Moderne Hilfsmittel der Softwareentwicklung

Thema: Ownership

Schriftliche Ausarbeitung

Kernfrage: Welchen Einfluss hat Ownership auf FLOSS Systeme im Vergleich zu industriellen/kommerziellen Systemen?

Von Alexander Hildebrandt

Inhaltsverzeichnis:

1 Einleitung

1.1 Was ist Ownership?

Ownership ist ein soziales Konzept. Es beschreibt, wie sich Entwickler die Arbeit in einem Projekt untereinander aufteilen. Wenn ein einziger Entwickler die Mehrheit einer Projektkomponente entwickelt hat, spricht man von hoher Ownership; dieser Entwickler „besitzt“ sozusagen jene Komponente. Allerdings ist Ownership allgemein ein sehr vages Konzept. Es gibt keine allgemeingültige Definition von „Besitz“ im Entwicklungskontext. D.h. wenn man sich über Ownership und dessen Auswirkungen unterhalten möchte, muss man sich auf eine bestimmte Definition einigen. Das Problem an konkreten Definitionen ist, dass sie soziale Aspekte von Ownership nicht unbedingt greifen können.

Deshalb betrachten viele Forscher nicht direkt die sozialen Aspekte von Ownership, sondern konzentrieren sich auf konkrete Definitionen, wobei jene Aspekte oft als Erklärung für bestimmte Muster genutzt werden können. Ein Beispiel für eine konkrete Definition ist die Anzahl von Commits pro Komponente von einem Entwickler. Wenn ein einzelner Entwickler mindestens 80% der Änderungen an einer Komponente vornimmt, spricht man dann von einer hohen Ownership. Auch wenn man durch Interpretationen wie diese einen simplifizierten Blick auf Ownership legt, lassen sich durch bestimmte Analysemethoden trotzdem signifikante Aussagen über Projekte machen.

Der Grund, warum Ownership trotz der eben genannten Schwierigkeiten immer mehr erforscht wird, ist die Tatsache, dass Ownership veränderbar ist. Projektmanager können Richtlinien einführen, wodurch die Effizienz des Projektes gesteigert werden könnte. Andere Entwicklungsaspekte wie Größe und Komplexität können im Gegensatz dazu nur bedingt bis gar nicht angepasst werden. Deshalb hat Ownership eine besondere Rolle in der Softwareentwicklung und viele große Firmen richten sich bereits in einigen Feldern nach Ownership, um ihre Effizienz zu steigern.

1.2 Bird und Foucault

Im Folgenden werden zwei Papers vorgestellt, die beide eine Interpretation von Ownership nach Anzahl von Commits benutzen und dadurch versuchen, Aussagen über Effizienz machen zu können. Das erste Paper von Bird et al. [1] betrachtet Ownership im kommerziellen Bereich am Beispiel von Windows Vista und Windows 7. Das zweite Paper von Foucault et al. [2] bezieht sich im Gegensatz dazu auf Ownership in einer Vielzahl von FLOSS Projekten.

Bird et al. benutzten bestimmte Analysemethoden, um Ownership Metriken mit klassischen Metriken wie Größe und Komplexität zu vergleichen. Dadurch wollten sie Aussagen über Ownership treffen, die zukünftigen Entwicklern helfen sollten, Software-Ausfälle in Projekten zu vermeiden. Nach ihren Studien sind sie zu der Schlussfolgerung gekommen, dass Komponenten eines Projektes fehleranfälliger sind, wenn es viele Entwickler mit einer geringen Anzahl an Commits gibt. D.h., dass es gibt nicht genug erfahrene Entwickler gibt, die Komponenten „besitzen“ und damit als Ansprechpartner und Korrekturleser von Commits fungieren können.

Ziel des Papers von Foucault et al. war es, die Methoden von Bird et al. im FLOSS Kontext zu benutzen, um dadurch allgemeingültige Aussagen über Ownership sowohl im kommerziellen und als auch im Open-Source Kontext machen zu können. Ihre Analyse hat jedoch nicht die gewünschten Ergebnisse geliefert, weshalb sie schlussfolgerten, dass Ownership Richtlinien für kommerzielle Projekte sich nicht auf den FLOSS Kontext generalisieren lassen.

