EIN ANWENDUNGSBEISPIEL FÜR EIN NEUES MODELL ZUR AUFWANDSCHÄTZUNG

Frank Padberg*

1 Einleitung

Stellen Sie sich vor, Sie sind Manager in einer Softwarefirma. Bei Ihrem letzten Projekt haben Sie erstmals objektorientierte Programmierung eingesetzt. Dadurch ist die durchschnittliche Produktivität Ihrer Entwickler im Vergleich zu vorangegangenen Projekten um zehn Prozent gestiegen. Können Sie davon ausgehen, daß Ihr nächstes Projekt um zehn Prozent schneller abgewickelt sein wird als ursprünglich geplant, wenn Sie erneut objektorientierte Programmierung einsetzen?

Die Frage ist schwierig. Wie sich objektorientiertes Programmieren auf die Produktivität der Entwickler auswirkt, ist umstritten. Es gibt nur sehr wenige empirische Arbeiten zu diesem Thema, etwa [2] [3] [6] [7] [11]. Auch ist nicht klar, wie sich Veränderungen der Produktivität der Entwickler auf die Gesamtdauer eines Projekts auswirken, denn die gängigen Modelle zur Aufwandschätzung sind unzuverlässig, siehe etwa [5] [9] [10].

In der vorliegenden Arbeit werden die Programmiersprachen C und Java verglichen. Dazu wird für jede der beiden Sprachen die Dauer eines fest gewählten Beispielprojekts geschätzt und die Schätzergebnisse verglichen. Die Projektdauer wird mit Hilfe eines neuen, wahrscheinlichkeitstheoretischen Modells für Softwareprojekte geschätzt. Dabei werden empirisch ermittelte Produktivitätsdaten für die beiden Sprachen als Eingangsdaten verwendet. Das wahrscheinlichkeitstheoretische Modell für Softwareprojekte ist in [12] ausführlich beschrieben. So gesehen ist das Ziel der vorliegenden Arbeit, mit empirischen Daten zu zeigen, wie das neue Modell eingesetzt wird.

Das betrachtete Beispielprojekt ist klein. Die empirischen Produktivitätsdaten stammen aus Programmierkursen für Informatik-Studenten im Hauptstudium. Die Datenerfassung in den Programmierkursen war nicht auf das neue Modell zugeschnitten. Insbesondere wurden nicht

---

* Fakultät für Informatik, Universität Karlsruhe, padberg@ira.uka.de
unterstützt durch ein Postdoktoranden-Stipendium der Deutschen Forschungsgemeinschaft DFG am Graduiertenkolleg Informatik der Uni Karlsruhe

alle für das Modell benötigten Daten aufgezeichnet, so daß sich die vorliegende Arbeit bisweilen mit Annahmen behelfen muß. Industrielle Projekte und die Produktivität industrieller Entwickler werden natürlich anders aussehen, daher zielt die Arbeit gar nicht erst darauf ab, die zahlenmäßigen Ergebnisse auf andere Situationen zu übertragen. Die *Methode* jedoch ist für größere Projekte mit industriellen Daten dieselbe, und um die Methode zu zeigen, genügt auch ein kleines Beispiel. Ferner lagen für Java weniger Datenpunkte vor als für C. Das Problem ungleich großer Datensätze tritt in der industriellen Praxis auch auf und sollte kein Grund sein, einen Vergleich nicht zu versuchen.

Nach einer kurzen Beschreibung des Modells wird auf folgende Fragen näher eingegangen:

- Wie sehen die benötigten empirischen Daten aus?
- Wie berechnet man aus den empirischen Daten die Parameter für das Modell?
- Wie berechnet man im Modell einen Schätzwert für die Dauer?
- Wie berechnet man im Modell das Risiko, daß eine festgesetzte Frist überschritten wird?
- Wie vergleicht man die Ergebnisse für verschiedene Entwicklungsmethoden, Programmiersprachen oder Werkzeuge?

Die letzten drei Fragen stellen sich, weil das neue Modell nicht einfach einen Schätzwert für die Dauer des Projekts liefert, sondern eine *Wahrscheinlichkeitsverteilung* für die Dauer.

Das hier verwendete neue Modell für Softwareprojekte zeichnet sich durch einige Eigenschaften aus, die andere Modelle nur teilweise bieten.

- Das Modell ist dynamisch. Die Dauer und die Kosten eines Projekts hängen unmittelbar von dem Verlauf ab, den das Projekt nimmt. Daher fußt das Modell auf einer Beschreibung des Projektverlaufs. Bestimmend für den Projektverlauf sind die Wechselwirkungen zwischen den Entwicklern.
- Das Modell ist wahrscheinlichkeitstheoretisch. Der Verlauf eines Projekts ist vorab nicht genau bestimmbar. Daher treten Ereignisse nur mit einer gewissen Wahrscheinlichkeit zu bestimmten Zeitpunkten auf.
- Der Architekturentwurf geht in das Modell ein. Die Architektur der zu entwickelnden Software hängt eng mit der Struktur und Dynamik des Projekts zusammen. Wichtig ist vor allem die Stärke der Kopplung zwischen den Komponenten.
- Für jeden Entwickler geht ein wahrscheinlichkeitstheoretisches, empirisch ermitteltes Produktivitätsprofil in das Modell ein, nicht nur eine Durchschnittsgröße für seine Produktivität. Das Profil hängt von verschiedenen Größen ab, etwa von der Funktionalität und dem Schwierigkeitsgrad der zu entwickelnden Komponente, oder auch von der gewählten Programmiersprache.

