Analyse von Programmieraufgaben durch Softwareproduktmetriken

Michael Striewe, Michael Goedicke
Universität Duisburg-Essen
{michael.striewe,michael.goedicke}@s3.uni-due.de

Zusammenfassung

Der Einsatz von Softwareproduktmetriken zur Beobachtung und Beurteilung von Softwareprojekten ist ein oft diskutiertes Thema. Es gibt vielfältige Vor- und Nachteile, die beim Einsatz zu berücksichtigen sind. Der vorliegende Artikel diskutiert an Fallbeispielen, wie Metriken zur Lehrunterstützung in einer Lehrveranstaltung zur Programmierung genutzt werden können.

Einleitung

Softwareentwicklung kann über Kennzahlen, die durch den Einsatz von Metriken erhoben werden, beobachtet und bewertet werden. Dabei kann sowohl das entwickelte Softwareprodukt als auch der Entwicklungsprozess Ziel der Beobachtung und Bewertung sein (Conte u. a., 1986). Der Einsatz von Metriken wird vielfach kritisch diskutiert: Auf der einen Seite steht die Aussage, dass ohne Messung keine Kontrolle möglich ist (z.B. (DeMarco, 1986)) und auf der andere Seite die Erkenntnis, dass die Messung das Ergebnis selber beeinflussen kann und viele Metriken gar nicht das messen, was sie zu messen vorgeben (z.B. (Kaner u. Bond, 2004)). Insgesamt kann der Einsatz von Softwaremetriken aber als etabliert betrachtet werden.

Nicht nur in der industriellen Praxis, sondern auch in der Lehre werden Softwareprodukte entwickelt. Es handelt sich hierbei zwar nicht um Produkte im ökonomischen Sinne, aber doch um Artefakte, deren Eigenschaften mit denselben Metriken messbar sind. Zudem ist der Begriff der Kontrolle in Form von Lernfortschrittskontrollen ein zentraler Bestandteil der Lehre. Es erscheint daher sinnvoll, Werkzeuge zur Unterstützung der Lehre zu konzipieren, mit denen Kennzahlen automatisch gewonnen und analysiert werden können. Dies ist insbesondere in E-Learning- und Blended-Learning-Szenarien wichtig, in denen eine (teil-)automatisierte Führung durch den Lernstoff angeboten werden soll und daher nicht immer ein Lehrender zur Verfügung steht, der die Leistungsfähigkeit seiner Studierenden einschätzt und ihnen passende Übungsaufgaben zuweist. Um hier eine Automatisierung zu erreichen sollte untersucht werden, wie Softwaremetriken zur Beobachtung des Lernprozesses und der dabei erzeugten Produkte einsetzbar sind. Der vorliegende Artikel stellt letzteres in den Fokus und befasst sich daher ausschließlich mit Softwareproduktmetriken. Aus praktischen Erwägungen heraus beziehen sich die im Folgenden diskutierten Ansätze und Ergebnisse auf Anfängervorlesungen zur Programmierung. Eine Generalisierung auf fortgeschrittene Lehrveranstaltungen zur Programmierung sowie Software Engineering im Allgemeinen bedarf zweifellos weiterer Forschungsarbeit, die weit über den Rahmen dieses Artikels hinaus geht.

Der Artikel ist wie folgt gegliedert: Zunächst werden einige Szenarien skizziert, in denen Metriken zur Lehrunterstützung angewandt werden können. Im selben Abschnitt werden zudem grundsätzliche Analysemethoden und existierende Ansätze genannt. Anschließend werden Metriken vorgestellt, die bei der Analyse von Programmieraufgaben genutzt werden können. Danach werden mehrere Fallbeispiele diskutiert, in denen eine Auswahl der Metriken auf tatsächliche Lösungen von Programmieraufgaben angewandt wurden. Aus den daraus gewonnenen Erkenntnissen wird am Ende des Beitrags ein Fazit gezogen.

Szenarien, Methoden und Verwandte Arbeiten

Es sind verschiedene Einsatzszenarien denkbar, in denen Softwareproduktmetriken bei der Analyse von Programmieraufgaben und ihren Lösungen zum Einsatz kommen können. Sie unterscheiden sich sowohl nach ihrer Zielsetzung als auch der Analysebasis: Als Zielsetzung gibt es die Rückmeldung an die Studierenden, die Rückmeldung an die Lehrenden und die systeminternen Nutzung zur automatischen Steuerung von Lernprozessen, während bei der Analysebasis zu unterscheiden ist, ob Aussagen über eine individuelle Lösung oder die Gesamtheit der Lösungen einer Aufgabe getroffen werden sollen. Aus diesen Merkmalen lassen sich zahlreiche Kombinationen bilden, z.B.:

- Durch die Analyse einzelner Lösungen und die Aufbereitung der Ergebnisse für die Studierenden können Aussagen zum Verhältnis von studenti-

scher Lösung und Musterlösung getroffen werden. Studierenden kann so zum Beispiel der Hinweis gegeben werden, dass ihre Lösung zwar alle funktionalen Anforderungen erfüllt, jedoch deutlich umfangreicher als die Musterlösung ist. Dies muss nicht mit einer negativen Bewertung der Lösung einhergehen, sondern kann auch als Hinweis verstanden werden, der die Studierenden zur Verbesserung ihrer Lösung anregen soll.

- Durch die Analyse aller Lösungen und die Aufbereitung der Ergebnisse für die Studierenden kann diesen angezeigt werden, ob sie eine typische Lösung gewählt haben, oder ob sie sich in bestimmten Merkmalen deutlich von anderen Lösungen unterscheidet. Auch dies kann den Studierenden einen Hinweis darauf geben, in welche Richtung sie ihre Lösungen verbessern können.

- Lehrende können dieselben Daten nutzen, um Häufungen bei Lösungsmerkmalen zu entdecken und damit einander ähnliche Lösungen zu finden. Handelt es sich dabei um Fehlerschwerpunkte, können diese in Lehrveranstaltungen gezielt aufgegriffen werden. Handelt es sich dagegen um gute Lösungen, die jedoch von der Musterlösung messbar abweichen, kann dies Hinweise auf einen alternativen Lösungsweg geben, der in einer zweiten Musterlösung berücksichtigt werden könnte.

- Ebenfalls durch die Analyse aller Lösungen und den Vergleich dieser Mengen über verschiedene Aufgaben hinweg können Aussagen über generelle Merkmale der jeweiligen Aufgaben getroffen werden. Zum Beispiel lässt sich so der minimale Umfang einer korrekten Lösung ableiten oder Aufgaben können anhand des Mittelwertes verschiedener Kennzahlen miteinander verglichen werden. Letzteres ist insbesondere auch in vollautomatisierten Systemen nutzbar, die auf diese Weise relevante Eigenschaften von Aufgaben lernen können.

Die letzte Kategorie steht im Fokus des vorliegenden Beitrags. Dabei wird davon ausgegangen, dass der Einsatz von Metriken in diesem Fall nur dann zu aussagekräftigen Ergebnissen kommt, wenn eine größere Menge von Aufgabenlösungen analysiert werden kann. Erst dann erscheint eine Verallgemeinerung der durch Kennzahlen gewonnen Aussagen auf die Aufgabe insgesamt sinnvoll, da in einer freien Aufgabenform wie Programmieraufgaben eine zu kleine Stichprobe kaum als repräsentativ angesehen werden kann. Aus der Gesamtheit aller Kennzahlen für eine große Menge von Lösungen sind dagegen Aussagen über die Aufgabe zu erwarten, die beispielsweise auch zur Qualitätssicherung genutzt werden können, wenn Aufgaben zu einem späteren Zeitpunkt erneut eingesetzt werden sollen.