1.3 Familiäre Studien:

Es gibt einige vorherige Studien, die sich mit dem Effekt von Contribution beschäftigt haben. Rahman & Devanbu [3] untersuchten den Effekt von Ownership und Erfahrung auf Qualität in verschiedenen Open-Source Projekten, wobei die Ergebnisse ähnliche wie die von Bird et al. waren. Allerdings wurde Ownership in ihrer Studie anders definiert und bereits bekannte Ownership Richtlinien unterscheiden sich in OSS und kommerziellen Projekten sehr stark.

Weyuker et al. [4] untersuchten, was für einen Einfluss die Teamgröße in Vorhersagemodellen hat. Sie fanden keine signifikante Korrelation, allerdings benutzten sie, anders als Bird et al., auch nicht die Proportion der Contributions von den einzelnen Entwicklern.

Boh et al. [5] fand heraus, dass Projekt-spezifisches Wissen einen größeren Einfluss auf hohe Qualität hat als ein großes Maß an allgemeinen Fachkenntnissen.

Laut Curtis et al. [6] ist eins der größten Probleme von Softwareentwicklung der Ausfall von Koordination und Kommunikation. Die Chance, dass es zu solchen Ausfällen kommt, steigt mit fehlenden Ansprechpartnern und einer größeren Anzahl von Entwicklern.

2. Analysen durch Ownership Metriken

2.1 Terminologie und Metriken:

Das Ziel ist es, die Beziehung zwischen Ownership und Softwarequalität zu verstehen. Dazu stellen wir folgende Fragen:

(1) Bedeutet hohe Ownership in einem Projekt automatisch auch weniger Fehler?
(2) Gibt es negative Effekte, wenn eine Software von vielen Entwicklern mit geringer Ownership entwickelt wird?

Dazu werden folgende Begriffe definiert:

Contributor: Ein Entwickler, der an einer Komponente arbeitet

Ownership: Das Verhältnis der Commits eines Contributor zu allen vorhandenen Commits pro Komponente

Minor Contributor: Contributors mit weniger als 5% Ownership

Major Contributor: Contributor mit mindestens 5% Ownership

Außerdem werden folgende Metriken eingeführt:

MINOR: Anzahl der Minor Contributors

MAJOR: Anzahl der Major Contributor s

TOTAL: Anzahl aller Contributors

OWNERSHIP: Ownership-Wert des Top-Contributors pro Komponente

2.2 Hypothesen:

Wir nehmen an, dass Entwickler mit geringen Fachkenntnissen eine höhere Chance haben, Bugs einzuführen. Außerdem ist eine geringe Anzahl von Commits für eine bestimmte Komponente ein Indiz dafür, dass geringe Fachkenntnisse für diese Komponente vorliegen. Zusätzlich bedeutet eine hohe Anzahl von Entwicklern, dass die Koordination zwischen Entwicklern schwieriger wird und dass Komponenten möglicherweise fragmentiert sind.

Hypothese 1: Software-Komponenten mit vielen Minor Contributors haben mehr Ausfälle.

Außerdem zeigt eine hohe Ownership, dass es Entwickler gibt, die viel an einer Komponente gearbeitet haben. Daraus ergibt sich, dass diese Entwickler wahrscheinlich hohe Fachkenntnisse über diese Komponente haben und auch als Ansprechpartner für andere Entwickler fungieren können.

Hypothese 2: Software-Komponenten mit hoher Ownership haben weniger Ausfälle.

Nun kommt die Frage auf, warum einige Komponenten so viele Minor Contributors haben, wenn dies doch ein Indiz für mehr Fehler ist. Ein Grund dafür ist, dass während Wartungen, Fehlerbehebung, etc. in einem Projekt Owner einer Komponente oft andere Komponenten modifizieren müssen, da diese Komponenten mit der eigenen in Verbindung stehen.

Hypothese 3: Ein Minor Contributor einer Komponente ist Major Contributor einer anderen Komponente, wobei diese Komponenten in einer Abhängigkeitsbeziehung stehen.

Wenn Minor Contributors einen großen Einfluss auf Software Qualität haben, werden Ausfallprognosen durch Einbeziehung oder Entnahme dieser Contributors beeinflusst. Deshalb werden verschiedene Ausfallprognosen reproduziert und wobei man bei mehreren Durchgängen jeweils allen Daten, nur MINOR Daten und nur MAJOR Daten berücksichtigt. Es wird erwartet, dass die Ausfallprognosen darunter leiden, wenn man MINOR entfernt.

Hypothese 4: Die Leistung von Ausfallprognosen wird drastisch sinken, wenn man MINOR Informationen entfernt.