- Das Modell liefert als Ergebnis eine Wahrscheinlichkeitsverteilung für die Dauer des Projekts. Daraus läßt sich das Risiko berechnen, eine festgesetzte Frist zur Fertigstellung zu überschreiten.

Zwei Modelle, die teilweise in dieselbe Richtung gehen wie das hier verwendete Modell, finden sich bei Abdel-Hamid und Madnick [1] sowie Raffo und Kellner [13]. Das Modell von Abdel-Hamid und Madnick ist nicht wahrscheinlichkeitstheoretisch, und der Architekturentwurf geht nur über einige skalare Parameter in das Modell ein. Das Modell von Raffo und Kellner ist zwar wahrscheinlichkeitstheoretisch, aber auch hier geht der Architekturentwurf nur über einige skalare Parameter ein. Keines der beiden Modelle beschreibt die Wechselwirkungen zwischen den Entwicklern in einem Projekt.

Das neue Modell benötigt neben empirischen Produktivitätsdaten zumindest einen groben Architekturentwurf als Eingabe. Deshalb muß die Dauer der frühen Projektphasen, etwa die Dauer der Anforderungsanalyse, gesondert abgeschätzt werden.

Wie beeinflußt ein neues Werkzeug oder eine neue Entwicklungsmethode die Modellparameter? Da gibt es mehrere Möglichkeiten. Ein neues Werkzeug wird sich in erster Linie auf die wahrscheinlichkeitstheoretischen Produktivitätsprofile der Entwickler auswirken. Ist das Werkzeug hilfreich, dann werden Fehler schneller gefunden und die Komponenten schneller fertiggestellt. Wieviel das Werkzeug bringt, wird von Entwickler zu Entwickler und von Komponente zu Komponente unterschiedlich sein. Eine neue Entwicklungsmethode kann sich sowohl auf den Architekturentwurf als auch auf die Produktivitätsprofile auswirken. Beispielsweise führt objektorientiertes Programmieren zu einer anderen Kopplung zwischen den Komponenten und zu Komponenten mit anderem Schwierigkeitsgrad als prozedurales Programmieren.

Im neuen Modell wird der Effekt einer Methode oder eines Werkzeugs auf die Dauer eines Projekts meist *nicht* durch einen skalaren Faktor beschreibbar sein, da das Modell die Daten in hohem Maß nicht-linear miteinander verknüpft. Das könnte auch erklären, warum die gängigen Modelle zur Aufwandschätzung mit ihren skalaren „Kostentreibern“ trotz allen Kalibrierens immer wieder falsche Vorhersagen liefern. Die Überlegung führt zurück zu der Frage, die zu Beginn der Einleitung gestellt wurde: wie günstig sich eine neue Entwicklungsmethode auf die Dauer des nächsten Projekts auswirkt, kann im allgemeinen nicht so einfach vom letzten Projekt abgeleitet werden.

2 Modell

2.1 Projekte

An einem Softwareprojekt arbeiten mehrere Entwickler. Der Architekturentwurf teilt die Software in Komponenten auf. Jeder Entwickler bearbeitet eine Komponente. Jeder Entwickler arbeitet so lange, bis seine Komponente fertiggestellt ist. Die Entwickler arbeiten gleichzeitig, aber nicht unabhängig voneinander. Es kommt zu einer gegenseitigen Beeinflussung, wenn einer der Entwickler ein Problem auf der Ebene des Architekturentwurfs entdeckt, das sich möglicherweise auf mehrere Komponenten auswirkt. Dabei werden nur solche Probleme berücksichtigt, die sich nicht von vornherein innerhalb einer Komponente lösen lassen. Beispielsweise könnte es nötig sein, die Schnittstelle einer Komponente zu erweitern. Von der Änderung wären *alle* Komponenten betroffen, die diese Schnittstelle benutzen. Die entsprechenden Entwickler müßten dann ihre Komponente abändern. Der Fortschritt eines Entwicklers hängt somit vom Fortschritt der anderen Entwickler ab.

2.2 Zustände

Im Verlauf eines Projekts wird der Architekturentwurf immer wieder überarbeitet. Die Zeitspanne zwischen zwei Überarbeitungen heißt eine Phase. Der Verlauf ω des Projekts wird beschrieben als die Folge

$$\zeta(1),\ \ \zeta(2),\ \ \ldots$$

der Zustände des Projekts am Ende der Phasen. Der Zustand des Projekts am Ende einer Phase ist ein Vektor, der sich aus den Zuständen der einzelnen Komponenten zusammensetzt. Der Zustand einer Komponente ist definiert als die bisher für die Komponente benötigte Entwicklungszeit, wobei Mehrarbeit, die durch Änderungen an der Architektur der Software bedingt war, abgezogen wird. Das Modell arbeitet mit diskreter Zeit. Die Einheit für den Zeittakt kann frei gewählt werden. Beispielsweise könnte ein Takt einem Tag entsprechen.

2.3 Wahrscheinlichkeiten

Die Übergangswahrscheinlichkeit

$$\mathrm{P}_{\zeta}(d,\ \eta)$$

gibt an, mit welcher Wahrscheinlichkeit das Projekt ausgehend vom Zustand ζ nach einer d Takte langen Phase im Zustand η ist. Zur Berechnung der Übergangswahrscheinlichkeiten

werden empirische Daten über den Verlauf früherer Projekte sowie Daten über die Architektur der Software benötigt, siehe unten. Die entsprechenden Formeln sind in [12] angegeben. Sind alle Übergangswahrscheinlichkeiten bekannt, dann kann die Wahrscheinlichkeit $\mathrm{P}(\omega)$ dafür, daß das Projekt einen bestimmten Verlauf ω nimmt, durch Multiplizieren der entsprechenden Übergangswahrscheinlichkeiten berechnet werden. Das Modell ist also eine Markovkette.

