilität* die Eigenschaft, dass ein Produkt nicht stört oder sogar mit anderen Produkten zusammenarbeiten kann. Dieses Merkmal ist besonders im Kontext der Softwarebibliotheken relevant, da Kompatibilitätsprobleme die Auswahl möglicher Softwarebibliotheken stark einschränken kann.

Die *Benutzerfreundlichkeit* wird definiert als der Grad, in dem ein Produkt oder System von bestimmten Benutzern verwendet werden kann, um bestimmte Ziele wie Wirksamkeit, Effizienz und Zufriedenheit in einem bestimmten Nutzungskontext zu erreichen. Laut Wagner [37] gehört unter anderem die Frage dazu, wie schnell die Benutzung erlernt werden kann und ob das Interface attraktiv ist.

Die *Portabilität* beschreibt nach Wagner [37], wie einfach es ist, ein Softwareprodukt auf eine andere Plattform (Programmiersprache, Betriebssystem, Hardware) zu bringen. Das bedeutet, dass die notwendigen Änderungen leicht durchgeführt werden können und das Produkt leicht installiert werden kann.

Der ISO-25010-Standard [18] definiert die *Wartbarkeit* als den Grad der Effektivität und Effizienz, mit dem ein Produkt oder System von den vorgesehenen Betreuern modifiziert werden kann. Nach Wagner [37] beschreibt die *Wartbarkeit*, ob ein Softwareprodukt für einen Softwareentwickler leicht zu verstehen, zu ändern und zu testen ist. Laut Fenton und Bieman[12] setzt sich die *Wartbarkeit* daraus zusammen, wie leicht eine Software zu verstehen, zu verbessern oder zu korrigieren ist.

Zur *Sicherheit* gehört laut Wagner [37], dass Daten intakt und geheim gehalten werden und dass ein Produkt sicherstellen können muss, dass seine Benutzer diejenigen sind, für die sie sich ausgeben. Der ISO-25010-Standard [18] beschreibt die *Sicherheit* als den Grad, in dem ein Produkt oder System Informationen und Daten schützt, so dass Personen, andere Produkte oder Systeme den ihren Arten und Berechtigungsstufen entsprechenden Grad des Datenzugriffs haben.

Probleme der *Zuverlässigkeit* sind nach Wagner [37] - zusammen mit der Leistungseffizienz - in der Regel die am unmittelbarsten auftretenden Probleme der Benutzung eines Softwareprodukts. Er spricht von einer schlechten Zuverlässigkeit, wenn dieses Ausfälle produziert und daher für den Benutzer möglicherweise nicht ausreichend verfügbar ist.

Diese acht Merkmale bilden die Grundlagen zur Bewertung der Qualität von Software jeglicher Art. Sie geben einen sinnvollen Rahmen für Qualitätsmessungen, sind in ihrer Allgemeinheit aber häufig nicht an die spezifischen Anforderungen und Bedürfnisse des jeweiligen Softwareprodukts angepasst. Vor diesem Hintergrund werden – aufbauend auf Basismodellen – zugeschnittene Modelle entwickelt.

2.3.2 Zugeschnittene Modelle und Open-Source-Software-Modelle

Laut Miguel et al. [28] ergeben sich die zugeschnittenen Modelle aus dem Bedarf der Softwareindustrie an spezifischen Softwarequalitätsmodellen. Sie werden auf der Grundlage der Basismodelle durch das Hinzufügen oder Modifizieren von Unterfaktoren und mit dem Ziel entwickelt, die Bedürfnisse einzelner Domänen oder spezialisierter Anwendungen zu erfüllen. Sie gründen sich nach Miguel et al. [28] meist auf Basismodellen, oft noch auf der Grundlage des ISO-9126-Modells. Gleichzeitig sollten neu zugeschnittene Modelle aber auf dem umfassendsten ISO-25010-Modell aufbauen, denn in einem Vergleich verschiedener Basismodelle bewerten Miguel et al. das ISO-25010-Software-Qualitätsmodell als das umfassendste. Da Softwarebibliotheken häufig Open-Source sind, sind für das Vorhaben dieser Arbeit vor allem die auf die Bewertung von OSS zugeschnittenen Modelle interessant. Diese sind nach Miguel et al. [28] eine besonders interessante Kategorie zugeschnittener Softwarequalitätsmodelle, da OSS immer relevanter werde. Die Qualitätsmodelle für OSS zeigen, dass hier weitere Qualitätsmerkmale wichtig werden, die nicht Teil der Basismodelle sind.

Ein wesentliches Unterscheidungsmerkmal von OSS sei es, so schreiben Haaland et al. [16], dass es von einer Community entwickelt und gewartet werde. Da der Nutzer von OSS im Normalfall keinen direkten Einfluss auf die Entwicklung dieses Softwareprodukts hat, muss dieser der Community rund um das Softwareprodukt vertrauen können. Damit sind in erster Linie die Entwickler der Software angesprochen. Entsprechend enthalten viele OSS-Qualitätsmodelle zusätzliche Merkmale rund um die Community der Software, um beispielsweise sicherzustellen, dass die Community einer OSS im Falle einer Sicherheitslücke diese schnell beheben kann oder um abschätzen zu können, ob die Community

in der Lage ist, das Softwareprodukt über einen längeren Zeitraum zu pflegen. Unterschiedliche Qualitätsmerkmale wurden bereits in diesem Zusammenhang definiert: Soto und Ciolkowski [34] beschreiben ein Qualitätsmerkmal zur *Qualität von Communities*. Dieses enthält als Unterkategorien *Instandhaltungskapazität*, *Nachhaltigkeit* und *Prozessreife*. Das Modell *SQO-OSS* von Samoladas et al. [31] aus dem Jahr 2008 hingegen misst die *Qualität der Mailingliste*, die *Qualität der Dokumentation* und die *Qualität der Entwicklerbasis*.

Das *EFFORT* Qualitätsmodell zur Bewertung der Qualität und Funktionalität von OSS-Systemen von Aversano und Tortorella aus dem Jahr 2013 [3] fügt zwei Kategorien hinzu, die nicht Teil des ISO-25010-Standards sind: *Vertrauenswürdigkeit der Community* und *Produktattraktivität*. Die erstgenannte Kategorie listet als Untermerkmale: *Entwickler*, *Gemeinschaftsaktivitäten*, *Support-Tools*, *Unterstützungsdienste* und *Dokumentation*. Die Kategorie *Produktattraktivität* führt die Untermerkmale an: *funktionale Angemessenheit*, *Verbreitung*, *Kostenwirksamkeit* und *rechtliche Wiederverwendbarkeit*.

Miguel et al. [28] schließen ihren Vergleich von OSS-Qualitätsmodellen aus dem Jahr 2014 mit der Beobachtung, dass im Falle von OSS die besonderen Anforderungsaspekte der Communities als ein wichtiges Merkmal betrachtet werden sollten, da der Grad des Einflusses der Community sowohl bei der Konstruktion als auch bei der Produktakzeptanz hoch ist. Zwei Jahre später, im Jahr 2016, haben Adewumi et al. [2] ebenfalls OSS-Qualitätsmodelle verglichen. Dabei beschreiben sie unter anderem, dass rund 50% der 19 verglichenen OSS-Qualitätsmodelle keine Qualitätsmerkmale der Community enthielten, obwohl dies nach ihrer Einschätzung das größte Unterscheidungsmerkmal von OSS zu proprietärer Software sei. Außerdem bildeten auffallend viele OSS-Qualitätsmodelle die Basismodelle nur teilweise ab.

Vor diesem Hintergrund erscheint für den Spezialfall von Open-Source Softwarebibliotheken ein Modell sinnvoll, das auf der Grundlage des ISO-Basismodells ein auf die Bewertung von Open-Source Softwarebibliotheken zugeschnittenes Modell entwickelt. Die Identifikation relevanter Qualitätsmerkmale reicht für die Entwicklung eines Qualitätsmodells für Softwarebibliotheken aber nicht aus. Ein Modell zur Bewertung der Qualität von Softwarebibliotheken muss darüber hinaus noch einer weiteren Besonderheit Rechnung tragen. Denn bei der Einbindung von Softwarebibliotheken kommt es nicht nur auf die Eigenschaften der Softwarebibliothek selbst an, sondern es stellen sich komplexe Fragen dazu, wie sich die Qualität der eingebundenen Bibliothek auf die „lokale“ Qualität

überträgt. Durch die Nutzung einer Softwarebibliothek entstehen komplexe Abhängigkeiten, die sich in unterschiedlichem Maße auf die lokale Qualität auswirken.

2.4 Das Abhängigkeitsmanagement von Eghan

Im Gegensatz zu den zuvor vorgestellten, insgesamt unbefriedigenden Ansätzen für die Auswahl von Softwarebibliotheken erkennt Eghan [11], dass aufgrund der Abhängigkeitsbeziehungen zwischen Softwareprojekten und kompletten Software-Ökosystemen ein übergreifender Ansatz gefunden werden muss, der diese Abhängigkeiten beachtet. In seiner Doktorarbeit aus dem Jahr 2019 beschreibt er „eine technologie-unabhängige Darstellung der Semantik der Software-Abhängigkeiten, die nahtlos mit anderen Software-Artefakten integriert ist, um die Informationen zur Projektabhängigkeit in Software-Aufgaben zu nutzen“ [11]. Dafür möchte er eine einheitliche Wissensrepräsentation von Abhängigkeitsrepositories etablieren. Auf der Grundlage dieser Repräsentation soll die Wissensbasis mit weiteren Ressourcen aufgewertet werden. Das Ergebnis soll ein flexibler Ansatz zur globalen Wirkungsanalyse sein.

Die Arbeit nennt mehrere Komponenten, die zusammen genommen dieses Ziel erreichen sollen. Dafür entwickelt Eghan im ersten Schritt ein *Software Build System Ontology (SBSON)* genanntes Modell, das auf dem Semantic Web, einer maschinenlesbaren Erweiterung des World-Wide-Web, basiert. Das soll einerseits dabei helfen, bidirektionale Abhängigkeitsanalysen durchzuführen, die es dem Betreuer einer Softwarebibliothek ermöglichen zu analysieren, in welchen Softwareprojekten seine Softwarebibliothek genutzt wird. Dadurch kann beispielsweise abgeschätzt werden, ob ein „Breaking Change“ große Auswirkungen auf die Softwareprodukte hat, die diese Bibliothek nutzen. Außerdem soll das Modell andererseits ermöglichen, unterschiedliche Artefakte zusammen zu betrachten, beispielsweise Quellcode-Metriken und Sicherheitsdatenbanken. Dieses Zusammenführen von Informationen aus verschiedenen Artefakten ermöglicht es, einfacher zu Aussagen zu gelangen, die auf der Basis mehrerer miteinander verknüpfter Artefakte basieren. Auf dieser Grundlage entwickelt er außerdem mit *OntTAM* ein weiteres Modell, das die Vertrauenswürdigkeit von Softwareprodukten beziehungsweise Softwarebibliotheken beschreiben soll. Es basiert auf dem Meta-Modell *SE-EQUAM* von Hmood et al. [17].

Der Ansatz von Eghan kann in verschiedener Hinsicht überzeugen. Durch den Bezug auf das Semantic-Web ergeben sich viele Möglichkeiten in der Nutzung. Vor allem im Zusam-

menhang mit der hier verwendeten Abfragesprache SPARQL. Und das darauf aufbauende *OntTAM* Modell bietet eine gute Möglichkeit, Softwarebibliotheken auf der Grundlage verschiedener Artefakte zu bewerten und auch mit der Abfragesprache SPARQL die Abhängigkeiten in der Analyse mit zu betrachten. In dem motivierenden Beispiel für das *OntTAM* Modell beschreibt Eghan, wie ein fiktiver Softwareentwickler versucht, externe Bibliotheken wiederzuverwenden, während er bei der Auswahl mit mehreren Herausforderungen konfrontiert ist und gleichzeitig darauf bedacht ist, deren negative Auswirkungen auf die Vertrauenswürdigkeit seines eigenen Projekts zu verringern. Obwohl Eghan die Propagation von Qualität (beziehungsweise Vertrauen) in diesem Beispiel anspricht, betrachtet er in seinem *OntTAM* Modell nicht die Propagation von Qualität durch den Abhängigkeitsgraphen, sondern lediglich die Verträglichkeit der einzelnen Lizenzmodelle aller Softwarebibliotheken in einem Softwareprodukt. Obwohl die Besonderheiten des Semantic-Web-Ansatzes diesen als Grundlage für diese Überlegungen der Qualitätspropagation geeignet erscheinen lassen, geht Eghan nicht weiter darauf ein.

Bei der Propagation von Qualität durch den Abhängigkeitsgraphen sind aber einige Eigenheiten und Besonderheiten explizit zu untersuchen. Es stellt sich unter anderem die Frage, wie Mehrfachabhängigkeiten oder zirkuläre Abhängigkeiten behandelt werden sollten, ein Problem, dem sich Eghan in seiner Arbeit nicht stellt.

2.5 Propagation/Vererbung von Softwarequalität durch einen Abhängigkeitsgraphen

Zur Beschreibung der Weiterreichung von Qualität von Softwarebibliotheken könnte der Begriff „Vererbung“ verwendet werden, da ein Softwareprodukt die Qualität seiner genutzten Softwarebibliotheken „erbt“. Die lineare Struktur, die diesem Begriff zugrunde liegt, kann aber komplexere Zusammenhänge der Weitergabe nicht erfassen. Zudem ist der Begriff in der Softwareentwicklung bereits in der Polymorphie geprägt[1].

Besser geeignet ist daher der Begriff der „Propagation“. Dieser leitet sich von dem lateinischen Verb „propagare“ (ausbreiten, ausdehnen) ab. In der Naturwissenschaft (Chemie) bezeichnet er eine Kettenreaktion. Der Begriff eignet sich, weil er den Prozess der „Weiterreichung“ gut beschreibt und dabei im Gegensatz zu dem Begriff der Vererbung, weder

[1] Vergleiche zum Beispiel [13, 22, 10]

im alltäglichen Sprachgebrauch noch in der Informatik bereits stark an andere Sachverhalte gebunden ist. Im weiteren Verlauf dieser Arbeit wird also von der „Propagation von Qualität“ gesprochen.

2.5.1 Die strukturelle Abhängigkeit

Es gibt verschiedene Möglichkeiten eine Software zu betrachten. Der Kontrollfluss bezeichnet die Reihenfolge, in der einzelne Anweisungen, Befehle oder Funktionsaufrufe eines Softwareprodukts ausgeführt werden. Auf der Grundlage dieses Kontrollflusses können Analysen durchgeführt werden, die beispielsweise unerreichbaren Quellcode identifizieren können. Über einen solchen Kontrollfluss können außerdem sehr viel genauere Aussagen über Abhängigkeiten gemacht werden. In dieser Arbeit wird allerdings nur die strukturelle Abhängigkeit zwischen Softwareprodukten und Softwarebibliotheken betrachtet und der Kontrollfluss weitgehend außer Acht gelassen. Die strukturellen Abhängigkeiten sind einfach zu ermitteln und in einem einfachen Abhängigkeitsgraphen abzubilden. Gleichzeitig können feingranulare Analysen nicht durchgeführt werden, bei denen die Propagation von Qualität über den Kontrollfluss betrachtet wird. Da eine an den Kontrollfluss angelegte Qualitätspropagation auch in der Softwareanalyse der lokalen Softwarequalität nicht beobachtet werden konnte (vergleiche [39]) wird diese Art der Qualitätspropagation auch nicht in dieser Arbeit betrachtet.

Hier wird die Abhängigkeit von Softwarebibliotheken nur strukturell behandelt, weil diese Form der Betrachtung die Grenzen der selbst- und fremdbestimmten Zuständigkeitsbereiche gut abbildet. Es kann also eine klare Abhängigkeitsstruktur betrachtet werden, in der die Grenzen zwischen lokaler Software und den verschiedenen genutzten Softwarebibliotheken deutlich gezogen werden kann. So entsteht eine für den Nutzer verständliche und nachvollziehbare Qualitätspropagation, und auszutauschende Softwarebibliotheken sind einfacher zu identifizieren. Das heißt, ein Softwareprodukt kann von einer Softwarebibliothek abhängen, der genaue Grad der Stärke dieser Abhängigkeit, also die Kopplung, wird aber in dieser Arbeit nicht unterschieden, um die Abhängigkeitsstruktur so simpel wie möglich zu halten. Unterschiedliche Abhängigkeiten können also nicht gewichtet werden. Das bedeutet auch, dass Unterscheidungen, welche Teile einer Softwarebibliothek genutzt werden und welche Teile ungenutzt bleiben oder die Frage nach dem Intensitätsgrad der Kopplung an diese Softwarebibliothek, in dieser Abhängigkeitsstruktur ebenfalls nicht abgebildet werden können.

Die betrachtete und hier wiedergegebene Abhängigkeitsstruktur sagt also nur aus, ob zwei Elemente in Abhängigkeit zueinander stehen und in welche Richtung diese Abhängigkeit besteht. Der Qualitätspropagation liegt in dieser Arbeit also auch keine Gewichtung zugrunde, welche Softwarebibliothek „besonders stark“ und welche „weniger stark“ propagiert.

2.5.2 Der Abhängigkeitsgraph

Zur Darstellung der Dependenzen eignet sich der Abhängigkeitsgraph. Ein solcher kann auf verschiedenen Ebenen erstellt werden. Er kann Abhängigkeiten von Klassen oder Modulen untereinander abbilden, solche zwischen verschiedenen Microservices beschreiben oder sich auch auf die Softwarebibliotheken beziehen, die ein Softwareprodukt nutzt. Im Zusammenhang mit Softwarebibliotheken wird in dieser Arbeit ausschließlich ein Abhängigkeitsgraph definiert, der Abhängigkeiten zu Softwarebibliotheken abbilden kann.

Das hier verwendete Modell des Abhängigkeitsgraphen besteht aus Knoten und gerichteten Kanten. Erstere entsprechen dabei den Softwarebibliotheken beziehungsweise dem Softwareprodukt. Die gerichteten Kanten zeigen die Abhängigkeitsbeziehungen an (siehe Abbildung 2.1). Die Abhängigkeitsbeziehung wird durch die Pfeilrichtung gezeigt. In diesem Beispiel hängt „A“ von „B“ und „C“ ab. „C“ wiederum hängt von „D“ ab.

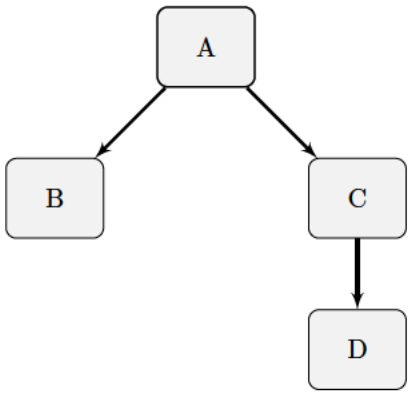


Abbildung 2.1: Ein Abhängigkeitsgraph – mit den Knoten A, B, C, und D

In den meisten Paketverwaltungssystemen wie Maven[2], Composer[3] oder Cargo[4] können Abhängigkeiten in Versionsbereichen definiert werden. Das heißt, dass eine Abhängig-

[2] https://maven.apache.org/
[3] https://getcomposer.org/
[4] https://doc.rust-lang.org/cargo

keit nicht auf eine spezifische Version einer Softwarebibliothek beschränkt ist, sondern mit verschiedenen Versionen arbeiten kann. Das hat unter anderem den Vorteil, dass die Kompatibilität zwischen verschiedenen Softwarebibliotheken steigt. Die Paketverwaltungssysteme entscheiden dann, in welcher Version die Softwarebibliotheken installiert werden können, so dass alle miteinander kompatibel sind.

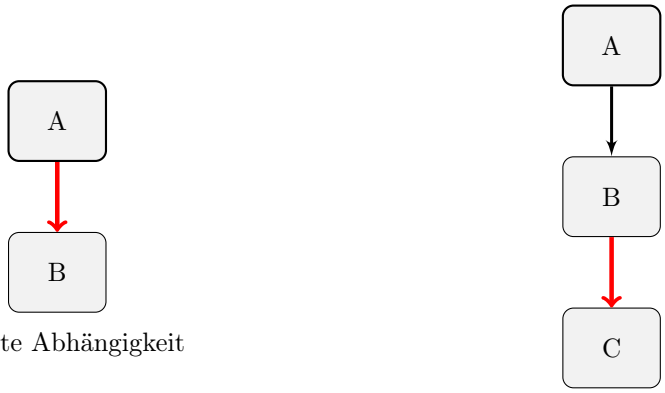
Ein Abhängigkeitsgraph kann also immer nur eine Momentaufnahme der Abhängigkeiten abbilden, deshalb muss der Kontext, in dem der Abhängigkeitsgraph betrachtet wird, durchgehend als relevant angesehen werden. Die Qualität einer Softwarebibliothek kann zwar für sich genommen analysiert werden, das Ergebnis dieser Analyse kann in einem anderen Kontext aber nicht wiederverwendet werden. Sofern auch die Qualitätspropagation berücksichtigt wird, muss also auch die Qualität einer Softwarebibliothek immer im gesamten aktuellen Kontext betrachtet werden.

Ein Abhängigkeitsgraph kann einige besondere Merkmale aufweisen, denen bei der Propagation von Qualität besondere Beachtung geschenkt werden muss. Darunter fallen Mehrfachabhängigkeiten, zirkuläre Abhängigkeiten, transitive und direkte Abhängigkeiten. Diese werden im Folgenden kurz erläutert.

2.5.3 Direkte und transitive Abhängigkeiten

In dem Abhängigkeitsgraphen wird zwischen direkten und transitiven Abhängigkeiten unterschieden (Siehe Abbildung 2.2). Im Falle der ersteren gibt es wie im Beispiel gezeigt, eine unmittelbare Dependenz (siehe Abbildung 2.2a). Hier hängt eine Softwarebibliothek „A“ direkt von einer Softwarebibliothek „B“ ab. Im Falle einer transitiven Abhängigkeit (siehe Abbildung 2.2b) ist eine Softwarebibliothek „A“ direkt von einer Softwarebibliothek „B“ und diese allerdings auch von der Softwarebibliothek „C“ abhängig. Die Softwarebibliothek „A“ ist also nicht direkt, sondern nur transitiv von der Softwarebibliothek „C“ abhängig.

In der Softwareentwicklung werden diese transitiven Abhängigkeiten in aller Regel kaum betrachtet, und dementsprechend hat ein Entwickler eines Softwareprodukts darüber also meist keinen Überblick. Auch wenn ein Softwareprodukt nicht direkt von dieser Softwarebibliothek abhängt, so kann es dennoch zu Problemen kommen, wenn in einer Softwarebibliothek, zu der eine transitive Abhängigkeit besteht, eine Sicherheitslücke existiert.



(a) Direkte Abhängigkeit

(b) Transitive Abhängigkeit

Abbildung 2.2: Direkte und transitive Abhängigkeit in Abhängigkeitsgraphen

Zwei Ausnahmen, die in der Praxis vorkommen können, werden in dieser Arbeit nicht behandelt: Die falsch-transitiven Abhängigkeiten und die optionalen Abhängigkeiten. In der Praxis kann es auch vorkommen, dass eine Abhängigkeit nicht als direkte Dependenz definiert wurde, diese transitive Abhängigkeit aber trotzdem direkt genutzt wird. So eine „falsch-transitive“ Abhängigkeit wird hier nicht berücksichtigt werden, um den Fokus in der Entwicklung der Konzepte auf die wesentlichen Punkte zu legen. Bei der Umsetzung der Konzepte sollten „falsch-transitive“ Abhängigkeiten allerdings beachtet werden.