2.3 Analyse Techniken:

Bird et al. untersuchten für ihre Studie im kommerziellen Bereich Windows Vista und Windows 7, da dies zwei unabhängige Projekte sind, die vergleichbare Rahmenbedingungen haben. Pre-/Post-Release Ausfälle wurden in einer Beziehungsanalyse sowohl mit Ownership Metriken als auch mit klassischen Metriken wie Größe und Komplexität verglichen.

Das Ergebnis der Analyse war, dass Ausfälle eine starke Beziehung mit MINOR, TOTAL und OWNERSHIP hatten. MINOR hatte eine größere Beziehung mit Pre-/Post-Release Ausfällen in Vista und Pre-Release Ausfällen in Windows 7 als jede andere Metrik, die Windows sammelt. Obwohl Post-Release Ausfälle von Windows 7 schwer zu analysieren sind, weil viele Binärdateien noch keine Berichte über Ausfälle haben, sind dessen Ergebnisse auch signifikant, abgesehen von OWNERSHIP. Allerdings kann man daraus noch keine direkten Aussagen machen, da viele Komponenten mit einer hohen Anzahl von Entwicklern auch automatisch an sich große Komponenten sind und man nicht genau sehen kann, ob die Anzahl der Entwickler oder die Größe der Komponente für die Beziehung zu Ausfällen sorgt.

Deshalb haben Bird et al. Multiple Lineare Regression [7] benutzt, um eine eindeutige Aussage zu machen. Dieses Verfahren wird benutzt, um zu zeigen, welche Variablen einen Effekt auf Ausfälle haben, wie groß dieser Effekt ist, in welche Richtung der Effekt geht und wie viel Varianz in der Anzahl der Ausfälle von den Metriken erklärt wird. In diesem Fall wird ein MLR Modell, das Ownership beinhaltet, mit einem Modell verglichen, das es nicht beinhaltet.

Die Ergebnisse der MLR Modelle zeigen, dass die klassischen Quellcode-Metriken wie Größe, Komplexität und Abwanderungsquote bereits einen großen Teil der Varianz erklären können. Die Teamgröße (ohne Blick auf Verteilung in Minor und Major Contributor) zeigt ebenfalls eine signifikante Verbesserung in erklärter Varianz. MINOR hatte wiederum auch einen Zuwachs, abgesehen von der Erklärung der Post-Release Ausfällen Windows 7. MAJOR und OWNERSHIP ergaben jeweils einen kleineren, aber noch signifikanten Zuwachs.

Daraus ergeben sich folgende Ergebnisse:

(1) Die Anzahl der Minor Contributors hat eine starke positive Beziehung mit Pre-/Post-Release Ausfällen.
(2) Eine hohe Ownership beim Top Contributor sorgt für weniger Ausfälle, aber dieser Effekt ist im Vergleich zur Minor Beziehung eher gering.

Die Werkzeuge und Informationen, die von Bird et al. benutzt wurden, sind Microsoft-spezifisch. Diese Informationen sind in Java FLOSS Systemen schwer zu sammeln, da es nicht die gleichen Ausfall Tracker und Konventionen gibt. Deshalb benutzten Foucault et al. sowohl den PROMISE Corpus [8] als auch den Corpus aus der Studie von D'Ambros et al. [9], wodurch Anzahl von Ausfällen und verschiedene Metriken zu geschriebenem Code und Komplexität dargestellt werden. Dadurch ergibt sich, dass im Corpus von Foucault et al. keine Pre-Release Ausfälle enthalten sind. Außerdem sind Java Klassen sehr viel kleiner als Windows Binärdateien, weshalb es für Vergleichszwecke mehr Sinn macht, Java Packages zu betrachten. Des Weiteren wird statt MLR ein sehr ähnliches Framework benutzt, um die gleichen Effekte zu messen; partielle Korrelation [10]. Mit diesem Framework lässt sich die Ownership Metrik mit der klassischen Metrik „Größe" vergleichen. Zusätzlich wird mit dem Versionskontrollsystem „Harmony" [11] die Contributions von Entwicklern deutlich.

