Reviews – ein Instrument zur Qualitätsverbesserung von UML-Diagrammen

Doris Schmedding, Anna Vasileva, Technische Universität Dortmund

doris.schmedding | anna.vasileva@tu-dortmund.de

Zusammenfassung

In modellgetriebenen Software-Entwicklungsprojekten in der Ausbildung besteht das Problem, dass die Qualität der Modelle oft schlecht ist. In diesem Beitrag wird ein Konzept vorgestellt, wie durch gegenseitige Reviews der Entwicklerteams die Qualität der Modelle verbessert wird. Die Reviews basieren auf einem Kriterienkatalog, der aus einem Qualitätsmodell entwickelt wurde. Das Software-Praktikum bietet einen geeigneten Rahmen, dieses interaktive Konzept einzusetzen. Wir stellen unsere Vorgehensweise und unsere Erfahrungen vor.

Motivation

Die intensive Auseinandersetzung mit der Code-Qualität in den studentischen Projekten im Software-Praktikum (Vasileva, Schmedding, 2016; Schmedding, Vasileva, Remmers, 2015) führte uns zu der Erkenntnis, dass manche Code-Mängel bereits im Modell zu finden sind. Im Software-Praktikum (kurz SoPra), das den Kontext unseres Beitrags bildet, entwickeln die Studierenden Software nach dem Konzept der modellgetriebene Software-Entwicklung. Modellgetriebene Software-Entwicklung bedeutet, dass automatisiert aus formalen Modellen vollständig oder wenigstens teilweise lauffähige Software erzeugt wird. (Bettin, Helsen, Kunz, 2007)

Daneben bieten die Modelle in der modellgetriebenen Software-Entwicklung eine gute Arbeits- und Kommunikationsgrundlage für das Entwicklerteam in den frühen Phasen der Software-Entwicklung. Außerdem können sie als Dokumentation der Software dienen, so dass spätere Überprüfungen der Anforderungen, Analysen des Codes, Verbesserungen und Erweiterungen oder Wiederverwendungen der Software in anderen Bereichen leichter möglich sind (Fieber, Huhn, Rumpe, 2008). Deshalb ist es wichtig, die Modelle bereits frühzeitig einer Qualitätsprüfung zu unterziehen. Diese kann teilweise automatisiert erfolgen, wie z.B. bei Arendt (Arendt, 2014), da die Modelle in digitaler Form vorliegen. Im Mittelpunkt dieses Beitrags steht allerdings die Qualitätskontrolle mittels gegenseitiger Reviews.

Wenn es möglich wäre, größere Mängel frühzeitig in den Modellen zu erkennen, so kann es sicherlich auch möglich sein, diese leichter und schneller zu beheben, als dies während der Code-Erstellung der Fall ist (Boehm, Basili, 2001). Dies könnte dann von vornherein zu einer besseren Code-Qualität und zu einer besseren Wartbarkeit des Programm-Codes führen, was insgesamt eine große Zeit- und Kostenersparnis bedeuten kann. Das setzt natürlich voraus, dass sich das Entwicklungsteam eng an die fertigen Modelle hält und nicht unzählige Veränderungen und neue Elemente in der Implementierungsphase hinzufügt, ohne die Modelle dementsprechend anzupassen und erneut zu prüfen.

Den Kontext unseres Erfahrungsberichts liefert das Software-Praktikum der Fakultät für Informatik der TU Dortmund. Das Ziel des SoPras ist es, im Rahmen von Software-Entwicklungsprojekten die Methoden und Verfahren, die in der vorangehenden Vorlesung Software-Technik kennen gelernt und geübt wurden, im Zusammenhang eines Projekts anzuwenden. Außerdem dient das SoPra dazu, die in der Einführungsveranstaltung „Datenstrukturen, Algorithmen und Programmierung“ erworbenen Programmierkenntnisse zu vertiefen.

Im SoPra führen acht Bachelorstudierende gemeinsam zwei Projekte durch. Etwa sechs bis zehn Gruppen bearbeiten gleichzeitig die gleichen Aufgaben. In einem Blockpraktikum in der vorlesungsfreien Zeit dauert ein Projekt drei Wochen. Es werden nacheinander in sechs Wochen zwei Projekte nach einem einfachen Wasserfallmodell durchgeführt. Dieses Software-Entwicklungsmodell hat sich bei der Arbeit mit den Studierenden, die zum ersten Mal ein Projekt durchführen, gut bewährt.

Wir verwenden die Programmiersprache Java aus der Einführungsveranstaltung. Zur Modellierung setzen wir die Sprache UML ein, die in der Software-Technik-Vorlesung eingeführt wird. Die Modellierung erfolgt mit dem Tool Astah (Astah,

2016), das leichtgewichtig, sehr benutzerfreundlich und intuitiv bedienbar ist. Deshalb kommen die Studierenden nach unserer Erfahrung damit gut zu recht.

Im Rahmen der modellgetriebenen Entwicklung werden aus dem mit Astah erstellten Strukturmodell Java-Code-Rahmen generiert. Als Entwicklungsumgebung verwenden wir Eclipse mit dem SVN-Plugin Subclipse. Die Dokumentation mit JavaDoc und JUnit-Tests sind weitere wichtige Bestandteile unseres Entwicklungsprozesses.

Der gesamte Entwicklungsprozess ist durchzogen von Maßnahmen zur Qualitätssicherung. Während des Software-Praktikums gibt es interne und externe Reviews, Abgaben, Korrekturen und Abnahmen. Nach jedem Entwicklungsschritt werden die erzeugten Dokumente, bei denen es sich oft um UML-Diagramme, aber auch um Textdokumente handelt, im internen Bereich des SoPra-Wikis (SoPra-Wiki, 2016) des Projektteams hochgeladen. Die Dokumente werden innerhalb der Gruppe diskutiert und vom Betreuer korrigiert. Sowohl das Anforderungsmodell als auch das Analysemodell werden von jeder Gruppe im öffentlichen Bereich des Gruppen-Wikis hochgeladen und sind damit für alle SoPra-Teilnehmer lesbar. Die Diagramme werden anschließend im Plenum vor anderen Gruppen vorgestellt und Unterschiede und Mängel diskutiert. Am Ende eines Projekts erfolgt die gegenseitige Abnahme und Bewertung der Projekte durch die Gruppen. Die Code-Qualität wird mit Hilfe von PMD (PMD, 2016), einem Tool zur statischen Code-Analyse, sowohl von den Teammitgliedern selbst als auch von den Lehrenden gemessen.

Auch wenn man nach drei Wochen nicht erwarten kann, ein perfekt funktionierendes Programm zu erhalten, besitzt die funktionale Korrektheit den höchsten Stellenwert und wird als vorrangiges Ziel in der Software-Entwicklung von keinem der Beteiligten in Frage gestellt. Für die Studierenden ist in ihrem eigenen Projekt und bei der Bewertung der Ergebnisse der anderen Gruppen auch das Aussehen eines Programms sehr wichtig, insbesondere bei Spielen.

