Empirische Grundlagen für das Klonmanagement

Jan Harder, Rainer Koschke
Universität Bremen, Arbeitsgruppe Softwaretechnik
http://www.informatik.uni-bremen.de/st
{harder, koschke}@informatik.uni-bremen.de

Abstract: Software-Systeme enthalten in der Praxis häufig einen hohen Grad redundanten Quelltextes - so genannte *Klone*. Von solcher Software-Redundanz wird angenommen, dass sie bei der Entwicklung und Wartung von Software zu zusätzlichem Aufwand und Problemen führt.

Ein Ziel der Klonforschung besteht darin, Methoden und Werkzeuge für den Umgang mit Klonen oder deren Vermeidung zu entwickeln. Für das so genannte *Klonmanagement* fehlen jedoch noch wichtige empirische Grundlagen, die Aufschluss über die Ursachen und Auswirkungen von Klonen geben und eine informierte Basis für die Entwicklung von Methoden und Werkzeugen bieten. In diesem Positionspapier nennen wir, ausgehend vom Stand der Forschung, offene Forschungsfragen, deren Beantwortung wesentliche Grundlagen für das Klonmanagement liefern wird.

1 Einleitung

Unter einem *Software-Klon* versteht man ein Programmfragment, das eine sehr hohe Quelltextähnlichkeit zu einem anderen existierenden Fragment aufweist, z.B. weil es durch *Copy & Paste* – gegebenenfalls mit nachfolgenden kleinen Modifikationen – entstanden ist. Änderungen im Original müssen gegebenenfalls in den Kopien nachgezogen werden. Das Finden und Abgleichen der Redundanzen führt zu zusätzlichem Arbeitsaufwand. Hieraus ergibt sich die Forderung nach Methoden und Werkzeugen zum geplanten Behandeln von existierenden Klonen - dem *Klonmanagement* [LPM+97, Gie03, Gie07, Kon07].

Die Ursachen für das Entstehen von Klonen sowie ihre Entwicklung sind individuell verschieden und ihre Entfernung ist nicht in jedem Fall sinnvoll und möglich [KBLN04, KSNM05, KG06]. Das Klonmanagement beinhaltet daher auch die Frage, welche Maßnahmen für den Umgang mit bestimmen Klonen gewählt werden sollten [KG06]. In der Praxis ist zu erwarten, dass die Menge an Redundanzen in existierender Software zu groß ist, um von den Entwicklern überblickt zu werden (berichtet wird, dass 7%-23% [Bak95, KMM+96, LPM+97] des Codes redundant sind, in einem extremen Fall sogar 59% [DRD99]). Das Klonmanagement benötigt daher Werkzeuge, die in der Lage sind, relevante Klone aus der Masse hervorzuheben und geeignete Maßnahmen vorzuschlagen.

Lague et al. [LPM+97] und Giesecke [Gie03] unterscheiden zwischen drei Formen des Klonmanagements: das präventive, kompensative und korrektive Klonmanagement. Das *präventive* Klonmanagement strebt an, neue Klone zu vermeiden. Das *kompensative* Klon-

management zielt darauf ab, negative Auswirkungen von Klonen, die im System verbleiben, zu vermeiden. Das *korrektive* Klonmanagement entfernt Klone.

Eine wesentliche Voraussetzung für das Klonmanagement ist es, Ursachen und Auswirkungen von Klonen zu verstehen. Dieses Wissen ist notwendig, um Klone nach diesen Kriterien zu kategorisieren und um operationale Definitionen zu schaffen, auf deren Grundlage Werkzeuge arbeiten können. Bislang haben sich jedoch nur wenige Studien mit dieser Frage befasst und weitere Untersuchungen sind notwendig.