Bei der Analyse von großen Mengen von Lösungen wird durch diese in der Regel ein Lösungsraum aufgespannt, der dann untersucht wird. Die Analyse des Raumes kann dabei wie oben skizziert auf Softwareproduktmetriken basieren, oder aber auch auf Ähnlichkeitsmaßen zwischen Lösungen. Ein möglicher Einsatzbereich ist dabei die Plagiatserkennung, bei der es nicht um die Eigenschaften der Lösungen selber geht, sondern nur um die Nähe der Lösungen zueinander (Leach, 1995). Die Nützlichkeit von Maßvergleichen in diesem Bereich ist jedoch strittig, so dass es auch zahlreiche Ansätze gibt, die Textvergleiche heranziehen.

Der Einsatz von Softwareproduktmetriken zur Qualitätssicherung und Unterstützung der Benotung wurde ebenfalls schon in Ansätzen untersucht (Mengel u. Yerramilli, 1999). Rückt die Kontrolle des Lernfortschritts oder die Generierung von Rückmeldungen in den Fokus, können auch komplexe mathematische Verfahren oder maschinelles Lernen zum Einsatz kommen (Martín u. a., 2009; Gross u. a., 2012). Im vorliegenden Beitrag werden jedoch einfachere Techniken der Datenanalyse eingesetzt.

Einige Metriken

Bei Softwareproduktmetriken wird zwischen Metriken zur Messung von Umfang und Größe, Metriken zur Messung von Struktur und Komplexität sowie Metriken zur Messung von Anwendung und Nutzen unterschieden. Die ersten beiden Kategorien lassen sich dabei den sogenannten internen Attributen von Softwareartefakten zuordnen, während die dritte Kategorie sogenannte externe Attribute berücksichtigt (Conte u. a., 1986; Fenton u. Pfleeger, 1998). Da in letzterer Kategorie die Interaktion eines Artefakts mit der Umwelt (d.h. zum Beispiel die Nutzerfreundlichkeit) gemessen wird, kann diese für die automatisierte Generierung von Kennzahlen nur begrenzt verwendet werden.

Im Folgenden werden einige Softwareproduktmetriken vorgestellt, die für die Anwendung auf Programmcode in objekt-orientierten Programmiersprachen geeignet sind. Dabei wird insbesondere untersucht, ob sie speziellen Anforderungen aus dem Kontext von Lehrveranstaltungen gerecht werden:

- Da in Übungsaufgaben häufig Codevorlagen eingesetzt werden, muss der Einfluss des vorgegebenen Programmcodes auf das Ergebnis der Metrik zuverlässig bestimmt werden können, um eine Verfälschung oder Verwässerung der Ergebnisse zu verhindern.

- Die Metriken müssen zudem geeignet sein, die Verfolgung eines didaktisch sinnvollen Ziels zu unterstützen, d.h. Kennzahlen liefern, die im Rahmen der Lehre überhaupt relevant sind.

Anzahl der Anweisungen

Die Zählung der Anweisungen im Programmcode („statements“ in Abgrenzung zu „expressions“) stellt

eine sehr einfache Umfangsmetrik dar. Im Gegensatz zur Zählung von Codezeilen („lines of code“) hat sie den Vorteil, dass sie resistent gegenüber Formatierungsänderungen am Programmcode ist und die Länge eventueller Code-Kommentare nicht berücksichtigt. Das Ergebnis dieser Metrik wird daher nur durch tatsächliche inhaltliche Änderungen am Programmcode beeinflusst. Allerdings haben nicht alle Änderungen am Programmcode Einfluss auf die Metrik, da Erweiterungen oder Vereinfachungen an Ausdrücken („expressions“) nicht berücksichtigt werden.

Die Metrik kann auch bei Aufgaben mit vorgegebenen Codevorlagen verwendet werden, da die Anzahl der dort vorgegebenen Anweisungen ebenfalls gemessen und gegebenenfalls vom Ergebnis für studentische Lösungen abgezogen werden kann. Ferner können die Kennzahlen didaktisch verwendet werden, da die Vermeidung unnötig aufwändiger und umfangreicher Lösungen zu den Lernzielen einer Veranstaltung gehören kann. Zudem hilft die Metrik zum Beispiel bei der Bestimmung der Schwierigkeit einer Aufgabe, da mehr Anweisungen im Programmcode in der Regel einen erhöhten Zeitaufwand bei der Erstellung der Lösung bedeuten.

Halstead-Metriken

Die nach ihrem Autor benannten Halstead-Metriken (Halstead, 1977) basieren ebenfalls auf der Messung der Länge eines Programms, ziehen aber zusätzlich noch die Menge der verschiedenen verwendeten Operatoren und Operanden in Betracht. Somit kann nicht nur der Umfang des Programms (Summe der Anzahl der Operanden und Operatoren), sondern auch der Umfang des verwendeten sogenannten Vokabulars (Anzahl unterschiedlicher Operanden und Operatoren) bestimmt werden. Beide Werte können verschieden miteinander in Beziehung gesetzt werden.

Bei der Analyse von Programmieraufgaben mit dieser Metrik ergibt sich das Problem, dass der Umfang des Vokabulars durch die Codevorlage nach unten begrenzt ist, auch wenn die Studierenden selber in ihrem Code ein kleineres Vokabular verwenden. Bei der Berechnung der Metrik müsste daher sorgfältig zwischen dem vorgegebenen Code und dem von Studierenden erstellten Code getrennt werden, was je nach Komplexität der Codevorlage nicht immer sauber möglich ist. Zudem ist nicht offensichtlich, ob der Umfang des Vokabulars in direkten Bezug zu einem möglichen Lernziel gesetzt werden kann. Bestenfalls kann noch angenommen werden, dass den Studierenden mit fortschreitenden Programmierkenntnissen ein größeres Vokabular bekannt ist, dass bei schwierigen Aufgaben dann auch zum Einsatz kommt. Da Aufgaben jedoch oft auch auf ein begrenztes Thema (z.B. Implementierung von Vererbungsstrukturen) abzielen, kann nicht einmal vorausgesetzt werden, dass fortgeschrittene Aufgaben tatsächlich ein umfangreiches Vokabular erfordern.

Zyklomatische Komplexität

Die zyklomatische Komplexität (McCabe, 1976) ist eine Metrik zur Komplexität des Kontrollflussgraphen eines Programms. Sie basiert auf der Anzahl der Knoten und Kanten dieses Graphen und damit auf der Entscheidungsstruktur des Programms. Das Ergebnis entspricht der maximalen Anzahl linear unabhängiger Pfade eines Programms und ist damit insbesondere unabhängig vom Hinzufügen oder Entfernen von Anweisungen, die keine Entscheidungsknoten darstellen. Sie gehört damit zu den Metriken, die die logische Struktur eines Programms unabhängig von seinem Umfang beurteilen.

Wie die oben diskutierte Anzahl der Anweisungen kann auch diese Metrik bei Aufgaben mit vorgegebenen Codevorlagen verwendet werden, indem die Komplexität dieser Vorlagen vorab bestimmt wird. Da die Vermeidung unnötig komplexer Lösungen zu den Lernzielen einer Veranstaltung gehören kann, lassen sich die Ergebnisse der Metrik auch direkt verwenden.

Die Metrik lässt sich auch ohne Aufbau des kompletten Kontrollflussgraphen eines Programms berechnen, indem alle `if`-Abfragen, bedingten Ausdrücke, Schleifen, `case`-Anweisungen, `catch`-Anweisungen (für Java) und logische Operatoren gezählt werden.

Objektorientierte Metriken

Es gibt weitere, speziell auf die Messung typischer Eigenschaften objektorientierter Programme ausgerichtete Metriken, z.B. zur Kopplung zwischen Klassen, Kohäsion innerhalb von Klassen oder Sichtbarkeit von vererbten Methoden (e Abreu u. Carapuça, 1994; Chidamber u. Kemerer, 1994). Da einige dieser Metriken so definiert sind, dass sie vom Umfang eines Programms unabhängig sind, lassen sie sich zur Bewertung der Komplexität und logischen Struktur eines Programms verwenden.