In den letzten Jahren ist es uns gelungen, auch die Code-Qualität erfolgreich als Ziel in der Veranstaltung zu verankern (Vasileva, Schmedding, 2016; Schmedding, Vasileva, Remmers, 2015). Da manche Mängel, die bei der Bestimmung der Code-Qualität gemessen werden, wie z.B. schlecht gewählte Bezeichner, bereits im Modell erkennbar sind, richtet sich unser Augenmerk in unserer aktuellen Arbeit auf die Modellqualität. Wir vermuten, dass sich auch gravierendere Mängel, wie zu komplexe Methoden und Klassen, bereits im Modell erkennen lassen. Voraussetzung dafür ist natürlich, dass das Modell aussagekräftig genug ist, d.h. dass es genügend durchdacht ist und genügend Details enthält.

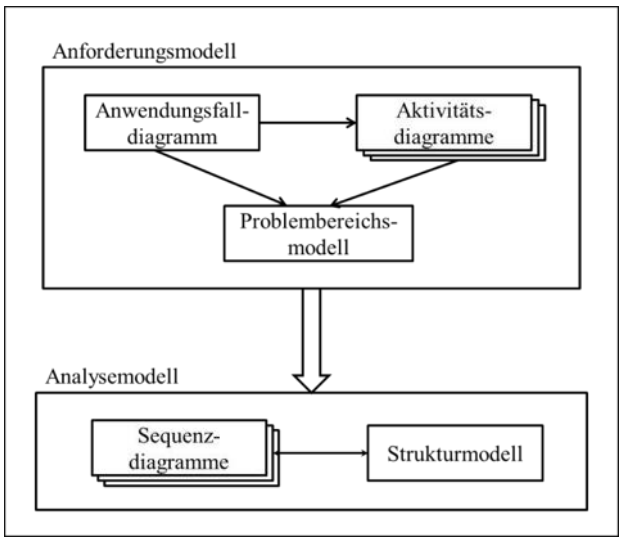


Abb. 1. UML-Modellierung im SoPra

Um von der Modellqualität auf die Code-Qualität schließen zu können, muss das Modell ein abstraktes Abbild dessen sein, was das Team entwickeln will.

Im folgenden Kapitel wird vorgestellt, wie UML im SoPra zur Modellbildung eingesetzt wird. Danach folgen Überlegungen zur Qualität von UML-Diagrammen. Anschließend wird das Konzept von Reviews und ihre Bedeutung im Software-Entwicklungsprozess diskutiert. Es folgt die Rolle, die die Reviews in der Lehrveranstaltung Software-Praktikum spielen. Danach werden die Erfahrungen mit dem gewählten Vorgehen im Software-Praktikum diskutiert. Der Artikel schießt mit einer Zusammenfassung und einem Ausblick.

Modellierung mit UML im SoPra

Die Modellierung mit UML findet im SoPra sowohl in der Phase Anforderungsdefinition als auch in der Analysephase statt. Abb. 1 zeigt die Entwicklungshasen und die Zusammenhänge zwischen den Diagrammen des Modells, die im SoPra verwendet werden.

In der ersten Phase des Software-Entwicklungsprozesses werden die Anforderungen erhoben und das Anforderungsmodell erstellt, das aus Anwendungsfalldiagramm, Aktivitätsdiagrammen und Problembereichsmodell besteht. Ein Anwendungsfalldiagramm stellt die funktionalen Anforderungen in Form von Anwendungsfällen dar, die die Nutzer des Systems in unterschiedlichen Rollen ausführen können. Anschließend wird jeder Anwendungsfall näher beschrieben, indem der genaue Ablauf durch ein Aktivitätsdiagramm dargestellt wird. Zu jedem Aktivitätsdiagramm gehört ein erläuternder Text, der u.a. die Vor- und Nachbedingungen enthält. Ein Beispiel für ein typisches Aktivitätsdiagramm aus dem SoPra befindet sich in Anhang A.

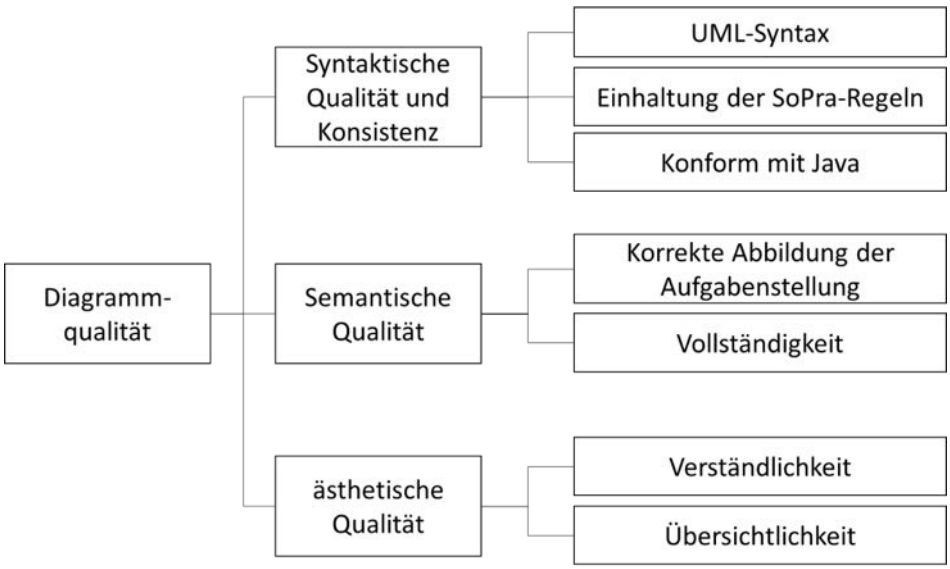


Abb. 2: Qualitätsmodell des Software-Praktikums

Die Aktivitätsdiagramme sind wichtige Bestandteile des Entwicklungsprozesses. Beim Erstellen dieser Diagramme stellen die Studierenden zum ersten Mal fest, dass einige Anwendungsfälle sehr komplex sind. Andere Anwendungsfälle funktionieren nicht so, wie sie zunächst gedacht haben.

Der Entwicklungsprozess startet also mit einer ausführlichen Erhebung der funktionalen Anforderungen, auf die im weiteren Verlauf des Prozesses immer wieder zurückgegriffen wird. Daneben wird ein Datenmodell, das so genannte Problembereichsmodell, erstellt, das die Klassen des Problembereichs mit ihren Attributen und den Beziehungen untereinander darstellt.