Eine weitere Besonderheit sind optionale Abhängigkeiten. Der Dependency Manager Composer[5] für die Programmiersprache PHP beispielsweise bietet Entwicklern die Möglichkeit, für die Nutzung ihrer Softwarebibliothek weitere Softwarebibliotheken vorzuschlagen[6]. Erstere kann ohne diese zusätzliche Abhängigkeit funktionieren, nutzt sie aber, sofern sie vorhanden ist. Auch solche optionalen Abhängigkeiten werden hier nicht weiter berücksichtigt.

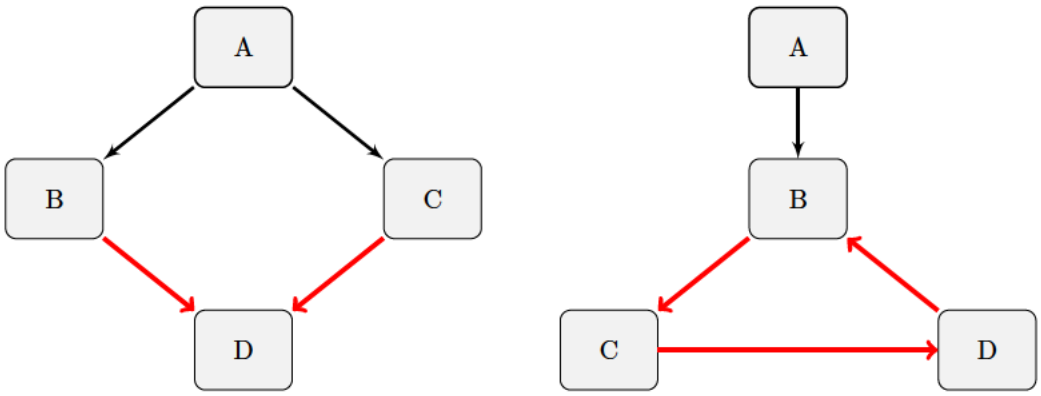
2.5.4 Mehrfachabhängigkeiten und zirkuläre Abhängigkeiten

Für diese Arbeit sind vor allem die Eigenschaften des gerichteten Graphen interessant, die einen Abhängigkeitsgraphen beschreiben. Zum einen kann es mehrere gerichtete Kanten zu einem Knoten geben. Von einem Ausgangspunkt ist also ein Knoten gegebenenfalls über mehrere Wege zu erreichen. Zum anderen kann es zirkuläre Beziehungen in einem

[5] `https://getcomposer.org/`
[6] https://getcomposer.org/doc/04-schema.md#suggest

Abhängigkeitsgraphen geben. In diesem Fall gibt es nicht endende Wege durch den Graphen. Diese beiden Phänomene sind in der Abbildung 2.3 abgebildet.

Eine einzelne Softwarebibliothek „D“ kann mehrfach als Abhängigkeit vorkommen (siehe Abbildung 2.3a). In diesem Fall hat eine Softwarebibliothek „A“ die beiden Abhängigkeiten Softwarebibliothek „B“ und Softwarebibliothek „C“. Beide Softwarebibliotheken haben als Abhängigkeit die Softwarebibliothek „D“. Dieses Szenario ist für die Propagation der Qualität wichtig, da in dem Zusammenhang bestimmt werden muss, ob die Qualität der Softwarebibliothek „D“ mehrfach propagiert wird, obwohl sie für die Softwarebibliothek „A“ nur einmal auftaucht.



(a) Mehrfache Abhängigkeit (b) Zirkuläre Abhängigkeit

Abbildung 2.3: Mehrfachabhängigkeit und zirkuläre Abhängigkeit in Abhängigkeitsgraphen

Im Falle der zirkulären Abhängigkeiten gibt es Zyklen im Abhängigkeitsgraphen. Auch wenn diese möglichst zu vermeiden sind, wie Martin [22] beschreibt, können sie im Abhängigkeitsgraphen auftauchen. Das Beispiel Abbildung 2.3b zeigt einen Abhängigkeitsgraphen mit einem Zyklus. In dem Beispiel hängt eine Softwarebibliothek „A“ von einer Softwarebibliothek „B“ ab. Diese ist von einer Softwarebibliothek „C“ abhängig und diese wiederum von einer Softwarebibliothek „D“. Zugleich hängt die Softwarebibliothek „D“ auch von der Softwarebibliothek „B“ ab. Es existiert also ein Zyklus zwischen den Softwarebibliotheken „B“, „C“ und „D“. Diese Form der Abhängigkeit kann auch zwischen zwei Softwarebibliotheken bestehen, indem sie direkt voneinander abhängen. Da solche Zyklen dazu führen, dass die Qualität von Knoten unendlich propagieren können, muss dieses Problem behandelt werden, denn diese unendliche Propagation kann als solche nicht berechnet werden.

2.6 Zusammenfassung: Ein zugeschnittenes Qualitätsmodell für Softwarebibliotheken

Trotz der grundsätzlichen Probleme in der Qualitätsbewertung gibt es die praktische und pragmatische Notwendigkeit der Bewertung von Software. Um dies zu leisten, legt die Forschung verschiedene Ansätze vor. Für den Spezialfall der Einbindung von Softwarebibliotheken, die überdies häufig open-source sind, werden in der Literatur aber noch keine Qualitätsmodelle diskutiert. Bisherige Ansätze zur Bewertung von Softwarebibliotheken sind primär praxisorientiert und dienen vornehmlich der Auswahl geeigneter Softwarebibliotheken im konkreten Fall. Den anspruchsvollsten Ansatz legt Eghan [11] vor. Aber auch sein Abhängigkeitsmanagement beschreibt kein Softwarequalitätsmodell, sondern geht einen praxisorientierten Weg. Die Besonderheiten, auf denen ein Qualitätsmodell für Softwarebibliotheken aufbauen sollte, werden deshalb in dieser Arbeit thematisiert.

Aus der vorangehenden Analyse zu bestehenden Modellen zur Bewertung von Softwarequalität sowie zu den spezifischen Herausforderungen, die aus den Abhängigkeiten für den Spezialfall „Softwarebibliothek“ gelten, lassen sich folgende Schlussfolgerungen für die Zwecke dieser Arbeit ziehen: Auf der Grundlage des ISO-25010-Qualitätsmodells als Basismodell können Ansätze für ein Softwarequalitätsmodell entwickelt werden, das auf die Domäne von Open-Source Softwarebibliotheken zugeschnitten ist, das aber nicht den Anspruch erhebt, für diese Domäne den Begriff der Softwarequalität eindeutig zu definieren. Ein umfassendes Qualitätsmodell zur Qualitätseinschätzung von Softwarebibliotheken sollte möglichst breit und differenziert aufgestellt sein, damit der Nutzer die für ihn wichtigen Merkmale herausfiltern kann. Um einen ersten Schritt in diese Richtung zu leisten, zeigt diese Arbeit anhand einzelner ausgewählter Qualitätsmerkmale auf, wie diese auf komplexe Weise propagieren. Dazu werden insbesondere die Aspekte der *Wartbarkeit* und *Sicherheit* in den Blick genommen. Um die Komplexität der Propagationsvorgänge beispielhaft aufzuzeigen, wird darüber hinaus die Stabilität diskutiert.

Diese Auswahl erfolgte aufgrund folgender Erwägungen. Für den spezifischen Fall der Bewertung von Softwarebibliotheken stellen sich die einzelnen Qualitätsmerkmale, die im Basismodell zur Bewertung von Softwarequalität herangezogen werden, zum Teil unterschiedlich dar. Sie eignen sich daher auch in unterschiedlichem Maße für ein auf Softwarebibliotheken zugeschnittenes Modell und damit auch zu einer näheren Betrachtung im Rahmen dieser Arbeit: Die *funktionale Eignung* ist ein Softwarequalitätsmerkmal, das sehr stark von den Anforderungen durch den Nutzer abhängt. Da vor allem bei

transitiven Abhängigkeiten die Anforderung nicht bekannt ist, wird sie in dieser Arbeit nicht näher betrachtet. *Leistungseffizienz* hängt stark mit der Art der Nutzung zusammen, deshalb ist sie in der Betrachtung über einen Abhängigkeitsgraphen nur schwer zu propagieren. Da die Softwarebibliotheken in den meisten Fällen auch kein Benutzerinterface bereitstellen, mit dem ein Nutzer einer Software direkt interagiert, wird die *Benutzerfreundlichkeit* hier gleichfalls nicht näher betrachtet. Der Aspekt der *Kompatibilität* ist über die strukturelle Betrachtungsweise dieser Arbeit nur schwer zu fassen. Obwohl sie für die Propagation wichtig ist, kann auch sie deshalb in dieser Arbeit nicht weiter betrachtet werden. Der Aspekt der *Sicherheit* ist von gesteigerter Bedeutung für die Qualitätspropagation bei der Einbindung von Softwarebibliotheken. Da der Nutzer wie eingangs beschrieben keinen direkten Einfluss auf die Entwicklung einer Softwarebibliothek hat, ist es für ihn auch wichtig, dass eine Softwarebibliothek, die er nutzt oder nutzen möchte, die *Sicherheit* seiner Software nicht beeinträchtigt. Deshalb ist die Propagation der *Sicherheit* ein wichtiger Punkt der Softwarequalitätspropagation und wird hier beispielhaft näher beleuchtet. *Zuverlässigkeit* und *Portabilität* ähneln in ihrem Propagationsverhalten aber der *Sicherheit* und finden deshalb hier keine weitere Beachtung. Schließlich ist die *Wartbarkeit* ein vom ISO-25010-Standard erfasstes Softwarequalitätsmerkmal, das insbesondere in der Analyse der lokalen Qualität viel Beachtung findet, weil die *Wartbarkeit* ein guter Indikator dafür ist, ob ein Softwareprodukt einfach erweitert oder gepflegt werden kann. Entsprechend ist die *Wartbarkeit* auch ein Anzeiger der notwendigen Ressourcen zur Weiterentwicklung oder Pflege eines Softwareprodukts, wie auch das Kostenmodell von Bakota et al. [4] zeigt. Deshalb ist sie neben dem Merkmal der *Sicherheit* besonders gut geeignet, um die Anforderungen an ein Qualitätsmodell für Softwarebibliotheken beispielhaft zu illustrieren.

Wie aus den obigen Darstellungen hervorgeht, stellen sich für den „Sonderfall“ Softwarebibliothek spezifische Herausforderungen in der Qualitätspropagation. Die komplexen Abhängigkeiten, die bei der Einbindung einer Softwarebibliothek in ein Softwareprodukt ergeben, rücken daher Aspekte in den Vordergrund, die vom ISO-25010-Standard nicht berücksichtigt werden, in einem zugeschnittenen Qualitätsmodell für Softwarebibliotheken aber erfasst werden sollten.

Stabilität wird in dem ISO-25010-Modell [18] weder als Qualitätsmerkmal noch als Untermerkmal genannt. Es wird lediglich als Teil des Untermerkmals der Modifizierbarkeit dargestellt und kann somit als ein Unter-Untermerkmal der *Wartbarkeit* im ISO-25010-Modell als nachrangiges Kriterium verstanden werden. Raemaekers et al. [29] zeigen allerdings, dass die Stabilität einer eingebundenen Softwarebibliothek sich unmittelbar auf

die *Wartbarkeit* des nutzenden Softwareprodukts auswirkt. Aus diesem Grund ist Stabilität für ein zugeschnittenes Qualitätsmodell für Softwarebibliotheken als wichtiger Aspekt der *Wartbarkeit* zu betrachten. Darüber hinaus ist die Qualität der Community für OSS ein ganz wesentliches Merkmal. Da hinter Softwarebibliotheken eine Community steht, die den Entwicklungsverlauf einer Softwarebibliothek maßgeblich bestimmt, sind bei der Bewertung von OSS-Bibliotheken durch ein auf diesen Anwendungsfall zugeschnittenes Softwarequalitätsmodell auch diese Community-Qualitätsmerkmale von besonderer Relevanz.

Vor diesem Hintergrund stehen *Wartbarkeit* und *Sicherheit* hier im Zentrum der Überlegungen zur Qualitätspropagation von Softwarebibliotheken. Um das Verhalten dieser beiden Aspekte in der Qualitätspropagation durch die Einbindung von Softwarebibliotheken darzustellen und zu untersuchen, werden die Abhängigkeiten in einem Abhängigkeitsgraphen dargestellt. Dabei werden insbesondere verschiedene Besonderheiten wie Mehrfachabhängigkeiten sowie zirkuläre Abhängigkeiten in den Blick genommen, die bei der Betrachtung der Qualitätspropagation besondere Berücksichtigung finden müssen. Auf diese Weise werden auf der Ebene einzelner Qualitätskriterien illustrativ praktikable Vorschläge für Ansätze zur Entwicklung eines Softwarequalitätsmodells für Softwarebibliotheken aufgezeigt.

3 Vorgehensweise

Um einen Vorschlag für die Propagation der Qualität von Softwarebibliotheken zu entwickeln, finden in dieser Arbeit Graphen Verwendung, außerdem werden zu diesem Zweck ein Qualitätspropagationsprogramm und Aggregationsmethoden entwickelt.

3.1 Zur Arbeit mit Graphen

Zum Betrachten von Abhängigkeitsgraphen von PHP-Softwareprodukten wurde das Werkzeug *graph-composer*[1] von Christian Lück genutzt. Es kommt allerdings eine noch nicht veröffentlichte Version zum Einsatz, die alle relevanten Funktionen enthält. Der noch nicht angenommene Pull-Request[2] dazu stammt von Markus Poerschke. Graph-composer hilft dabei, aus den Informationen, die das Paketverwaltungssystem Composer bereitstellt, einen Abhängigkeitsgraphen für ein Softwareprojekt zu erstellen, der alle direkten und transitiven Abhängigkeiten des Softwareprodukts in einem Abhängigkeitsgraphen anzeigt. Im Zuge dieser Arbeit hat das Werkzeug dabei geholfen, Abhängigkeitsgraphen für verschiedene Softwareprodukte der Programmiersprache PHP zu erstellen, und eine Vorstellung ihrer Komplexität zu entwickeln.

3.2 Entwicklung eines Qualitätspropagationsprogramms

Um verschiedene Ansätze zur Aggregation der propagierten Softwarequalität berechnen zu können wird ein Hilfswerkzeug entwickelt, das auf der Grundlage von Propagationsgraphen und der lokalen Softwarequalität die aggregierte Softwarequalität berechnen kann. Dieses Qualitätspropagationsprogramm dient als Grundlage zur Berechnung von Formeln zum Aggregieren propagierter Qualität. Gleichzeitig kann es die erstellten Graphen

[1] `https://github.com/clue/graph-composer`
[2] `https://github.com/clue/graph-composer/pull/45`

und die aggregierten Werte der Knoten als Grafik ausgeben, welche in dieser Arbeit an verschiedenen Stellen genutzt werden.

Das Qualitätspropagationsprogramm ist in der Programmiersprache PHP entwickelt und baut dabei in erster Linie auf den Softwarebibliotheken *GraPHP*[3] und *graphp/GraphViz*[4] von Christian Lück auf. Diese liefern zum einen eine Datenstruktur für die Propagationsgraphen, die das Qualitätspropagationsprogramm verarbeiten soll. Zum anderen können diese Graphen mit *graphp/GraphViz* grafisch dargestellt werden.

Außerdem wird die Softwarebibliothek *symfony/console*[5] genutzt, um aus dem Qualitätspropagationsprogramm eine Konsolenapplikation zu machen, die flexibel einsetzbar ist. Und die Softwarebibliothek *MathPHP*[6] dient der Erstellung von Rechenoperationen wie die Berechnung von Mittelwerten oder gewichteten Mittelwerten.

Dieses Qualitätspropagationsprogramm kommt im Laufe der Arbeit an verschiedenen Stellen zum Einsatz. Indem verschiedene „Aggregatoren“ implementiert werden, können auch unterschiedliche Aggregationsmethoden ausprobiert werden. Außerdem ist es mit dem Qualitätspropagationsprogramm möglich, die Qualitätsaggregation komplexerer Graphen ohne großen Aufwand zu simulieren. Das ist zum Beispiel beim Demonstrationsbeispiel (siehe Kapitel 4.1 „Ein Demonstrationsbeispiel“) hilfreich.

Das Qualitätspropagationsprogramm wurde so implementiert, dass es möglichst flexibel an verschiedene Parameter angepasst werden kann. Wichtigste Bestandteile des Programms sind die Aggregatoren, die Simulationskonfigurationen und die Programmlogik zur Anzeige des berechneten Graphen. Der Quellcode des Qualitätspropagationsprogramms ist dieser Arbeit in digitaler Form beigelegt.

Die Aggregatoren bilden eine zentrale Funktion, weil sie die eigentliche Aggregationsprogrammlogik für die verschiedenen Softwarequalitätsmerkmale beinhalten. Sie berechnen die aggregierte Softwarequalität in einer rekursiven Funktion, mit der durch einen Propagationsübergangsbaum (vergleiche 5.1 „Das Bilden eines Propagationsgraphen“) entgegen der Kantenrichtung traversiert wird. Dadurch wird es möglich, die Aggregationsformeln zu implementieren, die im Lauf dieser Arbeit entwickelt werden.

Die Simulationskonfigurationen können einen Simulationsdurchlauf beschreiben. Im Anhang A.2 „Qualitätspropagationsprogramm - Beispielkonfiguration“ ist ein Listing zu

[3] `https://github.com/graphp/graph`
[4] `https://github.com/graphp/graphviz`
[5] `https://github.com/symfony/console`
[6] `https://github.com/markrogoyski/math-php`

sehen, das die Konfiguration einer Testsimulation zeigt. In diesem Fall soll die einfache Propagation von einem Knoten B zum Knoten A simuliert werden. Dafür wird in Zeile 18 ein Titel definiert. In den Zeilen 21 bis 38 wird der zu simulierende Graph mit den lokalen Qualitäten beschrieben, in diesem Fall handelt es sich um die lokale *Wartbarkeit* der beiden Knoten. Außerdem werden an dieser Stelle die Kanten beschrieben (Zeile 35), der Knoten B propagiert zum Knoten A. In den Zeilen 41 bis 46 werden die Aggregatoren definiert, die in dieser Simulation genutzt werden. Hier handelt es sich nur um einen Aggregator für die *Wartbarkeit*. Außerdem wird noch der Zielknoten definiert, also der Knoten, dessen aggregierte Qualität in der Simulation betrachtet werden soll (Zeile 50). Zuletzt wird definiert, wie das Ergebnis in der Grafik dargestellt werden soll (Zeile 53 bis 56). In diesem Fall soll nur die lokale und aggregierte *Wartbarkeit* der Knoten dargestellt werden. Die Ausgabe dieser Konfiguration ist in der Abbildung 3.1 zu sehen. Das Qualitätspropagationsprogramm generiert dabei zwei verschiedene Graphen, einmal den Propagationsgraphen und zweitens den Propagationsübergangsbaum, der die möglichen Zyklen und Mehrfachabhängigkeiten des Graphen auffächert[7]. Für dieses Beispiel sind beiden Graphen gleich. Um sie voneinander unterscheiden zu können, wird der Propagationsübergangsbaum im Folgenden immer mit blauem Hintergrund angezeigt (siehe Abbildung 3.1b).

(a) Propagationsgraph (b) Propagationsübergangsbaum

Abbildung 3.1: Beispielhafte Ausgabe des Qualitätspropagationsprogramms

In der Ausgabe werden alle Knoten und Kanten des Graphen abgebildet. Der Knoten enthält dabei verschiedene Informationen, den Namen des Knotens, und eine Tabelle mit den Bewertungen der für den Zusammenhang relevanten Softwarequalitätsmerkmale. Diese sind in die lokalen und die aggregierten Bewertungen der Knoten aufgeteilt. Der Zielknoten, also der Knoten dessen aggregierte Qualität betrachtet werden soll, ist dabei mit einer stärkeren Umrandung dargestellt. Da ein Knoten in einem Graphen mit Zyklen unterschiedliche aggregierte Bewertungen enthalten kann, können die aggregierten Werte

[7] Mehr zum Propagationsuebergangsbaum in Kapitel 5.1 „Das Bilden eines Propagationsgraphen“

der entsprechenden Knoten in Graphen mit Zyklen in der Grafik nicht eindeutig angezeigt werden. Für diese Werte wird dann in dem Propagationsgraphen „Nicht eindeutig“ angezeigt (Vergleiche Abbildung 5.19, Seite 56). Im Propagationsübergangsbaum können allerdings alle aggregierten Werte angezeigt werden.

Sowohl die Methode zum Bilden dieser Graphen, als auch die Berechnungsgrundlage für die aggregierten Werte, die in dem Knoten angezeigt sind, werden im weiteren Verlauf der Arbeit entwickelt.

3.3 Entwicklung der Aggregationsmethoden

Das entwickelte Programm kann bei der Entwicklung der Aggregationsmethoden unterschiedliches Verhalten abbilden, um verschiedene Methoden gegenüberzustellen. In diesem Teil der Arbeit werden Formeln entwickelt, die die Aggregation der propagierten Qualität beschreiben.

Die mathematischen Formeln bauen auf dem Grundgerüst der Darstellung anhand eines Graphen auf. Softwareprodukte beziehungsweise Softwarebibliotheken werden dabei als Knoten und die Abhängigkeitsbeziehung durch die gerichteten Kanten zwischen den Knoten abgebildet. Die folgenden Formeln wurden auf der Basis der Definitionen zu gerichteten Graphen von Werners [38] gebildet.

Definition 1 (Propagationsgraph) *Ein* Propagationsgraph $P = (V, E)$ *ist ein gerichteter Graph und besteht somit aus einem geordneten Paar* (V, E)*, wobei* V *eine Menge von Knoten und* E *eine Menge von Kanten bezeichnet. Die Menge der Knoten* $V = \{v_1, \ldots, v_n\}$ *ist eine Zusammenfassung bestimmter, unterscheidbarer Knoten* v*, die Teil eines* Propagationsgraphens P *sind. Die Menge von Kanten* $E = \{e_1, \ldots, e_m\}$ *beschreibt die Beziehung zwischen den Knoten, wobei jede Kante* $e \in E$ *ein Knoten paar* $(i, j) \in V \times V$ *beschreibt.*

Zusätzlich sind einige Eigenschaften gerichteter Graphen für diese Arbeit relevant (nach Werner [38]):

- i heißt (unmittelbarer, direkter) Vorgänger von j (predecessor).
- Die Menge der direkten Vorgänger eines Knotens j ist $P(j)$.

- j heißt (unmittelbarer, direkter) Nachfolger von i (successor).
- Die Menge der direkten Nachfolger eines Knotens i ist $S(i)$.

Wie eingangs beschrieben betrachtet diese Arbeit die Softwarequalität in der Form eines Softwarequalitätsvektors, der aus den einzelnen Qualitätsbewertungen der verschiedenen Softwarequalitätsmerkmale besteht.