Wir liefern in diesem Artikel einen Überblick über den Stand der Forschung in diesen Bereichen und arbeiten aktuell offene Forschungsfragen heraus, deren Beantwortung Grundlagen für das Klonmanagement schaffen soll. Unser Beitrag baut auf den Ergebnissen des Dagstuhl Seminars *Duplication, Redundancy, and Similarity in Software* zum Thema Klonmanagement [Kos07] auf und präzisiert die dort aufgeworfenen Probleme zu konkreten Forschungsfragen.

2 Ursachen von Klonen

Die Ursachen für redundanten Code sind bis dato nur in zwei Untersuchungen genauer beleuchtet worden. In ihrer ethnographischen Studie beobachteten Kim et al. [KBLN04] neun Programmierer bei ihrer Programmierarbeit. In den untersuchten Fällen waren Programmierer oft gezwungen, Code zu kopieren, weil die verwendete Programmiersprache nicht ausdrucksstark genug ist. Insbesondere auch so genannte Querschnittsbelange (Cross-Cutting Concerns) lassen sich in den verbreiteten Sprachen nicht leicht ausdrücken. In vielen Fällen stellten Programmierer Restrukturierungen zurück, bis sie mehrere Male Code kopiert haben. Erst dann sahen sie sich in der Lage, die Unterschiede zu erkennen und heraus zu faktorisieren. Häufig benutzten sie den kopierten Text auch als eine Schablone, die sie dann im Kontext der Kopie anpassten.

Kapser und Godfrey [KG06] haben Klone in großen Systemen untersucht und dabei so genannte Muster des Klonens gefunden, d.h. Fälle, in denen Klonen als bewusste Implementierungsstrategie angewandt wird. Das Muster *Verzweigung* wird eingesetzt, wenn die Entwicklung ähnlicher, bereits existierender Funktionalität angeschoben werden soll, die im weiteren zeitlichen Verlauf jedoch eine unabhängige Weiterentwicklung erfahren soll. *Schablonen* werden gebildet, wenn existierendes Verhalten wieder verwendet werden soll, geeignete Abstraktionsmechanismen jedoch fehlen. *Kontextspezifische Anpassungen* sind anzutreffen, wenn existierende Funktionalität wieder verwendet werden soll, die aber an einen spezifischen Kontext angepasst werden muss.

Die Neigung von Programmierern, Code zu duplizieren, hat vermutlich noch weitere Ursachen: unzureichende Entwicklungswerkzeuge, organisatorische Mängel im Entwicklungsprozess, unzureichende Ausbildung oder fehlendes Problembewusstsein der Programmierer, Zeitdruck oder mangelnde Dokumentation von Architektur und Quelltext. Die Studie von Kapser und Godfrey [KG06] hat diese Möglichkeiten nicht untersucht, da nur der Quelltext untersucht wurde, ohne mit den Entwicklern selbst über die Gründe zu sprechen. Die Programmierer in der Studie von Kim et al. [KBLN04] waren Forscher aus kleinen

Teams. Ob deren Fähigkeiten mit dem Durchschnittsprogrammierer vergleichbar sind und ob eine Forschungsumgebung mit Kleingruppen die Situation industrieller Unternehmen widerspiegelt, ist fraglich.

Frage 1 *Was sind die Ursachen für Copy & Paste-Programmierung? Und insbesondere, wie häufig sind diese Ursachen?*

Die Kenntnis der Ursachen gäbe uns die Möglichkeit, entsprechende Gegenmaßnahmen zu entwickeln, also Klonen mit präventiven Maßnahmen zu begegnen. Die Kenntnis über die relative Häufigkeit der Ursachen würde es uns erlauben, die Gegenmaßnahmen entsprechend zu priorisieren.

3 Folgen von Klonen

Der Redundanz durch Klone werden in der Wartung oft negative Konsequenzen nachgesagt. Es gibt jedoch nur wenige Studien, die die Folgen von Klonen empirisch untersucht haben.

