# Wie kommen Entwickler mit Klonen zurecht?

Jan Harder, Saman Bazrafshan
Universität Bremen
harder,saba@informatik.uni-bremen.de

Rebecca Tiarks
Universität Hamburg
tiarks@informatik.uni-hamburg.de

## Zusammenfassung

Software-Klonen wird nachgesagt, dass sie die Wartung von Software-Systemen erschweren. Wir haben erstmals in einem kontrollierten Experiment untersucht wie sich die Gegenwart von Klonen auf die Leistung von Programmierern und die Korrektheit ihrer Änderungen auswirkt. Überraschend viele Teilnehmer haben Änderungen an Klonen dabei nur unvollständig durchgeführt. Daher sollten Entwicklern bei Ihrer Arbeit Informationen über Klone bereitgestellt werden. In unserer zukünftigen Forschung wollen wir daher empirisch untersuchen wie diese Informationen effektiv in den Entwicklungsprozess eingebunden werden können.

## 1 Einleitung

Copy&Paste ist ein unverzichtbares Hilfsmittel bei der Software-Entwicklung. Während diese Strategie auf kurzfristige Sicht die Entwicklung beschleunigt, führt sie mittel- und langfristig zu Redundanzen, die die Wartung erschweren können. Änderungen an solchen Klonen müssen dann mit erhöhtem Aufwand wiederholt ausgeführt werden. Zudem besteht das Risiko, dass nicht alle Kopien gleichmäßig geändert werden. Dem Einfluss dieser Faktoren auf die Wartbarkeit von Software wurde in der jüngeren Vergangenheit viel Aufmerksamkeit entgegengebracht. Eine Vielzahl von Studien hat Projektarchive hierzu retrospektiv untersucht. Einerseits lässt sich eine nicht unerhebliche Anzahl von Fällen nachweisen, in denen Klone Fehler verursacht haben. Andererseits ließ sich kein systematischer Zusammenhang zwischen Klonen und Fehlern oder einem erhöhtem Wartungsaufwand herstellen.

Die bisherigen Studien haben allesamt gemein, dass sie den Zusammenhang von Klonen und Wartbarkeit durch retrospektive Analysen von Versionskontrollsystemen und Fehlerdatenbanken untersuchen. Wartungsaufgaben erfordern komplexe Schritte wie Programmverstehen, Fehlersuche, Quelltext-Änderungen und Testen, die alle durch das Vorhandensein von Klonen erschwert werden können. Retrospektive Studien anhand der Versionshistorie können jedoch nur das Ergebnis dieses Prozesses – die Quelltext-Änderung – betrachten. Eine langwierige Fehlersuche kann hierbei als einfache, einzeilge Änderung erscheinen.

Kontrollierte Experimente sind ein geeignetes Mittel, um diese Lücke zu schließen. Wir haben daher ein erstes kontrolliertes Experiment durchgehführt, um die Auswirkungen von Klonen auf die Leistung von Entwicklern und die Korrektheit ihrer Lösungen zu untersuchen. Derartige Humanstudien wurden in der Forschung zu Software-Klonen bislang kaum eingesetzt, sind aber erforderlich um unser Wissen über die Auswirkungen von Klonen zu vervollständigen [1].

## 2 Das Experiment

Wir haben 33 Entwicklern jeweils zwei Aufgaben zur Beseitigung eines Fehlers in einfachen Java-Spielen durchführen lassen [3]. Unsere Teilnehmer waren zum einen Studenten der Universität Bremen sowie Experten auf dem Gebiet der Software-Klone. Letztere haben im Rahmen eines Dagstuhl-Seminars zu Klonen am Experiment teilgenommen.