In der Analysephase wird das Problembereichsmodell zum Strukturmodell ausgebaut, indem Steuerungsklassen ergänzt und Methoden hinzugefügt werden. Wir streben aus Gründen der Übersichtlichkeit und Einfachheit eine Dreischichtenarchitektur an, die sich an dem Model-View-Controller-Muster der Software-Entwicklung orientiert. Um zu überprüfen, ob im Strukturmodell alle Methoden vollständig erfasst sind, werden Sequenzdiagramme eingesetzt, die in der Regel die Anwendungsfälle aus dem Anwendungsfalldiagramm repräsentieren. Bei der Definition der Methoden im Strukturmodell werden die Signaturen der Methoden vollständig angegeben.

Das Strukturmodell und die Sequenzdiagramme sind inhaltlich gekoppelt. Bei Astah liegt diesen beiden Diagrammtypen ein gemeinsames Datenmodell zugrunde, so dass im Sequenzdiagramm die Klassen und Methoden aus dem Strukturmodell verwendet werden können.

Am Ende der Anforderungs- und der Analysephase finden Präsentationen statt. Bei der ersten Präsentation werden das Anwendungsfalldiagramm, drei Aktivitätsdiagramme sowie das Problembereichsmodell zwei anderen Gruppen im Plenum vorgestellt. In der zweiten Phase werden drei Sequenzdiagramme und das Strukturmodell präsentiert.

Zur Qualität von UML-Diagrammen

In der Literatur finden sich verschiedene Ansätze zur Definition und Bewertung der Modellqualität, von denen hier zwei genauer betrachtet werden. Der erste Ansatz stammt von Unhelkar (Unhelkar, 2005), der die Modelle unter den Gesichtspunkten ihrer Anwendung im Entwicklungsprozess betrachtet, d.h., für welchen Zweck sie erstellt werden und wie grob oder detailliert die Modelle dementsprechend sein sollten. Dazu unterteilt er die Qualität in drei Aspekte: Syntax, Semantik und Ästhetik. Syntax beschäftigt sich mit der Korrektheit der verwendeten Diagrammelemente und Semantik mit der korrekten und vollständigen Abbildung der Aufgabenstellung. Unter Ästhetik subsummiert Unhelkar Begriffe wie Übersichtlichkeit und Verständlichkeit, aber auch Granularität. Für jeden Diagrammtyp und für jeden Qualitätsaspekt benennt Unhelkar begründete Kriterien und Regeln für die Bewertung der Qualität, die er in Form von Checklisten zusammenstellt.

Der zweite Ansatz stammt von Arendt (Arendt, 2014), der stattdessen bei der Entwicklung eines Qualitätsmodells für UML-Modelle den Versuch unternimmt, die Metriken, die für die Untersuchung der Codequalität entwickelt wurden, auf Modelle zu übertragen. Im Rahmen seiner Arbeit beschränkt er sich auf Klassendiagramme. Dabei betrachtet er die Qualität von Modellen unter den Gesichtspunkten

der Qualitätsziele von Mohegheghi (Mohagheghi, Dehlen, Neple, 2009), den sogenannten 6C-Zielen. Diese umfassen Korrektheit (correctness), Vollständigkeit (completeness), Konsistenz (consistency), Verständlichkeit (comprehensibility), Beschränkung auf das Wesentliche (confinement) und Änderbarkeit (Changeability) (Arendt, 2014). Hinzu zieht Arendt die Qualitätsziele von Fieber (Fieber, Huhn, Rumpe, 2008), die er im Weiteren als Qualitätseigenschaften bezeichnet (zur besseren Unterscheidung von den 6C-Zielen) (Thorsten Arendt, 2014). Einige davon sind Präsentation, Einfachheit, Konformität, Kohäsion, Redundanz, semantische Angemessenheit, Korrektheit, Präzision, Vollständigkeit bzgl. Anforderungs- und Lösungsphase, Rückverfolgbarkeit und Veränderbarkeit.

Wir orientieren uns bei der Zusammenstellung der Qualitätskriterien an der Arbeit von Unhelkar, da bei uns alle Diagramme, auch diejenigen, die ganz früh im Entwicklungsprozess entstehen, begutachtet werden. Es geht im SoPra nicht nur um die Generierung von möglichst gutem Java-Code aus dem Klassendiagramm, sondern auch um die Qualität des gesamten Modells mit allen Diagrammen, die im SoPra erstellt werden, da diese Diagramme die Arbeits- und Kommunikationsgrundlage des Entwicklerteams in allen Phasen des Entwicklungsprozesses darstellen. Die Diagramme dokumentieren den Grat, mit dem das Team die Aufgabenstellung durchdrungen hat.

Diese Überlegungen führen zu dem in Abb. 2 dargestellten Qualitätsmodell für UML-Diagramme. Die einzelnen Qualitätseigenschaften werden im Folgenden näher erläutert.

Syntax und Konsistenz

Das im SoPra verwendete UML-Tool Astah arbeitet syntaxbasiert, aber nicht syntaxgesteuert. Da während des Modellierens zwangsweise immer wieder Diagrammzustände entstehen, die syntaktisch nicht korrekt sind, lässt Astah dem Modellierer viele Freiheitsgrade. Deshalb besteht für wenig erfahrene Modellierer die Gefahr, dass Diagramme entstehen, die nicht dem UML-Meta-Modell entsprechen und syntaktische Fehler beinhalten. Ein Beispiel für einen derartigen Mangel ist ein fehlender Endknoten in einem Aktivitätsdiagramm. Ein Diagramm mit diesem Fehler würde in die Klasse „Astah-konform“ fallen, da es ohne Fehlermeldung von Astah erstellbar ist, aber es würde wegen der Syntaxfehlers nicht in der Klasse „UML-konform“ liegen.

Insgesamt stellt die Komplexität der UML sowohl in der Ausbildung als auch in der Industrie ein großes Problem dar. Eine Studie (Petre, 2013) hat gezeigt, dass 35 von 50 befragten Unternehmen UML nicht verwenden. Ein Grund dafür ist, dass die Notation von UML zu komplex geworden ist. Einerseits macht die Komplexität die Einarbeitung in dem Bereich der Modellierung schwierig und zeitaufwendig. Andererseits können die Tools zur Modellierung von UML-Diagrammen den ganzen Sprachumfang nicht umsetzen, was zum Entstehung eigener teilweise komplett unterschiedlichen Versionen führt. Aus diesem Grund brauchen die Entwickler klare Richtlinien, welche Diagrammtypen und -elemente sie bei der Modellierung verwenden sollen.