Die Wahrscheinlichkeit $\varphi(x)$ dafür, daß das Projekt genau x Takte dauern wird, berechnet man, indem man die Wahrscheinlichkeiten $\mathrm{P}(\omega)$ aller Projektverläufe ω zusammenzählt, die genau x Takte dauern. Die Verteilung φ ist gerade die in der Einleitung angesprochene Wahrscheinlichkeitsverteilung, die das Modell als Ergebnis einer Berechnung liefert.

2.4 Empirische Daten

Zur Berechnung der Übergangswahrscheinlichkeiten werden empirische Daten über den Verlauf früherer Projekte benötigt. Im wesentlichen drücken diese Daten aus, wie schnell die Entwickler in früheren Projekten bei *vergleichbaren* Komponenten Fortschritte gemacht haben. Aus den empirischen Daten werden folgende Wahrscheinlichkeiten berechnet:

- die Wahrscheinlichkeit $\mathrm{P}(\mathrm{D}_k^i)$ dafür, daß Entwickler i nach genau k Takten fertig wird, sofern er nicht unterbrochen wird;
- die Wahrscheinlichkeit $\mathrm{P}(\mathrm{E}_k^i)$ dafür, daß Entwickler i nach genau k Takten ein Problem meldet, sofern er nicht unterbrochen wird;
- die Wahrscheinlichkeit $\mathrm{P}(\mathrm{R}_k^i)$ dafür, daß Entwickler i genau k Takte benötigt, um eine Änderung an der Architektur einzuarbeiten.

Die Wahrscheinlichkeiten bilden das Produktivitätsprofil eines Entwicklers. Sie hängen von der Erfahrung des Entwicklers ab, aber auch von der Funktionalität und dem Schwierigkeitsgrad der zu entwickelnden Komponente, sowie den eingesetzten Methoden, Werkzeugen und der Programmiersprache.

2.5 Architekturdaten

Zur Berechnung der Übergangswahrscheinlichen werden Daten über die Architektur der Software benötigt. Im wesentlichen drücken diese Daten aus, wie stark die Komponenten der Software miteinander gekoppelt sind. Je stärker die Komponenten gekoppelt sind, desto wahrscheinlicher ist es, daß sich Probleme mit dem Entwurf über mehrere Komponenten ausbreiten. Aus den Architekturdaten wird die Wahrscheinlichkeit

$$\alpha(K, X)$$

dafür berechnet, daß die Komponenten X überarbeitet werden müssen, wenn Probleme mit der Architektur in den Komponenten K auftreten. Dabei bezeichen K und X Mengen von Komponenten. Beispielsweise ist $\alpha(\{3\}, \{1, 2, 3\})$ die Wahrscheinlichkeit dafür, daß die ersten drei Komponenten überarbeitet werden müssen, wenn in der dritten Komponente ein Problem auftritt. Die Wahrscheinlichkeiten $\alpha(K, X)$ heißen Abhängigkeitsgrade.

3 Beispiel

Alle Wahrscheinlichkeiten sind in Prozent angegeben.

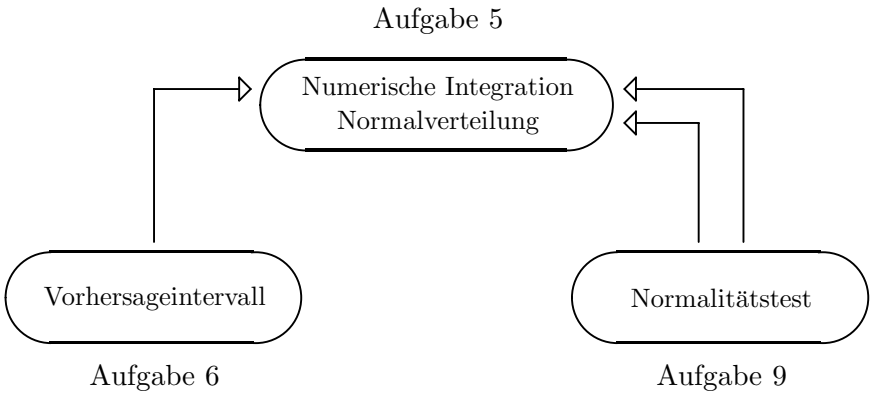
3.1 Datenbasis

Die empirischen Daten stammen aus Programmierkursen, die in den Jahren 1996 bis 1999 an der Universität Karlsruhe von Informatik-Studenten im Hauptstudium belegt wurden. Die Kurse waren am "Personal Software Process" [8] orientiert. Die Kurse hatten zum Ziel, den Studenten beim Erkennen ihrer typischen Fehler zu helfen. In jedem Kurs waren zehn Aufgaben zu bearbeiten. Die Studenten mußten zu jeder Aufgabe einige Formulare ausfüllen. Dabei mußten zwar auch Angaben zur benötigten Zeit für alle Phasen der Entwicklung von der Planung bis zum Testen gemacht werden, aber die Datenerfassung war nicht auf das in dieser Arbeit verwendete Modell zugeschnitten. Die Zeitangaben wurden nicht zur Scheinvergabe benutzt, daher kann man davon ausgehen, daß die Angaben nicht verfälscht wurden.

