Evaluierung und Aufwandsschätzung bei der Integration von Open Source Software-Komponenten

Stefan Koch und Christian Neumann

Abteilung für Informationswirtschaft
Wirtschaftsuniversität Wien
Augasse 2-6
1090 Wien, AUSTRIA
stefan.koch@wu-wien.ac.at

Abstract: In dieser Arbeit wird ein Ansatz zu einem Aufwandsschätzungmodell für die Integration von Open Source Komponenten vorgestellt. Ausgehend von der Entwicklung einer Klassifikation unterschiedlicher Arten dieser Integration werden mögliche Operationalisierungen für die Evaluierung und darauf basierend die Aufwandsschätzung vorgeschlagen.

1 Einleitung

Open Source Software hat in den letzten Jahren zunehmend an Bedeutung gewonnen, vor allem auch über die reine Benutzung hinaus. Es zeigen sich auch Auswirkungen auf die Software-Entwicklung, durch die Übernahme von entsprechenden Tools, von Teilen des Entwicklungsprozesses und auch durch die Entwicklung auf Basis von Open Source Komponenten [SS04]. Enwicklungstools, die zur Open Source Entwicklung verwendet werden und selbst Open Source sind, werden zunehmend auch in kommerziellen Projekten eingesetzt. Dazu zählen JBoss, GCC, Eclipse oder CVS. Auch entsprechende Elemente aus dem Open Source Entwicklungsprozess werden übernommen [Lu04], beispielsweise gemeinsame Code-Verantwortung oder Teile des Reviewprozesses. In diesem Artikel wird auf eine weitere Möglichkeit eingegangen, die Verwendung von Open Source Komponenten. Darunter ist zu verstehen, dass eine neue Entwicklung die Ergebnisse eines oder mehrerer Open Source Projekte einbindet, anstatt die Funktionalität selbst zu implementieren. Einige Beispiele für diese Art der Software-Entwicklung können auch bereits in der Literatur gefunden werden, beispielsweise im Zusammenhang mit der NASA Mars Exploration Rover Mission [No04], dem NOW-Projekt zur Entwicklung einer Modellierungsumgebung für Prozesse unter Verwendung der JGraph-Komponente [Ha04] oder einer Komplettlösung für ein Spital [FK04].

2 Integration von Open Source Komponenten

Open Source Komponenten eröffnen in der Software-Entwicklung wesentlich andere Möglichkeiten als die Integration von kommerziellen Fertigkomponenten (commercial-

off-the-shelf, COTS), da der Source Code hier verfügbar, einsehbar und änderbar ist. Es handelt sich damit um white-box reuse, womit mehrere Vorteile verbunden sind: Es ist so möglich, Einblick in den Code zu nehmen, diesen zu evaluieren und zu verbessern. Weiter entsteht keine Abhängigkeit von einem Hersteller und dessen Produktpolitik. Es sind jedoch auch einige Herausforderungen mit diesem Einsatz von Open Source Software verbunden [SS04,No04,Ha04]. Dazu zählt zuerst einmal die Auswahl und Evaluierung einer geeigneten Komponente. Insbesondere die Funktionalität und die Qualität der Implementierung aber auch der zugrunde liegenden Architektur sind festzustellen. Daneben ist es wesentlich, einen Eindruck von dem dahinter liegenden Projekt zu gewinnen, um die weitere Entwicklung abschätzen zu können. Dazu zählt die Größe der Community, deren Aktivitätsgrad sowie eventuell auch die Hilfsbereitschaft bei Problemen. Die nächste Herausforderung stellt die weitere Zusammenarbeit mit dem Projekt dar. Es ist zu entscheiden, welche Vorgehensweise bei selbst durchgeführten Änderungen erfolgt, insbesondere ob diese wiederum der Community zur Verfügung gestellt werden. Wenn dies nicht geschieht, entfernt man sich zwangsläufig vom Entwicklungspfad des Projektes, was zu Problemen führen kann, wenn man neue Releases integrieren möchte [SS04]. In diesem Artikel soll über eine Operationalisierung der angeführten Punkte ein erstes Aufwandsschätzungsmodell für die Software-Entwicklung mittels Integration von Open Source Komponenten entwickelt werden. Explizit nicht behandelt werden dabei rechtliche Fragen, also insbesondere welche Auswirkungen auf eine kommerzielle Verwertung sich durch diese Integration ergeben [RE04].