Eine dieser Studien, die sich mit dem Zusammenhang von Redundanz mit Fehlern und Wartung befasst ist [MNK+02]. Darin wurde ein 20 Jahre altes Cobol-System von ca. einer Million LOC auf Redundanz untersucht und Klone mit der Mindestlänge von 30 Zeilen Code betrachtet. Deren maximale Zeilenlänge pro Modul wurde mit der Fehlerzahl und der Änderungshäufigkeit (als eine Approximation von Wartungsaufwand) verglichen.

Die Gegenüberstellung der maximalen Länge eines Klons in Modulen und der Fehlerhäufigkeit der Module über die letzten sechs Jahre ergibt einen interessanten nichtlinearen Zusammenhang. Module mit einem Klon von mehr als 200 Zeilen haben am häufigsten Fehler. Interessanterweise haben aber auch solche Module, die keinen Klon länger als 30 Zeilen haben, eine relativ hohe Fehlerrate. Bei Modulen mit maximaler Klonlänge von 30-199 ist die Fehlerhäufigkeit dagegen geringer. Der Zusammenhang zwischen maximaler Klonlänge eines Moduls und seiner Fehlerrate entspricht daher eher einer Badewannenkurve. Außerdem konnte beobachtet werden, dass sich Module mit Klonen häufiger ändern als jene ohne Klone. Beim Vergleich von maximaler Klonlänge und Änderungshäufigkeit ergibt sich ein linearer Zusammenhang.

Diese Studie hat einen ersten Blick mit recht überraschenden Ergebnissen auf diese Zusammenhänge geworfen, die Verallgemeinerung der Ergebnisse ist jedoch noch fraglich. Zum einen wurden Fehler und Änderungshäufigkeit nur in Beziehung zur maximalen Klonlänge und nicht etwa der Menge und den Gesamtumfang der Klone gesetzt. Dann wurde die Änderungshäufigkeit als eine pragmatische, jedoch simplifizierende Approximation für Wartung verwendet. Die genauen Ursachen für die Zusammenhänge wurden nicht weiter analysiert. Die Klone wurden in der letzten Version des betrachteten Zeitraums ermittelt. Die Messung der Fehler und Änderungshäufigkeit erstreckt sich jedoch über den gesamten Zeitraum von sechs Jahren. Die Redundanz einzelner Module könnte jedoch über die Zeit zu- oder abgenommen haben, was die gemessenen Korrelationen verfälschen kann. Schließlich wurden die Ergebnisse an nur einem System ermittelt. Ob ähnliche Resultate auch bei anderen Systemen zu erzielen sind, ist nicht geklärt.

Es wird oft angenommen, dass mit zunehmender Redundanz die Fehlerrate ansteigt und die Wartbarkeit nachlässt. Diese Studie – trotz ihrer Einschränkungen – zeigt jedoch, dass möglicherweise kein derart einfacher Zusammenhang existiert.

Chou et al. [CYC+01] gingen in einer Studie der Hypothese nach, dass, wenn ein Modul einen Fehler enthält, die Wahrscheinlichkeit größer ist, dass es noch mehr Fehler enthält. In ihrer Untersuchung von Linux und OpenBSD stellten sie fest, dass sich diese Annahme besonders häufig bestätigt, wenn Verstöße gegen Schnittstellenregeln in Kombination mit Klonen auftreten. Sie erklären dieses Phänomen mit der Vermutung, dass Programmierer glauben, existierender Code sei korrekt und kann ohne Risiken kopiert werden. Wenn jedoch im Original ein Fehler vorhanden ist bzw. der Kontext, in den hinein kopiert wird, nicht passt, entstehen weitere Fehler.

Li et al. [LLMZ06] verwenden Klonerkennung, um Fehler durch *Copy & Paste* mit inkonsistenter Umbenennung zu finden. In ihrer Studie mehrerer größerer Open-Source-Projekte enthielten tatsächlich 13 % der von ihnen diagnostizierten potenziell fehlerhaften Code-Fragmente echte Fehler. In weiteren 14 % der Fälle ist noch offen, ob es sich um Fehler handelt, da diese Stellen noch von den Entwicklern untersucht werden. Die verbleibenden 73 % sind Falsch-Positive.