Die Studenten konnten zu Beginn des Kurses eine Programmiersprache wählen. Insgesamt 17 Studenten wählten C und insgesamt 9 Studenten Java. Für Java lagen also deutlich weniger Datenpunkte vor als für C. Das Problem ungleich großer Datensätze tritt bei industriellen Datensätzen auch auf, insbesondere, wenn neue Entwicklungsmethoden bewertet werden.

3.2 Komponenten

Die Programmieraufgaben bauen teilweise aufeinander auf, etwa die Aufgaben 5, 6 und 9. Aufgabe 5 implementiert die Simpson-Regel zur numerischen Integration von Funktionen und bietet ferner die Berechnung des Integrals der Normalverteilung an. Aufgabe 6 berechnet das sogenannte Vorhersageintervall zur linearen Regression und Aufgabe 9 implementiert den sogenannten χ^2–Test auf Normalität. Die genaue Funktionalität der Aufgaben 6 und 9 ist hier nicht so wichtig. Wichtig ist, daß Aufgabe 6 die numerische Integration von Aufgabe 5 benutzt, und Aufgabe 9 sowohl die numerische Integration als auch das Integral der Normalverteilung von Aufgabe 5. Die folgende Abbildung zeigt das Zusammenspiel der drei Aufgaben.



Die Aufgaben wurden von jedem Studenten nacheinander bearbeitet. Dabei kam es vor, daß ein Student Aufgabe 5 noch einmal überarbeiten mußte, während er an Aufgabe 6 arbeitete. Der Grund war, daß bei Aufgabe 6 (wie auch bei Aufgabe 9) nicht nur eine einzelne Funktion, sondern eine ganze Schar $(f_n)_{n=1,\,2,\,\ldots}$ verwandter Funktionen integriert werden mußte. Das Problem wurde von den Studenten entweder dadurch gelöst, daß der Parameter n in einer externen Variable oder der Klassendefinition der zu integrierenden Funktion versteckt wurde, oder die Schnittstelle zur numerischen Integration um den Parameter n erweitert und das Integrationsprogramm umgeschrieben wurde.

3.3 Projekt

Betrachtet man die Aufgaben 5, 6 und 9 als die drei Komponenten einer Software, dann entspricht das Erweitern der Schnittstelle von Aufgabe 5 einem Ändern der Architektur. Die Frage, die in dieser Arbeit als Beispiel untersucht werden soll, lautet nun:

> Wie lange würde ein Projekt dauern, bei dem die drei Komponenten gleichzeitig bearbeitet würden?

Die Frage soll mit Hilfe des neuen, wahrscheinlichkeitstheoretischen Modells sowohl für C als auch für Java untersucht und die Ergebnisse verglichen werden.

3.4 Zeittakt

Die Zeiten, die von den Studenten zum Bearbeiten der Aufgaben 5, 6 und 9 benötigt wurden, betragen zwischen 1 und 21 Stunden. Die in Stunden und Minuten gemessenen Zeiten werden zur Berechnung der Eingangsverteilungen zunächst in Takte umgerechnet. Wird die Einheit dabei zu klein gewählt, dann werden die Datenpunkte nicht genügend gebündelt. Die Folge wären Eingangsverteilungen mit vielen und großen Lücken, bei denen ein Balken häufig nur einem einzigen Datenpunkt entspricht. Andererseits will man nicht alle Datenpunkte zu einem einzigen Balken bündeln. Die geeignete Takteinheit muß durch Probieren gefunden werden.

Die Einheit für einen Takt muß für beide Programmiersprachen gleich gewählt werden, damit die Ergebnisse vergleichbar sind. Die geringe Anzahl der Datenpunkte für Java bestimmt also die Einheit. Durch Probieren wurde für das Beispielprojekt eine Einheit von 120 Minuten je Takt gefunden. Ferner werden die Zeiten vor der Umrechnung um bis zu 15 Minuten abgerundet. Eine Zeit von 4:12 entspricht dann 2 Takten, eine Zeit von 4:19 entspricht 3 Takten. Die erhaltenen Eingangsverteilungen (siehe die folgenden Abschnitte) zeigen noch gelegentlich eine Lücke oder einen Ausreißer, aber das ist für empirische Daten normal.

3.5 Wahrscheinlichkeit einer Problemmeldung

Die oben beschriebene Erweiterung der Schnittstelle von Aufgabe 5 war die einzige Änderung der Architektur, die während der Kurse an den drei betrachteten Aufgaben auftrat. Bei den C-Programmierern trat die Änderung in 5 von 17 Fällen auf, bei den Java-Programmierern in nur 1 von 9 Fällen. In Java war es also leichter möglich, das Problem „transparent“ zu lösen. Aus den Werten lassen sich trotz lückenhafter Zeitaufzeichnungen die Wahrscheinlichkeiten $\mathrm{P}(\mathrm{E}_k^i)$ für Problemmeldungen im Beispielprojekt wie folgt ansetzen.