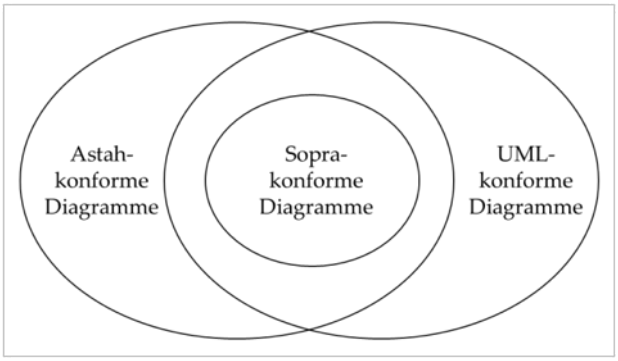


Abb. 3: Regelkonforme Diagramme im SoPra

Im SoPra haben sich zur Qualitätssicherung bestimmte „Best-Practice-Regeln“ bewährt, die insbesondere dafür sorgen, dass der Informationsgehalt der Diagramme genügend groß ist. Die Diagramme sollen viele, aber nicht zu viele Details enthalten, damit sie informativ, aber nicht unübersichtlich werden. Diese Gratwanderung zwischen Unübersichtlichkeit durch zu viele Details und Oberflächlichkeit, also die Wahl des richtigen Abstraktionsgrads, stellt für Modellierungsanfänger eine sehr große Herausforderung dar. Wir versuchen durch Regeln, z.B. wie viele Anwendungsfälle ein Anwendungsfalldiagramm mindestens und maximal beinhalten sollte, hier eine Hilfestellung zu geben.

Eine weitere Regel, die der inhaltliche Aussage von Aktivitätsdiagrammen zugutekommt, ist die Verwendung von Partitionen in Aktivitätsdiagrammen. Normalerweise verfügen Aktivitätsdiagramme im SoPra über zwei Partitionen – “Benutzer” und “System”. Auf diese Weise wird nicht nur dargestellt, wie der Benutzer mit dem System interagiert, sondern auch die Reaktion des Systems auf die Aktivitäten des Benutzers. Ein Beispiel findet sich in Anhang A.

Abb. 3 stellt dar, in welcher Beziehung die UML-konformen Diagramme, die Astah-konformen Diagramme und die mit den SoPra-Regeln konformen Diagramme zueinanderstehen. Alle SoPra-konforme Diagramme liegen Schnittmenge zwischen den UML- und Astah-konformen Diagrammen. Die im Rahmen des SoPras erstellten Diagrammen müssen eine echte Teilmenge dieser Menge sein, da sie die SoPra-Regeln einhalten müssen.

Eine besondere Rolle für das Klassendiagramm spielt unsere Implementierungssprache Java. Astah und auch UML lassen beispielsweise Mehrfacherbung zu, was Java ausschließt. Auch die Spezialisierung von Interfaces ist im Klassendiagramm von Astah möglich, in Java nicht.

Die nächste Gruppe von Mängeln in UML-Diagrammen bilden Inkonsistenzen. In einem Modell können Inkonsistenzen zwischen Diagrammen des gleichen Typs oder Inkonsistenzen zwischen Diagrammen unterschiedlichen Typs auftreten.

Weiterhin besteht bei Astah das Problem der Inkonsistenz zwischen der graphischen Repräsentation und den gespeicherten Daten. Teile der Grafik können gelöscht werden, ohne dass sie aus dem Modell entfernt werden. Auf diese Weise lassen sich übersichtliche Abbildungen gestalten. Die Code-Generierung basiert allerdings auf dem gespeicherten Modell. Bei unachtsame Modellierern tauchen bei der Code-Generierung plötzlich Java-Klassen oder Methoden auf, die sie vermeintlich längst gelöscht hatten.

Eine genaue Herleitung und Details sowie Beispiele wurden von Stefan Todorinski (Todorinski, 2016) in seiner Bachelorarbeit zusammengetragen. Allen bisher in diesem Kapitel dargestellten Mängeln ist gemeinsam, dass sie sich durch eine statische Analyse erkennen lassen.

Semantik

Die nächste Kategorie von Mängeln stellen semantische Mängel dar. Hier geht es vor allem um die Frage, ob die in der Aufgabenstellung festgelegten Anforderungen vollständig und korrekt im Modell umgesetzt werden. Striewe und Goedicke zeigen in (Striewe, Goedicke, 2011), dass sich bei der automatischen Korrektur von UML-Übungsaufgaben auch semantische Mängel finden lassen. Allerdings funktioniert das nur gut für wenig umfangreiche Aufgabenstellungen wie z.B. Klausuraufgaben. Die Lösung von Striewe und Goedicke basiert zwar nicht auf dem Vergleich mit einer Musterlösung, aber für die Korrektur müssen von den Lehrenden für die Überprüfung der Lösung konkrete Abfragen in einer Abfragesprache definiert werden, die die abgegebene Lösung erfüllen muss.

Mängel im semantischen Bereich sind nach unserer Erfahrung in studentischen Projekten in erster Linie darin begründet, dass die Modellierung nicht vollständig genug ist. Das erstellte Diagramm modelliert den Sachverhalt nur oberflächlich, es geht nicht genug in die Tiefe und enthält demnach nicht genug Details. Daneben kommt es immer wieder vor, dass die Aufgabenstellung missverstanden wird.

Ästhetik

Die Frage nach dem richtigen Abstraktionsgrad und der Menge von Details, die ein Modell enthalten sollte, ist besonders wichtig für die Bewertung der ästhetischen Qualität eines Diagramms. Sie wird geprägt durch die Qualitätskriterien „Übersichtlichkeit“ und „Verständlichkeit“. Im Gegensatz zu Unhelkar betrachten wir Granularität nicht als eigenständige Qualitätseigenschaft, vielmehr bestimmt die richtige Wahl der Granularität zusammen mit anderen Merkmalen die Eigenschaften „Übersichtlichkeit“ und „Verständlichkeit“. Wenn ein Diagramm zu viele Details enthält, kann es unübersichtlich und dadurch unverständlich werden. Zu wenige Details oder eine schlechte Bezeichnerwahl lassen es unverständlich werden. Ein zu komplexes Diagramm ist sowohl unübersichtlich als auch unverständlich.

Die Aspekte der syntaktischen Qualität und die Konsistenz der Diagramme lassen sich relativ einfach auch automatisch überprüfen, während sich ihre semantische und ästhetische Qualität nur durch Reviews feststellen lässt.

Reviews in der Software-Entwicklung und in der Lehre

Reviews sind in der Software-Entwicklung ein anerkanntes Instrument zur Qualitätskontrolle. Nach dem IEEE-Standard 1028 von 2008 (IEEE, 2008) versteht man unter dem Begriff *Review* eine formell organisierte Zusammenkunft von Personen zur inhaltlichen und formellen Überprüfung eines Produktteils (Dokument, Programmcode, usw.) nach vorgegebenen Prüfkriterien und -listen.

Reviews werden meist in Form von Code-Reviews zur Steigerung der Code-Qualität eingesetzt, aber auch zur Steigerung der Qualität der Dokumentation. Man unterscheidet zwischen informellen Reviews, wie das Gegenlesen von Dokumenten durch Kollegen, oder formellen Reviews, denen ein definierter Prozess mit festen Rollen und einer konkreten Checkliste zugrunde liegt. (Wiegers, 2001)

Auch die Stellung der beteiligten Personen dient als Kriterium für die Differenzierung von Review-Typen. Man unterscheidet Peer-Reviews, bei denen nur der Autor und/oder Kollegen, die auf gleicher Ebene arbeiten, beteiligt sind, und Management-Reviews, bei dem Personen aus dem Management als Gutachter beteiligt sind und bei denen es in erster Linie darum geht, sich ein Bild vom Stand des Projekts zu verschaffen. In der Lehre werden Reviews überwiegend im Form von Peer-Reviews, z. B. bei der Erstellung von Seminararbeiten, eingesetzt.