Eine Studie, die nicht nur die potentiellen, sondern die tatsächlichen Auswirkungen von Klonen untersucht und mit konkretem Zahlenmaterial unterlegt, würde uns helfen, die Grenzen tolerierbarer Redundanz auszuloten. Im Sinne der drei möglichen Grundarten des Klonmanagements (präventiv, kompensativ und korrektiv) muss hierbei näher untersucht werden, in welchen Situationen Klone zu negativen Konsequenzen führen.

Fehler, die bereits beim Kopieren entstehen, erfordern präventives Klonmanagement. Dies ist dann der Fall, wenn ein Code-Fragment nicht ausreichend an seine neue Umgebung angepasst wird oder bereits vorhandene Fehler kopiert werden, wie es Chou et al. [CYC+01] vermuten. Fehler, die durch inkonsistente Änderungen entstehen, erfordern dagegen kompensative oder korrektive Maßnahmen.

Frage 2 *Wie oft führt das Kopieren in einen anderen Kontext zu Fehlern? Wie oft werden bereits vorhandene Fehler an weitere Stellen kopiert? Wie oft führen inkonsistente Änderungen an Klonen zu Fehlern?*

Frage 3 *Wie korreliert die Fehlerrate von geklontem Code zu der von nicht-redundantem Code im Allgemeinen?*

Allgemein sollte auch der Wartungsaufwand in Bezug auf einzelne Klone untersucht werden, anstatt dies nur auf der Ebene von Modulen zu tun. Um verlässliche Daten über den zusätzlichen Wartungsaufwand, den Klone verursachen, zu erhalten.

Frage 4 *Wie korreliert das Ausmaß der Copy & Paste-Programmierung mit dem Änderungsaufwand?*

Für das werkzeugunterstützte Klonmanagement stellt sich insbesondere die Frage, ob und wie Klone hinsichtlich ihres Risikopotentials und der geeigneten Gegenmaßnahmen bewertet werden können.

Frage 5 *Was sind die Merkmale von Klonen, die einen negativen Einfluss auf Wartbarkeit haben? Wie können wir Klone danach kategorisieren? Wie können wir sie erkennen?*

4 Folgen der Klonentfernung

Bei der Betrachtung der Risiken, die von Klonen ausgehen, können wir uns nicht darauf beschränken, die Effekte zu untersuchen, die sich aus ihrem Vorkommen im Quelltext ergeben. Auch die Maßnahmen, die für das Klonmanagement in Frage kommen, müssen hinsichtlich ihrer Risiken betrachtet werden. Dies gilt insbesondere für das korrektive Klonmanagement, das einen Eingriff in den Code bedeutet und somit selbst – wie jede andere Änderung am Quelltext – ein Fehlerrisiko birgt. Die Gefahr, neue Fehler in existierenden Code einzuführen, kann ein Grund sein, von Umstrukturierungen abzusehen [Cor03].

Frage 6 *Wie hoch ist das Fehlerrisiko bei der Entfernung von Klonen? Wie verhält sich dieses zum Risiko, Änderungen an Klonen inkonsistent durchzuführen? Wann überwiegen die Nachteile von Klonen die Risiken ihrer Entfernung?*

Die Entfernung von Klonen soll die Wartbarkeit der Software verbessern. Bei der Entfernung von vollkommen identischen Klonen (*Typ-1-Klone*) ist dies in der Regel durch ein *Extract-Method*-Refactoring nach Fowler [Fow99] möglich. Oft unterscheiden sich Klone jedoch in Details, wie der Benennung von Bezeichnern (*Typ-2-Klone*) oder einzelnen Anweisungsfolgen (*Typ-3-Klone*). Die Entfernung solcher Klone und die Einführung einer unifizierten Lösung erfordern die Parametrisierung des Codes. In der Folge wird der Code komplizierter, was der Verständlichkeit abträglich ist. Daher stellt sich die Frage, ob und in welchen Fällen die Klon-Entfernung die Wartbarkeit sogar eher verschlechtert.