Zunächst berechnet man die Wahrscheinlichkeiten

$$p_i \;=\; \sum_k \mathrm{P}(\mathrm{E}_k^i).$$

Aufgabe 5 benutzt keinerlei Funktionalität der beiden anderen Aufgaben. Daher wird für das Beispielprojekt angenommen, daß von Aufgabe 5 keine Probleme ausgehen werden. Also ist p_1 gleich Null. Aufgabe 6 und Aufgabe 9 nutzen die numerische Integration auf dieselbe Art – beide Male wird eine Schar von Funktionen integriert. Da die Aufgaben im Beispielprojekt gleichzeitig bearbeitet werden, kann eine Problemmeldung mit derselben Wahrscheinlichkeit von Aufgabe 6 wie von Aufgabe 9 ausgehen. Also ist p_2 gleich p_3. Die Wahrscheinlichkeiten p_2 und p_3 ergeben zusammen die empirisch festgestellte relative Häufigkeit r von Änderungen an Aufgabe 5. Nimmt man an, daß Probleme unabhängig voneinander entdeckt werden, erhält man die Gleichung $p_2 + p_3 - p_2 \cdot p_3 = r$. Löst man nach p_2 (oder p_3) auf, erhält man folgende Werte:

	C	Java
r	29.4	11.1
p_1	0	0
p_2	16.0	5.7
p_3	16.0	5.7

Als nächstes werden die Wahrscheinlichkeiten $\mathrm{P}(\mathrm{E}_k^i)$ für die Problemmeldungen aus den Wahrscheinlichkeiten p_i berechnet. Im Modell müssen dazu die Zeitpunkte bekannt sein, zu

denen in früheren Projekten Probleme mit dem Architekturentwurf entdeckt und behoben wurden. Die Aufzeichnungen der Studenten, wann sie wieviel Zeit in den einzelnen Entwicklungsphasen aufgewendet hatten, waren jedoch unvollständig. Daher konnten im nachhinein die genauen Zeitpunkte, zu denen während der Arbeit an Aufgabe 6 eine Änderung an Aufgabe 5 durchgeführt wurde, nicht mehr ermittelt werden. Als Behelf wird angenommen, daß im Beispielprojekt jeder Zeitpunkt zwischen dem Beginn und dem spätesten beobachteten Ende der Arbeit an Aufgabe 6 (bzw. Aufgabe 9) gleich wahrscheinlich ist. Das ergibt schließlich folgende Wahrscheinlichkeiten $\mathrm{P}(\mathrm{E}_k^i)$ für die Problemmeldungen im Beispielprojekt:

	C	Java
$\mathrm{P}(\mathrm{E}_k^1)$	0	0
$\mathrm{P}(\mathrm{E}_k^2)$	1.60	0.95
	$k = 1 \ldots 10$	$k = 1 \ldots 6$
$\mathrm{P}(\mathrm{E}_k^3)$	2.67	0.64
	$k = 1 \ldots 6$	$k = 1 \ldots 9$

Bei der Auswertung von Daten aus industriellen Projekten wäre es kaum nötig, Gleichverteilungen anzunehmen. Man würde die Zeitpunkte, zu denen in früheren Projekten Probleme mit dem Architekturentwurf entdeckt und behoben wurden, aus den Projektaufzeichnungen ermitteln und daraus die Wahrscheinlichkeiten $\mathrm{P}(\mathrm{E}_k^i)$ nach demselben Schema berechnen, das im folgenden Abschnitt für die Wahrscheinlichkeiten $\mathrm{P}(\mathrm{D}_k^i)$ gezeigt wird.

3.6 Wahrscheinlichkeit der Fertigstellung

Ein Student wird, sofern er nicht unterbrochen wird, nach einiger Zeit entweder seine Komponente fertigstellen oder ein Entwurfsproblem entdecken. Die Wahrscheinlichkeiten

$$q_i \;=\; \sum_k \mathrm{P}(\mathrm{D}_k^i)$$

ergeben sich daher aus der Gleichung $q_i = 1 - p_i$. Das ergibt folgende Werte:

	C	Java
q_1	100.0	100.0
q_2	84.0	94.3
q_3	84.0	94.3

Nimmt man die Zeitaufzeichnungen der Studenten dazu, lassen sich die Wahrscheinlichkeiten $\mathrm{P}(\mathrm{D}_k^i)$ der Fertigstellung wie folgt berechnen.

In der folgenden Tabelle sind für Aufgabe 5 die Zeiten angegeben, die von den einzelnen C-Programmierern zur Fertigstellung der Aufgabe benötigt wurden, sowie für jede Zeitangabe die Anzahl der Takte, denen das entspricht.

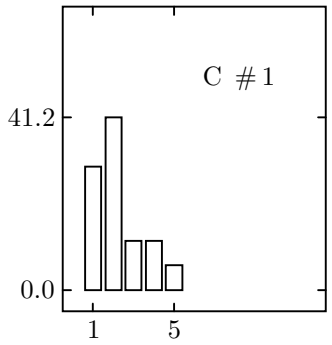
2:11	1	2:15	1	4:32	3	9:25	5	3:50	2	4:11	2
4:15	2	3:00	2	6:26	4	1:40	1	6:01	3	3:43	2
6:28	4	3:26	2	1:10	1	1:59	1	3:04	2		

Die Zeiten enthalten keine Mehrarbeit, da Aufgabe 5 erst während der Arbeit an Aufgabe 6 geändert wurde. Bezeichnet s_k^i die relative Häufigkeit, mit der für Komponente i genau k Takte benötigt wurden, dann ergibt sich die Wahrscheinlichkeit $\mathrm{P}(\mathrm{D}_k^i)$ aus der Formel

$$\mathrm{P}(\mathrm{D}_k^i) \;=\; q_i \cdot \mathrm{s}_k^i .$$

Beispielsweise hatten zwei der siebzehn Studenten vier Takte benötigt, also ist s_4^1 gleich $\frac{2}{17}$. Für die erste Komponente ergibt sich dann folgende Verteilung:

k	$\mathrm{P}(\mathrm{D}_k^1)$
1	29.41
2	41.17
3	11.77
4	11.77
5	5.88

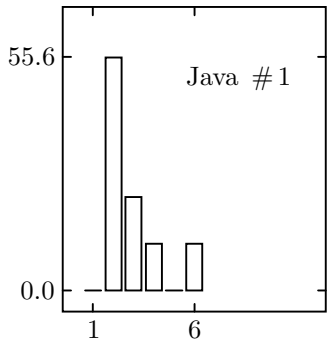


Die von den Java-Programmierern für die Aufgabe 5 erfaßten Zeiten sind:

3:59	10:45	2:56	7:46	2:51	5:54	3:08
2:55	5:53					

Das ergibt folgende Verteilung für Java:

k	$\mathrm{P}(\mathrm{D}_k^1)$
2	55.56
3	22.22
4	11.11
6	11.11



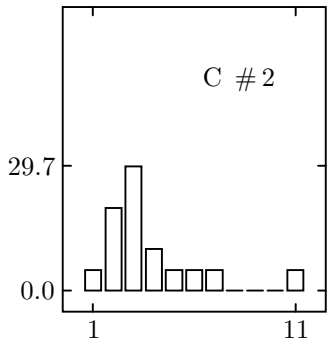
Einige Zeiten für Aufgabe 6 enthalten Mehrarbeit, da zwischendurch Aufgabe 5 überarbeitet wurde. Die Mehrarbeit muß von den Zeiten abgezogen werden, um die Wahrscheinlichkeiten $\mathrm{P}(\mathrm{D}_k^2)$ zu berechnen, aber die Studenten haben in ihren Formularen keine Angaben zur Mehrarbeit gemacht. Am Quelltext sieht man, daß der Änderungsaufwand eher gering war. Daher wird angenommen, daß zwischen 15 und 30 Minuten für die Änderung benötigt wurden.

Für Aufgabe 6 wurden von den C-Programmierern folgende Zeiten erfaßt:

∗8:30	1:41	4:48	20:25	∗4:57	9:15	10:45
4:43	12:54	3:59	∗7:40	∗5:29	∗5:16	6:04
2:45	2:23	4:15				

Die Zeiten, die Mehrarbeit enthalten, sind mit einem Sternchen gekennzeichnet. Das ergibt folgende Wahrscheinlichkeiten für die zweite Komponente:

k	$\mathrm{P}(\mathrm{D}_k^2)$
1	4.94
2	19.77
3	29.65
4	9.88
5	4.94
6	4.94
7	4.94
11	4.94

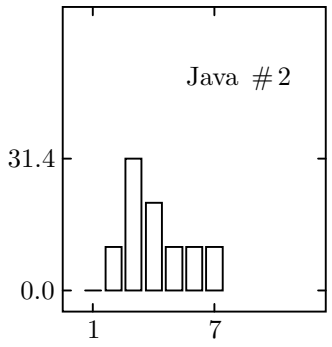


Die von den Java-Programmierern für die Aufgabe 6 erfaßten Zeiten sind:

4:31	7:28	8:57	12:43	3:33	8:06	∗6:17
6:10	12:09					

Das ergibt folgende Wahrscheinlichkeiten:

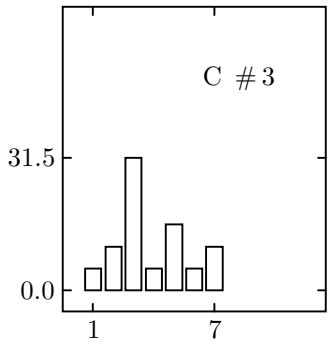
k	$\mathrm{P}(\mathrm{D}_k^2)$
2	10.48
3	31.43
4	20.95
5	10.48
6	10.48
7	10.48



Die Zeiten für Aufgabe 9 enthalten keine Mehrarbeit, da etwaige Änderungen an Aufgabe 5 schon während der Arbeit an Aufgabe 6 gemacht wurden. Die gemessenen Zeiten und daraus berechneten Wahrscheinlichkeiten für C sind:

4:21	6:32	13:42	5:58	14:03	9:33	5:09
11:31	3:50	9:14	4:44	10:09	5:17	2:11
3:36	5:03					

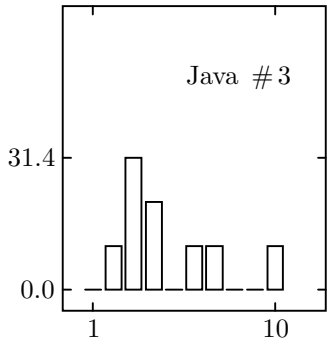
k	$\mathrm{P}(\mathrm{D}_k^3)$
1	5.24
2	10.50
3	31.50
4	5.24
5	15.76
6	5.24
7	10.50



Die gemessenen Zeiten und berechneten Wahrscheinlichkeiten für Java sind:

6:41	19:01	10:29	13:38	4:56	6:22	3:59
5:14	5:12					

k	$\mathrm{P}(\mathrm{D}_k^3)$
2	10.47
3	31.42
4	20.94
6	10.47
7	10.47
10	10.47



3.7 Wahrscheinlichkeit der Rückschritte

Der zeitliche Aufwand für die Änderung an Aufgabe 5 wurde nicht von den Studenten festgehalten, war aber dem Quelltext nach zu urteilen eher gering. Da die kleinste Einheit im Modell ein Takt ist, setzt man

$$\mathrm{P}(\mathrm{R}_1^i) \;=\; 100.0\,.$$

Im Beispielprojekt dauert ein Takt 120 Minuten, also wird der Einfluß der Mehrarbeit bei der Berechnung der Ergebnisse eher überschätzt.