3 Evaluierung von Open Source Software und Projekten

Als wesentlichste Punkte sind die Qualität des Source Codes selbst sowie Größe und Struktur des Projektes zu sehen. Es wird vorgeschlagen, beides gleichermaßen durch eine Analyse des Source Codes beziehungsweise der entsprechenden Meta-Daten abzudecken. Die Qualität des Codes Q selbst kann durch die Berechnung entsprechender Software-Metriken wie der zyklomatischen Komplexität [Mc76], oder auch durch Metriken der objekt-orientierten Programmierung und Designs [CK94] erfolgen. Diese können durch verschiedenste Programme automatisch erhoben werden. Wesentlicher ist es, zu einer Einschätzung der dahinter stehenden Programmiergemeinde zu gelangen. Es wird vorgeschlagen, dass dies anhand der im verwendeten Source-Code-Verwaltungssystem hinterlegten Meta-Daten erfolgt. Dort werden Daten hinsichtlich Datum, Größe und Autor jeder einzelnen Änderung festgehalten und sind einfach zugänglich [HK05]. Es wird vorgeschlagen, die Struktur der Programmiergemeinschaft einerseits durch die Anzahl der aktiven Programmierer, andererseits durch eine Maßzahl für die Ungleichverteilung der Leistung innerhalb dieser Gruppe zu beschreiben, da der Großteil der Arbeit oftmals von einigen wenigen erbracht wird [Mo02,Ko04]. Dies kann durch Berechnung des sogenannten Gini-Koeffizienten anhand der bisherigen Beiträge erfolgen. Eine hohe Ungleichverteilung, besonders bei geringer Gesamtgröße wäre als Risikofaktor anzusehen. Um zu einer Kennzeichnung der Entwicklungsgeschwindigkeit zu gelangen, wird vorgeschlagen, die Parameter einer simplen quadratischen Funktion zu schätzen, die das Wachstum der Software in Programmzeilen über die Zeit ausdrückt. Wie sich gezeigt hat, erfolgt bei einem Teil der Projekte das Wachtum über-linear, bei

anderen verlangsamt es sich [Ko04]. Im einfachsten Fall kann in diese 2 Gruppen geteilt werden. Diese Charakterisierung der Geschwindigkeit *G* hat wesentliche Auswirkungen auf die weitere Zusammenarbeit und auch, je nach Integration, den weiteren Aufwand.

4 Aufwandsschätzung für die Integration von Open Source Komponenten

4.1 Definitionen

Zusätzlich zu den Parametern *G* und *Q* wie oben beschrieben werden für die folgende Darstellung der jeweiligen Aufwände *A* die folgenden Definitionen verwendet: $F_{t=x}$ beschreibt die zum Zeitpunkt *t=x* im Vergleich zu *t=x-1* zusätzliche nötige Systemfunktionalität, $F^{Komp}_{t=x}$ die zu diesem Zeitpunkt in der Komponente implementierte. Der Zeitpunkt *t=0* entspricht der ersten zu erstellenden Version. Als Maßzahl für die Funktionalität werden entweder Programmzeilen oder Function Points herangezogen. Es werden entsprechende Kostenfunktionen *f(x)* definiert, in die jeweils die entsprechenden Parameter eingehen. Entweder sind diese selbst zu entwickeln, oder es können Aufwandsschätzungsverfahren aus der Literatur wie beispielsweise COCOMO [Bo00] zur Anwendung gebracht werden. Als wesentlichste Möglichkeiten ergeben sich die folgenden Kostenfunktionen:

- $f(F_{t=0} - F^{Komp}_{t=0})$*:* Dieser Aufwand entspricht jenem, der bei einer neuen Software-Entwicklung anfällt, wobei jedoch die zu implementierende Funktionalität um die bereits in der Open Source Komponente enthaltene reduziert wird. Daher ist nur der Rest zu implementieren. Im einfachsten Fall kann dies über ein bekanntes Aufwandsschätzungsverfahren wie COCOMO abgedeckt werden.