### 2.1 Versuchsaufbau

In den Open-Source-Spielen *FrozenBubble* und *Pacman* haben wir Fehler in Klone eingefügt, die bereits im System vorhanden waren. Jeder Klon führt hierbei zu einem Symptom des gleichen konzeptionellen Fehlers. Beispielsweise haben wir die geklonten Bewegungsroutinen von Pacman und den Geistern so verändert, dass sich beide Figuren fehlerhaft bewegen. Zusätzlich wurde für jedes System eine Variante erstellt, in der der Klon entfernt wurde, so dass derselbe Fehler nur noch an einer einzigen Stelle im Quelltext vorhanden ist. In Pacman wurde hierzu die Bewegungsroutine für Pacman und die Geister vereinheitlicht und enthielt nur einen Fehler, der aber die Bewegung beider Figuren korrumpierte.

Die Teilnehmer wurden in zwei Gruppen aufgeteilt, wobei jeweils eine die geklonte Variante erhielt und die andere die nicht geklonte. Für alle Teilnehmer wurden dann die benötigte Zeit für die Behebung des Fehlers, sowie die Korrektheit und Vollständigkeit ihrer Lösung bewertet und die Ergebnisse zwischen den Gruppen verglichen. Die unabhängige Variable des Experiments ist somit die Tatsache, ob der Fehler geklont ist oder nicht. Die abhängigen Variablen – also die zu messenden Unterschiede zwischen den Gruppen – waren die benötigte Zeit sowie die Korrektheit und Vollständigkeit der Lösung. Unvollständige Lösungen können auftreten, wenn geklonte Fehler nur an einer und nicht an beiden Stellen behoben wurden. Um eine möglichst realitätsnahe Durchführung der Aufgaben zu gewährleisten, haben die Teilnehmer

den vollständigen Prozess der Fehlerbehebung, inklusive, Programmverstehen, Fehlersuche, Änderungen, und Testen der Ergebnisse durchgeführt. Sie konnten die Programme jederzeit zur Ausführung bringen, um die Korrektheit ihrer Lösung zu prüfen. Ferner wurden sie dazu angehalten die Aufgabe so lange zu bearbeiten, bis sie zu der Überzeugung gelangt sind, dass ihre Lösung vollständig und korrekt ist. Die Teilnehmer wurden nicht darüber informiert, dass sich Klone im Code befinden oder dass das Experiment sich auf diese bezieht. Für die Bearbeitung der Aufgaben wurde ihnen keine Zeitbegrenzung auferlegt.

### 2.2 Ergebnisse

Aufgrund der unterschiedlichen Vorkenntnisse der beiden Teilnehmergruppen – Studenten und Experten – wurden ihre Ergebnisse getrennt ausgewertet. Mit einer Ausnahme benötigenten die Teilnehmer im Durchschnitt weniger Zeit für die Behebung des nicht geklonten Fehlers. Um eine vollständig korrekte Lösung zu erstellen benötigten sie bis zu 65% mehr Zeit (oder bis zu 5,5 Minuten mehr), wenn der Fehler geklont war. Allerdings behoben die Experten den geklonten Fehler in *FrozenBubble* durchschnittlich 5,9 Minuten schneller als den nicht geklonten. Aufgrund der kleinen Anzahl an Teilnehmern, sind die Ergebnisse für die Zeitmessung statistisch jedoch nicht signifikant. Sie deuten jedoch an, dass die Zeitunterschiede in absoluten Zahlen eher gering ausfallen. Der zusätzlich gemessene Zeitaufwand dürfte nur ein wesentliches Problem in der Wartung darstellen, wenn geklonte Fehler häufig vorkommen.