Ähnlich wie bei den oben diskutierten Halstead-Metrik besteht die Schwierigkeit, dass die ermittelten Merkmale bei Programmieraufgaben durch Codevorlagen weitgehend vorgegeben sein könnten. Gerade in Anfängervorlesungen ist es üblich, z.B. Felder einer Klasse und Methodensignaturen vorzugeben und nur die Methodenrümpfe programmieren zu lassen. Zudem sind solche Aufgaben in der Regel so begrenzt, dass die zu erstellenden Beziehungen zwischen Klassen überschaubar sind und die Studierenden kaum Wahlmöglichkeiten haben. Erst bei Aufgaben mit deutlich mehr Freiheit für die Studierenden können diese Metriken daher sinnvoll angewandt werden. Dann allerdings lassen sie sich gut in Beziehung zu einzelnen Lernzielen setzen, da zum Beispiel die Erzielung einer hohen Kohäsion ein direktes Thema des objektorientierten Designs sein kann.

Fallbeispiele

Im Folgenden wird an einigen Fallbeispielen diskutiert, wie konkrete Fragestellungen aus dem Aufgabenent-

wurf für eine Lehrveranstaltung durch den Einsatz von Metriken beantwortet werden können. Als Metriken kommen dabei stets die zyklomatische Komplexität und die Anzahl der Anweisungen zum Einsatz, d.h. eine Kombination aus Umfangs- und Komplexitätsbeobachtung. Auf die Verwendung von speziellen objektorientierten Metriken wurde verzichtet, da alle untersuchten Daten aus einer Anfängervorlesung stammen.

Alle Fallbeispiele beziehen sich auf Aufgaben aus der Erstsemestervorlesung „Programmierung“ im Wintersemester 2011/12 . Bei allen Aufgaben wurde die Einreichung von Lösungen in der Programmiersprache Java erwartet und den Studierenden war es erlaubt, beliebig viele Lösungen zu einer Aufgabe einzureichen. Bei einzelnen Aufgaben kamen so bis zu 1400 Lösungen zusammen, wobei im Schnitt 3,6 Lösungen pro Studierendem eingingen. Für die Analysen wurden solche Lösungen ausgefiltert, bei denen die Metriken aufgrund von syntaktischen Fehlern im Programmcode nicht bestimmt werden konnten. Duplikate von Lösungen wurden nicht ausgefiltert, führen aber zwangsläufig zu identischen Ergebnissen in den Metriken. Die Einreichung der Lösungen durch die Studierenden und die Berechnung der Kennzahlen aus den Metriken erfolgte über das Werkzeug JACK (Striewe u. a., 2009). Die Kennzahlen wurden den Studierenden nicht mitgeteilt, sondern nur für die rückblickende Analyse nach dem Semester verwendet.

Freiheitsgrade von Aufgaben

Beim Stellen von Übungs- und Prüfungsaufgaben ist relevant, wie viele Freiheitsgrade die Aufgaben den Studierenden beim Erstellen einer Lösung lassen. Je enger begrenzt die Aufgabenstellungen sind, desto leichter sind sie in der Regel zu korrigieren, da sich die Lehrenden in weniger verschiedene Lösungsansätze hineindenken müssen. Welche didaktischen Vor- und Nachteile sich aus eng begrenzten oder freieren Aufgaben ergeben und in welcher Phase des Lernprozesses welche Aufgaben am besten geeignet sind, soll hier nicht weiter diskutiert werden. Stattdessen soll beleuchtet werden, wie durch die Anwendung von Softwareproduktmetriken für Umfang und Komplexität überhaupt gemessen werden kann, welchen Freiheitsgrad eine Aufgabe hat.

Die Abbildungen 1 bis 3 zeigen drei Verteilungsdiagramme für Lösungen zu drei Übungsaufgaben, die in der zweiten Hälfte des Wintersemesters 2011/12 gestellt wurden und die durch die Studierenden als Hausaufgaben bearbeitet wurden. Aufgetragen sind jeweils Kreise, deren Position sich aus der Anzahl der Anweisungen und der zyklomatischen Komplexität ergibt. Je mehr Lösungen auf dieselbe Position fallen, umso größer ist der dort dargestellte Kreis. Dabei wird unterschieden, ob die Lösung als bestanden gewertet wurde oder nicht. Insbesondere ist zu beachten, dass unterschiedliche Lösungen dieselben Kennzahlen aufweisen können und an derselben Position sowohl bestandene als auch nicht bestandene Lösungen liegen können.

Es ist zu erkennen, dass die erste Übungsaufgabe (Abbildung 1; im Folgenden entsprechend der Benennung in der Lehrveranstaltung als „Miniprojekt 4“ bezeichnet) im Vergleich zu den anderen Aufgaben den geringsten Freiheitsgrad aufweist. Fast alle Lösungen liegen in einem kompakten Bereich zwischen 200 und etwas mehr als 300 Anweisungen und weisen einen Komplexitätswert zwischen knapp 70 und 100 auf. Die untere Grenze des Bereichs ergibt sich dabei automatisch aus Umfang und Komplexität der bereitgestellten Codevorlage.

Einen deutlich höheren Freiheitsgrad weist Miniprojekt 5 (Abbildung 2) auf. Umfang und Komplexität der Codevorlage entsprachen zwar in etwa den Werten aus Miniprojekt 4, aber die Lösungen verteilen sich in einem deutlich größeren Raum. Bis zu einem Umfang von 700 Anweisungen und einem Komplexitätswert von 200 ist der gesamte Lösungsraum abgedeckt, jedoch in einem ähnlich schmalen Band wie bei Miniprojekt 4. Dass es sich bei beiden Aufgaben um echte Freiheitsgrade und nicht nur die notwendige Differenz zwischen unvollständiger Codevorlage und minimaler korrekter Lösung handelt zeigt sich dadurch, dass es in beiden Aufgaben bereits im unteren Drittel der gemessenen Werte bestandene Lösungen gibt.

Dieselbe Beobachtung gilt auch für Miniprojekt 6 (Abbildung 3), auch wenn hier weniger hohe Werte für die Zahl der Anweisungen erreicht werden. Bemerkenswert ist hier jedoch die etwas breitere Streuung des Korridors, in dem die Lösungen liegen. Während in den ersten beiden betrachteten Aufgaben also eine recht starke Korrelation zwischen Umfang und Komplexität galt, lässt diese Aufgabe den Studierenden mehr Freiheiten zu umfangreicheren, aber weniger komplexen Lösungen beziehungsweise kleineren, aber komplexeren Ansätzen.

Bis hierhin kann also festgehalten werden, dass die Messung der Zahl der Anweisungen und der zyklomatischen Komplexität es erlaubt, Aussagen über den Freiheitsgrad einer Aufgabe zu treffen und somit gegebenenfalls gezielt Aufgaben auszuwählen.