- $f(G,F^{Komp}_{t=0})$*:* Diese Funktion spiegelt wieder, daß in Abhängigkeit von der Entwicklungsgeschwindigkeit im Open Source Projekt ein Aufwand anfällt. Beispielsweise kennt das COCOTS-Modell [Bo00] als Abwandlung von COCOMO für die Integration von Fertigkomponenten den Aufwand für Glue Code und jenen durch die Volatilität der Komponente während der Entwicklung. Beide sind wesentlich von dem Parameter CREVOL abhängig, der die prozentuelle Überarbeitung am System aufgrund Änderungen in der Komponente angibt. Dieser Parameter kann basierend auf der bereits in der Auswahl angefertigten Schätzung eines Modells *G* für die Evolution der Open Source Komponente gesetzt werden.

- $f(Q,F^{Komp}_{t=0})$*:* Dieser Faktor stellt einen Aufwand dar, der durch Verständnis und Überarbeitung der Open Source Komponente bedingt wird. Dieser Aufwand hängt wesentlich von einerseits Funktionalität beziehungsweise Größe der Komponente sowie ihrer Qualität, also Architektur, Verständlichkeit und Dokumentation ab. Beispielsweise kennt das Reuse-Modell von COCOMO II [Bo00] den Faktor Software Understanding, der von Struktur, Applikationsklarheit und Selbstbeschreibungsfähigkeit abhängt. Dieser Wert kann über die Analyse des Source Codes, wie vorgeschlagen, entsprechend parametrisiert werden.

4.2 Open Source Komponenten im black-box Reuse

Wenn die gewählte Komponente nicht verändert wird, also der Zugriff auf den Source Code nicht ausgenutzt wird, so handelt es sich um black-box reuse wie bei kommerzieller Software, sodass das COCOTS Modell [Bo00] anwendbar ist, das nur mehr entsprechend zu parametrisieren ist. Insbesondere wesentlich ist der Aufwand für die Entwicklung des Glue Codes und jener, der durch die Volatilität der Komponente, also neue Releases, entsteht. Dies gilt zu Beginn sowie für weitere Versionen des Systems.

$$A=f(F_{t=0} - F^{Komp}{}_{t=0})+f(G,F^{Komp}{}_{t=0})+ \sum_{t=1}^{t=n} (f(F_{t=n} - F^{Komp}{}_{t=n})+ f(G,F^{Komp}{}_{t=n}))$$

4.3 Open Source Komponenten im white-box Reuse ohne weitere Zusammenarbeit mit dem Projekt

In diesem Fall handelt es sich um klassisches Software Reuse. Diese Effekte können beispielsweise durch das in COCOMO II [Bo00] inkludierte Reuse-Modell berechnet werden. Die Ergebnisse der Evaluierung hinsichtlich Qualität des Designs und Dokumentationsgrad können hier über den Faktor Software Understanding direkt einfließen, um so das entsprechende Modell zu erhalten. Die weitere Systementwicklung und Wartung werden jedoch selbst getragen, sodass hier normale Aufwandsschätzungsmodelle zur Anwendung kommen und nur die Funktionalität zum Zeitpunkt $t=0$ abgezogen wird. Es fällt jedoch auch kein weiterer Aufwand für Verständnis und Überarbeitung an.

$$A=f(F_{t=0} - F^{Komp}{}_{t=0})+f(Q,F^{Komp}{}_{t=0})+ \sum_{t=1}^{t=n} (f(F_{t=n} - F^{Komp}{}_{t=0}))$$