Reviews haben im Software-Entwicklungsprozess zwei Funktionen. Im Vordergrund steht natürlich das Entdecken von Fehlern und Mängel, entweder durch die Person selbst, die die Fehler produziert hat, oder durch die Gutachter. Je frühzeitiger die Mängel entdeckt werden, desto geringere Kosten verursacht ihre Beseitigung (Boehm, Basili, 2001). Das sollte gerade bei der Begutachtung von Dokumenten der frühen Entwicklungsphase wie den UML-Diagrammen zur Anforderungsdefinition eine große Rolle spielen.

Die zweite wichtige Funktion besteht darin, dass die Reviewer durch die intensive Auseinandersetzung mit dem zu begutachtenden Dokument oder Programm-Code viel lernen. Sie lernen am Beispiel, indem sie Lösungsansätze sehen, deren Umsetzung sie sich vielleicht nicht zugetraut hätten oder die ihnen selbst schlichtweg nicht eingefallen sind.

Die Gefahr bei sehr informellen Reviews, wie das Gegenlesen durch den Autor selbst oder einen Kollegen/eine Kollegin, besteht darin, dass der Fokus schnell verloren geht und nicht kritisch genug begutachtet wird. Deshalb sollte der Review-Prozess auch in einer Lehrveranstaltung zumindest einen formalen Rahmen haben.

Die Gefahr bei stark vom Management dominierten Reviews besteht darin, dass das Ziel sich leicht verschieben kann. Das gemeinsame Anliegen, das konkret vorliegende Dokument oder Produkt zu verbessern und sich selbst zu verbessern, verschiebt sich hin zu einer Beurteilung und evtl. Abwertung des Entwicklers. Wenn sich ein derartiges Klima einstellt, stößt das Review auf Ablehnung und die Mitarbeiter verschließen sich gegenüber diesem Mittel zur Qualitätsverbesserung. (Wiegers, 2001)

Für eine Lehrveranstaltung steckt viel Potential für die Weiterentwicklung der Fähigkeiten der Studierenden in Reviews, die eine besondere Form des kollaborativen Lernens darstellen. Bauer et al. (Bauer, Figl, Derntl et al., 2009) berichten in ihrem Artikel über Online-Reviews in der Informatik-Ausbildung u.a. darüber, dass die Reviews die Entwicklung des Bewusstseins für Qualität bei der eigenen Arbeit erhöhen. Neben der Begutachtung anderer Lösungsansätze lernen die Studierenden ebenso viel von den Kommentaren und dem Feedback der Reviewer-Gruppe (Bauer, Figl, Derntl et al., 2009). Auf diese Weise wird die Motivation einerseits erhöht und andererseits vertiefen die Studierenden ihr schon vorhandenes Wissen (S. Trahasch, 2004). Durch diese Vorgehensweise, gegenseitig die Arbeiten zu begutachten, erfahren die Studierenden außerdem, dass Reviews eine wichtige Rolle im Bereich der Softwareentwicklung spielen.

Reviews im SoPra

Bereits im Motivationskapitel wurde darauf hingewiesen, dass Reviews im SoPra in vielfältiger Form vorkommen.

Zur Qualitätssicherung der UML-Diagramme setzen wir im SoPra sowohl interne als auch externe Reviews der UML-Diagramme ein. Interne Begutachtungen innerhalb der Gruppe finden statt, wenn Diagramme, die von einzelnen Teilnehmern zuhause erstellt werden, in der Gruppe betrachtet und kritisch diskutiert werden. Dabei wird eher informell vorgegangen.

Daneben gibt es auch externe, formelle Reviews. Die gegenseitige Begutachtung der UML-Diagramme durch die SoPra-Gruppen erfolgt strukturiert und Checklisten-basiert. Ausgehend von dem in Abb. 2 dargestellten Qualitätsmodell wurde eine Checkliste (siehe Anhang B) erarbeitet, die den Gruppen als Basis für die Suche nach Fehlern und Mängeln dient. Die Fragen beziehen sich auf die drei im Kapitel „Zur Qualität von UML-Diagrammen“ genannten Kategorien.

Wir haben folgende Vorgehensweise ausgewählt. Jede Gruppe begutachtet den Projektstatus einer anderen Gruppe. Es wird also für jede Gruppe eine Review-Gruppe festgelegt. Das Review der UML-Diagramme findet an zwei Zeitpunkten im Entwicklungsprozess statt: Am Ende der Anforderungsdefinition, wenn das Anwendungsfalldiagramm, die Aktivitätsdiagramme und das Problembereichsmodell fertig sind, und am Ende der Analysephase, wenn das Strukturmodell und die Sequenzdiagramme fertig sind. Am Ende des Projekts findet noch die gegenseitige Begutachtung der fertigen Produkte durch die Gruppen sowie eine statische Code-Analyse statt. Diese Begutachtung soll hier allerdings nicht betrachtet werden, ebenso wenig wie die Bewertung der Qualität des Quellcodes sowie die Begutachtung und die Korrektur der Diagramme und Dokumente durch die Gruppenbetreuer.

Der Begutachtungsprozess der UML-Diagramme läuft in zwei Schritten ab. Zur Vorbereitung werden die für die Präsentation vorgesehenen Diagramme im öffentlichen Bereich des Gruppen-Wikis hochgeladen und können damit von allen SoPra-Beteiligten eingesehen werden. In der letzten Gruppensitzung vor der Präsentation findet die Inspektion der Diagramme anhand der Checklisten statt. Als Beispiel ist in Anhang B die Checkliste für das Anwendungsfalldiagramm angegeben.

Alle Gruppenmitglieder betrachten das Diagramm. Ein Gruppenmitglied liest die Fragen

nacheinander vor. Die Gruppe diskutiert und einigt sich auf eine Antwort und trägt die gefundenen Mängel in das Formular ein. Abschließend wird das Diagramm allgemein bewertet. Für alle Diagramme (4 bzw. 5) sollten nicht mehr als 45 min. aufgewendet werden. Um zu vermeiden, dass Mängel übersehen werden oder Fehler gefunden werden, die keine sind, achten die GruppenbetreuerInnen auf die Qualität der Diskussion und die Begründungen bei der Begutachtung der UML-Diagramme.