Als Nebenbemerkung ist in allen drei Fällen zu beobachten, dass es in der „oberen“ Hälfte der Verteilung (d.h. in der jeweils oberen Hälfte des Wertebereichs der beiden verwendeten Metriken) je einen Punkt mit einer besonders hohen Zahl von Lösungen gibt, der bis zu knapp 20% aller Lösungen repräsentiert. Für diese auffällige Häufung konnten zwei Ursachen ermittelt werden: Zum einen konnten sich Studierende während der Bearbeitungsphase der Aufgabe Ratschläge und Hilfestellung von Tutoren holen, die sich ihrerseits an der Musterlösung einer Aufgabe orientierten. Die Merkmale dieser Musterlösung lassen sich in der Verteilung der studentischen Lösungen entsprechend

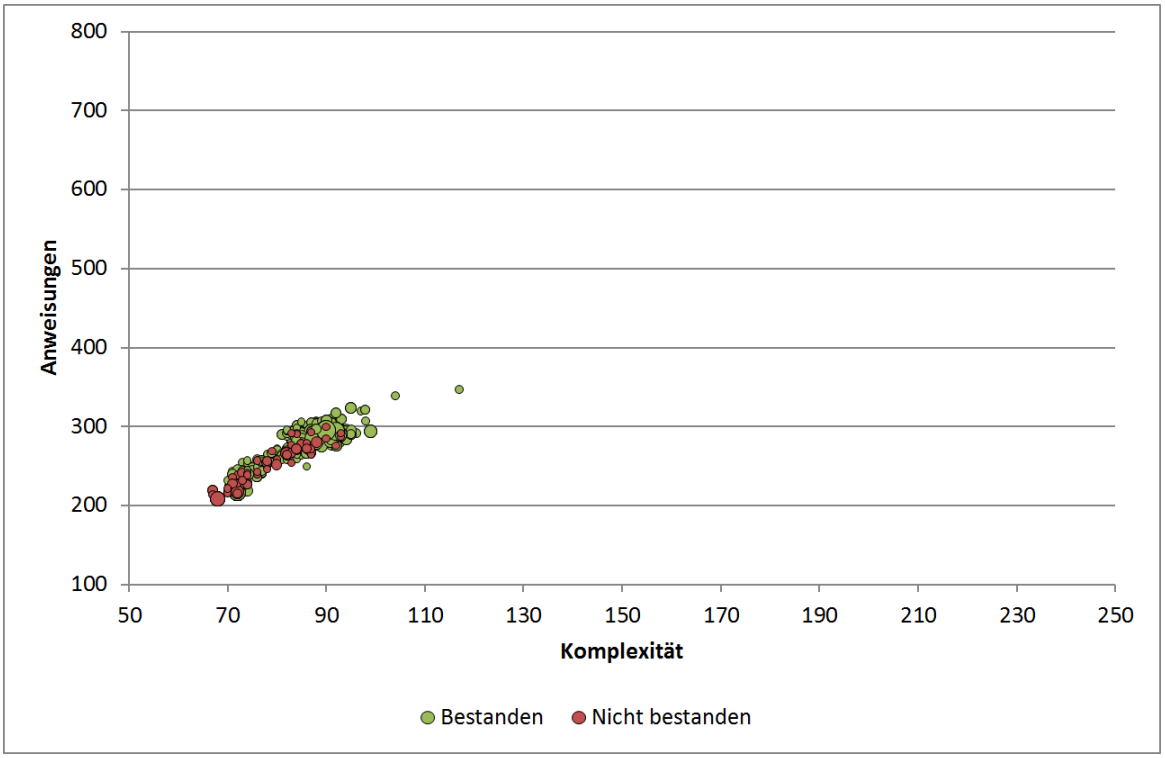

Abbildung 1: Verteilungsdiagramm für 1309 Lösungen zu einer Übungsaufgabe („Miniprojekt 4“). Größere Kreise bedeuten mehr Lösungen mit denselben metrischen Kennzahlen. Es wurden maximal 43 Lösungen mit denselben Kennzahlen eingereicht.

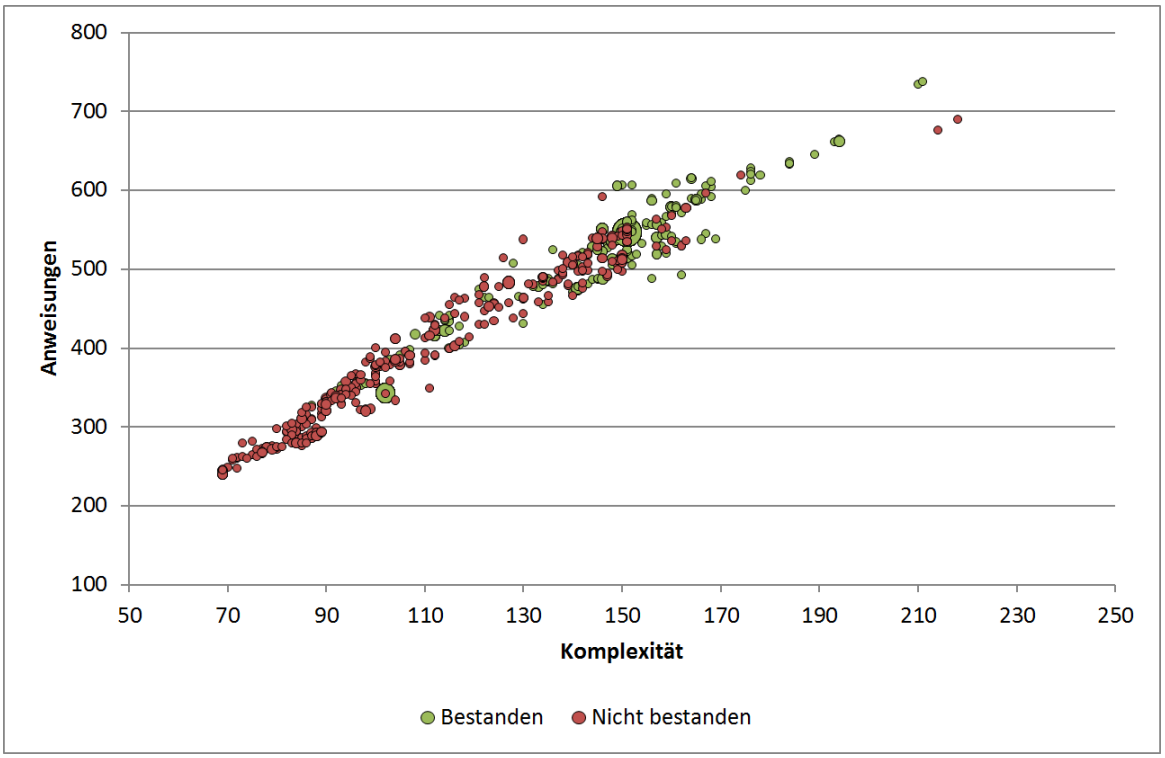

Abbildung 2: Verteilungsdiagramm für 815 Lösungen zu einer Übungsaufgabe („Miniprojekt 5“). Größere Kreise bedeuten mehr Lösungen mit denselben metrischen Kennzahlen. Es wurden maximal 155 Lösungen mit denselben Kennzahlen eingereicht.

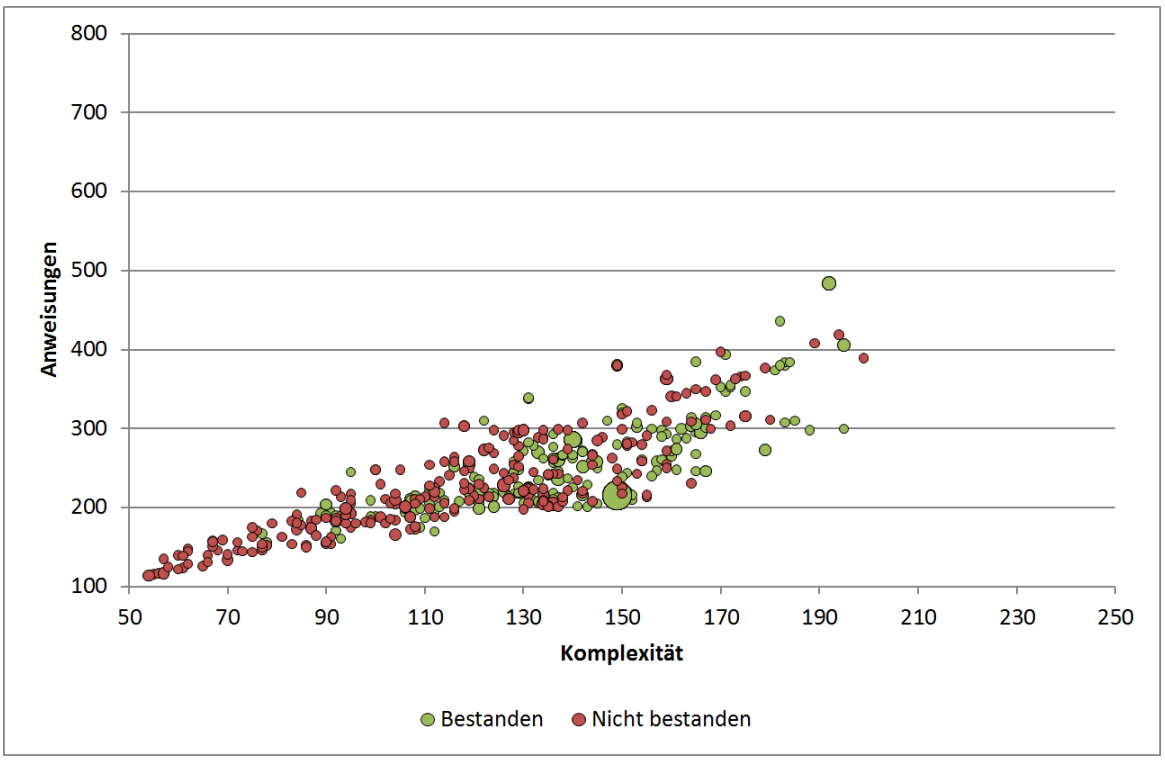

Abbildung 3: Verteilungsdiagramm für 600 Lösungen zu einer Übungsaufgabe („Miniprojekt 6“). Größere Kreise bedeuten mehr Lösungen mit denselben metrischen Kennzahlen. Es wurden maximal 85 Lösungen mit denselben Kennzahlen eingereicht.

wiederfinden. Zum anderen kursierten in den studentischen Internetforen inoffizielle Musterlösungen, an denen sich ebenfalls Studierende orientiert haben. Wie oben beschrieben, wurden die Daten nicht um Plagiate bereinigt. Stichproben zeigten jedoch, dass es sich bei den Anhäufungen nicht ausschließlich um Plagiate einer einzelnen Lösung handelt.

Relatives Aufgabenniveau

Das didaktische Konzept der untersuchten Lehrveranstaltung sah vor, dass zu jedem der oben diskutierten Miniprojekte eine kleinere Prüfungsaufgabe in Form eines Testats gestellt wird. Dieses war jeweils innerhalb von 45 Minuten im PC-Pool der Universität unter Prüfungsbedingungen zu bearbeiten. Da der PC-Pool nicht ausreichend viele Plätze bietet, um alle Studierenden gleichzeitig zu prüfen, mussten die Testate in Gruppen abgenommen werden. Daher mussten stets mehrere Aufgabenvarianten erstellt werden, damit Teilnehmer der ersten Gruppe nicht Details der Aufgabenstellung an die späteren Gruppen weitergeben konnten. Aus diesem Szenario ergeben sich zwei Anforderungen, die die Überprüfung eines relativen Aufgabenniveaus sinnvoll erscheinen lassen: Erstens muss sichergestellt sein, dass durch die verschiedenen Varianten keine Gruppe benachteiligt wird, indem ihre Aufgabe mehr Aufwand erfordert als die Aufgaben anderer Gruppen. Zweitens muss sichergestellt sein, dass die Testataufgaben insgesamt in einem angemessenen Verhältnis zum zugehörigen Miniprojekt stehen und tatsächlich einen Ausschnitt aus diesem bilden.

Um den Studierenden den Einstieg in die Testataufgaben zu erleichtern und zudem grobe Fehler beim Stellen der Testataufgaben zu vermeiden, orientierten sich diese sehr eng an den zugehörigen Miniprojekten. Es kam stets eine sehr ähnliche Codevorlage zum Einsatz und auch der narrative Kontext der Aufgabe war stets identisch mit dem des Miniprojektes. Der eigentliche Kern der Aufgabe war den Studierenden jedoch nicht vorab bekannt, d.h. es musste im Testat immer Funktionalität implementiert werden, die im Miniprojekt nicht diskutiert wurde. Daraus ergeben sich auch die Risiken bei der Aufgabenstellung, die es durch den Einsatz von Metriken abzuschätzen gilt: Zum einen können unterschiedliche Funktionalitäten mehr oder weniger Aufwand erfordern, was einzelnen Gruppen Vor- bzw. Nachteile bringt und zum anderen kann die Funktionalität in Relation zum Miniprojekt eher aufwendig oder eher einfach zu implementieren sein. Die konzeptionelle Arbeit, die von den Studierenden beim Entwurf ihrer Lösung erbracht werden muss, lässt sich freilich über die Metriken des Ergebnisses nicht bestimmen und folglich in den Betrachtungen auch nicht berücksichtigt werden.

Die Abbildungen 4 und 5 zeigen die Verteilungsdiagramme zu zwei Varianten von Testat 4. Die Verteilung der dritten Variante, an der deutlich weniger Studierende teilnahmen, sieht vergleichbar aus und wir aus

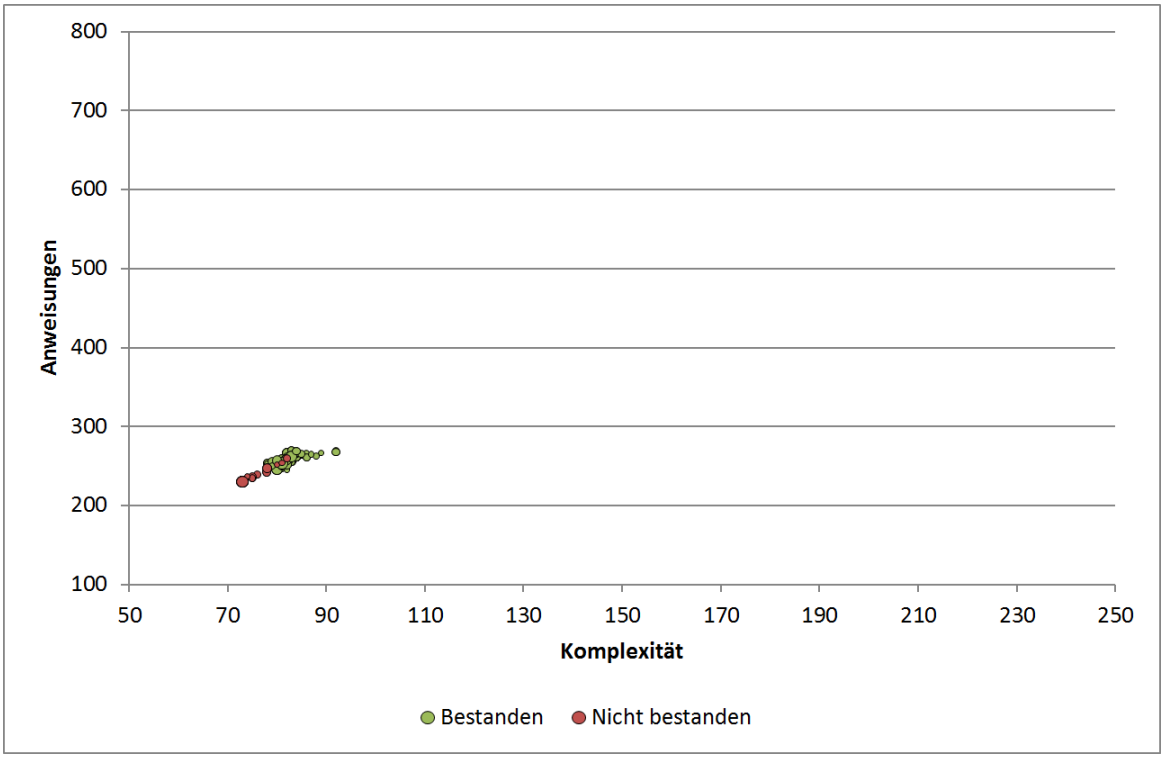

Abbildung 4: Verteilungsdiagramm für 378 Lösungen zu einer Prüfungsaufgabe („Testat 4, Variante 1“). Die Aufgabe griff die Aufgabenstellung aus Miniprojekt 4 (siehe Abbildung 1) auf. Zur Verteilung einer zweiten Aufgabenvariante vgl. Abbildung 5.

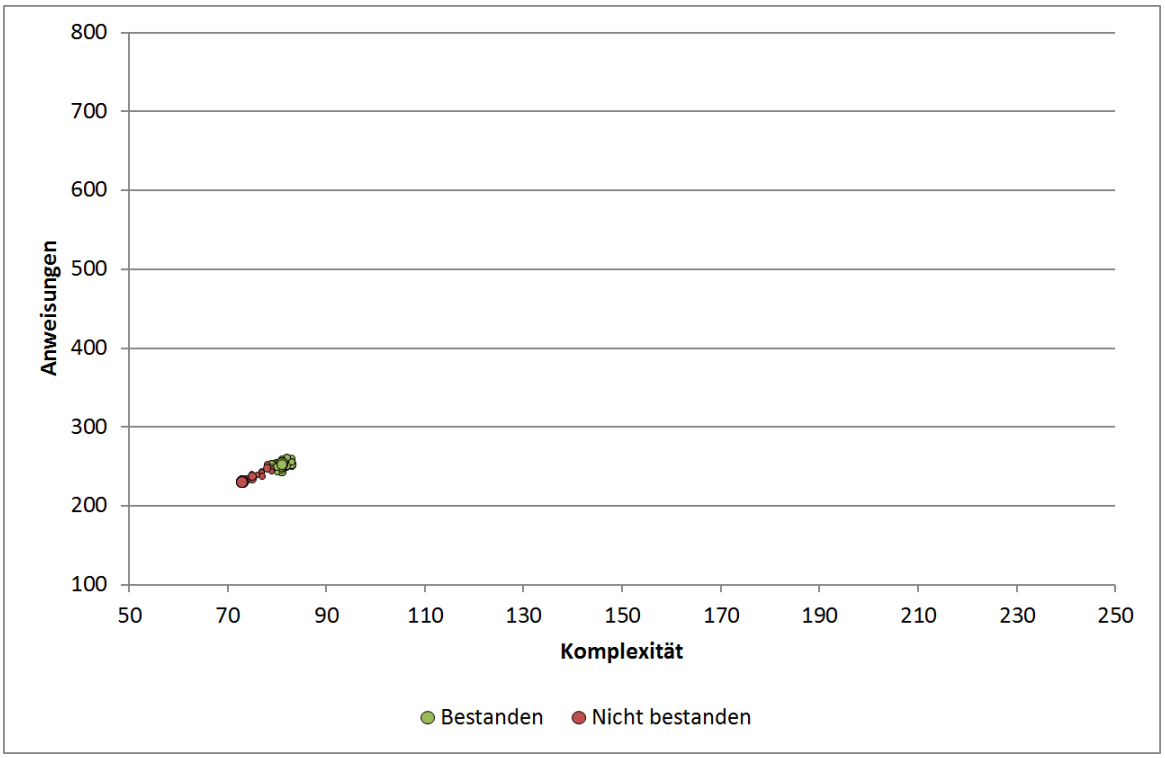

Abbildung 5: Verteilungsdiagramm für 298 Lösungen zu einer Prüfungsaufgabe („Testat 4, Variante 2“). Die Aufgabe griff die Aufgabenstellung aus Miniprojekt 4 (siehe Abbildung 1) auf. Zur Verteilung der ersten Aufgabenvariante vgl. Abbildung 4.

Platzgründen nicht dargestellt. Es ist zu erkennen, dass beide Varianten im Kern dieselbe Verteilung aufweisen. Insbesondere gibt es in beiden Aufgaben einen sehr schmalen Korridor nicht bestandener Lösungen im „unteren“ Bereich und eine größere Anhäufung bestandener Lösungen im „oberen“ Bereich der Verteilung. Es kann daher davon ausgegangen werden, dass die beiden Aufgaben zumindest im Bezug auf den Umfang und die Komplexität der Lösung identische Anforderungen an die Studierenden gestellt haben. Wie oben bereits angesprochen, kann diese Aussage nur dann auf das Aufgabenniveau insgesamt erweitert werden, wenn vorausgesetzt werden kann, dass die Herleitung der Lösung für beide Gruppen gleich schwierig ist. Bei der Interpretation der Verteilungen ist es also insbesondere als deutliches Warnsignal zu werten, wenn bei als gleichwertig vorgesehenen Aufgaben deutliche Unterschiede in den Metriken auftreten. Die Gleichheit von Umfang und Komplexität muss dagegen nicht zwangsläufig auch ein identisches Aufgabenniveau bedeuten.

Die Frage nach dem Verhältnis der Testataufgaben zum Miniprojekt kann durch einen Vergleich mit Abbildung 1 beziehungsweise durch einen Vergleich der Kennzahlen beantwortet werden: Der Komplexitätswert liegt mit 70 bis 90 nahezu mittig in den Grenzen des Miniprojektes und die Zahl der Anweisungen mit Werten zwischen 230 und 270 ebenfalls mittig im Rahmen des Miniprojektes. Es kann daher davon ausgegangen werden, dass die Testataufgaben bezogen auf Umfang und Komplexität sehr genau das Aufgabenniveau des Miniprojektes getroffen haben.

Für das Testat 5 (Abbildung 6; hier betrachtet an der dritten Aufgabenvariante dieses Testats) lässt sich eine leicht abweichende Beobachtung treffen: Der von der Verteilung abgedeckte Bereich liegt zwar im Korridor von Miniprojekt 5 und weißt (bei deutlicher kleiner Größe) eine ähnliche Form auf, liegt aber nicht mittig innerhalb der Werte des Miniprojektes, sondern deutlich nach „oben“ verschoben. Relativ zur Schwierigkeit des Miniprojektes kann hier also ein höheres Aufgabenniveau angenommen werden.

Auch der Vergleich der Testate 4 und 5 untereinander kann unter dem Gesichtspunkt des relativen Aufgabenniveaus angestellt werden. Die Verteilungen beider Testate weisen im Diagramm eine ähnliche Form auf und decken in etwa dieselbe Fläche ab, wobei Testat 5 etwas breiter gestreut ist. Aufgrund des deutlichen Abstands der Cluster kann trotzdem angenommen werden, dass Testat 5 ein höheres Niveau aufweist als Testat 4.

Da die Lehrenden nicht zwangsläufig dasselbe Bewertungsschema für Miniprojekte und Testate anlegten, lassen sich diese Annahmen nicht unmittelbar über die im Mittel erreichten Punktzahlen bestätigen oder widerlegen: Im Miniprojekt 4 wurden im arithmetischen Mittel 71,29 von 100 möglichen Punkten erreicht; im Testat 4 über alle drei Varianten im arithmetischen Mittel 63,38 Punkte. Im Miniprojekt 5 wurden im arithmetischen Mittel 48,57 Punkte erreicht; in allen drei Varianten der Testate im arithmetischen Mittel 45,28 Punkte. Das vermutete höhere relative Aufgabenniveau bei Testat 5 im Vergleich zu Miniprojekt 5 spiegelt sich also nicht in der Veränderung der Punktzahlen wider. Lediglich im Vergleich der beiden Testate untereinander tritt die erwartete Punkteänderung auf.

Die Schwierigkeit der Bestimmung des relativen Aufgabenniveaus unterstreicht auch Abbildung 7, die die Verteilung für die erste Variante von Testat 6 widergibt. Bei den Komplexitätswerten liegt diese in der Mitte des Bereichs für Miniprojekt 6; bei der Anzahl der Anweisungen im unteren Bereich. Im Vergleich mit Testat 4 weist sie insbesondere eine höhere Komplexität, aber einen geringeren Umfang der Lösungen aus. Im Bezug auf Miniprojekt 6 kann also ein leichteres Niveau angenommen werden, während im Bezug auf Testat 4 unklar ist, ob die höhere Komplexität oder der geringere Umfang mehr zum Niveau der Aufgabe beiträgt. In Miniprojekt 6 wurde im arithmetischen Mittel 50,84 Punkte erreicht, in allen Testatvarianten zusammen im arithmetischen Mittel 65,34 Punkte. Dies legt nahe, dass das Testat wie erwartet ein niedrigeres Niveau hatte als das Miniprojekt und im übrigen in etwa gleichwertig zu Testat 4 war.

Aus diesen Überlegungen lässt sich also festhalten, dass die Messung der Zahl der Anweisungen und der zyklomatischen Komplexität es erlaubt, Aussagen über das relative Niveau einer Aufgabe zu machen. Dies könnte in E-Learning-Systemen beispielsweise genutzt werden, um automatisch den Schwierigkeitsgrad von Aufgaben zu bestimmten und Studierenden damit gezielt schwierigere oder leichtere Aufgaben anbieten zu können. Noch einmal sei an dieser Stelle darauf hingewiesen, dass es andere Einflussfaktoren auf das Niveau einer Aufgabe gibt, die sich über die diskutierten Metriken nicht abbilden lassen, so dass die Verwendung von Softwareproduktmetriken trotz ihrer Nützlichkeit nicht als alleiniges Mittel zum Einsatz kommen sollte. Naheliegend ist beispielsweise die Erweiterung des Ansatzes auf Softwaremetriken, die nicht das Produkt, sondern den Erstellungsprozess messen und somit beispielsweise Auskunft darüber geben können, über welchen Zeitraum hinweg oder in welchen Einzelschritten eine Lösung erstellt wurde. Auch aus diesen Metriken kann eine Aussage über das Niveau einer Aufgabe erwartet werden.

Fazit und Ausblick

In diesem Beitrag wurde anhand von Fallbeispielen untersucht, welche Aussagen über Programmieraufgaben aus der Analyse ihrer Lösungen mit Softwareproduktmetriken gewonnen werden können. Es konnte festgestellt werden, dass mithilfe von Metriken zum Umfang und zur Komplexität von Programmen Aussagen über den Freiheitsgrad einer Aufgabe sowie das

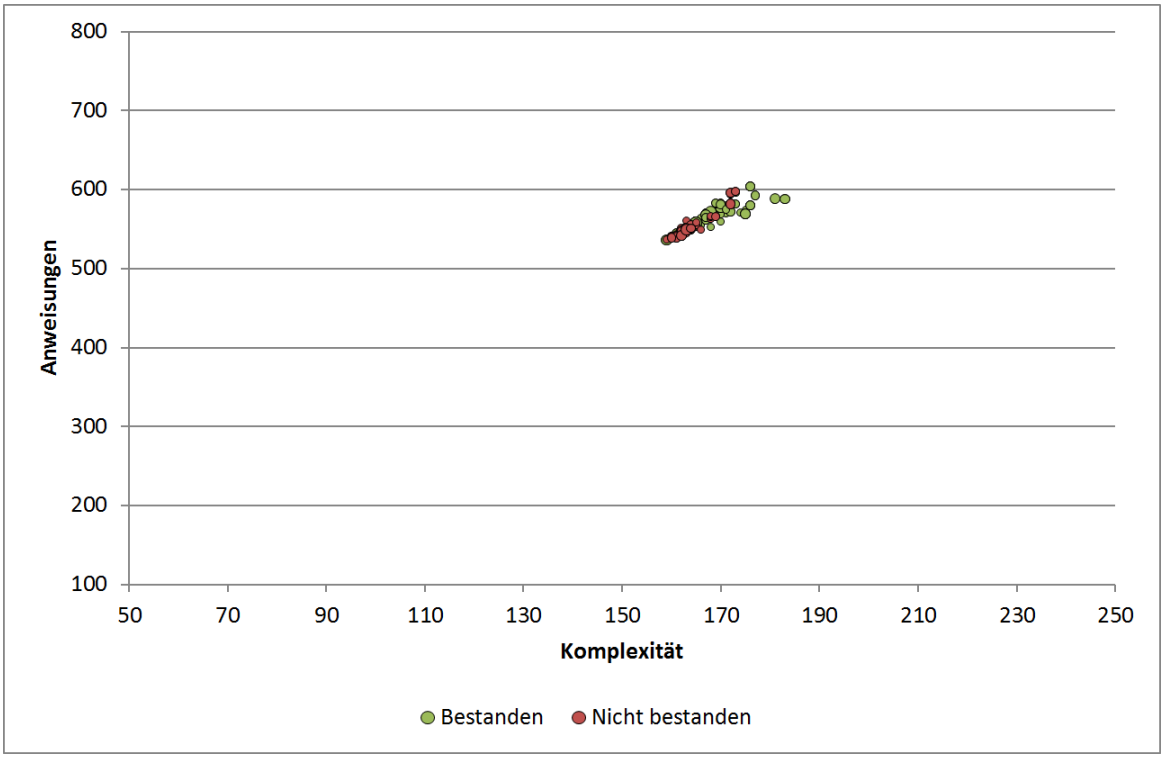

Abbildung 6: Verteilungsdiagramm für 162 Lösungen zu einer Prüfungsaufgabe („Testat 5, Variante 3“). Die Aufgabe griff die Aufgabenstellung aus Miniprojekt 5 (siehe Abbildung 2) auf.

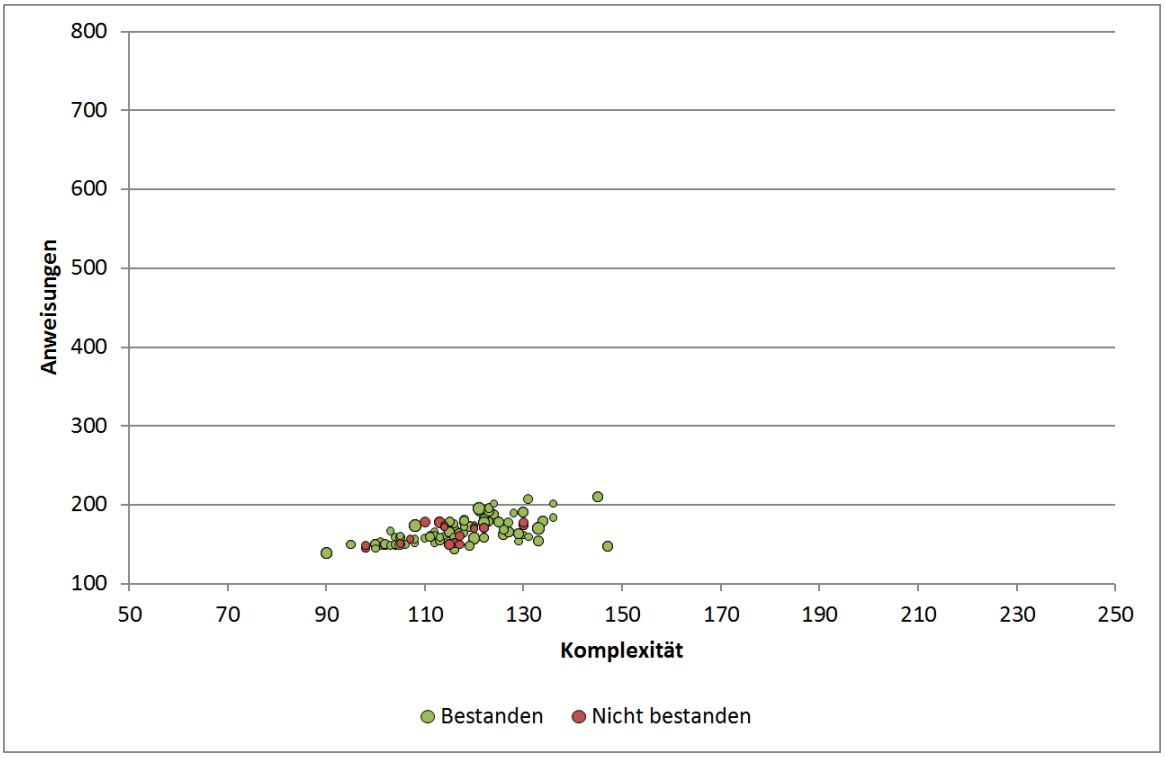

Abbildung 7: Verteilungsdiagramm für 222 Lösungen zu einer Prüfungsaufgabe („Testat 6, Variante 1“). Die Aufgabe griff die Aufgabenstellung aus Miniprojekt 6 (siehe Abbildung 3) auf.

Niveau der Aufgabe im Vergleich zu anderen Aufgaben getroffen werden können.

Daraus ergeben sich unmittelbare Einsatzmöglichkeiten in Werkzeugen zur Lehrunterstützung sowie weitere Forschungsansätze: Die gewonnenen Daten können unmittelbar genutzt werden, um Aufgaben zu klassifizieren und somit in einem E-Learning-System (teil-)automatisiert zur Bearbeitung vorzuschlagen. Im einfachsten Fall könnten Studierende dabei ein System explizit nach leichteren oder schwierigeren Aufgaben fragen. Daraus ergibt sich auch ein unmittelbarer Forschungsansatz, indem die Studierenden anschließend um eine (subjektive) Einschätzung der Schwierigkeit gebeten werden. Kennzahlen, Ergebnissen und subjektive Bewertungen können dann zur weiteren Validierung der Erkenntnisse dieses Artikels miteinander verglichen werden. Auch die oben diskutierten weiteren Einflussfaktoren sollten in diesen Betrachtungen Berücksichtigung finden.

Ferner kann untersucht werden, ob aus der Position einer einzelnen Lösung in Relation zum Verteilungsdiagramm der jeweiligen Aufgabe weitere Erkenntnisse, beispielsweise zur Generierung individueller Rückmeldungen an die Studierenden, gewonnen werden können. Über den Einsatz von Softwareproduktmetriken hinaus können dabei auch weitere Metriken zum Entwicklungsprozess der Lösung zum Einsatz kommen, mit denen beispielsweise eine Folge von mehreren Einreichungen mit inkrementellen Verbesserungen der Lösung untersucht wird. Dies würde auch die Bereiche der Analyse abdecken, die in diesem Artikel nicht besprochen wurden.

Danksagung Die Autoren danken Alexander Jung für seine umfangreiche Recherche zu Softwareproduktmetriken im Rahmen seiner Bachelor-Arbeit.

Literatur

[e Abreu u. Carapuça 1994] ABREU, Fernando B. ; CARAPUÇA, Rogério: *Object-Oriented Software Engineering: Measuring and Controlling the Development Process*. 1994

[Chidamber u. Kemerer 1994] CHIDAMBER, Shyam R. ; KEMERER, Chris F.: A metrics suite for object oriented design. In: *IEEE Transactions on Software Engineering* 20 (1994), jun, Nr. 6, S. 476 –493. http://dx.doi.org/10.1109/32.295895. – DOI 10.1109/32.295895. – ISSN 0098–5589

[Conte u. a. 1986] CONTE, Samuel D. ; DUNSMORE, Hubert E. ; SHEN, Vincent Y.: *Software engineering metrics and models*. Redwood City, CA, USA : Benjamin-Cummings Publishing Co., Inc., 1986. – ISBN 0–8053–2162–4

[DeMarco 1986] DEMARCO, Tom: *Controlling Software Projects: Management, Measurement, and Estimates*. Upper Saddle River, NJ, USA : Prentice Hall PTR, 1986. – ISBN 0131717111

[Fenton u. Pfleeger 1998] FENTON, Norman E. ; PFLEEGER, Shari L.: *Software Metrics: A Rigorous and Practical Approach*. 2nd. Boston, MA, USA : PWS Publishing Co., 1998. – ISBN 0534954251

[Gross u. a. 2012] GROSS, Sebastian ; MOKBEL, Bassam ; HAMMER, Barbara ; PINKWART, Niels: Feedback Provision Strategies in Intelligent Tutoring Systems Based on Clustered Solution Spaces. In: DESEL, Jörg (Hrsg.) ; HAAKE, Joerg M. (Hrsg.) ; SPANNAGEL, Christian (Hrsg.): *DeLFI 2012: Die 10. e-Learning Fachtagung Informatik*. Hagen, Germany, 2012. – ISBN 978–3885796015, S. 27–38

[Halstead 1977] HALSTEAD, Maurice H.: *Elements of software science*. Elsevier, 1977

[Kaner u. Bond 2004] KANER, Cem ; BOND, Walter P.: Software Engineering Metrics: What Do They Measure and How Do We Know? In: *In METRICS 2004. IEEE CS*, Press, 2004

[Leach 1995] LEACH, Ronald J.: Using metrics to evaluate student programs. In: *SIGCSE Bull.* 27 (1995), S. 41–43. http://dx.doi.org/10.1145/201998.202010. – DOI 10.1145/201998.202010. – ISSN 0097–8418

[Martín u. a. 2009] MARTÍN, David ; CORCHADO, Emilio ; MARTICORENA, Raúl: A Code-comparison of Student Assignments based on Neural Visualisation Models. In: CORDEIRO, José A. M. (Hrsg.) ; SHISHKOV, Boris (Hrsg.) ; VERBRAECK, Alexander (Hrsg.) ; HELFERT, Markus (Hrsg.) ; INSTICC (Veranst.): *Proceedings of the First International Conference on Computer Supported Eductation (CSEDU), 23 - 26 March 2009, Lisboa, Portugal* Bd. 1 INSTICC, INSTICC Press, 2009. – ISBN 978–989–8111–82–1, S. 47–54

[McCabe 1976] MCCABE, Thomas J.: A Complexity Measure. In: *IEEE Transactions on Software Engineering,* 2 (1976), Nr. 4, S. 308–320. http://dx.doi.org/10.1109/TSE.1976.233837. – DOI 10.1109/TSE.1976.233837. – ISSN 0098–5589

[Mengel u. Yerramilli 1999] MENGEL, Susan A. ; YERRAMILLI, Vinay: A case study of the static analysis of the quality of novice student programs. In: *The proceedings of the thirtieth SIGCSE technical symposium on Computer science education*. New York, NY, USA : ACM, 1999 (SIGCSE '99). – ISBN 1–58113–085–6, S. 78–82

[Striewe u. a. 2009] STRIEWE, Michael ; BALZ, Moritz ; GOEDICKE, Michael: A Flexible and Modular Software Architecture for Computer Aided Assessments and Automated Marking. In: *Proceedings of the First International Conference on Computer Supported Eductation (CSEDU), 23 - 26 March 2009, Lisboa, Portugal* Bd. 2 INSTICC, 2009, S. 54–61