4.4 Open Source Komponenten im white-box Reuse mit weiterer Zusammenarbeit mit dem Projekt

Diese Variante entspricht der vorangegangenen, es werden jedoch eigene Änderungen zum Open Source Projekt beigetragen, was die Kompatibilität mit neuen Releases weitgehend sichern sollte. Damit ist bei Ersterstellung der Software analog zu oben vorzugehen, jedoch der Aufwand im Bereich der Weiterentwicklung könnte jeweils in Abhängigkeit von neuen Versionen der Komponente reduziert werden. Jedoch fällt mit jeder neuen Version wiederum potentiell Aufwand für Verständnis und Überarbeitung an.

$$A=f(F_{t=0} - F^{Komp}{}_{t=0})+f(Q,F^{Komp}{}_{t=0})+ \sum_{t=1}^{t=n} (f(F_{t=n} - F^{Komp}{}_{t=n})+ f(Q,F^{Komp}{}_{t=n}))$$

5 Zusammenfassung

Die Integration von Open Source Komponenten eröffnet für die Software-Entwicklung neue Möglichkeiten der Wiederverwendung einer enormen Anzahl von Funktionalitäten und Implementierungen. Es sind damit jedoch auch eine Reihe von Fragen und Unsicherheiten verbunden. Dieser Artikel hat versucht aufzuzeigen, wie man in einigen Punkten von rein strategischen Überlegungen hin zu konkret quantifizierbaren Größen gelangen kann, die einen Vergleich unterschiedlicher Varianten ermöglichen können.

Literaturverzeichnis

[Bo00] Boehm, B.W.; Abts, C.; Brown, A.W.; Chulani, S.; Clark, B.K.; Horowitz, E.; Madachy, R.; Reifer, D.J.; Steece, B.: Software Cost Estimation with COCOMO II. Prentice Hall, Upper Saddle River, New Jersey, 2000.

[CK94] Chidamber, S.R.; Kemerer, C.F.: A Metrics Suite for Object-Oriented Design. IEEE Transactions on Software Engineering, 20(6), S. 476-493, 1994,

[FK04] Fitzgerald, B.; Kenny, T.: Developing an Information Systems Infrastructure with Open Source Software. IEEE Software, 21(1), S. 50-55, 2004.

[Ha04] Hang, J.; Hohensohn, H.; Mayr, K.; Wieland, T.: Benefits and pitfalls of open source in commercial contexts. In (Koch, S. Hrsg.): Free/Open Source Software Development. Idea Group Publishing, Hershey, PA, 2004, S. 222-241.

[HK05] Hahsler, M.; Koch, S.: Discussion of a Large-Scale Open Source Data Collection Methodology. Proceedings of the Hawaii International Conference on System Sciences (HICSS-38), Big Island, Hawaii, 2005.

[Ko04] Koch, S.: Profiling an open source project ecology and its programmers. Electronic Markets, 14(2), S. 77-88, 2004.

[Lu04] Lussier, S.: New Tricks: How Open Source Changed the Way My Team Works. IEEE Software, 21(1), S. 68-72, 2004.

[Mc76] McCabe, T.J.: A Complexity Measure. IEEE Transactions on Software Engineering, 2(4), S. 308-320, 1976.

[Mo02] Mockus, A.; Fielding, R.; Herbsleb, J.: Two case studies of open source software development: Apache and Mozilla. ACM Transactions on Software Engineering and Methodology, 11(3), S. 309-346, 2002.

[No04] Norris, J.S.: Mission-critical development with open source software: Lessons learned. IEEE Software, 21(1), S. 42-49, 2004.

[RE04] Ruffin, M.; Ebert, C.: Using Open Source Software in Product Development: A Primer. IEEE Software, 21(1), S. 82-86, 2004.

[SS04] Spinellis, D.; Szyperski, C: How Is Open Source Software Affecting Software Development? IEEE Software, 21(1), S. 28-33, 2004.