Der zweite Schritt ist die Präsentation der Diagramme im Plenum, das aus zwei bis drei anderen Gruppen besteht. Einer Gruppe stehen 15 min. zur Verfügung, die von ihr ausgewählten und im öffentlichen Bereich hochgeladenen 4 bzw. 5 Diagramme zu präsentieren und zur Diskussion zu stellen. Danach stellt das Plenum Verständnisfragen und macht auf Mängel aufmerksam, die durch die Reviews gefunden wurden. Auf diese Weise bekommt jede Gruppe Feedback zu der Qualität der vorgestellten Diagramme und die Verständnisfragen lassen sich unkompliziert in einer Face-to-Face-Situation klären.

Das durch Checklisten unterstütze Review wurde im Sommer 2016 erstmalig durchgeführt. An diesem SoPra nahmen 10 Gruppen teil. Alle Gruppen hatten jeweils die Aufgabe, in der Teamsitzung vor der Präsentation des Projektfortschritts mit Hilfe von Checklisten die UML-Diagramme zu begutachten. Zu bewerten waren von jeder Gruppe jeweils ein Anwendungsfalldiagramm, drei Aktivitätsdiagramme, ein Problembereichsmodell, ein Strukturmodell und drei Sequenzdiagramme. Also wurden insgesamt fast 90 Diagramme, allerdings nur jeweils einmal, begutachtet.

Von den ausgeteilten Checklisten wurden die meisten ausgefüllt zurückgegeben. Allerdings blieben drei Bögen zu den Aktivitätsdiagrammen leer und bei einem Bogen ist die Zuordnung zum zugehörigen Aktivitätsdiagramm unklar. Von den Fragebögen zu den Sequenzdiagrammen blieben fünf leer und bei zweien ist die Zuordnung zum zugehörigen Sequenzdiagramm wiederum etwas unsicher. Insgesamt konnten 82 von 90 Bögen ausgewertet werden.

Trotz dieser Probleme und Unvollständigkeiten ergibt die Auswertung der 82 Checklisten ein ungefähres Bild über die Beurteilung der Qualität der verschiedenen Diagramme durch die SoPra-Teilnehmer. Besonders auffallend ist, dass die häufigsten Mängel im Bereich der Semantik festgestellt wurden. Bei 43 Diagrammen wurde dieser Bereich als derjenige mit angegeben, in dem viele Mängel zu finden sind (wie die Abb. 4 zeigt), während viele ästhetische Mängel nur in 15 Diagrammen vorkommen. In 14 Diagrammen spielen auch syntaktische Fehler eine große Rolle. Dabei handelt es sich zur Hälfte um Aktivitätsdiagramme.

Dass nur bei einem Sequenzdiagramm viele Mängel im ästhetischen Bereich und bei zweien viele Syntaxmängel erkannt wurden, verwundert zunächst, da Sequenzdiagramme unter Studierenden als besonders schwierig gelten. Aber für die Gestaltung der Sequenzdiagramme gibt es besonders viele formale Regeln, für deren Einhaltung zum Teil auch der UML-Editor sorgt. Die beteiligten Objekte sind ganz oben nebeneinander angeordnet. Das Diagramm ist von oben nach unten zu lesen und zeigt einen zeitlichen Ablauf. Bei den Aktivitätsdiagrammen sind die Modellierer dagegen völlig

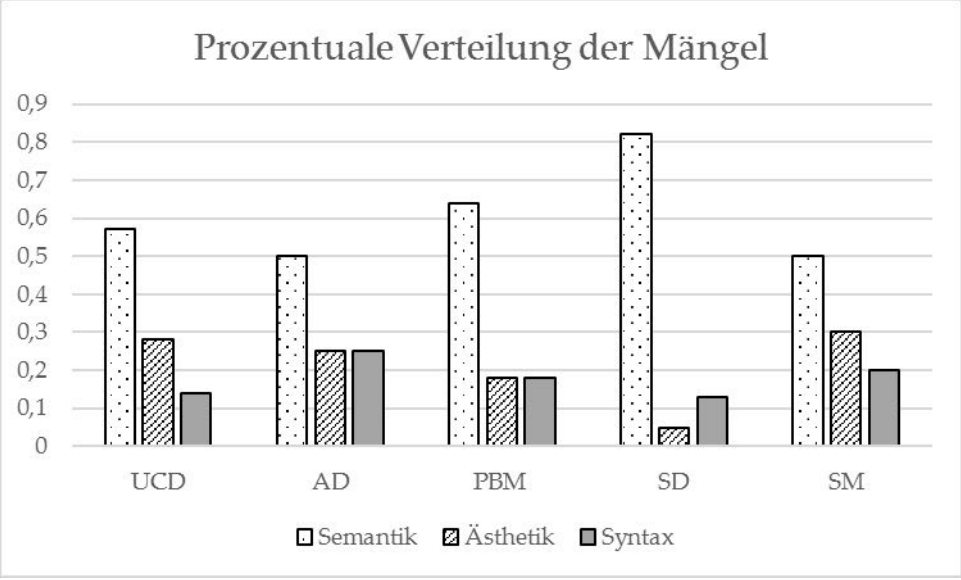


Abb. 4: Gefundene Mängel in den beurteilten Diagrammen (UCD: Anwendungsfalldiagramm, AD: Aktivitätsdiagram, PBM: Problembereichsmodell, SD: Sequenzdiagramm, SM: Strukturmodell)

frei in der Anordnung und der Verwendung der Komponenten.

Dass beim Anwendungsfalldiagramm, Problembereichsmodell und Strukturmodell wenige syntaktische Fehler gefunden wurden, wundert nicht so sehr. Für das Anwendungsfalldiagramm gelten nur wenige Regeln und die beiden Klassendiaramme sind nahe an Programmierung, dem Java-Code, dem Teil der Entwicklung, mit dem die Studierenden die meiste Erfahrung haben.

Erfahrungen

Neben den tieferen Einsichten in die Qualität der von den Studierenden erstellten Diagramme hat diese Vorgehensweise der gegenseitigen Begutachtung der UML-Diagramme anhand von Checklisten, die wir eigentlich nur eingeführt haben, um an Daten für eine Bachelorarbeit zu gelangen, sehr viele Vorteile für die Lehre mit sich gebracht.

Schon lange ist die Präsentation der Diagramme ein wichtiger Bestandteil des Software-Praktikums, der auch in der Modulbeschreibung auftaucht. Das Ziel der Präsentationen ist eine weitere Überprüfung, ob alle geforderten Funktionalitäten betrachtet wurden. Darüber hinaus wird geprüft, ob alle Studierenden die Aufgabestellung sowie das Entwicklungsprozess gut verstanden haben. Deswegen spielt die Qualität der Diagramme bei den Präsentationen eine wichtige Rolle, insbesondere ihre Lesbarkeit und Verständlichkeit. Allerdings verlief die Diskussion bei den Präsentationen meist schleppend. Die einzigen Anmerkungen und Fragen kamen von den Lehrenden. Ein erster Versuch, die Studierenden der anderen Gruppen mehr in Diskussion einzubeziehen, indem die Diagramme vor der Präsentation veröffentlicht wurden und immer eine Gruppe zum Reviewer bestimmt wurde, hat nur ein wenig Verbesserung gebracht. Manchmal kamen dann tatsächlich ein paar Fragen und Anmerkungen von dieser zuvor bestimmten Review-Gruppe.