Die Ergebnisse zeigen, dass Code Metriken mehr Korrelation mit der Anzahl an Ausfällen haben als die Ownership Metriken. Nur in wenigen Fällen haben Ownership Metriken eine etwas bessere Korrelation. Im Vergleich zu den Ergebnissen von Bird et al. ist dies ein starker Kontrast. Von allen Ownership Metriken lieferten TOTAL und MINOR die besten Ergebnisse. D.h. in Java FLOSS Systemen ist die Anzahl von Entwicklern ein genauso guter Indikator für Fehleranfälligkeit wie die Anzahl von Minor Contributors. Wenn man partielle Korrelation mit kontrollierter Größe anwendet, also nach Bugs sucht und dabei eine konstante Modulgröße behält, ändern sich die Ergebnisse nur sehr begrenzt. Damit wurden die Ergebnisse von Bird et al. für Java FLOSS Systeme nicht bestätigt. Dadurch ist die Analyse der Effekte von Minor Contributors für FLOSS Systeme hinfällig.

3 Effekte von Minor Contributors nach Bird et al.:

Es wurden zwei Analysen angelegt, um die Effekte von Minor Contributors zu untersuchen:

(1) Da fast alle Entwickler Major Contributors für einige Binärdateien und Minor Contributors für andere sind, untersucht die erste Analyse, ob es eine Beziehung zwischen zwei Komponenten gibt, wenn ein Entwickler Major Contributor für die eine und Minor Contributor für die andere ist.
(2) Die zweite Analyse benutzt die Ausfallprognosen von Pinzger et al., wobei es zu Modifikationen der Prognose kam, um Ownership berücksichtigen zu können.

3.1 Abhängigkeitsanalyse:

Entwickler bei Microsoft, die Top-Contributor einer Binärdatei sind, mussten oft eine andere Datei ändern, weil diese Datei die eigene nutzt oder von ihr genutzt wird. Die Frage ist, wie oft dieser Fall in der Praxis vorkommt, also wie viele der Minor Contributor durch diese Abhängigkeit erklärt werden.

Für diese Fragestellung wurde das Programm „MaX“ [12] benutzt, das die Beziehungen zwischen Binärdateien durch gegebene Informationen in einem Graph darstellen kann. „MaX“ wurde aber nur auf Windows Vista angewandt, da die benötigten Informationen für Windows 7 nicht zur Verfügung stehen.

Der Vista Graph, der von „MaX“ dargestellt wurde, zeigte, dass 52% der Binärdateien Minor Contributors haben, die gleichzeitig Major Contributors für andere Binärdateien sind, die mit der ersten Datei in Abhängigkeit stehen. Es wurden danach einige Zufallsgraphen erstellt, die mit den gleichen Voraussetzungen wie der Vista Graph gearbeitet haben (gleiche Anzahl von Entwicklern, Binärdateien, Commits, etc.). Durchschnittlich erreichten die Zufallsgraphen einen Wert von 24%. Außerdem gab es in 10000 Zufallsgraphen nur einen Höchstwert von 32%, wodurch sich die Signifikanz des Vista Graphen erkennen lässt. Daraus folgt, dass ein Minor Contributor von Vista, üblicherweise dadurch begründet wird, dass er ein Major Contributor einer abhängigen Datei ist.

3.2 Analyse der Ausfallprognosen:

Pinzger et al. berichten von einer Methode [13], die fehleranfällige Binärdateien in einem Contribution Netzwerk findet. Pinzger hat mit dieser Methode einen Prädiktor[13] gebaut, der fehleranfällige Binärdateien in diesen Netzwerken finden kann. Der Prädiktor hat dabei einen Prozentsatz für tatsächlich gefundene fehleranfällige Binärdateien (Recall) und einen weiteren Prozentsatz, der angibt wie viele der gemeldeten Binärdateien tatsächlich zu den fehleranfälligen gehören. Für Windows Vista hatte der Prädiktor 90% Recall und 85% Präzision.

Nun wurde analysiert, wie sich das Ergebnis geändert hat, wenn man Minor bzw. Major Informationen aus dem Netzwerk entfernt hat. Man sieht in den Netzwerken, dass viele Binärdatei-Paare, die eine kurze Verbindung über Minor Contributor hatten, jetzt komplett getrennt sind. Nun wurde ein neuer Prädiktor nach dem Schema von Pinzger gebaut, der mit den reduzierten Netzwerken für Vista und Windows 7 arbeiten kann. Die Werte für Vista ohne Minor Informationen beliefen sich dadurch auf 56% Recall und 44% Präzision. Mit dem Prädiktor von Pinzger kann man durch zufälliges Raten jedoch schon 50% für Recall und Präzision erreichen, also sind die Ergebnisse nur bedingt besser als Zufall. In beiden Versionen von Windows waren die Minor Informationen statistisch signifikant wichtiger als die Major Informationen. In einigen Fällen waren Minor

Netzwerke sogar besser als die kompletten Netzwerke.