3.8 Abhängigkeitsgrade

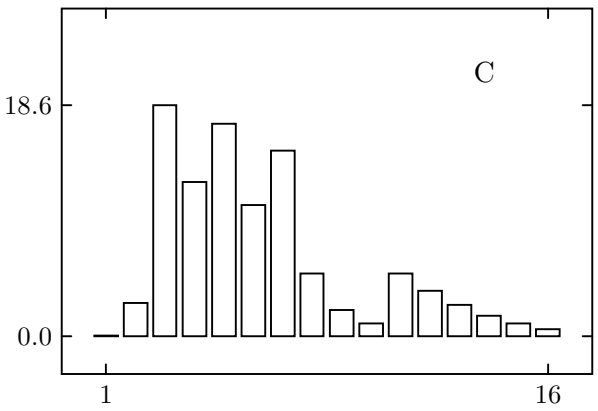
Die Abhängigkeitsgrade $\alpha(K, X)$ müssen aus dem Architekturentwurf einer Software abgeleitet werden. Für das Beispielprojekt ist das einfach, da es nur zwei Schnittstellen gibt, die beide von der ersten Komponente (Aufgabe 5) angeboten werden. Bezieht sich ein Problem auf die Schnittstelle zur numerischen Integration, dann sind sicher alle drei Komponenten davon betroffen. Bezieht sich ein Problem auf die Schnittstelle zum Integral der Normalverteilung, dann sind nur die erste und die dritte Komponente (Aufgaben 5 und 9) betroffen. Nimmt man an, daß die beiden Fälle gleich wahrscheinlich sind, erhält man folgende Tabelle für die Abhängigkeitsgrade.

	1	2	3	1,2	1,3	2,3	1,2,3
1	0	0	0	0	50	0	50
2	0	0	0	0	0	0	100
3	0	0	0	0	50	0	50
1,2	0	0	0	0	0	0	100
1,3	0	0	0	0	50	0	50
2,3	0	0	0	0	0	0	100
1,2,3	0	0	0	0	0	0	100

Die Tabelle enthält eine Zeile für jedes K und eine Spalte für jedes X.

3.9 Schätzungen

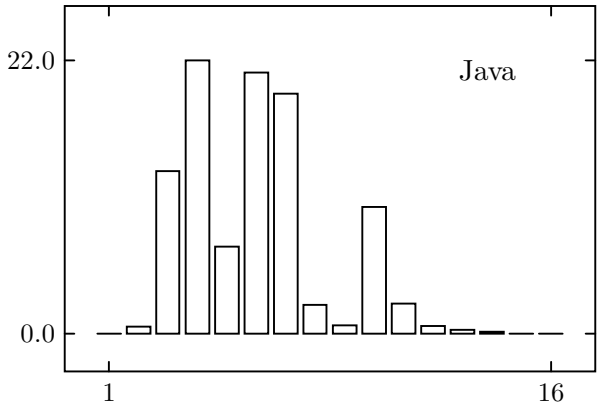
Ausgehend von den Verteilungen $\mathrm{P}(\mathrm{E}_k^i)$, $\mathrm{P}(\mathrm{D}_k^i)$ und $\mathrm{P}(\mathrm{R}_k^i)$ und den Abhängigkeitsgraden $\alpha(K, X)$ für die drei Komponenten des Beispielprojekts berechnet man nun mit Hilfe des Modells die Wahrscheinlichkeiten $\varphi(x)$ für die voraussichtliche Dauer des Beispielprojekts. Für C erhält man folgendes Diagramm der Verteilung φ:



Mit dem Diagramm läßt sich allerlei anfangen. Beispielsweise entspricht die Wahrscheinlichkeit dafür, das Projekt innerhalb von 12 Stunden abzuschließen, der Summe der ersten sechs Balken, also etwa 60 Prozent (ein Zeittakt ist 120 Minuten). Ferner wäre das Risiko, eine Frist von 12 Stunden zur Beendigung des Projekts zu überschreiten, etwa 40 Prozent.

Es gibt mehrere Möglichkeiten, aus dem Diagramm einen Schätzwert für die Dauer abzuleiten. Ein möglicher Schätzwert ist der Erwartungswert für das Diagramm, der hier 12.4 Stunden beträgt. Einen anderen Schätzwert erhält man, wenn man einen Risikowert vorgibt. Sei beispielsweise ein Risiko von 20 Prozent vorgegeben. Dann muß man 8 Takte, also 16 Stunden für das Projekt einplanen, wenn das Risiko, die geplante Dauer zu überschreiten, höchstens 20 Prozent hoch sein soll. Diesen Schätzwert von 8 Takten erhält man, indem man beginnend von links solange die Balken des Diagramms aufaddiert, bis die Summe 80 Prozent überschreitet. Will man das Risiko auf 10 Prozent begrenzen, muß man 24 Stunden einplanen. Nebenbei bemerkt hat es hier wenig Sinn, Streuungen zu betrachten, da für ein Projekt das Überschreiten der vorgesehenen Dauer kritisch ist, nicht das Unterschreiten.

Ausgehend von den statistischen Produktivitätsprofilen für Java erhält man ein anderes Diagramm der Verteilung φ:



Die Höhe der Balken schwankt bei Java viel stärker als bei C. Die Schwankung

$$\sum_{x=1}^{15} \mid \varphi(x+1) - \varphi(x) \mid$$

der Verteilung φ beträgt 62.7 für C und 92.6 für Java, also fast 50 Prozent mehr. Das liegt daran, daß die Eingangsverteilungen für Java mehr Löcher und Ausreißer haben als für C.