Frage 7 *Führen Klone zu mehr Aufwand und zu einem erhöhten Risiko bei der Änderung als vereinheitlichte Lösungen?*

5 Konklusion und Ausblick

Wir benötigen empirische Untersuchungen zur den Ursachen und Auswirkungen von Klonen sowie zu den in Frage kommenden Maßnahmen des Klonmanagements. Bisherige Untersuchungen wie die von Monden et al. [MNK+02] haben nur globale Kennziffern erhoben und in einen Zusammenhang gebracht.

Andere haben einzelne Klone inspiziert, stützen sich aber entweder auf nur wenige kleine Open-Source-Systeme [KBLN04, KSNM05] als Untersuchungsgegenstand oder berichten von allgemeinen Erfahrungswerten ohne konkrete Zahlen zu nennen Kapser und Godfrey [KG06].

Diese Studien haben jedoch einen wichtigen Beitrag geleistet, indem sie gezeigt haben, dass die Ursachen und Auswirkungen von Klonen offenbar komplex und vielfältig sind und dass zwischen Klonen und negativen Auswirkungen kein einfacher Zusammenhang zu bestehen scheint.

In unserer zukünftigen Forschung werden wir daher auf drei Aspekte ein besonderes Augenmerk legen:

1. Die Auswirkungen müssen *individuell für einzelne Klone und Klonklassen* untersucht werden. Nur auf dieser Ebene lässt sich den zuvor aufgeworfenen Fragen zu den individuellen Auswirkungen von Klonen und ihren spezifischen Merkmalen nachgehen.

2. Die Fragen nach den Ursachen und Folgen lassen sich nicht allein durch das Betrachten von einzelnen Versionen einer Software beantworten. Wir werden Klone daher im Kontext ihrer *evolutionären Entwicklung* betrachten. Erst die Entwicklung, die ein Klon über die Zeit genommen hat, erlaubt Rückschlüsse darauf, wie er entstanden ist, wie mit ihm umgegangen wurde und welche Probleme er hervorgerufen hat. Kennt man die Evolution eines Klons, kann man ihn mit anderen historischen Daten wie Fehlern und Änderungen am Code in Bezug setzen. Nur auf dieser Ebene lassen sich Merkmale von Klonen, die in Bezug zu ihren Ursachen und Auswirkungen stehen, herausarbeiten.

3. Zukünftige Studien müssen auf einer *umfangreicheren Basis von untersuchten Systemen* fußen. Wir erwarten, dass insbesondere in größeren Systemen ($\geq$ 1 Mio. LOC) und industriell entwickelter Software andere Rahmenbedingungen und Ursachen für Klone bestehen. Größere Codebasen können nicht mehr von einzelnen Entwicklern überblickt werden, in größeren Organisation ist der Zugang der Entwickler zum Code strenger reglementiert und in der Industrie kommen andere Entwicklungsmodelle und -werkzeuge zum Einsatz als in der Open-Source-Entwicklung. Wie sich diese Faktoren auf die Redundanzen auswirken, ist bislang nicht untersucht.

Die Fragen, die wir in diesem Beitrag nennen, haben auch über die Belange des Klonmanagements hinaus Relevanz. Noch immer fehlt der Klonforschung eine empirische Grundlage, die die Bestrebungen zur Erkennung und Behandlung von Klonen rechtfertigt. Darüber hinaus kann die Problematik der Redundanz softwareentwickelnden Unternehmen mit harten Fakten noch besser ins Bewusstsein gerufen werden.

Literatur

[Bak95] