Quantitative Analyse studentischer Projekte

Tilmann Hampp
Abteilung Software Engineering, Institut für Softwaretechnologie
Universität Stuttgart
www.iste.uni-stuttgart.de/se

1 Einführung und Motivation

Im Studiengang Softwaretechnik an der Universität Stuttgart müssen alle Studierenden im Hauptdiplom an zwei Studienprojekten teilnehmen. Das erste Studienprojekt (Studienprojekt A) wird von den Informatikinstituten ausgegeben, das zweite Studienprojekt (B) wird im Anwendungsfach von Instituten anderer Fakultäten durchgeführt [Stu05].

Die Teilnehmer an einem Studienprojekt sollen die wesentlichen Aktivitäten von Softwareprojekten selbstständig planen und durchführen [Lud01]. Neben allen Tätigkeiten zur Entwicklung mit Vorprojekt, Spezifikation, Entwurf, Implementierung, Integration, Test und Auslieferung gehören dazu auch Projektplanung und -kontrolle, Qualitätssicherung und weitere projektbegleitende Maßnahmen. Unterstützend stehen ihnen Betreuer der ausgebenden Abteilung zur Seite. Ein Teilnehmer übernimmt die Rolle des Projektleiters. Die Kundenrolle wird von einem Mitarbeiter der Abteilung übernommen, aber auch von externen Kunden aus der Industrie [Ham05].

An jedem Studienprojekt nehmen 6 bis 12 Studierende teil. Die Studienordnung [Stu05] nennt für Studienprojekte eine Dauer von zwei Semestern und einen Aufwand von 16 SWS (Semesterwochenstunden) pro Teilnehmer. Davon entfallen 6 SWS auf konventionelle Lehrveranstaltungen. Die Projektarbeit mit 10 SWS entspricht 400 Eh (Entwicklerstunden) pro Teilnehmer.

Bislang wurden Metriken in den Studienprojekten von den Teilnehmern erhoben und anekdotisch in den Abschlusspräsentationen gezeigt. Da diese Daten aber nicht zentral gesammelt wurden, war ein Vergleich zwischen mehreren Projekten nicht möglich. Eine erste systematische Analyse des Aufwands und des Umfangs wurde für die Studienprojekte der Abteilung Programmiersprachen durchgeführt [Sim05].

In einer Studienarbeit wurden umfangreich Metriken aus abgeschlossenen Studienprojekten A erhoben [Thu05]. Die Ergebnisse der Arbeit und die Erfahrungen mit der Metrikerhebung und -analyse werden in diesem Artikel vorgestellt. Die Ziele dieser Erhebung waren:

Unterstützung der Kostenschätzung. Die Aufgabenstellung wird von den Betreuern und Kunden auf die Rahmenbedingungen des Studienprojekts zugeschnitten, der Umfang der Anforderungen wird aber im Vorprojekt und während der Spezifikation endgültig geklärt. Für das Angebot im Vorprojekt werden die Teilnehmer mit den Schwierigkeiten der Kostenschätzung konfrontiert, weil sie alle Anforderungen des Kunden realisieren sollen, der mögliche Aufwand aber begrenzt ist. Für die Planung müssen einzelne Phasen und Arbeitspakete definiert und mit Aufwand und Dauer geplant werden. Während in der Praxis überwiegend Expertenschätzungen durchgeführt werden [Jør04], können die Studienprojektteilnehmer nicht auf eigene Erfahrungen mit Projekten dieser Größe zurückgreifen. Darum sollen die Teilnehmer durch Daten aus abgeschlossenen Studienprojekten unterstützt werden.

Vergleich mit Erfahrungswerten. Interessiert hat uns außerdem der Vergleich mit Daten aus der Industrie: Weichen Produktivität und Dauer der Projekte deutlich von Daten aus typischen Industrieprojekten ab? Verteilen sich die Aufwände für einzelne Aktivitäten ähnlich, oder gibt es deutliche Abweichungen?

Um diese Ziele zu erreichen, wurden in der Studienarbeit zuerst relevante Metriken identifiziert (Abschnitt 2) und dann nachträglich die Metrikwerte erhoben (Abschnitt 3). Diese Metriken wurden für die Kostenschätzung, Phasenverteilung und Termineinhaltung analysiert (Abschnitt 4). Folgerungen aus den erhobenen Daten und aus den bei der Analyse aufgetretenen Problemen beschreibt der Abschnitt 5.

2 Identifikation relevanter Metriken

Die relevanten Metriken leiten sich aus den oben genannten Zielen ab. Für die Kostenschätzung sind Daten zu Dauer und Aufwand des gesamten Projekts und einzelner Aktivitäten notwendig.

In den Vorlesungen des Studiengangs Softwaretechnik werden die Kostenschätzverfahren Cocomo [Boe81], Cocomo II [Boe00] und Function Points [Gar00, IFP94] vorgestellt. Cocomo basiert auf Delivered Source Instructions, die als ausgelieferte Zei-

len Code definiert sind. Diese Metrik wird im Folgenden als Lines of Code (LOC) bezeichnet. Cocomo II basiert auf Delivered Source Instructions, die als logische Source Statements definiert sind, im Folgenden als Statements (Stmts) bezeichnet. Für das Function-Point-Verfahren [Gar00, IFP94] werden Aufwand und Function Points aus abgeschlossenen Projekten verwendet. Alle Kostenschätzverfahren benötigen zusätzlich spezielle Parameter.

Für den Vergleich zwischen den Projekten wird die Programmiersprache, die Anzahl der Teilnehmer und das Vorgehen im Projekt benötigt. Unterschieden wird zwischen Vorgehen nach dem Wasserfallmodell und iterativem Vorgehen.

Ergänzt werden diese Metriken um den Umfang der Dokumente in Seiten, den Umfang des Codes als Anzahl Klassen oder Module und Anzahl Routinen (Methoden, Prozeduren, Funktionen). Die Termineinhaltung wird anhand der Vorgaben aus dem ersten Projektplan bewertet.

Zur Erhebung und Analyse wurden diese Metriken nach den wesentlichen Aktivitäten in Studienprojekten gegliedert: Den Schritten der Entwicklung (Vorprojekt, Spezifikation, Entwurf, Implementierung und Nachbearbeitung), den Qualitätssicherungsmaßnahmen (Reviews, Tests, projektumspannende Maßnahmen wie Schulungen und Configuration Management) und dem Projektmanagement. Jeder Kategorie werden Metriken zur Phase in einer bestimmten Iteration, zu einzelnen Aktivitäten und zum entstandenen Dokument zugeordnet. Diese Gliederung bildet die Grundlage für die ER-Modellierung der Datenbasis, um die Metriken zu speichern.

3 Nachträgliche Erhebung der Metriken

Zur Erhebung dieser Metriken stellten die Institute, die Studienprojekte ausgegeben hatten, die archivierten Dokumente zur Verfügung. 22 Studienprojekte aus den Jahren 1998 bis 2004 konnten analysiert werden. Wesentliche Dokumente für die Erhebung der Metriken waren primär der Abschlussbericht und die Abschlusspräsentation, ergänzend der Projektplan und die Angebote als Ergebnis der Vorprojekte. Zusätzlich wurden Stundenzettel und Protokolle ausgewertet. Produktmetriken zum Umfang konnten an den Dokumenten (Spezifikation, Entwurf, Handbuch) und am Code gemessen werden. Für die Messung der Lines of Code und der Statements wurde das Werkzeug CodeCount [USC98] verwendet.

Problematisch waren unvollständige, ungenaue und widersprüchliche Daten.

Unvollständige Daten: Dokumente waren nicht mehr auffindbar, weil die Betreuer nicht mehr am Institut beschäftigt waren und Teilnehmer des Studienprojekts ihr Studium abgeschlossen hatten. Teilweise wurden relevante Metriken nicht dokumentiert. Häufig fehlten detaillierte Daten zu Reviews und Tests. Daten zu einzelnen Aktivitäten, beispielsweise Grobentwurf und Feinentwurf, wurden nicht getrennt erhoben oder dokumentiert. Erstaunlicherweise fehlen in einigen Projekten selbst Dauer und Aufwand einzelner Projektphasen. Die nachträgliche Erhebung der Function Points konnte nur erfolgen, wenn die Spezifikation verfügbar war. Sie war schwierig für Systeme mit wenigen Schnittstellen nach außen, aber komplexer Verarbeitung von Daten.

Ungenaue Daten: Ungenauigkeiten entstanden bei der Erhebung, wenn die Daten nur als Schaubild dokumentiert oder Aussagen unpräzise formuliert waren, so dass geschätzte Daten abgeleitet werden mussten. Diese geschätzten Daten wurden in der Datenbasis entsprechend kommentiert.

Widerspüchliche Daten: Bei der Analyse wurden widersprüchliche Angaben in den Dokumenten entdeckt, beispielsweise wurde von unterschiedlichen Teilnehmern in einem Studienprojekt der Integrationsaufwand der Implementierung oder dem Test zugeordnet. Unklare Begriffsdefinitionen und -verwendungen erschweren den Vergleich zwischen Studienprojekten: Als Aufwand für Projektmanagement wurde teilweise nur der Aufwand für die Planung verbucht, in anderen Projekten auch der Aufwand für die Abnahme oder den Abschlussbericht. Für eine konsistente Erfassung wurden darum die Tätigkeiten des Projektleiters betrachtet. Ähnliche Schwierigkeiten zeigen sich bei der Aufwandserhebung für die Qualitätssicherung.

Insgesamt sind alle erhobenen Daten mit erheblicher Ungenauigkeit belastet. Unklar ist bei einer nachträglichen Erhebung auch, wie weit die dokumentierten Ergebnisse der Realität entsprechen oder geschönt wurden, um das Projekt möglichst erfolgreich abzuschliessen. Ich halte die erhobenen Metriken trotz dieser Schwächen für nützlich, weil sie als Anhaltspunkt für die Kostenschätzung und Planung dienen können. Eine gründliche Analyse zur Kostenschätzung muss für ein neues Projekt auf jeden Fall durchgeführt werden, die historischen Daten erlauben aber, diese Kostenschätzung auf Plausibilität zu prüfen. Eine bewusste Verfälschung halte ich für unwahrscheinlich, da ein Studienprojekt nicht anhand dieser Metriken bewertet wird.

4 Ergebnisse der Metrikenanalyse

4.1 Studienprojekte im Überblick

Analysiert werden konnten Daten aus 21 der 22 untersuchten Projekte. Ein Projekt konnte nicht berücksichtigt werden, weil keine Dokumentation zur Verfügung stand. Der Aufwand eines Studienprojekts beträgt im Median 4070 Eh (Entwicklerstunden), also

etwa 2 Entwicklerjahre. Der Aufwand pro Teilnehmer liegt mit durchschnittlich 488 Eh (Median: 460 Eh) um rund 20 % über dem in der Prüfungsordnung veranschlagten Aufwand. Die Dauer liegt bei etwa 11 Monaten vom ersten Termin bis zur Abnahme (Median: 321 Tage, Durchschnitt: 327 Tage).

Abbildung 1 zeigt diese Metriken als Boxplot. Die Box ist begrenzt durch das erste und dritte Quartil. Ausreißer weichen um mehr als den 1,5-fachen Quartilsabstand von diesen Grenzen ab und sind als Punkt oder Stern dargestellt. In drei Studienprojekten wurde von den Teilnehmern erheblich mehr Aufwand als verlangt geleistet, in einem Projekt mehr als das Doppelte des in der Studienordnung veranschlagten Aufwands von 400 Eh.

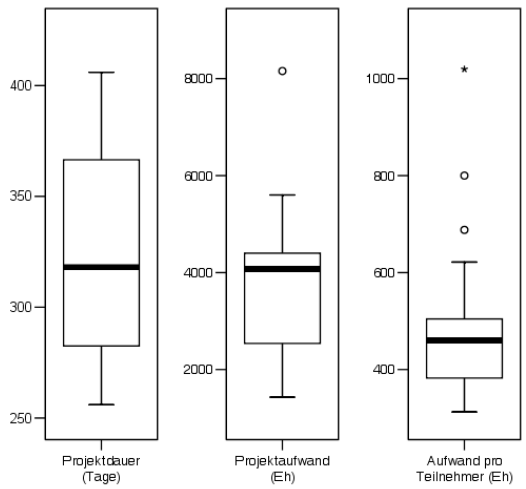

Abbildung 1: Dauer, Aufwand, Aufwand pro Teilnehmer

In den Studienprojekten wurden fünf Programmiersprachen verwendet (Abb. 2). In einigen Projekten wurden mehrere Sprachen kombiniert: Zwei Projekte verwendeten Ada95 und Tcl/tk, zwei Projekte kombinierten Ada95 und C++.

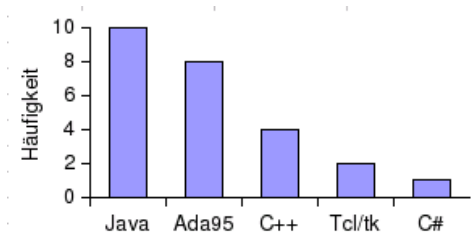

Abbildung 2: Häufigkeit verwendeter Programmiersprachen

4.2 Aufwandsverteilung

In 14 der 21 Projekte wurde der Aufwand getrennt für einzelne Phasen erhoben. Zur Berechnung der Aufwandsverteilung in Abbildung 3 wurden die Aufwände aus Phasen mit der gleichen Aktivität aus mehreren Iterationen summiert. Unter Sonstiges wurden Aufwände für Projektmanagement, Configurationmanagement und begleitende Maßnahmen eingeordnet, falls sie getrennt erhoben werden konnten. Die Integration wurde der Implementierung zugeordnet.

Phase	Median	Mittelwert	Minimum	Maximum
Vorprojekt	11 %	11 %	4 %	19 %
Spezifikation	16 %	16 %	9 %	29 %
Entwurf	15 %	15 %	9 %	40 %
Implementierung	39 %	42 %	25 %	59 %
Test	8 %	10 %	4 %	19 %
Handbuch	4 %	3 %	0 %	7 %
Sonstiges	-	3 %	-	-

Abbildung 3: Aufwandsverteilung

Das Vorprojekt hat einen hohen Anteil am Gesamtaufwand, weil in dieser Phase das Studienprojekt in bis zu vier Gruppen geteilt wird, die das Vorprojekt getrennt durchführen und konkurrierend ein Angebot erstellen. Die Implementierung benötigt mit rund 42 % im Mittel den größten Anteil, enthält aber auch zusätzliche Aufwände: Falls die Integration getrennt erhoben wurde, benötigt sie im Mittel 9 % des Gesamtaufwands. Enthalten ist teilweise auch Testaufwand: Der Aufwand für Modultest wurde nur in vier Projekten getrennt erhoben, zwei Projekte ordneten alle Aufwände für Test und Integration dem Implementierungsaufwand zu, der dann rund 50 % des Gesamtaufwands ausmacht.

Der Vergleich mit Erfahrungswerten zeigt ähnliche Aufwandsverteilungen mit leichten Abweichungen: Cocomo [Boe81] schätzt in organic- und medium-size-Projekten rund 33 % für Planung, Spezifikation und Entwurf, 36 % für Implementierung und Modultest und 22 % für Integration und Test, bezogen auf den Gesamtaufwand. Jalote [Jal00] zählt Projektmanagement und begleitende Maßnahmen getrennt und nennt 20 % für Aufwände vor Implementierung, 38 % für Implementierung und Modultest und 14 % für Aufwände nach Implementierung einschließlich Integration. In beiden Verteilungen ist der Anteil der frühen Phasen niedriger als in den Studienprojekten, selbst wenn das Vorprojekt nicht berücksichtigt wird. Der Anteil der späten Phasen ist etwas höher.

Ein möglicher Grund für die Abweichungen in den späten Phasen ist die unterschiedliche und uneinheitliche Erhebung von Integration und Modultest. In den von Jalote genannten Projekten ist die Prozessreife hoch. Der Korrekturaufwand ist niedrig, so dass der geringe Anteil der frühen und späten Phasen erklärt werden kann.

4.3 Termineinhaltung

Daten zur Termineinhaltung wurden durch Vergleich des tatsächlichen und des im ersten Plan dokumentierten Abgabedatums gemessen. Diese Daten standen in 19 Studienprojekten zur Verfügung. Der Median beträgt zwischen zwei und drei Wochen. Ein Ausreißer in den Daten zeigt eine Verzögerung von 150 Tagen; Ursache ist eine zu ehrgeizige Planung, nach der das Projekt innerhalb eines halben Jahres durchgeführt werden sollte. 7 Studienprojekte beendeten das Projekt zum geplanten Abgabetermin oder früher.

Hinweise auf die Ursache dieser Verzögerungen gibt die Termineinhaltung einzelner Phasen. Um die Verzögerung einer Phase in iterativen Projekten darzustellen und mit anderen Projekten zu vergleichen, wurde der Mittelwert der jeweiligen Phasenverzögerung in allen Iterationen gebildet. Abbildung 4 stellt die aus diesen Werten berechnete zusätzliche Verzögerung jeder Phase dar.

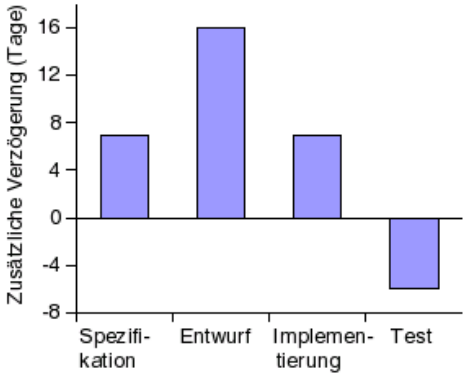

Abbildung 4: Verzögerung über die Projektphasen

Der Entwurf verursacht die größte Verzögerung in den Projekten. Zum Teil wird diese Verzögerung im Test wieder aufgeholt. Diese Verkürzung des Tests bietet eine mögliche Erklärung für den geringen Testaufwand: Um den Abgabetermin zu halten, wird der Test vernachlässigt.

4.4 Umfang der Produkte

Für die Messung des Umfangs wurden Lines of Code (LOC), Statements (Stmts), Unadjusted Function Points (UFP) und Adjusted Function Points (AFP) erhoben (Abb. 5). Function Points konnten für 14 Studienprojekte, Statements für 15 Studienprojekte und Lines of Code für alle betrachteten 21 Studienprojekte erhoben werden. Für Statements und Lines of Code wurde in Abbildung 5 ein Projekt nicht berücksichtigt, weil in diesem Projekt Code automatisch generiert wurde, der nicht von manuell erstelltem Code getrennt erhoben werden konnte.

Der Umfang der Studienprojekte ist vergleichbar mit dem Umfang typischer kleiner und mittlerer Auftragsprojekte: Jones [Jon03] ordnet Projekte in Klassen mit 1, 10, 100, 1000, 10000 und 100000 Function Points ein. Ein Projekt wird der Klasse mit dem nächstliegenden Umfang zugeordnet. 42 % der untersuchten Auftragsprojekte werden der Klasse mit 100 Function Points zugeordnet; bis auf ein Studienprojekt fallen alle Projekte, deren Umfang in Function Points erhoben werden konnte, in diese Klasse. Das größte Studienprojekt mit 530 AFP wird der nächstgrößeren Klasse zugeordnet. Nach Jones fallen in diese Klasse 36 % der Projekte.

Umfangs-metrik	Median	Mittel-wert	Mini-mum	Maxi-mum
KLOC	18,9	23,1	9,1	47,5
KStmts	10,8	12,6	6,8	29,1
UFP	232	244	126	541
AFP	233	242	125	530

Abbildung 5: Produktumfang

4.5 Kostenschätzverfahren

Für die Kostenschätzung werden Function Points, Cocomo und Cocomo II betrachtet. Von allen folgenden Analysen ist ein Projekt mit generiertem Code, der nicht getrennt gezählt werden konnte, ausgeschlossen. Abbildung 6 zeigt die Zuordnung des Aufwands zu Adjusted Function Points. Pro realisiertem Function Point werden rund 16 Eh benötigt. Dieser Wert entspricht dem von Jones [Jon03] für Auftragsprojekte genannten Mittelwert von rund 17 Eh. Für Projekte in der 100-Function-Point-Klasse nennt Jones einen niedrigeren Mittelwert von rund 4 Eh pro Function Point, aber auch weniger Projektmitarbeiter.

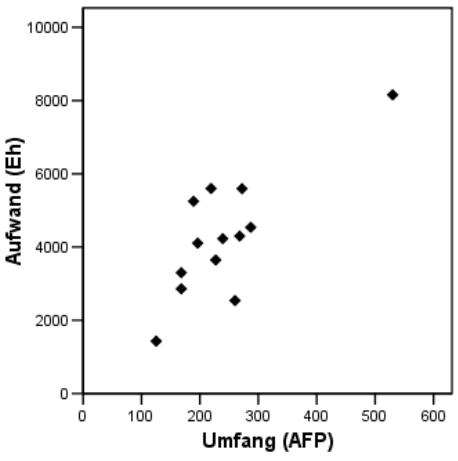

Abbildung 6: Aufwand und Umfang in AFP

Bei der Kostenschätzung mit Cocomo II wird zur Aufwandschätzung die folgende Formel verwendet:

$$Effort = A \cdot SIZE^{E} \cdot EM$$

Aufwand und Umfang in KStmts (1000 Statements) sind in Abbildung 7 dargestellt. Für 13 Studienprojekte wurden der Exponent E und die Einflussfaktoren EM individuell ermittelt. Da die Teilnehmer nur einen Teil ihrer Zeit am Projekt arbeiten, wurde ein Entwicklermonat als 160 Eh definiert. Für den Vergleich der Produktivität und zur Kalibrierung wurde der Produktivitätskoeffizient A individuell berechnet. Der Median liegt mit 2,1 deutlich unter dem Standardwert von 2,94 [Boe00]. Eine lineare Regression durch den Ursprung ergibt einen Produktivitätskoeffizienten von 1,8. Für eine getrennte Analyse nach den häufigsten Programmiersprachen in Studienprojekten konnten 6 Java-Projekte und 3 Ada95-Projekte ausgewertet werden. Die Projekte verwendeten jeweils nur eine Programmiersprache. Es zeigt sich kein deutlicher Unterschied: Für die Java-Projekte ist der Median des Produktivitätskoeffizienten 2,1, für Ada95-Projekte 2,2.

Obwohl die Teilnehmer unerfahren sind, ist die Produktivität höher als die in Cocomo II angegebene. Die Gründe können nur vermutet werden: Eine Rolle spielt möglicherweise, dass die Produkte in der Mehrzahl nicht für den externen Gebrauch bestimmt sind, so dass für Tests oder Benutzerdokumentation weniger Aufwand eingesetzt wird und dass auf wenige bestehende Systeme und Rahmenbedingungen Rücksicht genommen werden muss.

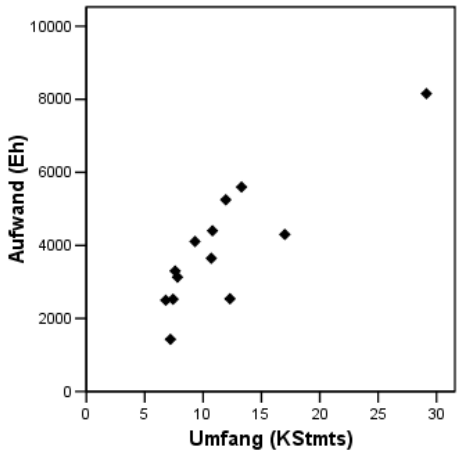

Abbildung 7: Aufwand und Umfang in Statements

Für Cocomo wurden 20 Studienprojekte für die Projektarten organic und semidetached mit den beiden folgenden Formeln analysiert:

Projektart	
organic:	$Effort = 2,4 \cdot SIZE^{1,05}$
semidetached:	$Effort = 3,0 \cdot SIZE^{1,12}$

Abbildung 8 zeigt Aufwand in Eh und Umfang in KLOC (1000 Lines of Code). Der Produktivitätskoeffizient weicht mit einem Median von 1,1 anstatt 2,4 für die Projektart organic und 0,8 anstatt 3,0 für die Projektart semidetached deutlich vom Standardwert ab. Die lineare Regression durch den Ursprung ergibt einen Koeffizienten von 0,8 für organic und 0,6 für semidetached.

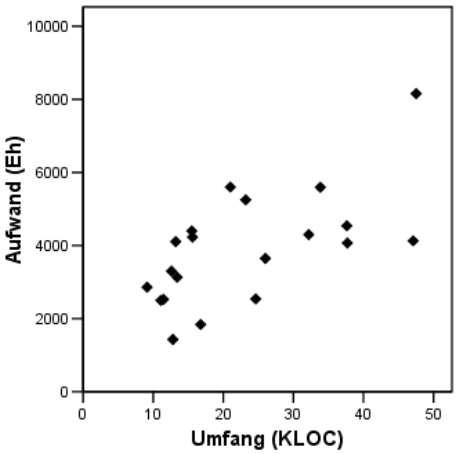

Abbildung 8: Aufwand und Umfang in Lines of Code

Der starke Unterschied zum Standardwert kann durch die Kalibrierung von Cocomo erklärt werden, die auf Projekten vor 1981 basiert. Die Abweichung zwischen dem Median des Koeffizienten und dem durch Regression geschätzten Wert für Cocomo und Cocomo II ist durch Ausreißer der Produktivität nach oben bestimmt.

Zu Beginn des Projekts den Umfang in Statements oder Lines of Code zu schätzen, ist für die Teilnehmer ähnlich schwierig, wie den Aufwand zu schätzen. Cocomo II erlaubt, aus Unadjusted Function Points die Anzahl der Statements zu schätzen und gibt dazu Umrechnungsfaktoren vor. Das Verhältnis zwischen Function Points und Statements konnte für 9 Projekte berechnet werden, unter denen 2 Ada95- und 6 Java-Projekte sind; ein Projekt wurde in C++ implementiert. Abbildung 9 zeigt den Median für Java und Ada95 im Vergleich zu den Angaben in [Boe00].

Statements pro Function Point für	Studienprojekte	[Boe00]
Java	58	53
Ada95	54	49

Abbildung 9: Statements pro Function Point

Zum Vergleich der unterschiedlichen Verfahren wurde der Aufwand dieser 9 Projekte nachgeschätzt. Zur Beurteilung dient der Korrelationskoeffizient r nach Pearson für den Zusammenhang zwischen dem tatsächlichen und dem geschätzten Aufwand. Verwendet wurde für Cocomo II der Median des Produktivitätskoeffizienten $A = 2,1$. Die Schätzung ba-

sierend auf Statements korreliert stark mit dem tatsächlichen Aufwand ($r = 0,92$, $r^2 = 0,84$). Werden Function Points mit den Werten aus [Boe00] auf Statements umgerechnet, ergibt sich mit Cocomo II ein ähnlich starker Zusammenhang ($r = 0,91$, $r^2 = 0,83$). Der direkte Zusammenhang zwischen Function Points und Aufwand mit einer Produktivität von 16 Eh pro AFP ist weniger stark ($r = 0.79$, $r^2 = 0.62$).

5 Folgerungen

Die Daten zeigen, dass Umfang und Aufwand der Studienprojekte mit typischen Auftragsprojekten aus der Industrie vergleichbar sind.

Die erhobenen Daten geben Anhaltspunkte für die Kostenschätzung. Die Kalibrierung der Verfahren verbessert meiner Meinung nach die Aussagekraft der Kostenschätzverfahren: Bislang war unklar, ob und wie genau sich mit den algorithmischen Verfahren der tatsächliche Aufwand abschätzen ließ. Erfahrungen bei der Verwendung liegen aber noch nicht vor. Das Projekt mit dem höchsten Aufwand (1020 Eh pro Teilnehmer) zeigt, dass historische Daten hilfreich sein können: Es hat den größten Umfang (530 AFP und rund 29000 Statements). Dass dieser Umfang nicht im Rahmen der Vorgabe zu schaffen ist, hätte nicht nur erkannt, sondern auch belegt werden können. Die Kostenschätzung bleibt trotzdem schwierig, selbst wenn das Function-Point-Verfahren angewendet werden kann, weil dazu die Anforderungen detailliert vorliegen müssen. Die Analyse der Daten mit Cocomo II zeigt zwar eine hohe Korrelation, erklärt aber nicht alle Einflüsse; eine Prognose wird ungenau sein.