Der Erwartungswert für Java beträgt 11.8 Stunden, der 20-Prozent-Schätzwert 14 Stunden und der 10-Prozent-Schätzwert 16 Stunden.

3.10 Vergleich

Hier sind noch einmal die Schätzergebnisse für das Beispielprojekt im tabellarischen Vergleich:

	C	Java
Erwartungswert	12.4 h	11.8 h
20-Prozent-Risiko	16 h	14 h
10-Prozent-Risiko	24 h	20 h

Der Unterschied zwischen den Erwartungswerten ist gering. Interessanter ist ein Vergleich der Risikowerte. Laut Modellberechnung muß man 16 Stunden bei C, aber nur 14 Stunden bei Java einplanen, wenn man das Risiko, nicht rechtzeitig fertig zu werden, auf 20 Prozent begrenzen will. Java schneidet auch besser ab, wenn man das Risiko auf 10 Prozent begrenzt. Beim Vergleichen der Ergebnisse darf man nicht vergessen, daß für Java nur wenige Datenpunkte vorlagen.

4 Bemerkungen

4.1 Datenerfassung

Der Aufwand für das *laufende* Erfassen der benötigten empirischen Rohdaten ist gering. Ich habe genau zum Modell passende Erfassungsbögen entwickelt, die zur Zeit erprobt werden. Mit den Bögen werden die einzelnen Tätigkeiten der Entwickler und die zugehörigen Zeiten erfaßt. Es ist klar, daß sich Ungenauigkeiten bei der Zeiterfassung durch die Entwickler in der Praxis nicht ganz vermeiden lassen. Ich gehe aber davon aus, daß sich solche Ungenauigkeiten aufgrund der Mittelbildung bei der Berechnung der Produktivitätsprofile wenig auf die Ergebnisse auswirken. Außerdem werden die erfaßten Zeiten im Modell zunächst in Takte umgerechnet (Abschnitte 3.4 und 3.6), wodurch kleinere Schwankungen bei den Zeiten verschwinden.

Schwieriger ist das Auswerten *bestehender* Datensätze, die nicht im Hinblick auf das neue Modell erfasst wurden. Meistens werden dann einige Daten für das Modell fehlen. Zum Beispiel war die Datenerfassung in den Programmierkursen nicht auf das neue Modell zugeschnitten. Nur deshalb war es in dieser Arbeit nötig, Annahmen über die zeitliche Verteilung von Problemmeldungen (Abschnitt 3.5) und über die Mehrarbeit zu machen, die mit Entwurfsänderungen verbunden ist (Abschnitte 3.6 und 3.7). Nur deshalb war es nötig, den Quelltext der Programmieraufgaben heranzuziehen, um festzustellen, ob die betrachtete Entwurfsänderung (Abschnitt 3.2) bei einem Studenten aufgetreten war oder nicht. Die größte Schwierigkeit beim Auswerten bestehender Datensätze sehe ich darin, die von den Entwicklern erfaßten Zeiten mit den in den Projekten aufgetretenen Entwurfsänderungen in Beziehung zu setzen.

4.2 Modellprüfung

Bisher wurden die Aufgaben in den Programmierkursen von jedem Studenten nacheinander bearbeitet. Zur Zeit ändern wir den Aufbau der Programmierkurse, damit das in dieser Arbeit betrachtete Beispielprojekt (Abschnitt 3.3) tatsächlich und für beide Programmiersprachen durchgeführt werden kann. Obwohl die vorhandene Datenbasis klein ist, sehe ich das als einen ersten Schritt, um die Vorhersagen des Modells zu überprüfen. Außerdem arbeite ich daran, das Modell auf industriellen Projektdaten einzusetzen.

Literatur

[1] ABDEL-HAMID, MADNICK: *Software Project Dynamics. An Integrated Approach*, Prentice Hall 1991

[2] BASILI, CALDIERA, MC GARRY, PAJERSKI, PAGE, WALIGORA: "The Software Engineering Laboratory – An Operational Software Experience Factory", Proceedings ICSE 14 (1992) 370-381

[3] BASILI, BRIAND, MELO: "How Reuse Influences Productivity in Object-Oriented Systems", Communications of the ACM 39-10 (1996) 104-116

[4] EL EMAM, MADHAVJI: *Elements of Software Process Assessment and Improvement*, IEEE Computer Society 1999

[5] GRAY, MAC DONELL: "A Comparison of Techniques for Developing Predictive Models of Software Metrics", Information and Software Technology 39 (1997) 425-437

[6] HATTON: "Does OO Sync with How We Think?", IEEE Software 15-3 (1998) 46-54

[7] HUDAK, JONES: "Haskell vs. Ada vs. C++ vs. Awk. An Experiment in Software Prototyping Productivity", Technical Report, Yale University, July 1994

[8] HUMPHREY: *A Discipline for Software Engineering*, Addison-Wesley 1995

[9] KEMERER: "An Empirical Validation of Software Cost Estimation Models", Communications of the ACM 30-5 (1987) 416-429

[10] LEDERER, PRASAD: "Nine Management Guidelines for Better Cost Estimating", Communications of the ACM 35-2 (1992) 51-59

[11] LLOYD, HARTLINE: "On Language Choice for Software Projects", Preprint, University of Washington, March 1999

[12] PADBERG: "A Probabilistic Model for Software Projects", Proceedings ESEC/FSE 7 (1999) 109-126, Springer LNCS 1687

[13] RAFFO, KELLNER: "Modeling Software Processes Quantitatively and Evaluating the Performance of Process Alternatives", enthalten in [4] 297-341