Definition 2 (Qualitätsvektor) *Ein* Qualitätsvektor $Q(v)$ *beschreibt die Qualität eines Knotens. Da dessen Qualität in verschiedenen Softwarequalitätsmerkmalen beschrieben wird, bildet sich daraus der Qualitätsvektor. Der Qualitätsvektor setzt sich wie folgt zusammen:*
$Q(v) = (Q^{Funktionale\_Eignung}(v), Q^{Leistungseffizienz}(v), Q^{Kompatibilität}(v),$
$Q^{Benutzerfreundlichkeit}(v), Q^{Portabilität}(v), Q^{Wartbarkeit}(v), Q^{Sicherheit}(v),$
$Q^{Zuverlässigkeit}(v)).$

Dieser Qualitätsvektor beinhaltet alle in dieser Arbeit zu betrachtenden Softwarequalitätsmerkmale. Auch wenn in dieser Arbeit immer nur Teile des Qualitätsvektors betrachtet werden, so ist es dennoch sinnvoll, hier den vollständigen Softwarequalitätsvektor zu definieren. Neben einem Qualitätsvektor, der die Qualität eines Knotens inklusive der propagierten Qualität abbildet, definieren wir für diese Arbeit außerdem auch den Qualitätsvektor für die lokale Qualität eines Knotens $Q_{lokal}(v)$, auch $Q_l(v)$. Der lokale Qualitätsvektor setzt sich wie folgt zusammen:
$Q_{lokal}(v) = (Q_{lokal}^{Funktionale\_Eignung}(v), Q_{lokal}^{Leistungseffizienz}(v), Q_{lokal}^{Kompatibilität}(v),$
$Q_{lokal}^{Benutzerfreundlichkeit}(v), Q_{lokal}^{Portabilität}(v), Q_{lokal}^{Wartbarkeit}(v), Q_{lokal}^{Sicherheit}(v),$
$Q_{lokal}^{Zuverlässigkeit}(v)).$

Da einige Teile des Qualitätsvektors, wie etwa die *Wartbarkeit* eines Knotens besonders häufig verwendet werden, finden hier folgende Kurzformen Verwendung:

- $Q^{Wartbarkeit}(v) = Q^{W}(v)$ und $Q_{lokal}^{Wartbarkeit}(v) = Q_l^{W}(v)$
- $Q^{Sicherheit}(v) = Q^{S}(v)$ und $Q_{lokal}^{Sicherheit}(v) = Q_l^{S}(v)$

Die Entwicklung der Formeln zur Aggregation basieren auf diesen Definitionen des Qualitätsvektors.

4 Probleme der Qualitätspropagation

Zur Konkretisierung der Fragestellung ist es notwendig, sich mit Problemen zu beschäftigen, die aus der Struktur eines Abhängigkeitsgraphen abgeleitet werden können, beziehungsweise mit solchen Problemen, die aus der Diversität der Softwarequalität entstehen. Dabei handelt es sich um strukturelle ebenso wie nicht-strukturelle Probleme der Qualitätspropagation. Um sich diesen Problemen anzunähern wird zunächst ein Demonstrationsbeispiel entworfen, welches eine Beispielarchitektur abbildet.

4.1 Ein Demonstrationsbeispiel

An einem Demonstrationsbeispiel können die zuvor beschriebenen Formen der Abhängigkeiten zwischen Softwarebibliotheken nachgezeichnet und damit kann zugleich auf die strukturellen Probleme der Qualitätspropagation hingewiesen werden.

In der Abbildung 4.1 ist der Abhängigkeitsgraph des für diese Arbeit entwickelten Demonstrationsbeispiels zu sehen. Die Funktionalitäten der Softwarebibliotheken sind an dieser Stelle nicht relevant, da es lediglich um die Propagation bereits aggregierter Metriken geht. Dennoch wurde dieses Beispiel in der Programmiersprache PHP umgesetzt, um bei der Entwicklung eine praxisnahe Grundlage zu haben. Als Demonstrationsapplikation wurde ein Rechner entwickelt, der Addition und Subtraktion von Fließkommazahlen beherrscht. Die Implementierung wurde auf verschiedene Softwarebibliotheken aufgeteilt. Das Demonstrationsbeispiel ist explizit für die Zwecke dieser Arbeit konstruiert, die Architektur simplifiziert und dient hier also nur zur Veranschaulichung.

Die Applikation ist in der Abbildung 4.2 zu sehen. Sie teilt sich in die folgenden Elemente auf: Den Einstiegspunkt bildet das Softwarepaket *CalculatorCli* in Form eines Command Line Interface (CLI). Der Screenshot zeigt die Applikation, in der einige Berechnungen durchgeführt werden. Um zu einem Ergebnis zu kommen, nutzt *CalculatorCli* zwei Softwarebibliotheken direkt. Zum einen den *Calculator*, der das Parsen der Eingabe

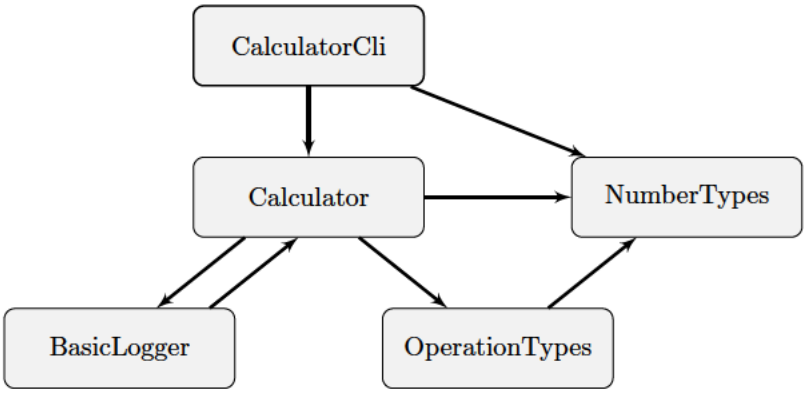


Abbildung 4.1: Demonstrationsbeispiel – Abhängigkeitsgraph des Softwareprodukts *CalculatorCli* mit den genutzten Softwarebibliotheken

und die Berechnung übernimmt, und zum anderen die *NumberTypes*, die verschiedene Nummertypen enthält. Der *Calculator* ist wiederum abhängig vom *BasicLogger*, der die Eingaben des *Calculator* loggen soll und von den *OperationTypes*, welche die möglichen Operationen abbilden, die der *Calculator* durchführen kann. Dafür hängen die *OperationTypes* ebenso wie der *Calculator* auch von den *Numbertypes* ab. Der *BasicLogger* hängt nur vom Calculator ab. Da dieser *Calculator* wiederum auch vom *BasicLogger* abhängt, entsteht eine zirkuläre Abhängigkeit.

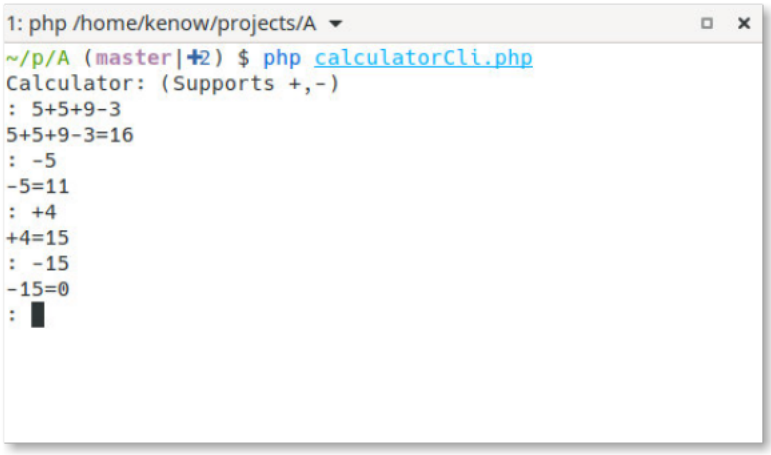


Abbildung 4.2: Demonstrationsbeispiel – Screenshot des Softwareprodukts *CalculatorCli* mit einigen Berechnungen

Die zuvor besprochenen Eigenheiten der Propagation tauchen in diesem Demonstrationsbeispiel auf, das direkte und transitive Abhängigkeiten enthält: *CalculatorCli* ist direkt abhängig von *Calculator* und *Numbertypes* transitiv ist es zusätzlich ab-

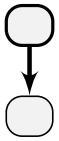
hängig von *BasicLogger* und *OperationTypes*. Außerdem gibt es zirkuläre Abhängigkeiten und Mehrfachabhängigkeiten. Die *CalculatorCli* ist mehrfach abhängig von den *Numbertypes*, einmal direkt und zweimal transitiv über die Softwarebibliotheken *Calculator* und *OperationTypes*. Eine zirkuläre Abhängigkeit besteht zwischen den Softwarebibliotheken *Calculator* und *BasicLogger*, sie sind voneinander abhängig.

Dieses Demonstrationsbeispiel belegt zum einen deutlich, dass die Qualität eines Softwareprodukts in der Regel nicht nur auf der Grundlage eines einzelnen Softwarepakets gemessen werden kann. Wenn die Qualitätsbewertung entsprechend durchgeführt würde, würde in dem Demonstrationsbeispiel nur die Qualität der *CalculatorCli* bewertet werden. Diese enthält allerdings nur einen kleinen Bruchteil der Logik, aus der die gesamte Applikation besteht. Zum anderen wird hier gezeigt, dass die zuvor beschriebenen Probleme der Mehrfachabhängigkeit und der zirkulären Abhängigkeiten reale Probleme sind.

4.2 Strukturelle Probleme der Qualitätspropagation

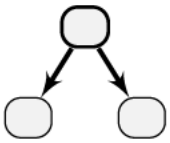
Um die Propagation von Qualität über die Struktur des Abhängigkeitsgraphen diskutieren zu können, muss zuerst die lokale Qualität einzelner Softwarebibliotheken bestimmt werden können. Da sich diese Arbeit allerdings nur mit den strukturellen Problemen beschäftigen wird, werden an dieser Stelle bereits lokale Qualitätsbewertungen der einzelnen Knoten im Abhängigkeitsgraphen vorausgesetzt. Diese Qualitätsbewertungen haben dabei die Form eines lokalen Qualitätsvektors (Vergleiche 3.3 „Entwicklung der Aggregationsmethoden“).

Die strukturellen Probleme der Softwarequalitätspropagation, die sich aus dem Demonstrationsbeispiel ergeben sind:



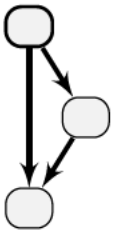
1. Im ersten Schritt der Propagation stellt sich die Frage, wie die Qualität bei einer Abhängigkeit propagiert. In dem oben angeführten Demonstrationsbeispiel findet sich eine solche Beziehung zwischen den beiden Softwarebibliotheken *OperationTypes* und *NumberTypes*. Die *OperationTypes* haben als einzige Abhängigkeit die *NumberTypes*. Für diesen Fall der Abhängigkeit muss die Aggregation der lokalen Qualität der *OperationTypes* mit der propagierten Qualität der *NumberTypes* betrachtet werden.

2. Im nächsten Schritt muss betrachtet werden wie die Qualität mehrerer Abhängigkeiten aggregiert wird. Dafür kann in dem Abhängigkeitsgraphen der *Calculator* betrachtet werden. Um den Graphen für die Betrachtung dieses Problems zu simplifizieren, wird die Softwarebibliothek *BasicLogger* ebenso wie die Abhängigkeit von *OperationTypes* zu *NumberTypes* ausgeblendet. In diesem Fall muss betrachtet werden, wie die Qualität mehrerer Softwarebibliotheken mit der lokalen aggregiert wird.

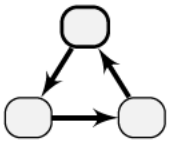


3. Weiterhin sollte geprüft werden, wie die Qualität propagiert, wenn sie nicht direkt, sondern transitiv propagiert. In dem Demonstrationsbeispiel also beispielsweise von *OperationTypes* über *Calculator* zu *CalculatorCli*.

4. Wie wird die Qualität einer Softwarebibliothek aggregiert, wenn diese über mehrere Wege zu einem Knoten propagiert? Eine Antwort auf die Frage nach diesem Verhalten ist in dem Demonstrationsbeispiel unter anderem zwischen *Calculator*, *OperationTypes* und *NumberTypes* zu beobachten. Die Qualität von *NumberTypes* propagiert einmal direkt zum *Calculator* und einmal transitiv über *OperationTypes*.



5. Das nächste Problem, das betrachtet werden muss, ist die zirkuläre Abhängigkeit zwischen Softwarebibliotheken. Wenn es einen Zyklus in dem Abhängigkeitsgraphen gibt, so wird die Qualität theoretisch unendlich propagiert. Wie kann dieser Zyklus aufgebrochen werden, ohne das Ergebnis zu sehr zu verfälschen?



6. Abschließend können die Ergebnisse des Demonstrationsbeispiels zusammen betrachtet, verglichen, bewertet und ihre Plausibilität im Hinblick auf strukturelle Probleme der Qualitätspropagation untersucht werden.

4.3 Nicht-strukturelle Probleme der Qualitätspropagation

Wie eingangs beschrieben ist Qualität kein einfach zu definierender Begriff. Die verschiedenen Softwarequalitätsmerkmale bezeichnen deshalb auch sehr unterschiedliche Aspekte der Softwarequalität. Aus dieser Diversität entstehen neue Probleme in der Qualitätspropagation.

Da die Softwarequalitätsmerkmale unterschiedlicher Natur sind, verhalten sie sich in der Propagation auch unterschiedlich. Während bei einem Softwarequalitätsmerkmal vielleicht ein Mittelwert oder gewichteter Mittelwert zwischen der aggregierten und lokalen Bewertung genutzt werden kann (zum Beispiel bei der *Wartbarkeit*, vergleiche dazu Kapitel 5.2.1 „Die Aggregation der lokalen und propagierten Qualität“), so muss bei einem anderen gegebenenfalls ein Minimum genutzt werden. Das ist beispielsweise der Fall bei der *Sicherheit* (vergleiche Kapitel 5.3 „Die Propagation der Sicherheit“). Diese Arbeit soll klären, inwiefern die Softwarequalitätsmerkmale auf unterschiedlichen Aggregationsmethoden aufbauen.

Die Signifikanz bestimmter Softwarequalitätsmerkmale ist in der Propagation größer. Solche Merkmale beeinflussen im Zuge der Propagation die Qualität eines Knotens entsprechend stärker als andere. Die Aggregation der propagierten Qualität muss deshalb an die verschiedenen Softwarequalitätsmerkmale angepasst werden. Dafür werden hier verschiedene Softwarequalitätsmerkmale näher betrachtet und Konzepte zum Behandeln der propagierten Qualität erarbeitet.

Maßgeblich für die Betrachtung der Qualitätspropagation ist hier eine Unterteilung der Softwarequalität in die verschiedenen Softwarequalitätsmerkmale des ISO-25010-Standards. Dabei stellt sich die Frage, ob diese Betrachtung in ihrer Genauigkeit ausreicht, oder ob die Propagation der Qualität in kleineren Einheiten der Softwarequalität, wie etwa den Softwarequalitätsuntermerkmalen oder Unter-Untermerkmalen betrachtet werden müsste. Die Betrachtung der Softwarequalitätspropagation auf Basis der Unter-Unter-Merkmale des ISO-25010 würde sicherlich eine differenziertere Aussage über die aggregierte Qualität ermöglichen. Gleichzeitig erfordert das allerdings auch ein komplexeres Vorgehen, weil für eine größere Zahl unterschiedlicher Merkmale eine Aggregationsmethodik entwickelt werden muss. Da sich die Problematiken dieser Arbeit grundsätzlich auch auf eine Betrachtung der einzelnen Unter-Unter-Merkmale übertragen lassen und die Betrachtung auf Basis der Softwarequalitätsmerkmale verständlicher ist, wird hier auf deren Basis aggregiert.

5 Die Propagation von Qualität

Weil im Folgenden die Propagation von Softwarequalität und die zuvor beschriebenen Probleme näher untersucht werden, muss dafür zunächst eine Struktur entwickelt werden, in der die Propagation von Qualität abgebildet werden kann, der Propagationsgraph. Darauf aufbauend können strukturelle und nicht-strukturelle Probleme der Softwareuqualitätspropagation untersucht werden.

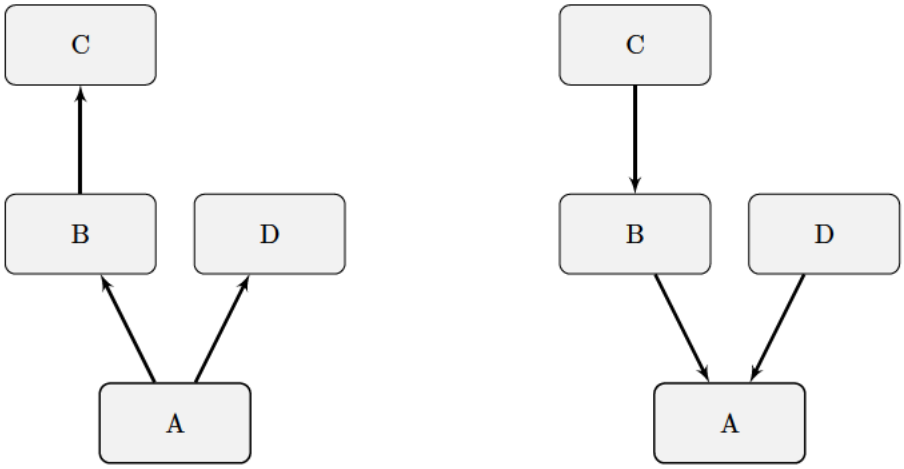
Zur Untersuchung der strukturellen Probleme der Softwareuqualitätspropagation wird das Softwarequalitätsmerkmal der *Wartbarkeit* genauer betrachtet und das Verhalten der *Wartbarkeit* im Zusammenhang mit den verschiedenen strukturellen Problemen der Softwarequalitätspropagation betrachtet. Im Anschluss daran wird zur Verdeutlichung der Diversität der Softwarequalitätsmerkmale die *Sicherheit* genauer betrachtet und untersucht, inwiefern es sich von der *Wartbarkeit* unterscheidet.

Zum Untersuchen der Softwarequalitätspropagation werden jeweils fiktive Beispielgraphen verwendet bei denen jeder Knoten bereits mit den für das Beispiel relevanten lokalen Qualitäten bewertet ist. Wie bereits beschrieben liegt die Skala bei $0,00$ für die schlechteste, bis $10,00$ für die beste Bewertung.

5.1 Das Bilden eines Propagationsgraphen

Während sich Abhängigkeiten über den Graphen abbilden lassen, wird darüber hinaus auch eine Struktur benötigt, die die Softwarequalitätspropagation abbilden kann. Dafür wird hier nachfolgend ein Propagationsgraph entwickelt. Als solcher wird in dieser Arbeit ein Graph definiert, der beschreibt, wie die Qualität in einer Abhängigkeitsstruktur von einem Softwareprodukt in das nächste propagiert wird. Unter Bezugnahme auf die grundsätzliche These, dass die Softwarequalität einer genutzten Softwarebibliothek Einfluss auf die Qualität der vorliegenden Software hat entspricht das Propagationsverhältnis dem umgekehrten Abhängigkeitsverhältnis. Deshalb, kann ein Propagationsgraph

aus einem Abhängigkeitsgraphen gebildet werden, indem die gerichteten Kanten eines Abhängigkeitsgraphen in ihrer Richtung umgekehrt werden.



(a) Abhängigkeitsgraph (b) Propagationsgraph

Abbildung 5.1: Das Bilden eines Propagationsgraphens

Dieser Prozess ist in Abbildung 5.1 dargestellt. Während Abbildung 5.1a einen Abhängigkeitsgraphen zeigt, in dem der Knoten A von den beiden Knoten B und D direkt und transitiv über B von C abhängt, zeigt Abbildung 5.1b den daraus resultierenden Propagationsgraphen, in dem die gerichteten Kanten umgekehrt sind und so die Qualitätspropagation nachvollzogen werden kann. Der Knoten C propagiert also über den Knoten B transitiv die Qualität zu A und die beiden Knoten B und D ihre Qualität direkt zu A.

Ebenso wie der Abhängigkeitsgraph kann auch der Propagationsgraph Zyklen enthalten, so dass die Qualität unendlich im Graphen propagieren kann.

Der Propagationsgraph Abbildung 5.3a enthält einen Zyklus (B propagiert an C und C propagiert an B). In diesem Graphen wird A betrachtet. Entsprechend sind die Wörter, die in dem Graphen gebildet werden können und mit „A“ aufhören, relevant für die Propagation zu A. Dazu gehören unter anderem die Wörter A, BA, CBA oder $BCBA$. Da C zu B und B zu C propagiert, existiert hier ein Zyklus. Es können also auch unendlich lange Wörter gebildet werden, die mit einer unendlich langen Folge des Teilwortes CB beginnen. Die möglichen Wörter für den Graphen lassen sich so zusammenfassen: $B^n \cdot CB^k \cdot A$ wobei $n \in 0,1$ und $k \in \mathbb{N}$. Auch wenn der Graph mit dem Zyklus die Propagation der Qualität wiedergibt, kann mit dem Zyklus in der Darstellung der Qua-

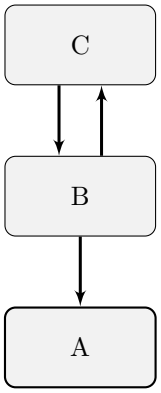
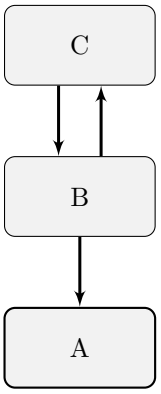
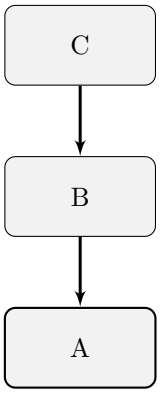


Abbildung 5.2: Propagationsgraph mit Zyklus

litätspropagation nicht gearbeitet werden, da die Qualität unendlich propagieren und deshalb nicht abgebildet werden kann.



(a) Propagationsgraph unmodifiziert

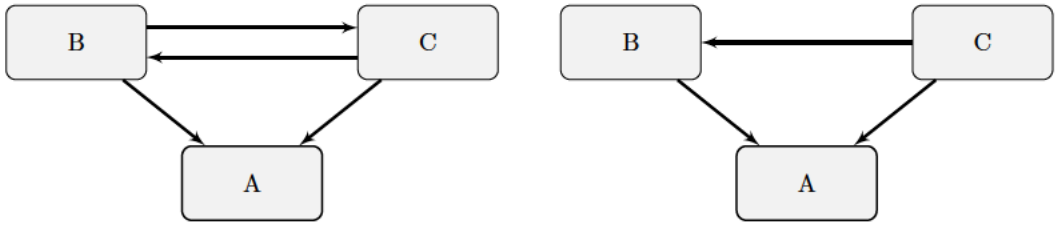


(b) Modifizierter Propagationsgraph

Abbildung 5.3: Beispiel 1 - Entfernen von Kanten um Zyklen zu verhindern

Es muss also eine Form der statischen Abbildung gefunden werden, die sich dennoch der unendlichen Propagation der Qualität annähert. Einen solchen Zyklus aufzulösen, bedeutet in jedem Fall den Verlust von Information. Der Zyklus ist Teil des Abhängigkeitsgraphen und somit auch Teil des Propagationsgraphen. Eine Betrachtung der Qualitätspropagation, die den Zyklus nicht vollständig betrachtet, kann also nicht komplett sein und führt zu einer Verfälschung des Ergebnisses. Im Weiteren werden verschiedene Ansätze beschrieben, wie die Zyklen im Propagationsgraphen dennoch behandelt werden können und sich dabei dem Zyklus besser annähern.

Der Zyklus könnte beispielsweise schon beim Erstellen, durch das Entfernen einzelner Kanten des Propagationsgraphen zu einem modifizierten Propagationsgraphen umgeformt werden, der keine Zyklen enthält. Das ermöglicht es, mit den Graphen zu rechnen. In Abbildung 5.3 ist eine solche Modifizierung zu sehen. Hier wird die gerichtete Kante von B nach C entfernt. In dem modifizierten Graphen ist weiterhin das Wort CBA enthalten, welches auch im unmodifizierten Graphen das längste zyklenlose Wort bildet. Und auch wenn B jetzt nicht mehr zu C propagiert, so wird die Qualität von B dennoch zu A propagiert.



(a) Propagationsgraph unmodifiziert (b) Modifizierter Propagationsgraph

Abbildung 5.4: Beispiel 2 - Entfernen von Kanten um Zyklen zu verhindern

Diese Modifizierung birgt allerdings einige Probleme. Zum einen kann in dem Propagationsgraphen nun nur noch der Knoten A betrachtet werden, da der Graph für diesen angepasst wurde. Wenn man sich also den Knoten C anschaut, so lässt sich für ihn im modifizierten Graphen nur noch das Wort C bilden, während im unmodifizierten Graphen auch das Wort BC möglich ist. Hinzu kommt, dass sich diese Methode nicht bei jedem Graphen anwenden lässt. Ein Beispiel dafür ist in Abbildung 5.4 zu sehen. In diesem unmodifizierten Graphen (Abbildung 5.4a) propagiert B zu C und A. Der Knoten C wiederum propagiert zu B und A. Es besteht also ein Zyklus zwischen den Knoten B und C. Dabei können unter anderem die zyklusfreien Worte BCA und CBA aus dem Graphen gebildet werden. Wenn nun eine der beiden Kanten von C nach B oder von B nach C entfernt werden, so kann eins der beiden Worte BCA und CBA nicht mehr gebildet werden. Nach dieser Methode würde der Propagationsgraph also nicht nur Zyklen entfernen, sondern den Graphen noch weiter verfälschen. Wenn der Propagationsgraph auf diese Weise verändert wird, weicht er stark von der tatsächlichen Qualitätspropagation ab und verliert also an Aussagekraft. Dieser Weg kann also nicht weiter beschritten werden.

Ein weiterer Ansatz zum Arbeiten mit dem Propagationsgraphen basiert auf Regeln zur Bildung der Wörter aus diesem Graphen. Dabei bleibt der Propagationsgraph unangetastet. Die Regeln zum Bilden der Wörter sind dabei wie folgt:

1. In einem Propagationsgraphen werden alle Wörter betrachtet, die keine Zyklen enthalten, in denen ein Knoten also nicht mehrfach auftaucht.

2. Dabei werden nur jene Wörter betrachtet, die auf den zu betrachtenden Knoten enden und dabei keine Teilmenge eines anderen Wortes sind.

Indem diese Menge von Wörtern betrachtet wird, kann die Propagation von Qualität durch den Abhängigkeitsgraphen betrachtet werden. Die nach diesen Regeln relevanten Wörter für die Propagation für den Graphen aus Abbildung 5.4a sind: BCA, CBA. Im Gegensatz zu dem ersten Ansatz wird hier sowohl die Propagation von B nach C als auch die Propagation von C nach B berücksichtigt. Außerdem bleibt der Graph auch für die Betrachtung anderer Knoten intakt.

Allerdings geht auch diese Variante nicht weit genug, da, wenn man einen Propagationsgraphen betrachtet, der zwei ineinander greifende Zyklen enthält (siehe Abbildung 5.5), viele Kanten beim Bilden des Graphen ignoriert würden. Nach dem letzten Ansatz hätte dieser Graph die Wörter $DCBA$ und $FEBA$. Das heißt im Falle der Knoten F und B würden nur die lokalen Qualitäten genutzt. Das Teilwort des Graphen $FEBD$ ist also nicht enthalten.

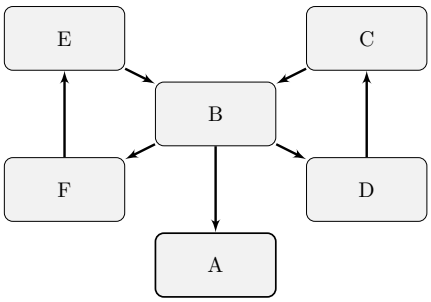


Abbildung 5.5: Propagationsgraph mit ineinander verschachtelten Zyklen

Auch in anderen Bereichen der Softwareentwicklung gibt es Probleme mit Zyklen in einem Graphen. So zum Beispiel beim zustandsbasierten Testen, wie Sneed und Winter [33] es beschreiben. Hier wird auf der Basis von Zustandsdiagrammen getestet, und, da diese Automaten auch Zyklen enthalten, kommt ein Übergangsbaum zum Einsatz. Dieser macht es laut Sneed und Winter insbesondere bei Zustandsdiagrammen, die Schleifen enthalten, möglich, endlich viele Testfälle zu generieren. Auch wenn ein Zustandsdiagramm etwas anders aufgebaut ist, so lässt sich doch die Methode mit der ein solcher Übergangsbaum erstellt wird auf den Propagationsgraphen übertragen. Laut Sneed und Winter ist es das Ziel des Übergangsbaums, aus den bei zyklischen Zustandsdiagrammen

potenziell unendlich vielen möglichen Folgen von Übergängen eine repräsentative Menge auszuwählen. Insbesondere sollen demnach alle Zustände mindestens einmal eingenommen und alle Zustandsübergänge mindestens einmal ausgeführt worden sein.

Auch wenn dieses Prinzip des Übergangsbaums in angepasster Variante auf den Propagationsgraphen angewandt werden kann, gibt es doch einige Aspekte, in denen sich der Propagationsgraph von einem Zustandsdiagramm unterscheidet.

Der Übergangsbaum für Zustandsdiagramme ist mit einem Startpunkt definiert, der Propagationsgraph enthält dagegen keinen Startpunkt, aber einen zu betrachtenden Zielknoten. Dennoch kann entgegen der Richtung der gerichteten Kanten prinzipiell das gleiche Vorgehen zum Bilden eines Propagationsübergangsbaums eingesetzt werden.

Beim Bilden des Übergangsbaums definieren Sneed und Winter als Endbedingung, dass ein Zustand im Übergangsbaum auf dem Weg von der Wurzel zum Blatt nur einmal auftauchen darf. Ein Zustand in einem Zustandsdiagramm ist im Vergleich zum Knoten im Propagationsgraphen allerdings statisch, das heißt, dass der Weg mit dem ein Zustand erreicht wird, den Zustand selbst nicht beeinflusst. In einem Propagationsgraphen beschreiben die gerichteten Kanten zwischen den Knoten allerdings die Beeinflussung des Zielknotens. Das heißt, ein Knoten in einem Propagationsgraphen hat je nachdem, welcher Weg durch den Propagationsgraphen gegangen wird, unterschiedliche Qualitätsbewertungen. Deshalb muss beim Bilden des Propagationsübergangsbaums eine andere Endbedingung gewählt werden: Die Kondition, dass ein Übergang auf dem Weg zwischen Wurzel und Blatt bereits existiert, ist eine passendere Endbedingung, da die Qualitätspropagation so genauer abgebildet werden kann.

Bei dem hier vorgestellten Propagationsübergangsbaum handelt es sich um einen „In-Tree“, also um einen Baum, bei dem alle Kanten zum Wurzelknoten hin gerichtet sind. Außerdem kann ein Knoten eines Propagationsgraphen mehrfach in dem Propagationsübergangsbaum repräsentiert werden, da die verschiedenen Wege zu einem Knoten in einzelne Äste aufgefächert werden (Vergleiche Abbildung 5.6).

Dementsprechend kann zur Bildung eines Propagationsübergangsbaums – abgeleitet von Sneed und Winter [33] – folgender Ablauf verwendet werden:

1. Der Zielknoten wird die Wurzel des Propagationsübergangsbaums.

2. Für jeden möglichen Übergang eines Knotens zu dem zu betrachtenden Knoten erhält der Baum eine Verzweigung zu einem neuen Knoten. Die neue Kante zwischen den beiden Knoten ist immer zur Wurzel hin gerichtet.

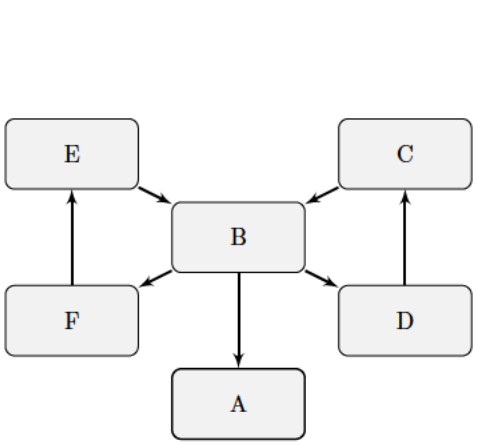
3. Der letzte Schritt wird für jedes Blatt des Übergangsbaums so lange wiederholt, bis eine der beiden Endbedingungen eingetreten ist:

   - Der der Kante entsprechende Übergang im Propagationsgraphen ist auf dem Weg von der Wurzel zum Blatt bereits einmal im Baum enthalten.
   - Der dem Blatt entsprechende Knoten hat im Propagationsgraphen keine weiteren Übergänge.

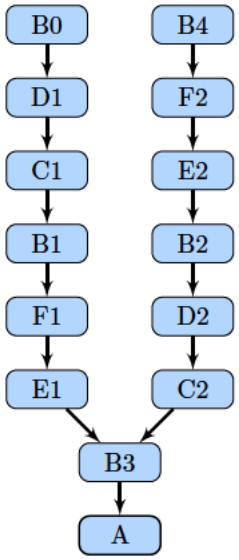
In der Abbildung Abbildung 5.6 ist ein Propagationsgraph und der dazu erstellte Propagationsübergangsbaum zu sehen. Der Propagationsübergangsbaum wird dabei – wie bei der Ausgabe aus dem Qualitätspropagationsprogramm angekündigt – zur besseren Unterscheidung mit blauem Hintergrund angezeigt. In dem Propagationsübergangsbaum (Abbildung 5.6b) sind die Knoten mit Ziffern als Suffix benannt, damit sie unterscheidbar sind. Auch wenn sie den gleichen Knoten des Propagationsgraphen beschreiben, so können sie im Propagationsübergangsbaum unterschiedliche Qualitätsbewertungen haben. Der Knoten B taucht in beiden Wörtern dieses Beispiels dreimal statt einmal auf. Damit nähert man sich stärker der tatsächlichen Propagation der Qualität als mit dem oben aufgezeigten Ansatz, denn jetzt können in dem Propagationsübergangsbaum Abbildung 5.6a die Worte $BFEBDCBA$ und $BDCBFEBA$ gebildet werden.

Definition 3 (Propagationsübergangsbaum) *Ein* Propagationsübergangsbaum $\ddot{U} = (V, E)$ *ist ein gerichteter gewurzelter Baum und besteht somit aus einem geordneten Paar* (V, E)*, wobei* V *eine Menge von Knoten und* E *eine Menge von Kanten bezeichnet. Die Menge der Knoten* $V = \{v_1, \ldots, v_n\}$ *ist eine Zusammenfassung bestimmter, unterscheidbarer Knoten* v*, die Teil eines* Propagationsübergangsbaums P *sind. Die Menge von Kanten* $E = \{e_1, \ldots, e_m\}$ *beschreibt die Beziehung zwischen den Knoten, wobei jede Kante* $e \in E$ *ein Knotenpaar* $(i, j) \in V \times V$ *beschreibt. Jede Kante* $e \in E$ *ist dabei in Richtung des Wurzelknotens gerichtet.*

Zusammenfassend lässt sich festhalten, dass sich durch die Umkehrung der gerichteten Kanten aus einem Abhängigkeitsgraphen ein Propagationsgraph erstellen lässt. Bei



(a) Propagationsgraph



(b) Propagationsübergangsbaum

Abbildung 5.6: Das Bilden des Propagationsübergangsbaums (b) aus einem Propagationsgraph (a) mit zwei ineinandergreifende Zyklen

Rechenoperationen auf der Grundlage eines Propagationsgraphen muss allerdings entschieden werden, wie mit möglichen Zyklen umgegangen wird. Im letzten der drei hier beschriebenen Ansätze wurde eine größere Annäherung in der Qualitätspropagation an den Propagationszyklus erreicht. Deshalb werden im Folgenden die zur Propagation und Aggregation notwendigen Formeln auf der Grundlage des Propagationsübergangsbaums entwickelt. Zur besseren Verständlichkeit wird hier aber in einigen Fällen auch der Propagationsgraph gezeigt.

5.2 Qualitätspropagation und Qualitätsaggregation

Auf der Grundlage dieses Propagationsübergangsbaums können nun die Besonderheiten der Qualitätspropagation behandelt werden. Dazu wird zuerst die *Wartbarkeit* und ihr Propagationsverhalten betrachtet und untersucht, wie die propagierten Qualitäten an einem Knoten aggregiert werden können. Dabei liegt der Fokus auf den strukturellen Problemen wie der Mehrfachabhängigkeit und zirkulären Abhängigkeit. Im nächsten Schritt werden die Diversität der Softwarequalitätsmerkmale und die daraus resultierenden Unterschiede in der Aggregationsmethodik untersucht.

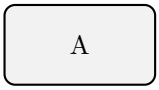
5.2.1 Die Aggregation der lokalen und propagierten Qualität

Im ersten Schritt wird die Qualitätspropagation im einfachen Propagationsübergangsbaum betrachtet. Der simpelste Propagationsgraph enthält nur einen Knoten und keine Kante. Da dieser Knoten also keine Vorgängerknoten hat, kann es keine Qualitätspropagation geben. Es wird in diesem Fall also nur die lokale Qualität des Knotens betrachtet. In diesem Fall entspricht die aggregierte *Wartbarkeit* der lokalen *Wartbarkeit* eines Knotens.

Wir definieren: *Die aggregierte Wartbarkeit eines Knotens v in einem Propagationsübergangsbaum entspricht der lokalen Wartbarkeit von v, sofern die Menge der direkten Vorgänger von v ($P(v)$) leer ist.*

$$Q^W(v) = Q_l^W(v) \qquad wenn\ P(v) = \{\} \tag{5.1}$$

In Abbildung 5.7a ist ein solcher Graph zu sehen. Für dieses Beispiel hat der Knoten A eine Bewertung der lokalen *Wartbarkeit* von $7,6$. Mit der zuvor definierten Formel zur Berechnung der *Wartbarkeit* eines Knotens entspricht die aggregierte *Wartbarkeit* der lokalen *Wartbarkeit* des Knotens A in dem Graphen. Das Qualitätspropagationsprogramm gibt also den Graphen Abbildung 5.7b aus. Dabei entspricht nach der Formel (5.1) die aggregierte *Wartbarkeit* von A, der lokalen *Wartbarkeit* – also einem Wert von $7,6$.



(a) Der Propagationsgraph

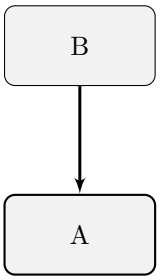
A		
	$Q_{lokal}(A)$	$Q(A)$
Wartbarkeit	7,60	7,60

(b) Die Ausgabe des Qualitätspropagationsprogramms

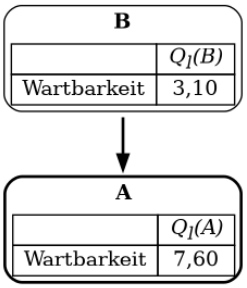
Abbildung 5.7: Qualitätspropagation ohne Kanten

Der einfachste Propagationsgraph, der mindestens eine Kante enthält, besteht aus lediglich zwei Knoten, in diesem Fall A und B, und einer Kante (siehe Abbildung 5.8). In diesem Fall propagiert die Qualität von B zu A, die Kante ist also von B zu A gerichtet. In diesem Szenario muss die lokale *Wartbarkeit* mit der propagierten *Wartbarkeit* aggregiert werden. Es wird davon ausgegangen, dass die lokalen *Wartbarkeiten* der beiden Knoten bereits bestimmt wurden.

Für dieses Beispiel gehen wir von einer Bewertung der *Wartbarkeit* von $7,6$ für den Knoten A und $3,1$ für den Knoten B aus. Die *Wartbarkeit* von B ist also schlechter als die *Wartbarkeit* von A. Da die Qualität von B die Qualität von A beeinflusst, ist die aggregierte *Wartbarkeit* von A also schlechter.



(a) Der Propagationsgraph



(b) Die lokalen Wartbarkeiten

Abbildung 5.8: Einfache Qualitätspropagation

Die Aggregationsmethodik, die genutzt wird, um aus der von B propagierten *Wartbarkeit* und der lokalen *Wartbarkeit* von A die aggregierte *Wartbarkeit* von A zu berechnen, muss also diese Beeinflussung beachten. Der einfachste Ansatz wäre der Mittelwert der beiden *Wartbarkeiten*. Mit der Hilfe der Formel (5.2) würde dabei in dem hier gewählten Beispiel mit einer *Wartbarkeit* von $7,6$ für den Knoten A und $3,1$ für den Knoten B einem Mittelwert von $5,35$ entsprechen. Die Formel (5.2) beschreibt dabei den Mittelwert der lokalen *Wartbarkeit* von A und der *Wartbarkeit* von B.

$$\frac{Q_l^W(A) + Q^W(B)}{2} \tag{5.2}$$

Unter Verwendung dieser Formel wären die beiden *Wartbarkeiten* allerdings gleich stark gewichtet. Wenn man sich aber die *Wartbarkeit* genauer ansieht, erkennt man, dass die *Wartbarkeit* einer genutzten Softwarebibliothek keinen besonders großen Einfluss auf die *Wartbarkeit* des nutzenden Softwareprodukts hat. Es wäre also falsch anzunehmen, dass die propagierte und die lokale *Wartbarkeit* gleich relevant für eine Bewertung der *Wartbarkeit* des nutzenden Softwareprodukts wären. Es gibt also ein Verhältnis lokaler zu propagierter *Wartbarkeit*, das in der Formel beachtet werden muss. Dieses Verhältnis sollte allerdings je nach Anwendungsfall einstellbar sein, da die Gewichtung der lokalen und propagierten *Wartbarkeit* unterschiedlich sein kann. Ein einfacher Weg ein solches Verhältnis abzubilden, ist das gewichtete arithmetische Mittel, welches Büchter [7] genauer erläutert und mit dem die Relevanz beziehungsweise das Gewicht der miteinander

zu verrechnenden Bewertungen vorgegeben werden kann. Dafür wird die Formel (5.3) genutzt.

$$\frac{n \cdot Q_l^W(A) + 1 \cdot Q^W(B)}{n+1} \tag{5.3}$$

Mit der Hilfe des gewichteten Mittelwerts ist es also möglich, über die Variable n das Verhältnis zwischen lokaler und propagierter *Wartbarkeit* zu bestimmen. Die konkrete Festlegung der Gewichtung der propagierten Qualität hängt vom Betrachter und seinen Zielen ab und muss hier offen bleiben. Es muss hier also auf eine qualifizierte Begründung der gesetzten Gewichtungen verzichtet werden. Vielmehr geht es grundsätzlich zunächst einmal nur darum zu überprüfen, ob sich die Propagation der Qualität berechnen lässt. Deshalb wird hier im Sinne einer „Arbeitshypothese“ eine Gewichtung von lokaler zu propagierter *Wartbarkeit* in einem Verhältnis von 3 : 1 gesetzt, weil die propagierte *Wartbarkeit* nach Einschätzung des Autors etwa in diesem Verhältnis weniger Auswirkungen auf die aggregierte *Wartbarkeit* ($Q^W(A)$) hat als die lokale *Wartbarkeit* ($Q_l^W(A)$). Es muss an dieser Stelle betont werden, dass diese Setzung beliebig ist. In einer weiteren Studie sollten dem Nutzer transparent nachvollziehbare Vorschläge für angepasste und angemessene Gewichtungen gemacht werden.

Auf der Grundlage dieser gesetzten Gewichtung entspricht die aggregierte *Wartbarkeit* von A einem Wert von $6,48$ (siehe auch Abbildung 5.9). Die schlechtere *Wartbarkeit* von B hat in diesem Beispiel Auswirkungen auf die *Wartbarkeit* von A. Allerdings ist diese geringer als in dem vorherigen Beispiel ohne die Gewichtung.

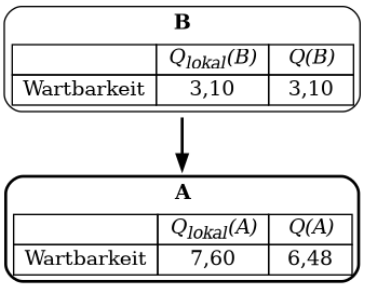
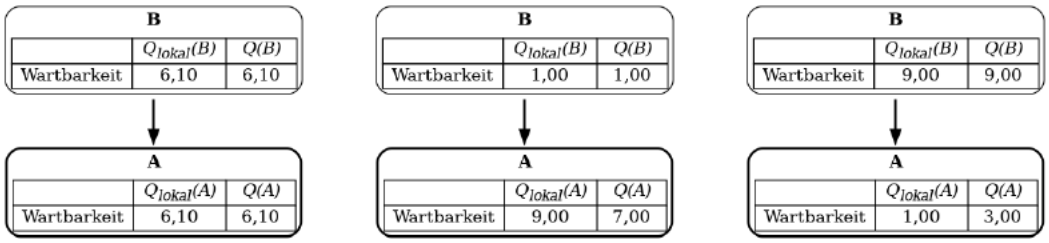


Abbildung 5.9: Einfache Qualitätspropagation mit aggregierter Wartbarkeit

Zur Validität der Berechnung müssen noch einige Edge-Cases betrachtet werden. Wenn die beiden Knoten A und B die gleichen Bewertungen der lokalen *Wartbarkeit* haben, dann sollte im Ergebnis die aggregierte *Wartbarkeit* beider Knoten dieser Bewertung

entsprechen. Außerdem sollte bei stark unterschiedlichen Bewertungen der lokalen *Wartbarkeit* der beiden Knoten die aggregierte *Wartbarkeit* von A immer näher an der lokalen *Wartbarkeit* von A liegen als an der lokalen *Wartbarkeit* von B.



(a) $Q_l^W(B) = 6,10$
$Q_l^W(A) = 6,10$

(b) $Q_l^W(B) = 1,00$
$Q_l^W(A) = 9,00$

(c) $Q_l^W(B) = 9,00$
$Q_l^W(A) = 1,00$

Abbildung 5.10: Mehrere Beispiele der Wartbarkeitspropagation

Zu diesen Edge-Cases zeigt die Abbildung 5.10 verschiedene Berechnungen aus dem Qualitätspropagationsprogramm. Aus Abbildung 5.10a ergibt sich, dass die aggregierten *Wartbarkeiten* bei gleicher lokaler *Wartbarkeit* der beiden Knoten B und A diesen entsprechen, und aus den beiden Abbildungen 5.10b und 5.10c lässt sich schließen, dass die Bewertung der lokalen *Wartbarkeit* des Knotens A bei großem Unterschied zur von B propagierten *Wartbarkeit* stärker gewichtet wird, also einen größeren Einfluss auf die aggregierte *Wartbarkeit* von A hat.

Sobald in einem Propagationsgraphen mehr als eine gerichtete Kante auf einen Knoten zeigt, eignet sich die oben entwickelte Formel zur Propagation der *Wartbarkeit* nicht mehr zur Berechnung. Ein Beispielgraph dazu ist in Abbildung 5.11 zu sehen. In diesem Fall propagieren die drei Knoten B, C und D zu A. Um die aggregierte *Wartbarkeit* von A zu berechnen, gibt es verschiedene Möglichkeiten.

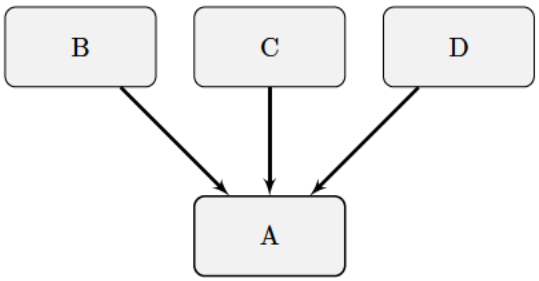


Abbildung 5.11: Propagationsgraph mit mehreren direkten Vorgängerknoten

Eine Möglichkeit wäre, den in Formel (5.3) beschriebenen gewichteten Mittelwert so zu erweitern, dass alle Vorgängerknoten jeweils mit einem Gewicht von 1 hinzuaddiert würden und der Nenner der Formel um die Anzahl der hinzugefügten Vorgängerknoten addiert wird. Für das Beispiel aus Abbildung 5.11 würde sich also die folgende Formel ergeben: $\frac{n \cdot Q_l^W(A)+1 \cdot Q_l^W(B)+1 \cdot Q_l^W(C)+1 \cdot Q_l^W(D)}{n+3}$.

Das würde allerdings auch bedeuten, dass mit steigender Anzahl an Vorgängerknoten der Einfluss der lokalen *Wartbarkeit* des Knotens A immer weiter abnimmt. Dieses Ergebnis ist nachvollziehbar, sofern sich das Verhältnis lokaler Software gegenüber genutzter Softwarebibliotheken entsprechend verhält. Wie auch immer dieses Verhältnis gemessen werden könnte – die Softwaregröße könnte ein Anhaltspunkt sein –, so ließe es die Betrachtung der Qualitätspropagation über den Propagationsgraphen beziehungsweise den Propagationsübergangsbaum nicht zu, diese Beobachtungen in Zahlen zu erfassen und in die Berechnung mit aufzunehmen.

Deshalb wird hier eine statische Gewichtung der lokalen *Wartbarkeit* gegenüber der *Wartbarkeit* aller Vorgängerknoten gewählt. Also müssen die *Wartbarkeiten* aller Vorgängerknoten aggregiert werden, bevor der gewichtete Mittelwert aus lokaler und propagierter *Wartbarkeit* berechnet werden kann.

Da die gerichteten Kanten eines Propagationsgraphen nicht gewichtet sind, sind alle Kanten gleichwertig. Entsprechend wird für diesen Fall ein einfacher Mittelwert aller propagierten *Wartbarkeiten* vorgeschlagen. Dafür wird im ersten Schritt ein Mittelwert aller zu einem Knoten v propagierten *Wartbarkeiten* berechnet. Die Formel (5.4) berechnet die propagierte *Wartbarkeit* ($Q^W_{propagiert}(v)$). Dafür werden die *Wartbarkeiten* aller Vorgängerknoten aufaddiert und im Anschluss durch die Anzahl der Vorgängerknoten geteilt. Auf dieser Grundlage kann die Formel zur Bestimmung der *Wartbarkeit* eines Knotens im Propagationsübergangsbaum entwickelt werden. Die Formel (5.5) beschreibt diese Berechnung und enthält dabei eine Fallunterscheidung: Sofern ein Knoten v keine Vorgängerknoten ($P(v)$) hat, wird, wie zuvor definiert, die lokale *Wartbarkeit* als aggregierte *Wartbarkeit* des Knotens angenommen. Wenn ein Knoten allerdings Vorgängerknoten hat, so wird ein gewichteter Mittelwert der lokalen *Wartbarkeit* und der zu v propagierten, *Wartbarkeit* ($Q^W_{propagiert}(v)$) gebildet. Dabei stellt n das Gewicht der lokalen *Wartbarkeit* gegenüber der propagierten *Wartbarkeit* dar.

$$Q^W_{propagiert}(v) = \frac{\sum_{i=1}^{|P(v)|} Q^W(P(v)_i)}{|P(v)|} \tag{5.4}$$

$$Q^W(v) = \begin{cases} Q_l^W(v) & P(v) = \{\} \\ \frac{n \cdot Q_l^W(v) + 1 \cdot (Q^W_{propagiert}(v))}{n+1} & \text{sonst} \end{cases} \tag{5.5}$$

In der Abbildung 5.12 ist ein Beispielgraph mit verschiedenen lokalen *Wartbarkeiten* und den jeweils aggregierten *Wartbarkeiten* zu sehen. Die aggregierten *Wartbarkeiten* der Knoten, auf die keine Kante gerichtet ist, entsprechen den lokalen Wartbarkeiten. Der Mittelwert der propagierten *Wartbarkeiten* ($Q^W_{propagiert}(A)$) entspricht in diesem Beispiel dem Wert $8,7$. Die propagierte *Wartbarkeit* ist also besser als die lokale *Wartbarkeit* von A $(7,6)$. Unter der Verwendung der Formel (5.5) entspricht die aggregierte *Wartbarkeit* von A $(Q^W(A))$ einer Bewertung von $7,88$. Die aggregierte Bewertung der *Wartbarkeit* von A ist also um $0,28$ besser als die lokale Bewertung der *Wartbarkeit*.

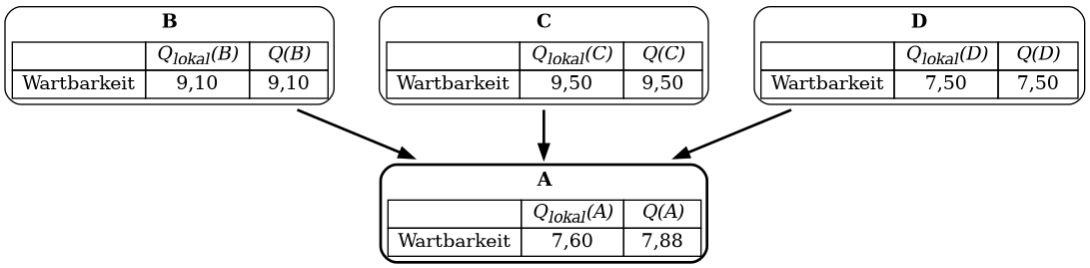


Abbildung 5.12: Qualitätspropagation mit mehreren direkten Vorgängerknoten

Durch dieses Vorgehen ist der Einfluss der propagierten *Wartbarkeit* eines Knotens davon abhängig, wie viele andere Knoten zum gleichen Zielknoten propagieren. Dieses Verhalten ist darin begründet, dass insgesamt auch mehrere Knoten in die Bewertung mit aufgenommen werden. Außerdem kann in diesem Fall von einem „größeren“ Softwareprodukt ausgegangen werden.

5.2.2 Die Aggregation lokaler und transitiv propagierter Qualität

Es muss auch betrachtet werden, wie die transitive Abhängigkeit aggregiert wird. In der Abbildung 5.13 ist ein Graph zu sehen, in dem der Knoten C zu B propagiert, B wiederum propagiert zu A. Wird nun A betrachtet, so ist zu überlegen, ob die Qualitätsaggregation von B zuvor umgesetzt werden muss, damit nachvollzogen werden kann, welche Qualität zu A propagiert wird.

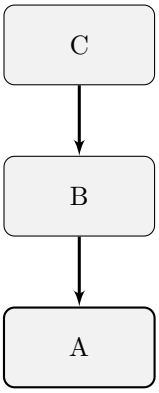


Abbildung 5.13: Propagationsgraph mit einem transitiven Vorgängerknoten

An dieser Stelle kann das oben bereits entwickelte Verfahren genutzt werden. Die Formeln (5.5) und (5.4) bilden eine rekursive Formel auf dem Propagationsgraphen ab. So wird die Qualität ausgehend vom zu betrachtenden Knoten rekursiv berechnet. Im Folgenden wird dieses Vorgehen näher erläutert und diskutiert.

Die transitive Qualitätspropagation kann als mehrschrittige Form der direkten Qualitätspropagation betrachtet werden. Die Abbildung 5.14 zeigt den Propagationsgraphen aus Abbildung 5.13 aufgeteilt in zwei Propagationsgraphen, wobei jeweils die ausgeblendeten Teile des Graphen ausgegraut sind. Indem in diesem Beispiel die Qualität von C zu B und die Qualität von B zu A propagiert, sollte die Qualität von C automatisch in der propagierten Qualität von A enthalten sein. Damit die Qualität von C auch transitiv zu A propagiert, muss die von B zu A propagierte Qualität nicht der lokalen Qualität von B entsprechen, sondern der bereits aggregierten.

(a) Propagationsgraph (C zu B) (b) Propagationsgraph (B zu A)

Abbildung 5.14: Transitive Qualitätspropagation in zwei Graphen dargestellt

Da zur Berechnung der propagierten *Wartbarkeit* ($Q^W_{propagiert}$) nicht die lokale *Wartbarkeit* (Q^W_l) der Vorgängerknoten, sondern die aggregierte *Wartbarkeit* (Q^W) genutzt wird, kann mit dieser Formel auch die transitive Propagation der Qualität abgebildet werden.

Diese rekursive Herangehensweise führt dazu, dass die Auswirkungen eines im Graphen weiter entfernten Knotens geringer ausfallen. In dem hier angeführten Beispiel hat deshalb, dass die lokale *Wartbarkeit* des Knotens C weniger Einfluss auf die aggregierte *Wartbarkeit* von A als die lokale *Wartbarkeit* von B. Die lokale *Wartbarkeit* von C ist nur in der aggregierten *Wartbarkeit* von A enthalten, weil sie zu einem Anteil in der aggregierten *Wartbarkeit* von B enthalten ist. Nachdem die Formel angewandt wurde, kommt man also zu folgender Anteilsverteilung der lokalen *Wartbarkeiten*: Die aggregierte *Wartbarkeit* von A setzt sich zu $\frac{3}{4}$ aus der lokalen *Wartbarkeit* von A, zu $\frac{3}{16}$ aus der lokalen *Wartbarkeit* von B und zu $\frac{1}{16}$ aus der lokalenm *Wartbarkeit* von C zusammen.

Dieses Verhalten ist aus der Sicht eines Entwicklers gut nachzuvollziehen: Beim Warten des Quellcodes eines Softwareprodukts ist es von Vorteil, wenn die *Wartbarkeit* der direkt genutzten Softwarebibliotheken gut bewertet ist, weil die *Wartbarkeit* auch Maße für die Verständlichkeit und Nachvollziehbarkeit von Quellcode enthält. Um die Abläufe im Quellcode einer Softwarebibliothek besser nachvollziehen zu können, und somit den lokalen Quellcode besser warten zu können, ist es hilfreich, wenn die Softwarebibliothek eine gute Bewertung der *Wartbarkeit* hat. In den meisten Fällen ist die *Wartbarkeit* der Softwarebibliotheken, von denen das Softwareprodukt nur transitiv abhängig ist, in diesem Schritt weniger wichtig, da der Softwareentwickler meist nicht so tief in die Programmabläufe hinein schauen muss. Wenn wie in diesem Beispiel die transitiv propagierte lokale *Wartbarkeit* des Knotens C nur $\frac{1}{16}$ der Bewertung der aggregierten *Wartbarkeit* von A ausmacht, erscheint das aus der Perspektive des Autors entsprechend plausibel.

Abbildung 5.15 zeigt ein Beispiel für die transitive Propagation der *Wartbarkeit*. Während die beiden Knoten A und B jeweils über eine bessere Bewertung der lokalen *Wartbarkeit* von $7,6$ verfügen, hat C eine schlechtere Bewertung der lokalen *Wartbarkeit* von nur $3,1$. Die schlechtere lokale *Wartbarkeit* von C gegenüber der anderen beiden Knoten hat einen größeren direkten Einfluss auf die *Wartbarkeit* von B als auf die *Wartbarkeit* von A.

Bei der transitiven Propagation wurde in dieser Arbeit ein stufenweises Vorgehen gewählt, da die Qualität einer Softwarebibliothek nur jenen Knoten beeinflusst, der eine direkte Abhängigkeit zu dieser Softwarebibliothek hat. Je nach Softwarequalitätsmerkmal wird dabei die Qualität unterschiedlich im Propagationsgraphen weitergereicht. Bei der *Wartbarkeit* wurde beispielsweise eine abnehmende Signifikanz der Softwarequalität mit

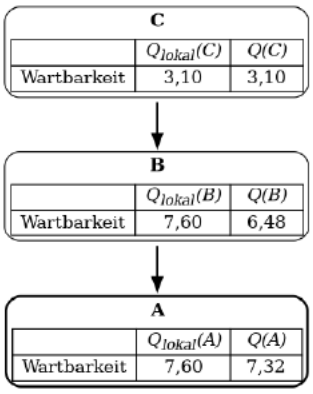
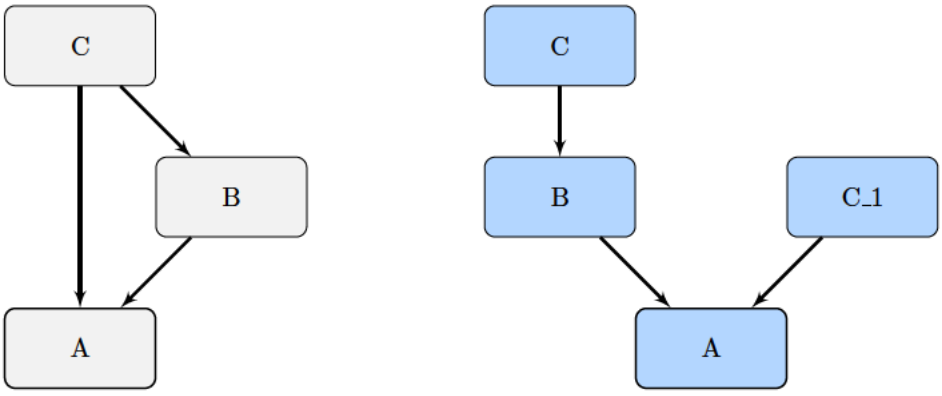


Abbildung 5.15: Beispiel der Qualitätspropagation mit transitiver Propagation

jedem Propagationsschritt beschrieben. Bei anderen Softwarequalitätsmerkmalen kann das anders aussehen (vergleiche 5.3 „Die Propagation der Sicherheit“).

5.2.3 Die Aggregation lokaler und mehrfach propagierter Qualität

Mehrfachpropagation meint im Rahmen dieser Arbeit den Zustand, in dem ein Knoten die Qualität über mehrere Wege zu einem anderen Knoten propagiert. Diese Form der Propagation kann nur vorkommen, wenn mindestens einer der Wege über mindestens einen anderen Knoten, also transitiv, verläuft. In der Abbildung 5.16 ist ein Propagationsgraph zu sehen, der eine solche Mehrfachpropagation enthält. Während die Qualität von B direkt zu A propagiert, propagiert sie zusätzlich transitiv über C zu A.

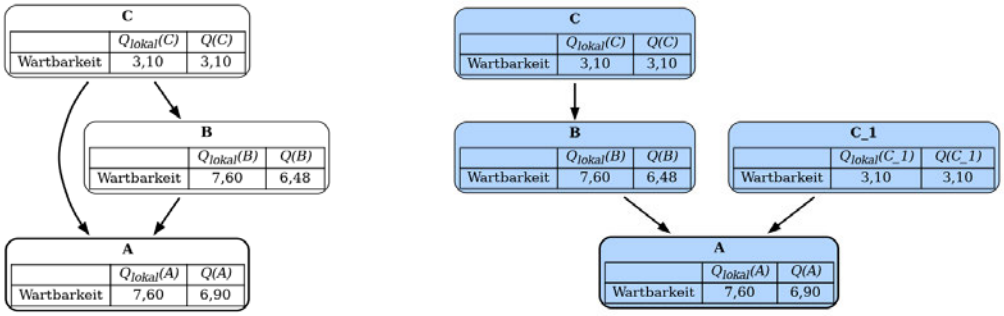


(a) Propagationsgraph (b) Propagationsübergangsbaum

Abbildung 5.16: Qualitätspropagation mit Mehrfachpropagation

In diesem Fall kann also die Qualität durch die Propagation mehrfach bei dem zu betrachtenden Knoten ankommen. Dieses Verhalten ist in dem Propagationsübergangsbaum gut zu beobachten (Abbildung 5.16), in dem zwei dem Knoten C des Propagationsgraphen entsprechende Knoten enthalten sind, C und C_1. Aufgrund der strukturellen Gegebenheiten erscheint ein solches Verhalten durchaus plausibel. Im Regelfall ist in einem Softwareprodukt eine Abhängigkeit nur einmal installiert, der zu bewertende Quellcode also einmal enthalten, aber er beeinflusst durch die Mehrfachpropagation auch mehrfach die Qualität des zu betrachtenden Knotens.

Die bisher aufgestellte Formel zur Berechnung der *Wartbarkeit* (5.5) kann also auch in diesem Fall ohne weitere Anpassungen genutzt werden. In Abbildung 5.17 ist ein Beispiel der Mehrfachpropagation zu sehen. Es ähnelt dem letzten Beispiel zur transitiven Propagation und zeigt die gleichen lokalen Wartbarkeiten, hinzugefügt wurde allerdings eine zusätzliche Kante von dem Knoten C direkt zum Knoten A.



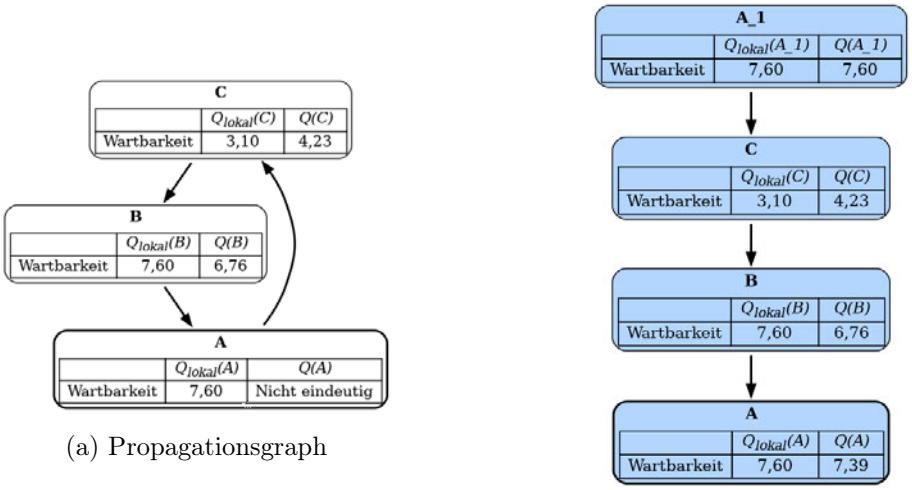
(a) Propagationsgraph (b) Propagationsübergangsbaum

Abbildung 5.17: Beispiel der Qualitätspropagation mit Mehrfachpropagation

Im direkten Vergleich der aggregierten *Wartbarkeiten* ist zu beobachten, dass die lokale *Wartbarkeit* des Knotens C mehr Auswirkungen auf die aggregierte Qualität von A hat. Statt zu einem Anteil von $\frac{1}{16} = \frac{2}{32}$, ist in diesem Beispiel die lokale *Wartbarkeit* von C zu einem Anteil von $\frac{5}{32}$ in der *Wartbarkeit* von A, die von B hingegen zu $\frac{3}{32}$ enthalten und die lokale *Wartbarkeit* von A zu $\frac{24}{32}$. Die *Wartbarkeit* von C ist also zu einem größeren Anteil in der *Wartbarkeit* von A enthalten als wenn A nur direkt oder nur transitiv von C abhängen würde.

5.2.4 Der Umgang mit zirkulären Abhängigkeiten

Ein strukturelles Problem der Softwarequalitätspropagation ist der Umgang mit Zyklen im Abhängigkeits- beziehungsweise Propagationsgraphen (vergleiche Kapitel 4.2 „Strukturelle Probleme der Qualitätspropagation“). Beim Aufstellen des Propagationsgraphen wurde der Umgang mit Zyklen bereits behandelt (vergleiche Kapitel 5.1 „Das Bilden eines Propagationsgraphen“). Mit dem dort beschriebenen Ansatz muss man zwar einen Informationsverlust in Kauf nehmen, da der Zyklus nicht bis ins unendliche aufgelöst wird, es werden allerdings alle Übergänge über die Qualität propagiert. Es stellt sich die Frage, ob diese Herangehensweise bei der Propagation der *Wartbarkeit* mit den zuvor beschriebenen Formeln plausibel ist. Das Qualitätspropagationsprogramm nutzt die Formel (5.5) und transformiert den definierten Propagationsgraphen zu einem Propagationsübergangsbaum, bevor die Qualität berechnet wird.



(a) Propagationsgraph

(b) Propagationsübergangsbaum

Abbildung 5.18: Beispiel der Qualitätspropagation bei zirklärer Propagation

Die Abbildung 5.18 zeigt den von dem Qualitätspropagationsprogramm ausgegebenen Graphen mit aggregierten Wartbarkeiten. Dabei wird über den Weg $ACBA$ propagiert. Wenn also die *Wartbarkeit* von A betrachtet wird, so ist sie in diesem Fall doppelt enthalten. Da die lokale *Wartbarkeit* von A höher als die zu A propagierte *Wartbarkeit* ist, ist die aggregierte *Wartbarkeit* in diesem Beispiel auch höher als die *Wartbarkeit* aus dem Beispiel Abbildung 5.15, bei dem nur über den Weg CBA propagiert wird. Der Zyklus wirkt sich also auf das Gesamtergebnis aus. Die Anteile der lokalen *Wartbarkeiten* der Knoten in der aggregierten *Wartbarkeit* von A sehen wie folgt aus: Die lokale *Wartbarkeit*

von A ist zu $\frac{3}{4} + \frac{1}{64} = \frac{49}{64}$ enthalten, die von B zu $\frac{12}{64}$ und die von C zu $\frac{3}{64}$. Das Ergebnis kann also mit dem Beispiel zur transitiven Propagation der *Wartbarkeit* verglichen werden (vergleiche Abbildung 5.15). Da hier die gleichen lokalen *Wartbarkeiten* gesetzt wurden, sieht man, dass die aggregierte Qualität $Q^W(A) = 7,39$ dieses Beispiels durch den Zyklus, also durch den größeren Einfluss der lokalen Qualität von A, besser bewertet wird als in dem Beispiel ohne Zyklus.

Ein komplexeres Exempel ist in Abbildung 5.19 zu sehen. Es verknüpft die zirkuläre Abhängigkeit mit der Mehrfachabhängigkeit und bildet den Beispielgraphen der Abbildung 5.6 (Seite 44) ab. Die aggregierten Werte der Knoten B, C, D, E, F können nicht eindeutig im Graphen angezeigt werden, da sie je nach Weg durch den Graphen unterschiedliche Werte annehmen können. Wie in Kapitel 5.1 „Das Bilden eines Propagationsgraphen“ bereits beschrieben werden in diesem Graphen die Worte $BFEBDCBA$ und $BDCBFEBA$ propagiert. Der Propagationsübergangsbaum, in dem auch die aggregierten Werte angezeigt werden können, ist im Anhang A.3 (Seite 84) zu sehen.

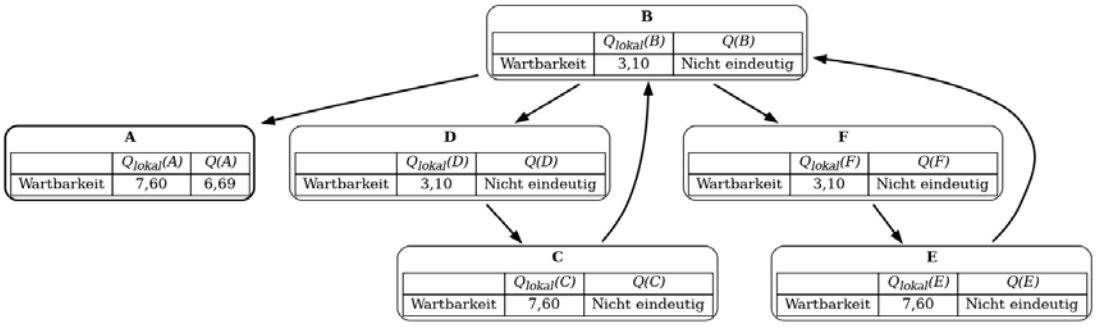


Abbildung 5.19: Propagationsgraph (zirkuläre Propagation). Der entsprechende Propagationsübergangsbaum ist in Anhang A.3 zu sehen.

Bei der in diesem Beispiel gewählten Betrachtung der *Wartbarkeit* ist eine Auflösung des Zyklus in dieser Form möglich, und die Ergebnisse erscheinen plausibel. Das ist auch dadurch zu erklären, dass die Gewichtung lokaler zu propagierter *Wartbarkeit* hier zugunsten der lokalen *Wartbarkeit* ausfällt, weshalb der Anteil an der aggregierten *Wartbarkeit* mit der steigenden Entfernung vom zu betrachtenden Knoten stark abfällt.

Da die bisher aufgestellte Formel zur Berechnung der aggregierten *Wartbarkeit* im Propagationsübergangsbaum auch dann plausible Ergebnisse liefert, wenn dieser aus einem Propagationsgraphen mit Zyklen erstellt wurde, wird die Propagation der *Wartbarkeit* abschließend wie folgt definiert:

Definition 4 (Die Propagation der Wartbarkeit) *Sei v ein Knoten eines Propagationsübergangsbaums, so beschreibt $Q^W(v)$ die Wartbarkeit dieses Knotens. Es gilt:*

$$Q^W(v) = \begin{cases} Q_l^W(v) & P(v) = \{\} \\ \frac{n \cdot Q_l^W(v) + 1 \cdot (Q^W_{propagiert}(v))}{n+1} & sonst \end{cases} \tag{5.6}$$

$$Mit \quad Q^W_{propagiert}(v) = \frac{\sum_{i=1}^{|P(v)|} Q_l^W(P(v)_i)}{b} \tag{5.7}$$

Wobei n die Gewichtung der lokalen gegenüber der propagierten Wartbarkeit sei.

5.2.5 Die Aggregationsmethoden im Zusammenspiel

Nachdem die Methoden zur Behandlung der strukturellen Problemfälle anhand des Beispiels der *Wartbarkeit* nacheinander entwickelt wurden, können hier noch einmal die Probleme im Zusammenspiel betrachtet werden.

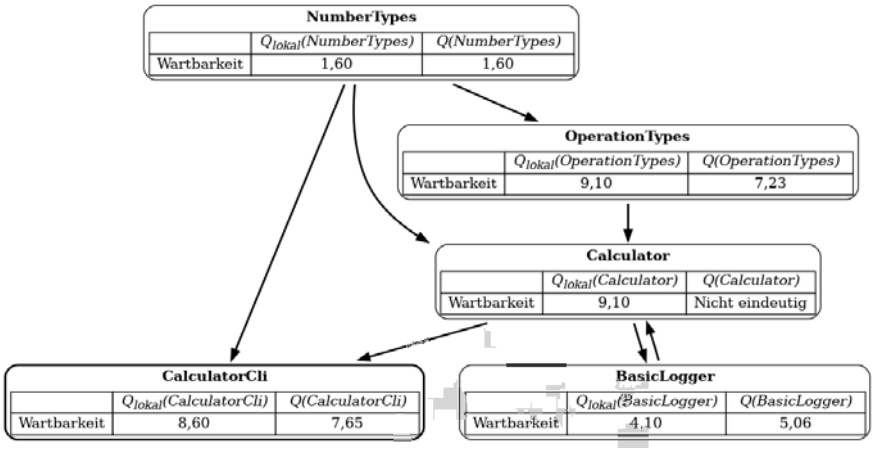


Abbildung 5.20: Demonstrationsbeispiel mit propagierter *Wartbarkeit*. Der entsprechende Propagationsübergangsbaum ist in Abbildung 5.21 zu sehen.

In der Abbildung 5.20 ist der Propagationsgraph des Demonstrationsbeispiels mit gesetzter lokaler *Wartbarkeit* und berechneter aggregierter *Wartbarkeit* zu sehen. Die Abbildung 5.21 zeigt den dazugehörigen Propagationsübergangsbaum. Während die beiden Knoten *OperationTypes* und *Calculator* beide relativ hohe Werte für lokale *Wartbarkeiten* von $9,1$ haben und auch *CalculatorCli* eine relativ hohe Bewertung der lokalen

Wartbarkeit von 8, 6 hat, so zeigen die beiden Knoten *NumberTypes* und *BasicLogger* geringere Bewertungen der lokalen *Wartbarkeit* mit lediglich 1, 6 und 4, 1. Die aggregierte *Wartbarkeit* des Knotens *Calculator* kann im Propagationsgraphen nicht angezeigt werden, da sie in dem Graphen unterschiedliche Werte annehmen kann. Dieses Verhalten ist im Propagationsübergangsbaum zu beobachten. Hier hat der Knoten *Calculator* eine aggregierte *Wartbarkeit* von 7, 98, der Knoten *Calculator_1* hat allerdings eine aggregierte *Wartbarkeit* von 7, 93. Die zum Knoten *CalculatorCli* propagierte *Wartbarkeit* von *Calculator* beträgt jedoch 7, 98. Trotz des relativ geringen Einflusses der propagierten *Wartbarkeit* werden durch diese Bewertungen die *Wartbarkeiten* der anderen Knoten sichtlich beeinflusst. Vor allem der mehrfach zum Knoten *CalculatorCli* propagierende Knoten *Numbertypes* hat einen starken negativen Einfluss auf die Bewertung der *Wartbarkeit*.

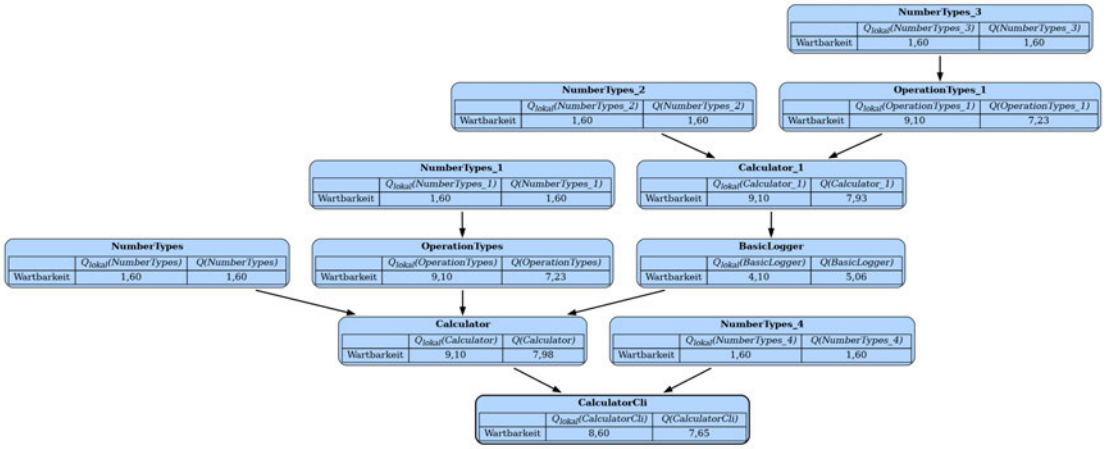


Abbildung 5.21: Demonstrationsbeispiel mit propagierter *Wartbarkeit* (Propagationsübergangsbaum)

Die verschiedenen Aggregationsmethoden, die in diesem Kapitel zum Behandeln der strukturellen Probleme der Qualitätspropagation entwickelt wurden, funktionieren also auch im Zusammenspiel. Außerdem produzieren sie nachvollziehbare und dadurch glaubwürdige Ergebnisse, wobei die Gewichtung der lokalen gegenüber der propagierten Qualität je nach Anwendungszweck und Domäne vom Nutzer eingestellt werden muss.

Die Abbildung 5.22 zeigt das gleiche Demonstrationsbeispiel mit den gleichen lokalen Bewertungen der Qualität wie im vorangegangenen Beispiel. Allerdings wurde der Berechnung ein anderes Verhältnis zwischen lokaler und propagierter *Wartbarkeit* zugrunde gelegt. Die lokale Qualität steht in diesem Beispiel in einem Verhältnis von 1 : 1 zur

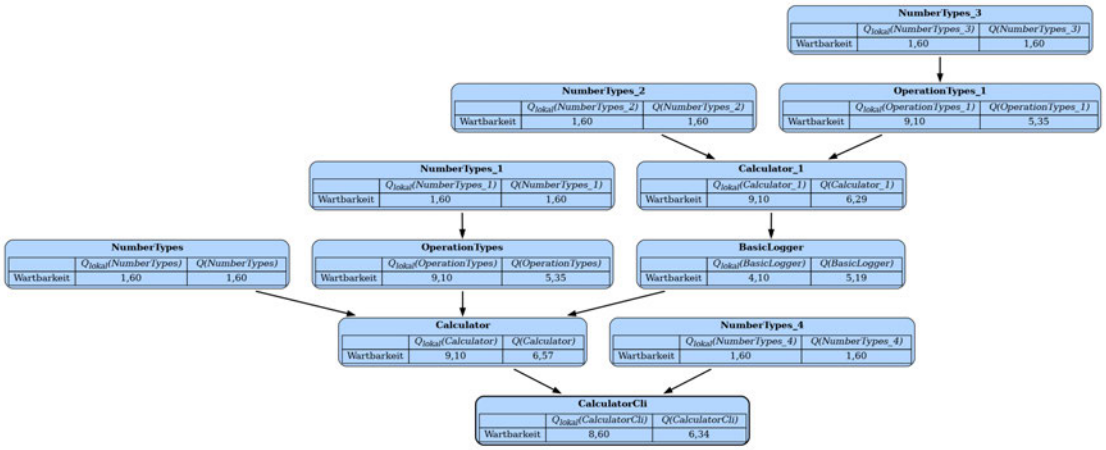


Abbildung 5.22: Propagationsübergangsbaum des Demonstrationsbeispiels, berechnet mit einem Verhältnis von 1 : 1 lokaler zu propagierter Wartbarkeit

propagierten Qualität. Ein Nutzer, der ein größeres Interesse an der propagierten *Wartbarkeit* hat, kann so das Verhältnis der propagierten *Wartbarkeit* erhöhen. Während im ersten Beispiel bei einem Verhältnis von 3 : 1 von lokaler zu propagierter *Wartbarkeit* der Knoten *CalculatorCli* mit einer aggregierten *Wartbarkeit* von $7,65$ bewertet wurde, beträgt dieser Wert im zweiten Beispiel $6,34$.

Es ist also für den Nutzer möglich, die Signifikanz der propagierten *Wartbarkeit* in dem betrachteten Propagationsübergangsbaum nach seinen Bedürfnissen einzustellen und somit verschiedenen Anwendungszwecken gerecht zu werden.

5.3 Die Propagation der Sicherheit

Beim Betrachten der strukturellen Probleme wurde nur das Softwarequalitätsmerkmal der *Wartbarkeit* betrachtet. Um ein vollständiges Modell zur Behandlung der Softwarequalitätspropagation zu erstellen, ist es aber notwendig, auch für die anderen Merkmale entsprechende Formeln zur Aggregation der propagierten Softwarequalitätsmerkmale zu entwickeln. Diese propagieren allerdings nicht unbedingt genauso wie die *Wartbarkeit*. Dafür müssen also alle Softwarequalitätsmerkmale genau untersucht und passende Aggregationsmethoden gefunden werden. In dieser Arbeit können aber nicht alle Merkmale thematisiert werden, das würde hier den Rahmen sprengen. Im Folgenden wird daher nur die *Sicherheit* als weiteres und besonders wichtiges Softwarequalitätsmerkmal untersucht.

Die *Sicherheit* (Q^S) propagiert als Softwarequalitätsmerkmal im Vergleich zur *Wartbarkeit* mit einem viel höheren Gewicht. Während der Einfluss der *Wartbarkeit* mit jedem Propagationsschritt abnimmt, sollte eine Sicherheitsbewertung nicht mit steigender Knotendistanz an Relevanz verlieren, weshalb die für die *Wartbarkeit* definierte Formel nicht verwendet werden kann. Selbst bei einem Verhältnis von 1 : 1 würde nach dem Schema der *Wartbarkeit* der Einfluss der Sicherheitsbewertung mit jeder Propagation nachlassen.

Stattdessen kann zur Entwicklung einer Aggregationsmethode der Umgang mit Sicherheitsbewertungen in der Analyse der lokalen Qualität eines Knotens genauer betrachtet werden. In einer früheren Arbeit, in der das Softwareanalysewerkzeug SonarCloud[1] untersucht wurde, konnte festgestellt werden, dass dieses Werkzeug zur Berechnung der lokalen *Sicherheit* (Q_l^S) nach einem Minimum-Prinzip aggregiert [39]. Die lokale Bewertung der *Sicherheit* basiert auf dem Minimum der verschiedenen lokal bewerteten *Sicherheiten*. Die *Sicherheit* des gesamten Softwareprodukts wird an der Stelle also durch die Bewertung des schwächsten Glieds bestimmt.

In der Propagation der *Sicherheit* über die Struktur sollte ein ähnliches Verfahren eingesetzt werden, damit eine schlechte Bewertung der *Sicherheit*, die als kritisch eingestuft werden kann, bei der transitiven Propagation zum betrachtenden Knoten nicht „verwässert“. Deshalb wird an dieser Stelle auch ein Aggregationsverhalten über ein Minimum vorgeschlagen. Das heißt, die *Sicherheit* eines Knotens setzt sich aus dem Minimum der propagierten und lokalen *Sicherheit* zusammen. Die Formel (5.8) zeigt die Propagation der *Sicherheit*. Die *Sicherheit* des Knotens v wird entsprechend aus dem Minimum der lokalen *Sicherheit* von v ($Q_l^S(v)$) und dem Minimum aller propagierten *Sicherheiten* bestimmt.

$$Q^S(v) = min(Q_l^S(v), min(Q^S(P(v)_1), \ldots, Q^S(P(v)_n))) \tag{5.8}$$
$$Wobei \quad n = |P(v)|$$

Da diese Formel in der Propagation keine Veränderung durchführt, kann sie auch vereinfacht als das Minimum der *Sicherheiten* aller direkten oder transitiven Vorgänger von v dargestellt werden. Sei $P^P(v)$ die Menge aller direkten oder transitiven Vorgänger von v. In der daraus resultierenden Formel (5.9) entfällt die Rekursion. Somit gilt:

[1] `https://sonarcloud.io`

$$Q^S(v) = min(Q_l^S(v), min(Q_l^S(P^P(v)_1), \ldots, Q_l^S(P^P(v)_n))) \tag{5.9}$$

$$Wobei \;\; P^P(v) = die\ Menge\ aller\ direkten\ oder\ transitiven\ Vorgänger\ von\ v$$

$$Und \;\; n = |P^P(v)|$$

Die Propagation der *Sicherheit* wird wie folgt definiert:

Definition 5 (Die Propagation der Sicherheit) *Sei v ein Knoten eines Propagationsübergangsbaums, so beschreibt $Q^S(v)$ die Wartbarkeit dieses Knotens. Es gilt:*

$$Q^S(v) = min(Q_l^S(v), min(Q_l^S(P^P(v)_1), \ldots, Q_l^S(P^P(v)_n))) \tag{5.10}$$

$$Wobei \;\; P^P(v) = die\ Menge\ aller\ direkten\ oder\ transitiven\ Vorgänger\ von\ v$$

$$Und \;\; n = |P^P(v)|$$

Abbildung 5.23 zeigt einen Graphen, in dem die Aggregation der propagierten Sicherheitsbewertungen umgesetzt wurde. Abgesehen von dem Knoten C haben alle anderen Knoten eine Sicherheitsbewertung von $7,6$ oder höher. Die schlechtere Bewertung von $3,1$ des Knotens C propagiert allerdings über den Knoten B zu A. So ist die aggregierte Sicherheitsbewertung $Q^S(A) = 3,1$. Die Bewertung der *Sicherheit* ist in dieser Betrachtungsweise also fokussiert auf den schwächsten Vorgängerknoten im Propagationsgraphen beziehungsweise Propagationsübergangsbaum.

Die hier präsentierte Methode zur Aggregation propagierter Bewertungen der *Sicherheit* ist also grundsätzlich anders als die zuvor präsentierte Methode zur Aggregation der Wartbarkeiten, und das ist mit der Diversität der Softwarequalitätsmerkmale zu begründen. Die *Sicherheit* direkter und transitiver Vorgängerknoten muss – abhängig vom Betrachter und der Domäne – einen sehr viel größeren Einfluss auf die *Sicherheit* des Knotens haben. In diesem Fall kann eine besonders gute Bewertung der *Sicherheit* eines Vorgängerknotens (D) die besonders schlechte Bewertung der *Sicherheit* eines anderen Vorgängerknotens (C) nicht ausgleichen.

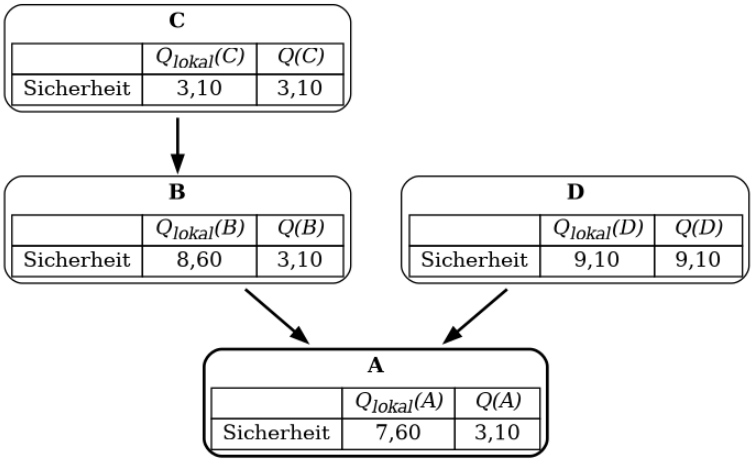


Abbildung 5.23: Propagationsgraph mit aggregierten Sicherheitsbewertungen

5.4 Ergebnis

Im Ergebnis zeigt diese Arbeit, dass es möglich ist, einen Propagationsgraphen aus einem Abhängigkeitsgraphen zu erstellen. Darüber hinaus konnte gezeigt werden, dass sich die Zyklen eines Propagationsgraphen mit dem hier entwickelten Propagationsübergangsbaum so auffächern lassen, dass eine Annäherung an die durch Zyklen entstehende unendliche Qualitätspropagation möglich ist. Außerdem wurde anhand zweier Merkmale beispielhaft gezeigt, dass eine einfache Qualitätspropagation der Softwarequalitätsmerkmale anhand des Propagationsübergangsbaums plausibel nachvollziehbar dargestellt werden kann. Dafür wurde anhand der sehr unterschiedlichen Beispiele *Wartbarkeit* und *Sicherheit* das Propagationsverhalten für transitive Propagation und Mehrfachpropagation genauer untersucht, und es wurden Formeln entwickelt, die ihre Qualitätspropagation beschreiben.

Für die *Wartbarkeit* wurde eine Formel zur schrittweisen Propagation entwickelt, die darauf basiert, dass die aggregierte *Wartbarkeit* (Q^W) eines Knotens aus seiner lokalen *Wartbarkeit* (Q^W_l) und den aggregierten *Wartbarkeiten* seiner Vorgängerknoten besteht. Da die aggregierte *Wartbarkeit* seiner Vorgängerknoten wiederum berechnet werden muss, handelt es sich in diesem Fall um eine rekursive Formel zur Berechnung der aggregierten *Wartbarkeit*. Diese Herangehensweise zeigt im Ergebnis, dass die *Wartbarkeit* einer Softwarebibliothek, von der ein Softwareprodukt abhängig ist, zu einem gewissen Anteil die *Wartbarkeit* des Softwareprodukts beeinflusst. Die *Wartbarkeit* transitiver Abhängigkei-

ten beeinflusst also das Softwareprodukt ebenfalls und zwar indirekt, da sie einen Anteil der *Wartbarkeit* der direkten Abhängigkeiten ausmacht.

Für die *Sicherheit* wurde eine Formel entwickelt, die eine Bewertung der aggregierten *Sicherheit* (Q^S) anhand der schlechtesten lokalen Sicherheitsbewertung aller direkten und indirekten Vorgängerknoten im Propagationsübergangsbaum sowie der eigenen lokalen *Sicherheit* (Q_l^S) bildet. Die Propagation der *Sicherheit* propagiert also sehr verschieden im Vergleich zur *Wartbarkeit*. Bei der *Sicherheit* macht es keinen Unterschied, ob die Abhängigkeit zu einer Softwarebibliothek direkt oder transitiv ist, weil eine schlechte Sicherheitsbewertung immer Einfluss auf das Softwareprodukt hat, unabhängig davon, ob diese aus einer direkten oder transitiven Abhängigkeit zu einer Softwarebibliothek entsteht.

Zum Abschluss soll an dieser Stelle noch einmal die Plausibilität der in dieser Arbeit entwickelten Konzepte im Zusammenspiel reflektiert werden. Dafür wird das eingangs vorgestellte Demonstrationsbeispiel herangezogen, und es werden die bisher entwickelten Methoden zur Softwarequalitätsaggregation genutzt. Die gesetzte Gewichtung der lokalen zur propagierten *Wartbarkeit* liegt dabei wie zuvor auch bei einem Verhältnis 3 : 1. Abbildung 5.24 zeigt einen Propagationsübergangsbaum des Demonstrationsbeispiels mit den in dieser Arbeit entwickelten Propagations- und Aggregationsmethoden.

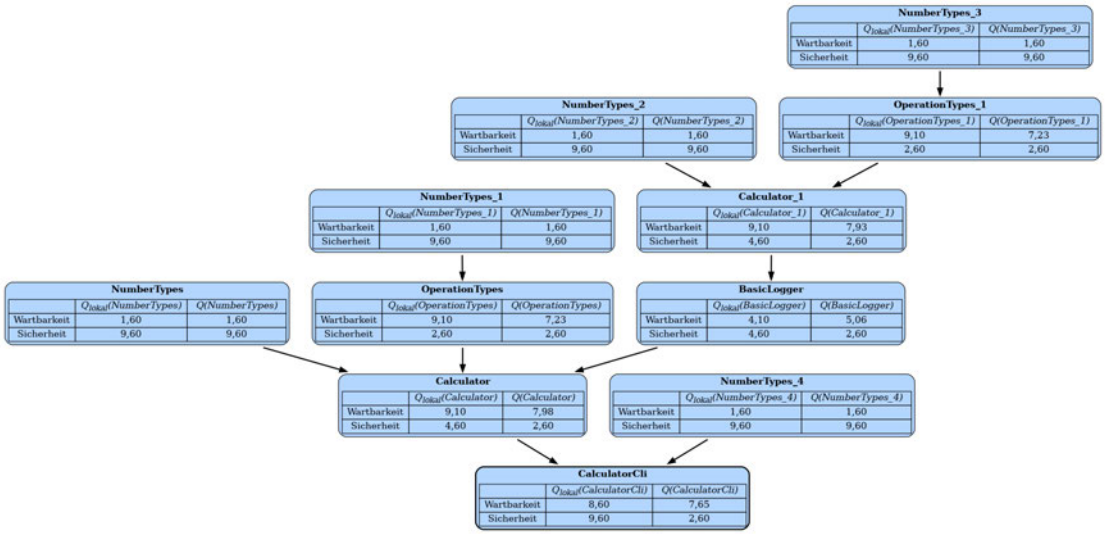


Abbildung 5.24: Demonstrationsbeispiel (*Wartbarkeit* und *Sicherheit* propagiert)

Wie eingangs beschrieben ist die Softwarequalitätsbewertung in der Praxis meist auf die lokale Qualität eines Knotens beschränkt. Beim Demonstrationsbeispiel wird in dem

Fall also bei der Qualitätsbewertung des Knotens *CalculatorCli* lediglich die Qualität dieses Knotens betrachtet, obwohl ein Großteil des Programmcodes aus den verschiedenen benutzten Softwarebibliotheken bereitgestellt wird. Der auf die hier zu betrachtenden Softwarequalitätsmerkmale reduzierte lokale Qualitätsvektor des Knotens sieht wie folgt aus: $Q_l(CalculatorCli) = \left(\; Q_l^W(v) = 8,6 \quad Q_l^S(v) = 9,6 \; \right)$.