Den wirklichen Durchbruch brachten die Checklisten. Während sich bisher bei der Betrachtung von Diagrammen anderer Gruppen nur wenige interessiert gezeigt haben, entstand plötzlich in der Inspektionssitzung eine lebhafte Diskussion über die dargestellte Lösung. Sehr akribisch und ernsthaft, aber auch durchaus fair, wurden die Diagramme der anderen Gruppe inspiziert. Die differenzierte Betrachtung der einzelnen Qualitätsaspekte ermöglichte eine tiefere Diskussion.

Auch der erhoffte Effekt, dass sich durch die Beschäftigung mit der Qualität von fremden Diagrammen der Blick für die Diagrammqualität schärft und dass auch bei der Erstellung der eigenen Diagramme in der nächsten Projektphase mehr Sorgfalt verwendet wird, ist eingetreten. Das wurde in allen Gruppen beobachtet, so dass die Checklisten-basierten Reviews auf jeden Fall ein fester Bestandteil des SoPra bleiben werden. Die Zeit von etwa 45 min., die für die Inspektionssitzung aufgewendet wird, ist sehr nützlich investiert, da in der nächsten Iteration die Qualität der abgegebenen Diagramme steigt. Dass das Bewusstsein für Qualität gestiegen ist, kann man den Diskussionen in den Gruppen entnehmen.

Wir haben uns gegen das Weiterleiten der ausgefüllten Checklisten an die Entwickler entschieden, da die Gutachter in den Präsentationen im Plenum die Gelegenheit haben, ihre Fragen zu stellen und ihre Kritik öffentlich zu äußern. Die Verantwortlichen für die Lehrveranstaltung nehmen an diesen Plenumssitzungen teil und können jederzeit bei unsachgemäßer Kritik eingreifen.

Zusammenfassung und Ausblick

In diesem Beitrag haben wir gezeigt, wie Checklisten-basierte gegenseitige Reviews im SoPra eingesetzt werden, um die Qualität der UML-Diagramme zu erhöhen. Die in UML-Diagrammen vorzufindenden Mängel werden in drei Kategorien aufgeteilt. Wir präsentieren, welche dieser Kategorien beim gegenseitigen Review von den Studierenden am häufigsten gefunden wurden.

Die Abgrenzung der Mängelklassen in Syntax, Semantik und Ästhetik ist nicht einfach. Wenn beispielsweise die Assoziationsrichtung in einem Klassendiagramm falsch eingezeichnet ist, könnte dieser im Ursprung syntaktische Fehler wie ein semantischer Fehler wirken, da ein anderer Sachverhalt dargestellt ist. Ähnliches gilt für die Verwechselung von *include* und *extend* im Anwendungsfalldiagramm.

Wenn die Studierenden insgesamt nur wenige syntaktische Mängel gefunden haben, kann es daran liegen, dass sie selbst die UML-Syntax nicht so ganz genau beherrschen. Deshalb arbeiten wir an einem Eclipse-Plugin zur Syntax- und Konsistenzprüfung von UML-Diagrammen. Damit könnten die Studierenden dann selbst ihre Diagramme vor der Abgabe prüfen.

Die semantische und ästhetische Qualität der Diagramme lässt sich unserer Meinung nach am besten durch eine gegenseitige Begutachtung mit Diskussion bewerten. Da alle Gruppen an der gleichen Aufgabe arbeiten, sind alle SoPra-Teilnehmer Experten auf dem Anwendungsgebiet. Der Einblick in die fremde Lösung ermöglicht zudem das Erkennen von Unvollständigkeiten und Mängel im eigenen Modell.

Der besondere Zugewinn durch die Reviews besteht darin, dass sich insgesamt das Bewusstsein für Diagramm-Qualität sehr erhöht hat, wie alle

Gruppenbetreuer übereinstimmend berichten. Die Zeit für die Inspektion der fremden Diagramme geht nur scheinbar dem eigenen Projekt verloren, vielmehr werden durch die verbesserte Qualität Korrekturzyklen eingespart.

Literatur

Arendt T. (2014): Quality Assurance of Software Models. A Structured Quality Assurance Process supported by a flexible Tool Environment in the Eclipse Modeling Project. Dissertation. Marburg.

Astah. (2016) Zugriff am 31.10.2016. Verfügbar unter http://astah.net/

Bauer, B., Figl, K., Derntl, M., Beran, P., Kabicher S. (2009): The student view on online peer reviews. SIGCSE Bull.

Bettin, J., Helsen, S., Kunz, M. (2007): Modellgetriebene Softwareentwicklung. dpunkt.verlag.

Boehm, B., Basili, V. R. (2001): Software Defect ReductionTop 10 List. Software's complexity andaccelerated developmentschedules make avoiding defectsdifficult. These 10 techniques can help reduce the flaws in your code. IEEE Computer, 135-137.

Briand, L., Labiche, Y., Leduc, J. (2006): Toward the Reverse Engineering of UML Sequence Diagrams for Distributed Java Software. IEEE Trans. Softw. Eng.

Fieber, F., Huhn, M. & Rumpe, B. (2008): Modellqualität als Indikator für Softwarequalität: eine Taxonomie. Informatik-Spektrum – Online.

IEEE. (2008): IEEE Standard for Software Reviews and Audits. IEEE Std 1028-2008.

Marian, P. (2013): UML in Practice. International Conference on Software Engineering.

Mohagheghi, P., Dehlen, V., Neple, T. (2009): Definitions and approaches to model quality in model-based software development – A review of literature. Quality of UML Models.

PMD. (2016): Zugriff am 26.10.2016. Verfügbar unter https://pmd.github.io/

Schmedding, D., Vasileva, A., Remmers, J. (2015): Clean Code - ein neues Ziel im Software-Praktikum. Tagungsband des 14. Workshops "Software Engineering im Unterricht der Hochschulen".

SoPra-Wiki. (2016): Software Praktikum. Zugriff am 01.11.2016. Verfügbar unter https://sopra.cs.tu-dortmund.de/wiki/

Striewe, M., Goedicke, M. (2011): Automated checks on UML diagrams. ACM.

Todorinski, S. (2016): Syntax- und Konsistenzchecks von UML-Diagrammen. Interne Berichte / Universität Dortmund, Fakultät Informatik.