Die Analyse der Studienprojekte zeigt Schwierigkeiten bei der Erhebung und Zuordnung des Aufwands in den Projekten. Dadurch wird der Vergleich mit anderen Projekten erschwert. Auch diese Schwierigkeiten sind typisch für Industrieprojekte: Jones [Jon03] nennt unvollständige Daten als typisches Problem. Standardkategorien für die Aufwandserhebung sind notwendig, um vergleichbare Daten für die Analyse der Aufwandsverteilung und Qualitätskosten zu erheben [Jal00]. Darum sollen als Fortsetzung dieser Arbeit Kategorien für den Aufwand in Studienprojekten standardisiert und Werkzeuge zur Erfassung zur Verfügung gestellt werden.

Nicht erhoben wurden in dieser Arbeit Metriken zur Qualität, beispielsweise Fehlerzahlen. Diese wurden in den Projekten nicht erfasst, Reviews, Tests und die Qualität des ausgelieferten Produkts können somit nicht quantifiziert werden. Unklar ist darum, ob die hohe Produktivität zu Lasten der Produktqualität geht. Unsere Erfahrung zeigt aber, dass die Produktqualität gut, teilweise sogar sehr gut ist.

Literatur

[Boe81] Boehm, B. W.: *Software Engineering Economics.* Prentice Hall, 1981.

[Boe00] Boehm, B. W. et al.: *Software cost estimation with Cocomo II.* Prentice Hall, 2000.

[Gar00] Garmus, D.; Herron, D: *Function Point Analysis.* Addison-Wesley information technology series, 2000.

[Ham05] Hampp, T.; Opferkuch, S.; Schmidberger, R.: Projekte der Lehre mit hochschulexternen Kunden. *Software Engineering im Unterricht der Hochschulen (SEUH 9)*, dpunkt.verlag, 2005.

[IFP94] *Function Point Counting Practices Manual.* Release 4.0, International Function Point Users Group (IFPUG), 1994.

[Jal00] Jalote, P.: *CMM in Practice: Processes for executing software projects at Infosys.* Addison-Wesley, 2000.

[Jon03] Jones, C.: *Software Assessments, Benchmarks, and Best Practices.* 2nd Ed., Addison-Wesley, 2003.

[Jør04] Jørgensen, M: A review of studies on expert estimation of software development effort. *Journal of Systems and Software 70(1-2)*, pp. 37-60, 2004.

[Lud01] Ludewig, J. (Hrsg): *Praktische Lehrveranstaltungen im Studiengang Softwaretechnik: Programmierkurs, Software-Praktikum, Studienprojekte, Fachstudie.* Mit Beiträgen von S. Krauß, J. Ludewig, P. Mandl-Striegnitz, R. Melchisedech, R. Reißing. Bericht der Fakultät Informatik, Universität Stuttgart, 3. Auflage, 2001.

[Sim05] Simon, D.; Vogel, G.; Plödereder, E.: Teaching Software Engineering with Ada95. *10th International Conference on Reliable Software Technologies: Ada-Europe 2005*, 2005.

[Stu05] Studienkommission Informatik und Softwaretechnik: *Studienplan und Prüfungsordnung, Diplomstudiengang Informatik, Diplomstudiengang Softwaretechnik.* Fakultät Informatik, Elektrotechnik und Informationstechnologie, Universität Stuttgart, 2005.

[Thu05] Thumm, A.: *Quantitative Analyse von Studienprojekten.* Studienarbeit, Institut für Softwaretechnologie, Universität Stuttgart, 2005.

[USC98] *CodeCount.* University of Southern California Center for Software Engineering, 1998. http://sunset.usc.edu/research/CODECOUNT/