Anhand eines solchen Qualitätsvektors wird in der Softwarequalitätsanalyse, die nur die lokale Softwarequalität betrachtet, die Qualität eines Softwareprodukts beschrieben. Ihm fehlen allerdings für eine umfassendere Betrachtung der Softwarequalität die Möglichkeiten zur Bewertung der Qualität der genutzten Softwarebibliotheken. Die in dieser Arbeit entwickelten Methoden der Softwarequalitätspropagation helfen dabei, einen aussagekräftigeren Qualitätsvektor zu bilden. In diesem ist die von anderen Knoten propagierte Softwarequalität mit in die Bewertung aufgenommen. So ergibt sich für die gleichen Knoten in dem gleichen Szenario ein Softwarequalitätsvektor der wie folgt aussieht: $Q(CalculatorCli) = \left(\; Q^W(v) = 7,82 \quad Q^S(v) = 2,6 \; \right)$.

Im direkten Vergleich der beiden Softwarequalitätsvektoren fällt auf, dass die Sicherheitsbewertung stark von der Bewertung der lokalen Qualität abweicht. Während die lokale *Sicherheit* mit $9,6$ bewertet ist, wird für die aggregierte *Sicherheit* ein Wert von $2,6$ angeführt. Je nach Anwendungsfall könnte eine Bewertung von $9,6$ für die *Sicherheit* sehr gut sein und signalisieren, dass kein Handlungsbedarf bestünde, eine Sicherheitsbewertung von $2,6$ aber sofortigen Handlungsbedarf signalisieren. Dieses Beispiel macht deutlich, dass gerade bei der Sicherheitsbewertung eine Betrachtung des gesamten Softwaresystems wichtig ist und der Softwarequalitätsvektor der aggregierten Softwarequalität als Grundlage der Qualitätseinschätzung genutzt werden sollte.

Nicht für jedes Softwarequalitätsmerkmal ist der Unterschied durch die Qualitätspropagation so eindeutig wie bei dem Merkmal *Sicherheit*. Die *Wartbarkeit* ist beispielsweise ein Softwarequalitätsmerkmal, das weniger stark von der Qualitätspropagation profitiert, beziehungsweise beeinträchtigt wird. Hier fallen die Einflüsse der genutzten Softwarebibliotheken sehr viel geringer aus, weil die lokale gegenüber der propagierten *Wartbarkeit* stärker gewichtet wird. In dem oben angeführten Beispiel reduziert sich die Bewertung der *Wartbarkeit* des Knotens *CalculatorCli* von $8,6$ auf $7,82$. Dieser Unterschied könnte gravierend genug sein, einen Nutzer zu motivieren, aktiv an einer Verbesserung der Bewertung zu arbeiten. Eine umfangreichere Bewertung der *Wartbarkeit*, die auch die *Wartbarkeit* der genutzten Softwarebibliotheken beinhaltet, ist also auch hier sinnvoll.

Es ergeben sich also aus dieser Arbeit die folgenden Erkenntnisse:

Die Bewertung nur der lokalen Softwarequalität kann nicht als umfassende Betrachtung der Qualität eines Softwareprodukts angesehen werden, da zumindest die Qualität der genutzten Softwarebibliotheken auch unter dem Aspekt der Propagation mitbetrachtet werden muss. Die genutzten Softwarebibliotheken können im Rahmen der Gesamtbewertung genauer daraufhin untersucht werden, wie die Beeinflussung der Qualität durch die Softwarebibliotheken geschieht. Dabei kommt diese Arbeit zu dem Ergebnis, dass die Softwarequalität, aufgeschlüsselt auf verschiedene Softwarequalitätsmerkmale, grundsätzlich durch einen Graphen propagieren kann, der eine Abhängigkeitsstruktur abbildet. Dabei muss allerdings auf die Besonderheiten der einzelnen Softwarequalitätsmerkmale eingegangen werden, und es müssen jeweils passende Methoden zur Aggregation der propagierten Qualität gefunden werden. Die resultierende aggregierte Qualität führt also zu einer deutlich umfassenderen Qualitätsaussage über ein Softwareprodukt, als die ausschließliche Betrachtung der lokalen Softwarequalität.

Was hier beispielhaft an zwei als besonders relevant identifizierten Qualitätsmerkmalen aufgezeigt wurde bedarf allerdings weiterer Forschung um zu einem Gesamtbild zur Bewertung des Einflusses von Softwarequalitäten auf lokale Software zu gelangen. Die Untersuchung weiterer Aspekte, wie etwa der unterschiedlichen Softwarequalitätsmerkmale des ISO-25010-Standards auf ihr Propagations- und Aggregationsverhalten, sind noch zu leisten. Je nach Genauigkeit der angestrebten Beurteilung müssen dabei auch die verschiedenen Softwarequalitätsuntermerkmale betrachtet werden.

6 Diskussion und Ausblick

Obwohl Open-Source Softwarebibliotheken in fast jeder modernen Software eingebaut werden, gibt es bislang keine Methode, die erfasst, wie sich dies auf die Qualität des lokalen Softwareprodukts auswirkt. Diese Arbeit hat sich die Frage vorgelegt, wie Ansätze entwickelt werden können, die die Qualität von Softwarebibliotheken und deren Propagation beurteilen können. Insgesamt zeigt sich, dass es mit den in dieser Arbeit entwickelten Methoden der Bewertung der Qualität von Softwarebibliotheken möglich ist, eine umfassendere Bewertung von Softwarequalität durchzuführen. Dazu ist es grundsätzlich möglich, Softwarequalität in einer Abhängigkeitsstruktur so darzustellen und zu betrachten, dass die Qualität entgegen dem Abhängigkeitsverhältnis propagiert. Diese Betrachtung der Softwarequalität eröffnet neue Wege in der Softwarequalitätsbewertung.

Dabei kann sowohl strukturellen als auch nicht-strukturellen Herausforderungen der Qualitätspropagation Rechnung getragen werden. Die Handhabung der Zyklen im Propagationsgraphen über das Bilden des Propagationsübergangsbaums bewirkt, dass auch Propagationsgraphen mit Zyklen berechnet werden können. Dabei bildet der Propagationsübergangsbaum die Zyklen des Propagationsgraphen bestmöglich ab, ohne dass Kanten des Propagationsgraphen mehrfach in den möglichen Wegen eines Propagationsübergangsbaums vorhanden sind.

Die Ergebnisse der hier entwickelten Methoden zur Qualitätspropagation halten einer ersten und vorläufigen Plausibilitätsprüfung stand. Zwar basieren die Einschätzungen der Plausibilität auf subjektiv gesetzten Beurteilungen des Autors, was bedeutet, dass ein tatsächlicher Grad von Plausibilität zu gegebenem späterem Zeitpunkt empirisch verifiziert werden müsste. Dies kann beispielsweise durch ein auf Umfragen gestütztes Verfahren geschehen, in dem die Einschätzung mehrerer unterschiedlicher qualifizierter Nutzer zu verschiedenen Aspekten der Softwarequalität eines Softwareprodukts abgefragt wird.

Die vorgeschlagen Methoden wurden beispielhaft anhand der Kriterien der *Wartbarkeit* und der *Sicherheit* entwickelt. Daraus ergeben sich verschiedene Einschränkungen, die

im Folgenden reflektiert und mit Blick auf Anknüpfungspunkte für künftige Forschung diskutiert werden. Insbesondere stellen sich Fragen zur Konfigurierbarkeit, Genauigkeit und Übertragbarkeit der Methoden auf andere Bereiche.

6.1 Konfigurierbarkeit

Bei der hier vorgeschlagenen Methode zur Aggregation propagierter und lokaler *Wartbarkeit* bleibt es dem Nutzer überlassen, wie er die lokale zur propagierten *Wartbarkeit* gewichtet. Die hier entwickelte Methode zur *Sicherheit* bietet allerdings keine Möglichkeit der Konfiguration. Hier könnte alternativ eine andere Aggregationsmethode entwickelt werden, die nicht auf der Basis eines Minimums arbeitet und dem Nutzer eine Alternative zu der hier vorgestellten, recht strikten Methode zur Sicherheitsaggregation böte.

Die Konfigurierbarkeit hat allerdings Vor- und Nachteile. Zum einen kann die Softwarequalitätspropagation über eine Konfiguration genau auf die Bedürfnisse des Nutzers eingestellt werden, zum anderen muss der Nutzer aber detailliertes Wissen über die verschiedenen Softwarequalitätsmerkmale haben, um die Gewichtung seinen Bedürfnissen anpassen zu können. Es sollte also in einem Softwarequalitätsmodell immer eine vorgeschlagene Standardeinstellung geben, damit der Nutzer sich nicht zwingend mit dem Propagationsverhalten der verschiedenen Softwarequalitätsmerkmale auseinandersetzen muss.

6.2 Genauigkeit

Während die Plausibilität der in dieser Arbeit besprochenen Ansätze bereits grundsätzlich bejaht wurde, bedarf die Genauigkeit der Ergebnisse auf verschiedenen Ebenen noch einer eingehenden Betrachtung.

6.2.1 Softwarequalitätsmerkmale

Die Softwarequalitätspropagation wurde anhand von zwei verschiedenen Softwarequalitätsmerkmalen des ISO-25010-Standards untersucht, und mögliche Methoden zur Aggregation der propagierten Qualität wurden aufgezeigt. Da die Softwarequalitätsmerkmale

nicht in ihrer vollen Komplexität betrachtet und somit auch Untermerkmale mit weiteren Ausdifferenzierungen nicht weiter beachtet werden konnten, sollte in einem weiteren Schritt auch untersucht werden, ob die hier gewählte Auflösung der Propagation und Aggregation den relativ diversifizierten Softwarequalitätsmerkmalen, -untermerkmalen und Unter-Untermerkmalen gerecht wird.

Um die Genauigkeit besser einschätzen zu können, wird im Folgenden ein Beispiel aus der *Wartbarkeit* nach ISO-25010 genauer betrachtet. Die Untermerkmale der *Wartbarkeit* sind *Modularität*, *Wiederverwendbarkeit*, *Analysierbarkeit*, *Modifizierbarkeit* und *Testbarkeit*. Außerdem ließen sich aus dem ISO-25010-Standard noch weitere Untermerkmale der *Wartbarkeit* ableiten, die nicht direkt als solche gekennzeichnet sind. So beschreibt der Standard beispielsweise, dass die *Wartbarkeit* auch die Installation von Updates oder Upgrades beinhaltet (hier im Weiteren als *Aktualisierbarkeit* bezeichnet).

Das oben näher betrachtete und für Softwarebibliotheken als besonders wichtiges Element identifizierte Kriterium *Stabilität* wird vom ISO-25010-Standard als Teil der *Modifizierbarkeit* beschrieben. Gerade durch den hohen Stellenwert der *Stabilität* in der Propagation der Qualität, wie in der Betrachtung der Softwareauswahlmethoden festgestellt (siehe Kapitel 2.1 „Analyse von Softwarebibliotheken“), ergibt sich die Frage, ob alle Softwarequalitätsuntermerkmale in derselben Weise propagieren oder ob es vielleicht Abweichungen gibt.

Wie Raemaekers et al. [29] beschreiben, wirkt es sich negativ auf die *Wartbarkeit* aus, wenn die *Stabilität* einer genutzten Softwarebibliothek schlecht ist. Wenn allerdings die Propagation der Qualität aufgeteilt auf die Softwarequalitätsuntermerkmale oder Unter-Untermerkmale betrachtet wird, dann propagiert die *Stabilität* offenbar nicht direkt zur *Stabilität* des Nachfolgeknotens. Die *Stabilität* einer genutzten Softwarebibliothek propagiert also nicht zwingend zur *Stabilität* des nutzenden Softwareprodukts. Vielmehr beeinflusst sie die *Aktualisierbarkeit* des nutzenden Softwareprodukts. Diese spezielle Form der Propagation, in der im zu nutzenden Softwareprodukt ein anderes Softwarequalitätsmerkmal beeinflusst wird, wird im Folgenden *Querpropagation* genannt.

Eine *Querpropagation* zeichnet sich dadurch aus, dass ein Qualitätsmerkmal einer Softwarebibliothek speziell durch den Akt der Propagation ein anderes Qualitätsmerkmal beeinflusst. Die beiden Qualitätsmerkmale können dabei auch ohne Betrachtung der Propagation in Beziehung zueinander stehen, sie müssen es aber nicht.

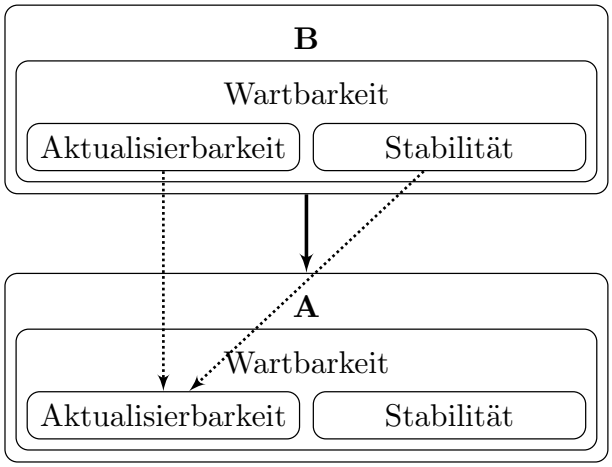


Abbildung 6.1: Schaubild Querpropagation

Die Abbildung 6.1 macht dieses Verhältnis deutlich. Dabei werden die Beeinflussungen zwischen Untermerkmalen im Schritt der Propagation als gestrichelte Linie dargestellt. Danach propagiert die Qualität des Knotens B zum Knoten A. Die *Stabilität* und die *Aktualisierbarkeit* des Knotens B beeinflussen die *Aktualisierbarkeit* des Knotens A. Die *Stabilität* beeinflusst aber nicht die *Stabilität* des Knotens A.

Beim Betrachten der Qualitätspropagation müssen also Beziehungen zwischen verschiedenen Softwarequalitätsmerkmalen beziehungsweise Qualitätsuntermerkmalen oder Unter-Untermerkmalen untersucht werden, die bei der herkömmlichen Softwarequalitätsbewertung nicht vorhanden sind. Insbesondere eine Betrachtung auf der Ebene von Untermerkmalen sollte zukünftig noch genauer sein. Auch die Querpropagation eines Untermerkmals zu einem Untermerkmal eines anderen Hauptmerkmals ist dabei vorstellbar. Diese differenzierten Beziehungen können im Rahmen dieser Arbeit allerdings nicht näher analysiert werden. Im Sinne der Machbarkeit und Transparenz wurde das Gewicht auf eine Betrachtung der “Haupt”-Softewarequalitätsmerkmale gelegt.

6.2.2 Andere Aggregationsmethoden

Den hier präsentierten Methoden für die Einschätzung der Qualität der beiden Softwarequalitätsmerkmale *Wartbarkeit* und *Sicherheit*, müssen noch weitere Aggregationsmethoden gegenüber gestellt werden. Für die propagierte *Wartbarkeit* wurde hier beispielsweise ein einfaches arithmetisches Mittel der Wartbarkeiten aller zu einem Knoten propagierenden Knoten gewählt. Darüber hinaus werden in der Literatur eine Reihe weiterer Ansätze

zur Metrikaggregation diskutiert. Unter anderem werden dafür verschiedene Methoden aus der Ökonomie auf Softwaremetriken übertragen. So beschreiben einige Autoren Ansätze zur Aggregation, beispielsweise mit dem Theil-Index[32], oder dem Gini-Index[36]. Weitere Forschung kann zeigen, inwiefern die verschiedenen Methoden zur Bildung eines Mittelwerts bei der Aggregation der *Wartbarkeit* – beziehungsweise im nächsten Schritt auch weiterer Qualitätsmerkmale – angewandt werden können und welches Mittel jeweils am besten geeignet ist.

6.2.3 Gewichtete Kanten

Diese Arbeit baut auf einem Modell des Abhängigkeitsgraphen auf, der keine gewichteten Kanten enthält. Aus unterschiedlichen Gründen können Softwarebibliotheken in einem Softwareprodukt aber auch von unterschiedlicher Bedeutung für die aggregierte Qualität sein. Das kann zum Beispiel durch die unterschiedliche Menge des genutzten Quellcodes, den unterschiedlichen Grad der Kopplung oder die Größe der Softwarebibliotheken motiviert sein. Eine Gewichtung der Kanten und eine daraus resultierende Anpassung der Aggregationsmethoden könnte also notwendig sein. Je nach Softwarequalitätsmerkmal könnten unterschiedliche Möglichkeiten der Gewichtung von Kanten im Abhängigkeitsgraphen untersucht werden, um eine genauere Aussage über die aggregierte Qualität zu ermöglichen.

Wenn man zum Beispiel die Größe der Softwarebibliotheken zugrunde legt, kann man in diesem Zusammenhang das Demonstrationsbeispiel aus Kapitel 5.4 „Ergebnis“ heranziehen. Es wurde festgestellt, dass der Knoten $CalculatorCli$ eine bessere lokale *Wartbarkeit* ($Q_l^W(CalculatorCli) = 8,6$) als aggregierte *Wartbarkeit* ($Q^W(CalculatorCli) = 7,82$) hat. Diese Reduzierung der *Wartbarkeit* hängt vor allem mit der sehr schlecht bewerteten *Wartbarkeit* des Knotens $NumberTypes$ zusammen. Es ist allerdings fraglich, ob die *Wartbarkeit* von $NumberTypes$ tatsächlich so viel Einfluss auf die *Wartbarkeit* von $CalculatorCli$ hat. Da der $Calculator$ die vermutlich „größere“ Softwarebibliothek ist, hat seine *Wartbarkeit* vermutlich auch einen stärkeren Einfluss auf diejenige von $CalculatorCli$. Mit ungewichteten Kanten kann dieser Einfluss aber nicht abgebildet werden. Außerdem ist fraglich, ob die Softwaregröße als Basis für die Gewichtung der Kanten passend ist.

Eine andere Möglichkeit die Kanten zu gewichten und so eine genauere Aussage über den Grad der Abhängigkeit zu erhalten, wäre die Kopplung von Softwareprodukt und genutz-

ter Softwarebibliothek. So könnte die Berechnung der aggregierten *Wartbarkeit* gewichtet werden. Wenn man davon ausgeht, dass die propagierte *Wartbarkeit* bei größerer Abhängigkeit einen größeren Einfluss hat, dann würde diese Gewichtung also dabei helfen, die zu einem Knoten propagierte *Wartbarkeit* genauer zu bestimmen.

6.2.4 Knotendistanz

Ein Vorteil der schrittweisen Softwarequalitätspropagation, wie sie hier vorgestellt wurde, ist, dass sie sich organisch aus dem Abhängigkeitsverhältnis bildet, ein Knoten also nur seinen direkten Nachfolger auch direkt und alle transitiven Nachfolger nur indirekt beeinflusst.

Es stellt sich in diesem Zusammenhang aber die Frage, welchen Einfluss die Knotendistanz auf die Aggregation hat. Dieser Aspekt kann beispielhaft anhand der *Wartbarkeit* illustriert werden; die Knotendistanz zum Aggregieren der *Sicherheit* ist in dieser Arbeit nicht relevant, da angenommen wird, dass die Distanz keinen Einfluss auf die Relevanz hat.

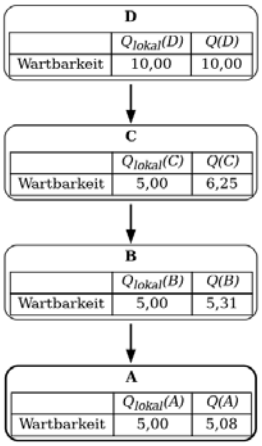


Abbildung 6.2: Propagationsgraph mit propagierter *Wartbarkeit*

In der Abbildung 6.2 ist ein Propagationsgraph zu sehen, in dem der Knoten D transitiv zu A propagiert. Alle Knoten bis auf D haben eine Bewertung der *Wartbarkeit* von $5,00$, D ist allerdings mit $10,00$ angegeben. Diese sehr gute Bewertung von D wirkt sich allerdings nur durch eine Verbesserung der *Wartbarkeit* von A um $0,08$ aus. Während dieses geringe Maß der Auswirkung bei dieser Knotendistanz bei der *Wartbarkeit* erwartbar ist,

so wird bei der *Sicherheit* eine Aggregationsformel genutzt, bei der die Knotendistanz keinen Unterschied macht. Es müssen also für die verschiedenen Softwarequalitätsmerkmale unterschiedliche Wege gefunden werden, mit der Knotendistanz umzugehen.

6.3 Übertragbarkeit

Der Abhängigkeitsgraph, auf dem diese Arbeit aufbaut, dient der Abbildung der Propagation von Qualitätsmerkmalen. Soweit er eine spezifische Software-Architektur abbildet, ist er aber auch auf andere Architektur beschreibende Graphen übertragbar. Zu diesen gehört auch der Graph, der Abhängigkeiten einzelner Module eines Softwareprodukts untereinander abbilden kann. Darüber hinaus könnte ein Graph betrachtet werden, der die Abhängigkeit verschiedener Services in einer Microservice-Architektur untereinander darstellt. Es kommen also Graphen in den Blick, die entweder in ihrer Betrachtungsstruktur kleinmaschiger (Modulabhängigkeitsgraph) oder grobmaschiger (Microservice-Architekturgraph) sind. Beide Graphen werden ähnlich gebildet wie der Abhängigkeitsgraph in dieser Arbeit: gerichtete Graphen, die Zyklen enthalten dürfen und ein Abhängigkeitsverhältnis beschreiben.

Die Übertragung der hier entwickelten Methoden auf Microservice-Architekturgraphen könnte dabei helfen, die Gesamtqualität innerhalb eines Microservice-Clusters besser überblicken zu können. Ähnlich wie bei der Betrachtung der Softwarebibliotheken könnte es hier sinnvoll sein, die Kopplung zwischen den Services als Basis für die Gewichtung der Kanten des Graphs zu nutzen, um eine genauere Qualitätsaggregation zu ermöglichen. In diesem Bereich ist die Forschung von Ma et al. [21] relevant, welche sich mit der Bildung eines „Service Dependency Graph“ also eines Serviceabhängigkeitsgraphen beschäftigt, um Entwicklern beim Auffinden von Fehlerursachen bei Ausfällen zu helfen.