Eine größere Relevanz für die Wartbarkeit dürfte die Auswertung der Korrektheit liefern. Denn obwohl die Teilnehmer das Ergebnis ihrer Änderungen prüfen konnten und unvollständige Fehlerbehebungen direkt ins Auge fallen sollten, haben viele Teilnehmer unvollständige Lösungen erstellt. Mehr als die Hälfte der Studenten behoben nur eine Instanz des geklonten Fehlers in *FrozenBubble*. Einem Drittel gelang es in *Pacman* nicht beide geklonte Fehlerinstanzen zu beheben. Sogar bei den Experten, die im Rahmen eines Klon-Seminars teilnahmen und Klone erwartet haben mussten, lieferten vier von 16 Teilnehmern unvollständige Lösungen. Auswertungen unserer automatischen Aufzeichnungen ergeben, dass oft nicht nach weiteren Fehlervorkommnissen gesucht wurde, nachdem eines gefunden und behoben wurde. Auch unsere Gespräche mit den Teilnehmern bestätigen dies. Ein Teilnehmer gab sogar an, dass seine (unvollständige) Lösung so offensichtlich korrekt war, dass er sie nicht einmal am laufenden Programm getestet hat. Zwar leidet die statistische Auswertung auch hier unter der geringen Teilnehmerzahl, allerdings scheinen die ersten Ergebnisse einer Wiederholung des Experiments in Zusammenarbeit mit der University of Alabama unsere Beobachtungen zu bestätigen.

## 3 Konklusion und Ausblick

Obwohl unsere Ergebnisse in diesem ersten Experiment zu den Auswirkungen von Klonen auf die Leistung von Programmierern statistisch keine Signifikanz aufweisen und ihre Verallgemeinerbarkeit weiterer Untersuchungen bedarf, deuten sie darauf hin, dass ein reales Risiko für unvollständige Fehlerbehebungen bei geklonten Fehlern besteht. Dabei ist zu beachten, dass wir aus pragmatischen Gründen eher kleine Systeme und Klone mit nur zwei Vorkommen verwendet haben. Bei größeren und schwerer zu verstehenden Systemen und häufiger kopierten Klonen ist zu vermuten, dass das Risiko solcher unvollständigen Fehlerbehebungen größer ist.

Um derartige Fehler in der Software-Entwicklung zu vermeiden, liegt es nahe den Entwicklern Informationen über die Klone im System bereitzustellen. Erste Untersuchungen hierzu deuten jedoch darauf hin, dass einfache textuelle Berichte hierzu offenbar nicht geeignet sind. So zeigt sich, dass Entwickler solche Klonberichte durchaus fehlinterpretieren oder sogar ignorieren. Werden sie jedoch korrekt eingesetzt, ist ein Nutzen messbar [2].

Hieraus ergibt sie die Frage, in welcher Weise Kloninformationen effektiv und verständlich bereitgestellt werden können. In der Vergangenheit wurden verschiedene Werkzeuge zur Integration von Kloninformationen in Entwicklungsumgebungen vorgeschlagen. Allerdings wurde deren Nutzwert in der Regel nicht untersucht. Wir werden unsere Forschung daher fortsetzen, indem wir, aufbauend auf unserem ersten kontrollierten Experiment, einzelne Maßnahmen zur IDE-Integration von Kloninformationen in weiteren Experimenten auf ihre Eignung hin untersuchen werden. Hierbei soll die Vermeidung unvollständiger Fehlerbehbungen im Mittelpunkt stehen. Zunächst werden wir untersuchen inwieweit sich solche unvollständigen Änderungen vermeiden lassen, indem Klonpositionen im Quelltexteditor permanent angezeigt werden und Funktionen angeboten werden, mit denen zu den jeweiligen Kopien gesprungen werden kann. Zukünftige Werkzeuge für das Klonmanagement sollten auf solchen empirischen Untersuchungen basieren, um einen tatsächlichen Nutzen sicherzustellen.

## Literatur


[1] J. Carver, D. Chatterji, and N. A. Kraft. On the need for human-based empirical validation of techniques and tools for code clone analysis. In *Int. Workshop on Software Clones*, pages 61–62. ACM, 2011.

[2] D. Chatterji, J. Carver, B. Massengil, J. Oslin, and N. Kraft. Measuring the efficacy of code clone information in a bug localization task: An empirical study. In *International Symposium on Empirical Software Engineering and Measurement*, pages 20–29, 2011.

[3] J. Harder and R. Tiarks. A controlled experiment on software clones. In *20th International Conference on Program Comprehension*, pages 219–228. IEEE, 2012.