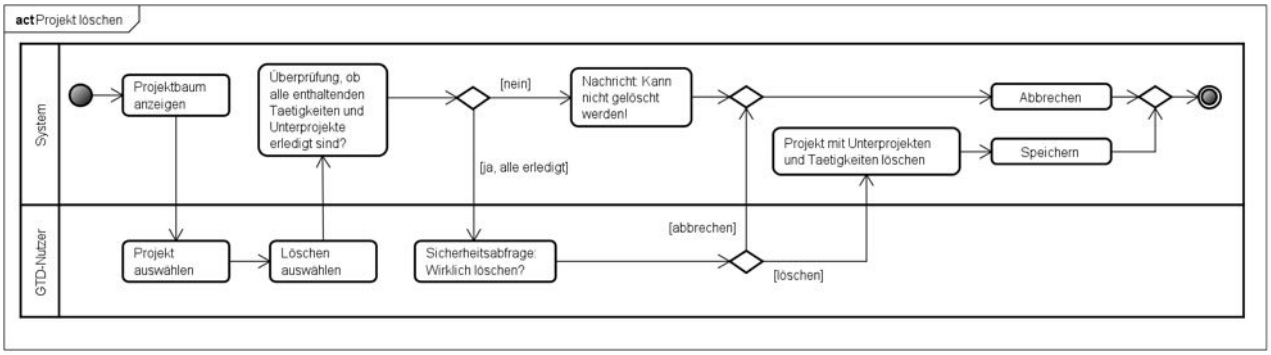
Trahasch, S. (2004): (Hrsg.). From peer assessment towards collaborative learning (34th Annual Frontiers in Education, 2004. FIE 2004).

Unhelkar, B. (2005): Verification and validation for quality of UML 2.0 Models. New Jersey: Hoboken.

Vasileva, A., Schmedding, D. (2016) How to Improve Code Quality by Measurement and Refactoring. Quality of Information and Communications Technology (QUATIC).

Wiegers, K. (2001): Peer Reviews in Software: A Practical Guide. Addison-Wesley Professional.

Anhang A: Aktivitätsdiagramm



Autoren: Alice, Bob

Kurzbeschreibung:
Löschen eines Projekts mit allen Unterprojekten und zugeordneten Tätigkeiten, sofern alle den Status erledigt tragen. Ein Unterprojekt ist erledigt, wenn alle seine Tätigkeiten und Unterprojekte erledigt sind.

Akteure:
Benutzer des Programms

Auslöser:
Der Benutzer wählt die Aktion „Projekt löschen“ aus.

Vorbedingung:
Das zu löschende Projekt muss angezeigt werden.

Nachbedingung:
Waren alle Unterprojekte und Tätigkeiten erledigt, ist das Projekt gelöscht.
Ansonsten keine Änderung des Zustands.

Standardablauf:

1. Auswahl des zu löschenden Projekts aus der Projektliste.
2. Überprüfung, ob alle Tätigkeiten und Unterprojekte erledigt sind.
3. Falls alles erledigt ist, muss Benutzer eine Sicherheitsabfrage zum Löschen des Projekts bestätigen.
4. Entfernen des Projekts mit allen seinen Unterprojekten und Tätigkeiten aus dem System.

Mögliche Fehler:
gewähltes Projekt wird nicht gefunden

Alternativabläufe:

1. Wenn nicht alle Tätigkeiten und Unterprojekte erledigt sind, wird nicht gelöscht.
2. Bestätigt der Benutzer das Löschen nicht, wird die Aktion „Projekt löschen“ abgebrochen.

Anhang B: Fragebogen zum Anwendungsfalldiagramm

Fragebogen zum Anwendungsfalldiagramm

	1	2	3	4	5	6	7	8	9	10
Das Diagramm ist von der Gruppe:	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

	ja	eher ja	eher nein	nein	schlecht zu sagen
Ist das Diagramm informativ? (viele Informationen, vollständig)	☐	☐	☐	☐	☐
Ist das Diagramm leicht verständlich? (kein Erklärungsbedarf notwendig)	☐	☐	☐	☐	☐
Ist das Diagramm übersichtlich? (z.B. Elemente sinnvoll angeordnet)	☐	☐	☐	☐	☐

Welche der folgenden Mängel finden sich im Diagramm?

(Es sind stets Mehrfachantworten möglich, aber bitte keine syntaktischen Fehler bewerten! Zur Syntax zählen alle Fehler, die die inkorrekte Verwendung der Diagrammelemente betreffen wie z.B. Kasten für Anwendungsfall statt Oval)

Im Bereich <u>Ästhetik</u> ist das Diagramm …

… zu detailliert / zu komplex … ☐

… denn es gibt zu viele:
Akteure ☐ **System-Akteure** ☐ **Anwendungsfälle** ☐ **Assoziationen** ☐
Generalisierungen ☐ **Extend-/Include-Bez.** ☐ **Notizzettel** ☐

Andere Gründe:

… zu wenig detailliert / unvollständig … ☐

… denn es gibt zu wenige / es fehlen:
Akteure ☐ **System-Akteure** ☐ **Anwendungsfälle** ☐ **Assoziationen** ☐
Generalisierungen ☐ **Extend-/Include-Bez.** ☐ **Notizzettel** ☐

Andere Gründe:

… unübersichtlich … ☐

… wegen: **sich überschneidender Linien** ☐ **ungünstiger Anordnung der Elemente** ☐

Andere Gründe:

Im Bereich <u>Semantik …</u>

… ist die Bedeutung der folgenden Elemente unklar:
(z.B. wegen schlecht gewählter Bezeichner)
Akteure ☐ **System-Akteure** ☐ **Anwendungsfälle** ☐ **Assoziationen** ☐
Generalisierungen ☐ **Extend-/Include-Bez.** ☐ **Notizzettel** ☐

Grund/Gründe für die Unklarheit/en:

Gewählter Name / Erklärung ist zu … ☐ **kurz** ☐ **lang** ☐ **ungenau**

☐ **Element macht keinen erkennbaren Sinn**

Andere Gründe:

☐ **Es existieren Duplikate/ Wiederholungen / Redundanzen**

Sonstige Mängel (außer Syntaxfehler!):

☐ **Keine Mängel erkennbar**

Falls Mängel vorhanden sind:

Welche Mängel überwiegen: (Mehrfachantworten möglich, falls gleich viele Mängel in verschiedenen Bereichen vorhanden sind)

Im Bereich der …

☐ **Syntax** ☐ **Semantik** (Bedeutung) ☐ **Ästhetik** (z.B. Layout, Abstraktheit, ...)

Insgesamte Bewertung des Diagramms:

(sehr gut) ☐ **1** ☐ **2** ☐ **3** ☐ **4** ☐ **5** ☐ **6** **(schlecht)**

Das Diagramm ist …

... gut, denn es ist: (Mehrfachantworten möglich)

☐ **informativ** ☐ **leicht verständlich** ☐ **übersichtlich** ☐ **nichts von allem**

... nicht gut, denn es ist: (Mehrfachantworten möglich)

☐ **nicht informativ** ☐ **schwer verständlich** ☐ **unübersichtlich** ☐ **nichts von allem**

Weitere Anmerkungen sind hier möglich!