Die Übertragung in einen Modulabhängigkeitsgraphen würde hingegen einen kleinteiligen Ansatz wählen und eher ergänzend in der Aggregation der „lokalen Softwarequalität“ eingesetzt werden. Hier würde das lokale Softwareprodukt, ohne die genutzten Softwarebibliotheken in verschiedene Module aufgeteilt und deren Abhängigkeit untereinander in einem Graphen betrachtet. Wenn nun die Softwarequalität dieser Module bewertet wird, so kann die Propagation der Softwarequalität hier genauso aggregiert werden wie in dieser Arbeit beschrieben. Kazem und Lotfi [20] beschreiben beispielsweise einen „weighted module dependency graph“, also einen gewichteten Modulabhängigkeitsgraphen. Auf dieser Grundlage könnte die Softwarequalitätspropagation genauer betrachtet werden. Auch

die Verwendung eines noch kleinmaschiger abbildenden Graphen, der die Abhängigkeiten der in einem Softwareprodukt genutzten Klassen abbildet, ist denkbar.

Die methodischen Ergebnisse dieser Arbeit lassen sich also nicht nur im Bereich der Softwarebibliotheken anwenden, sondern auch in andere Umgebungen übertragen.

7 Fazit

Bei der Qualitätsbewertung von Software wird die Softwarequalität genutzter Softwarebibliotheken bisher kaum miteinbezogen. Stattdessen wird meist nur die lokale Qualität eines Softwareprodukts betrachtet. Auch in der Literatur werden für die Einbindung von externen Softwarebibliotheken in Softwareprodukte noch keine Qualitätsmodelle diskutiert. Die bekannten Ansätze zur Bewertung von Softwarebibliotheken sind praxisorientiert und dienen zur Auswahl von jeweils geeignet erscheinenden Softwarebibliotheken. Diese Arbeit soll einen Beitrag zu der Debatte leisten, inwiefern die Qualität genutzter Softwarebibliotheken in der Qualitätsbewertung von Software mit einbezogen werden kann, um eine aussagekräftigere Bewertung der Qualität eines Softwareprodukts machen zu können. Für die Beantwortung der Frage nach den Anforderungen für ein Qualitätsmodell für Softwarebibliotheken wurden auf der Basis des ISO-25010-Qualitätsmodells Ansätze für ein Softwarequalitätsmodell entwickelt, das breit und differenziert und damit nutzerfreundlich aufgestellt sein soll. Um eine qualifiziertere Qualitätsaussage zu ermöglichen, reicht die Identifikation der einschlägigen Qualitätsmerkmale allerdings nicht aus. Denn die Qualität von eingebundenen Softwarebibliotheken überträgt sich nicht unmittelbar auf die lokale Software, sondern es ergeben sich komplexe Propagationseffekte, die auch nachvollzogen werden müssen.

Aufgrund ihres Propagationsverhaltens eignen sich die einzelnen Qualitätsmerkmale, die in bestehenden Softwarequalitätsmodellen zur Bewertung von Softwarequalität herangezogen werden, in unterschiedlichem Maße für ein auf Softwarebibliotheken zugeschnittenes Modell. Um das Verhalten zweier Aspekte in der Qualitätspropagation bei der Einbindung von Softwarebibliotheken beispielhaft zu untersuchen, wurden die Qualitätsmerkmale *Wartbarkeit* und *Sicherheit* ausgewählt. Dazu ist auf der Basis eines Abhängigkeitsgraphen ein Propagationsgraph entwickelt worden. Da dieser Zyklen enthält, die das Rechnen auf der Grundlage eines Propagationsgraphen erschweren, ist außerdem parallel dazu ein Propagationsübergangsbaum entwickelt worden, der die Zyklen des Propagationsgraphen aufbricht, sich dabei aber dem Propagationsgraphen soweit wie möglich annähert.

Auf der Grundlage des Propagationsübergangsbaums konnten am Beispiel der beiden Softwarequalitätsmerkmale *Wartbarkeit* und *Sicherheit* Methoden zur Aggregation der lokalen und propagierten Qualität entwickelt werden. Dabei wurde ein besonderes Augenmerk auf die strukturellen Probleme der Softwarequalitätspropagation gelegt. Diese schließen Mehrfachabhängigkeiten und zirkuläre Abhängigkeiten ein. Es zeigte sich, dass die beiden Softwarequalitätsmerkmale sehr unterschiedlich propagieren, und entsprechend wurden jeweils geeignete Aggregationsmethoden entwickelt. Während die *Wartbarkeit* hier vorrangig über ein gewichtetes arithmetisches Mittel aggregiert wurde, ist die *Sicherheit* dagegen über ein Minimum aggregiert worden. Die beiden grundverschiedenen Methoden zeigen, dass einzelne Softwarequalitätsmerkmale diverse Aggregationsmethoden erfordern, um abbilden zu können, wie sich ihre Einbindung auf die Softwarequalität auswirkt.

Das Ergebnis der Auswertung bestätigt, dass – zumindest bei den hier gewählten Beispielen – die Berücksichtigung der Qualität der genutzten Softwarebibliotheken unter dem Aspekt der Propagation neben der lokalen Softwarequalität zu differenzierteren Bewertungen der Qualität von Softwareprodukten führen kann. Mit den hier entwickelten Ansätzen kann genauer beschrieben werden, wie genutzte Softwarebibliotheken die Softwarequalität eines Softwareprodukts beeinflussen. Wie gezeigt wurde, kann der Einfluss der Qualität von genutzten „externen“ Softwarebibliotheken auf ein Softwareprodukt grundsätzlich durch einen Graphen dargestellt werden, der die Propagationsstruktur abbildet. Die Qualität wird dabei aufgeschlüsselt auf verschiedene Merkmale als ein Qualitätsvektor betrachtet. Dabei können die spezifischen Besonderheiten der jeweiligen Softwarequalitätsmerkmale auch durch jeweils passende Methoden zur Aggregation der propagierten Qualität dargestellt werden. Die resultierende aggregierte Qualität ermöglicht eine deutlich umfassendere Qualitätsaussage über ein Softwareprodukt als die ausschließliche Betrachtung der lokalen Softwarequalität. Die Untersuchung weiterer Softwarequalitätsmerkmale neben der *Wartbarkeit* und der *Sicherheit* steht aber noch aus. Dabei könnten dem jeweiligen Propagationsverhalten entsprechend alternativ andere Aggregationsmethoden entwickelt werden.

In der Reflexion der Ergebnisse zeigt sich, dass verschiedene Aspekte der hier besprochenen Softwarequalitätspropagation noch intensiver untersucht werden sollten, um zum Beispiel zu klären, ob die Softwarequalitätsmerkmale in ihrer Auflösung der Qualitätsaspekte genau genug sind oder ob die Softwarequalitätspropagation auf der Ebene der Untermerkmale der Softwarequalitätsmerkmale betrachtet werden müsste. In diesem Zusammenhang ist auch das Phänomen der Querpropagation zu berücksichtigen. Diese beschreibt

die Querbeziehungen zwischen Qualitätsuntermerkmalen oder -unteruntermerkmalen, die nur in der Betrachtung der Qualitätspropagation zu sehen sind. Außerdem wird die Frage zu klären sein, ob eine Gewichtung der Kanten des Propagationsübergangsbaums sinnvoll wäre, um bei der Aggregation der propagierten Qualitäten ein genaueres Abbild der Beziehung zweier Knoten zueinander abbilden zu können.

Fragen, wie sich die Distanz einer transitiven Abhängigkeit auf die aggregierte Qualität auswirkt oder ob eine Betrachtung eines Propagationsgraphen mit gewichteten Kanten besser für die Qualitätspropagation geeignet wäre, konnten im Rahmen dieser Arbeit nur angeschnitten werden. Außerdem müssen für die hier unbeachtet gebliebenen Qualitätsmerkmale noch Aggregationsmethoden entwickelt werden, die das Propagationsverhalten abbilden. Aber das im Zuge dieser Arbeit entwickelte Qualitätspropagationsprogramm kann eine Vorlage dafür liefern, diese Methoden zu entwickeln und zu testen.

Die hier gewonnenen Erkenntnisse und Fragestellungen zur Bedeutung „externer“ Softwarebibliotheken für die Softwarequalität eines Softwareprodukts können damit als Anstoß für die weitere Auseinandersetzung mit der Softwarequalitätspropagation von Softwarebibliotheken betrachtet werden und als ein erster Schritt in Richtung eines vollständigen Softwarequalitätsmodells für Softwarebibliotheken dienen, das die Besonderheiten der Softwarequalität bei Softwarebibliotheken beachtet und die Effekte der Qualitätspropagation reflektiert.

Literaturverzeichnis

[1] Abdalkareem, Rabe ; Nourry, Olivier ; Wehaibi, Sultan ; Mujahid, Suhaib ; Shihab, Emad: Why Do Developers Use Trivial Packages ? An Empirical Case Study on Npm. In: *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. New York, NY, USA : Association for Computing Machinery, 2017 (ESEC / FSE 2017), S. 385–395. – ISBN 978-1-4503-5105-8

[2] Adewumi, Adewole ; Misra, Sanjay ; Omoregbe, Nicholas ; Crawford, Broderick ; Soto, Ricardo: A systematic literature review of open source software quality assessment models. In: *SpringerPlus* 5 (2016), November, Nr. 1, S. 1936. – ISSN 2193-1801

[3] Aversano, Lerina ; Tortorella, Maria: Quality evaluation of floss projects: Application to ERP systems. In: *Information and Software Technology* 55 (2013), Nr. 7, S. 1260–1276

[4] Bakota, T. ; Hegedűs, P. ; Ladányi, G. ; Körtvélyesi, P. ; Ferenc, R. ; Gyimóthy, T.: A cost model based on software maintainability. In: *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, 2012, S. 316–325

[5] Bloch, Josh ; Jäarvi, Jaakko ; Musser, David ; Schupp, Sibylle ; Siek, Jeremy: LCSD: Library-Centric Software Design. In: *Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications*. New York, NY, USA : Association for Computing Machinery, 2006 (OOPSLA '06), S. 618. – ISBN 159593491X

[6] Boehm, Barry W.: *Characteristics of software quality*. North-Holland Pub. Co., 1978

[7] Büchter, Andreas ; Henn, Hans-Wolfgang: *Elementare Stochastik : eine Einführung in die Mathematik der Daten und des Zufalls*. Springer-Verlag, 2006

[8] Crosby, Philip B.: *Quality is free: The art of making quality certain*. Bd. 94. McGraw-hill New York, 1979

[9] Deming, William E.: *Out of the Crisis*. MIT press, 1986

[10] Doberkat, Ernst-Erich ; Dissmann, Stefan: *Einführung in die objektorientierte Programmierung mit Java*. Walter de Gruyter, Januar 2009. – ISBN 978-3-486-59382-2

[11] Eghan, Ellis E.: *Dependency Management 2.0 – A Semantic Web Enabled Approach*, Concordia University, phd, Juli 2019

[12] Fenton, Norman ; Bieman, James: *Software metrics: a rigorous and practical approach*. Third Edition. CRC press, 2014

[13] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design patterns: Entwurfsmuster als Elemente wiederverwendbarer objektorientierter Software*. MITP-Verlags GmbH & Co. KG, 2015

[14] Garvin, David A.: What Does "Product Quality "Really Mean. In: *Sloan management review* 25 (1984), S. 25–43

[15] Gerasimou, S. ; Kechagia, M. ; Kolovos, D. ; Paige, R. ; Gousios, G.: On Software Modernisation due to Library Obsolescence. In: *2018 IEEE / ACM 2nd International Workshop on API Usage and Evolution (WAPI)*, Juni 2018, S. 6–9

[16] Haaland, Kirsten ; Groven, Arne-Kristian ; Regnesentral, Norsk ; Glott, Ruediger ; Tannenberg, Anna ; FreeCode, AS: Free/libre open source quality models-a comparison between two approaches. In: *4th FLOS International Workshop on Free / Libre / Open Source Software*, 2010, S. 1–17

[17] Hmood, A. ; Keivanloo, I. ; Rilling, J.: SE - EQUAM - An Evolvable Quality Metamodel. In: *2012 IEEE 36th Annual Computer Software and Applications Conference Workshops*, 2012, S. 334–339

[18] ISO: *ISO / IEC 25010:2011 Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models*. 2011

[19] Iso, ISO: *Iec 9126-1: Software engineering-product quality-part 1: Quality model*. 2001

[20] Kazem, A. A. P. ; Lotfi, S.: An Evolutionary Approach for Partitioning Weighted Module Dependency Graphs. In: *2007 Innovations in Information Technologies (IIT)*, 2007, S. 252–256

[21] Ma, S. ; Fan, C. ; Chuang, Y. ; Lee, W. ; Lee, S. ; Hsueh, N.: Using Service Dependency Graph to Analyze and Test Microservices. In: *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)* Bd. 02, 2018, S. 81–86

[22] Martin, Robert C.: *Clean Architecture - Das Praxis - Handbuch für gutes Softwaredesign.* mitp, 2018. – ISBN 978-3-95845-724-9

[23] McCall, Jim A. ; Richards, Paul K. ; Walters, Gene F.: Factors in Software Quality . Volume I . Concepts and Definitions of Software Quality / GENERAL ELECTRIC CO SUNNYVALE CA. November 1977. – Forschungsbericht

[24] Mileva, Yana M. ; Dallmeier, Valentin ; Burger, Martin ; Zeller, Andreas: Mining Trends of Library Usage. In: *Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution (IWPSE) and Software Evolution (Evol) Workshops.* New York, NY, USA : Association for Computing Machinery, 2009 (IWPSE - Evol '09), S. 57–62. – ISBN 978-1-60558-678-6

[25] Mohagheghi, P. ; Conradi, R. ; Killi, O. M. ; Schwarz, H.: An empirical study of software reuse vs. defect-density and stability. In: *Proceedings. 26th International Conference on Software Engineering*, 2004, S. 282–291

[26] Mora, Fernando L. de la ; Nadi, Sarah: An Empirical Study of Metric - Based Comparisons of Software Libraries. In: *Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering.* New York, NY, USA : Association for Computing Machinery, 2018 (PROMISE '18), S. 22–31. – event-place: Oulu, Finland. – ISBN 978-1-4503-6593-2

[27] Mora, Fernando L. de la ; Nadi, Sarah: Which Library Should I Use ? A Metric - Based Comparison of Software Libraries. In: *Proceedings of the 40th International Conference on Software Engineering : New Ideas and Emerging Results.* New York, NY, USA : Association for Computing Machinery, 2018 (ICSE - NIER '18), S. 37–40. – event-place: Gothenburg, Sweden. – ISBN 978-1-4503-5662-6

[28] P. Miguel, José ; Mauricio, David ; Rodríguez, Glen: A Review of Software Quality Models for the Evaluation of Software Products. In: *International Journal of Software Engineering & Applications* 5 (2014), November, Nr. 6, S. 31–53. – ISSN 0975-9018

[29] Raemaekers, S. ; Deursen, A. v. ; Visser, J.: Measuring software library stability through historical version analysis. In: *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, September 2012, S. 378–387

[30] Rodríguez, Moisés ; Oviedo, Jesús R. ; Piattini, Mario: Evaluation of software product functional suitability: a case study. In: *Software Quality Professional* 18 (2016), Nr. 3, S. 18

[31] Samoladas, Ioannis ; Gousios, Georgios ; Spinellis, Diomidis ; Stamelos, Ioannis: The SQO - OSS Quality Model : Measurement Based Open Source Software Evaluation. In: Russo, Barbara (Hrsg.) ; Damiani, Ernesto (Hrsg.) ; Hissam, Scott (Hrsg.) ; Lundell, Björn (Hrsg.) ; Succi, Giancarlo (Hrsg.): *Open Source Development , Communities and Quality*. Boston, MA : Springer US, 2008, S. 237–248. – ISBN 978-0-387-09684-1

[32] Serebrenik, Alexander ; Brand, Mark van den: Theil index for aggregation of software metrics values. In: *2010 IEEE International Conference on Software Maintenance*, September 2010, S. 1–9. – ISSN: 1063-6773, 1063-6773

[33] Sneed, H.M. ; Winter, M.: *Testen objektorientierter Software : das Praxishandbuch für den Test objektorientierter Client - Server - Systeme*. Hanser, 2002. – ISBN 978-3-446-21820-8

[34] Soto, M. ; Ciolkowski, M.: The QualOSS open source assessment model measuring the performance of open source communities. In: *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, 2009, S. 498–501

[35] Thung, F.: API recommendation system for software development. In: *2016 31st IEEE / ACM International Conference on Automated Software Engineering (ASE)*, September 2016, S. 896–899

[36] Vasa, R. ; Lumpe, M. ; Branch, P. ; Nierstrasz, O.: Comparative analysis of evolving software systems using the Gini coefficient. In: *2009 IEEE International Conference on Software Maintenance*, 2009, S. 179–188

[37] Wagner, Stefan: *Software product quality control*. Springer, 2013

[38] Werners, Brigitte: *Grundlagen des Operations Research : Mit Aufgaben und Lösungen*. Springer-Verlag, 2013

[39] Wessels, Keno: Analyse von Metrik-Aggregation in den Softwareanalysewerkzeugen SonarCloud, Scrutinizer und Better Code Hub. (2020), Mai. – Projektarbeit

[40] Wessels, Keno: Analyse von PHP-Bibliotheken durch LibraryCheck. (2020), Februar. – Projektarbeit

[41] Wittern, Erik ; Suter, Philippe ; Rajagopalan, Shriram: A look at the dynamics of the JavaScript package ecosystem. In: *Proceedings of the 13th International Conference on Mining Software Repositories*. Austin, Texas : Association for Computing Machinery, Mai 2016 (MSR '16), S. 351–361. – ISBN 978-1-4503-4186-8

A Anhang

A.1 Abhängigkeitsgraph Laravel

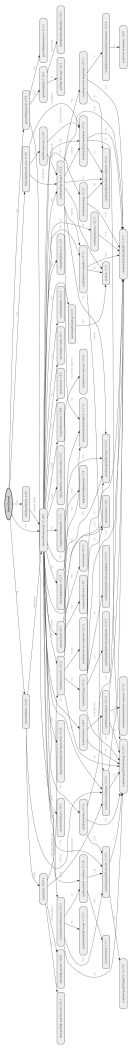


Abbildung A.1: Abhängigkeitsgraph Laravel Skeleton (erstellt mit Graph-Composer)

A.2 Qualitätspropagationsprogramm - Beispielkonfiguration

Listing A.1: Auszug aus dem Qualitätspropagationsprogramm; Konfiguration eines Testszenarios (Siehe auch Abbildung 5.8b)

```php
<?php

declare(strict_types=1);

namespace App\Simulations;

use App\Aggregators\AbstractAggregator;
use App\Aggregators\Wartbarkeit;
use App\Dtos\PrintSettings;
use App\GraphFactory;
use App\QualityAttributes;
use Graphp\Graph\Graph;

class EinfacheAbhaengigkeit implements Simulatable
{
    public function getTitle(): string
    {
        return 'Einfache Abhängigkeit';
    }

    public function getGraph(): Graph
    {
        return GraphFactory::fromArray(
            [
                [
                    'id' => 'A',
                    'q_local_Wartbarkeit' => 7.6,
                ],
                [
                    'id' => 'B',
                    'q_local_Wartbarkeit' => 3.1,
                ],
            ],
            [
                ['from' => 'B', 'to' => 'A']
            ],
        );
    }

    /** @return AbstractAggregator[] */
    public function getAggregators(): array
    {
        return [
            new Wartbarkeit(),
        ];
    }

    public function getTargetVertexId(): string
    {
        return 'A';
    }

    public function getPrintSettings(): PrintSettings
    {
        return new PrintSettings([QualityAttributes::WARTBARKEIT], $this->getTargetVertexId());
    }
}
```

A.3 Propagationsübergangsbaum

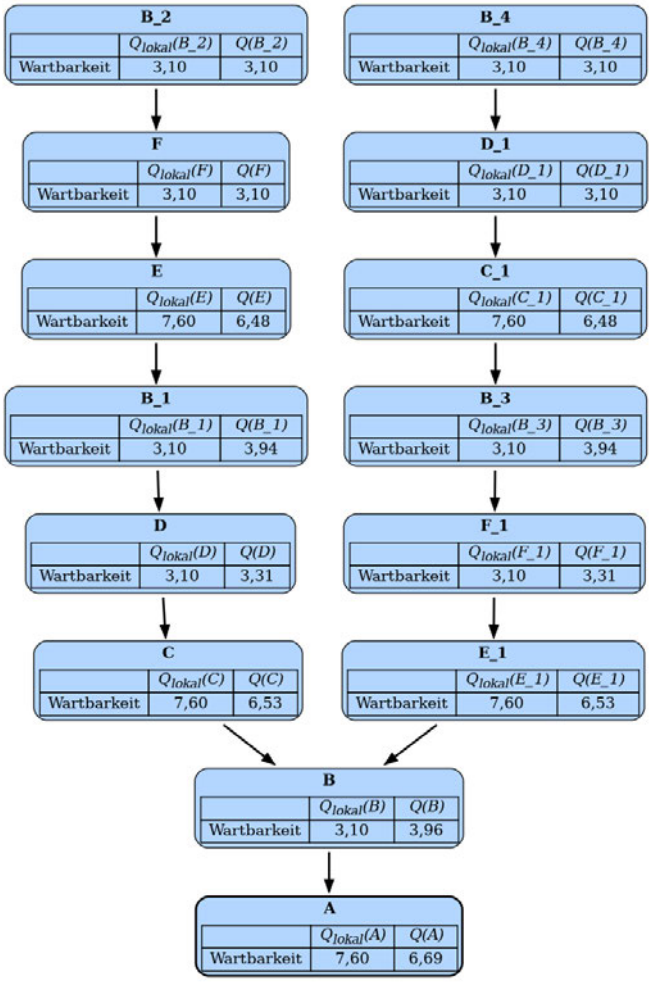


Abbildung A.2: Propagationsübergangsbaum (mehrfach zirkuläre Propagation)

Glossar

Abhängigkeitsgraph Ein Graph der direkt und transitiv genutzten Softwarebibliotheken eines Softwareprodukts abbildet.

Aggregation Das Zusammenführen von Informationen zu einem einzigen repräsentativen Wert, der anschließend für verschiedene Zwecke verwendet werden kann.

lokale Softwarequalität Die Softwarequalität eines Softwareprodukts, welche die propagierte Softwarequalität nicht miteinbezieht. Sie wird hier in Form eines Qualitätsvektors betrachtet und mit dem Formelzeichen $Q_{lokal}(v)$ oder $Q_l(v)$ beschrieben.

Mehrfachabhängigkeit Die mehrfache Abhängigkeit eines Softwareprodukts von einer Softwarebibliothek. Diese mehrfache Abhängigkeit ist möglich, wenn eine Abhängigkeit sowohl direkt als auch transitiv besteht.

Propagationsgraph Ein Graph der die Qualitätspropagation der genutzten Softwarebibliotheken eines Softwareprodukts abbildet.

Propagationsübergangsbaum Ein modifizierter Propagationsgraph, der keine Zyklen enthält.

Qualitätspropagation Die Weitergabe der Softwarequalität von Softwarebibliotheken an die Softwareprodukte die diese nutzen.

Softwarequalitätsvektor Die Softwarequalität in der Form eines Vektors der sich aus verschiedenen Softwarequalitätsmerkmalen zusammensetzt.

zirkuläre Abhängigkeit Abhängigkeiten die in einem Abhängigkeitsgraphen Zyklen bilden. Das ist beispielsweise bei Softwarebibliotheken der Fall die voneinander abhängig sind.

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Masterarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Entwicklung eines Qualitätsmodells zur Bewertung von Softwarebibliotheken

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original