Als Ergebnis lässt sich also festhalten, dass Minor Informationen die „Signale“ liefern, die von Ausfallprognosen mit Contribution Netzwerk benutzt werden. Ohne diese Informationen sind Prognosen stark beeinträchtigt, was die Hypothese unterstützt, dass Minor Contribution stark mit Softwarequalität zusammenhängt.

4. Diskussion:

In beiden Versionen von Windows gab es eine signifikante Beziehung zwischen Ownership und Codequalität. Komponenten mit mehr Minor Contributors hatten auch mehr Pre-/Post-Release Ausfälle. Dadurch ist die erste Hypothese empirisch unterstützt. Außerdem konnte man hohe Ownership mit einer geringeren Anzahl von Ausfällen assoziieren, wodurch die zweite Hypothese ebenfalls unterstützt ist.

Schlussendlich lassen sich für kommerzielle Projekte folgende Empfehlungen auf Basis von Ownership machen:

(1) Major Contributors sollten Änderungen an ihren Binärdateien stärker überprüfen. Außerdem sollten sie sich auf Code von Minor Contributors fokussieren, wenn der komplette Code zeitlich nicht überprüft werden kann.
(2) Potenzielle Minor Contributors sollten ihre gewollten Änderungen an Entwickler weitergeben, die mit der vorliegenden Binärdatei mehr Erfahrung haben anstatt selbst ein Minor Contributor zu werden.
(3) Komponenten mit geringer Ownership sollten während QA besonders genau getestet werden. Wenn es nur begrenzte Mittel für QA gibt, sollte man Binärdateien mit potenziell vielen Post-Release Ausfällen identifizieren und mehr Priorität beim Testen geben.

Es ist sicherlich nicht immer möglich, alle Empfehlungen umzusetzen. Dies gilt zum Beispiel, wenn es nicht genügend Major Contributors gibt, um Code zu kontrollieren oder wenn man keine Zeit für Ausfallprognosen vor QA gibt. Aber die Empfehlungen sind ein solider Leitfaden für kommerzielle IT-Projekte im Allgemeinen.

Foucault et al. fanden hingegen komplett andere Ergebnisse, was wahrscheinlich an dem inhärenten Unterschied zwischen industriellen und FLOSS Systemen. Einer der wesentlichen Unterschiede ist die Verteilung von Arbeitsanfall. In industriellen Systemen investieren Entwickler 100% ihrer Zeit in ein Projekt, während in FLOSS Systemen es zwei Arten von Entwicklern gibt; einige wenige Helden, die etwas zu allen Modulen beisteuern, und sehr viele Minor Contributors, die ein einziges Feature entwickeln oder einen Bug fixen und danach aufhören, für das System zu entwickeln. Dadurch sind die Ownership Metriken keine effizienten Indikatoren für Softwarequalität in FLOSS Systemen. Es wird angenommen, dass die Qualität von Software in diesen Fällen von der Qualität der Helden abhängig ist.

5. Referenzen:

[1] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu. Don't touch my code!: examining the effects of ownership on software quality.
[2] M. Foucault, J. R. Falleri, X. Blanc. Code Ownership in Open-Source Software
[3] F. Rahman and P. Devanbu. Ownership, Experience and Defects: a fine-grained study of Authorship.
[4] E. J. Weyuker, T. J. Ostrand, and R. M. Bell. Do too many cooks spoil the broth?
[5] W. Boh, S. Slaughter, and J. Espinosa. Learning from experience in software development: A multilevel analysis.
[6] B. Curtis, H. Krasner, and N. Iscoe. A field study of the software design process for large systems.
[7] http://www.stat.yale.edu/Courses/1997-98/101/linmult.htm
[8] T. Menzies, B. Caglayan, Z. He, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan. The promise repository of empirical software engineering data.
[9] M. D'Ambros, M. Lanza, and R. Robbes. Evaluating defect prediction approaches: a benchmark and an extensive comparison.
[10] M. G. Kendall and S. Alan. The advanced theory of statistics. Vols. II and III.
[11] J.-R. Falleri, C. Teyton, M. Foucault, M. Palyart, F. Morandat, and X. Blanc. The harmony platform.
[12] A. Srivastava, J. Thiagarajan, and C. Schertz. Efficient Integration Testing using Dependency Analysis.
[13] M. Pinzger, N. Nagappan, and B. Murphy. Can developer-module networks predict failures?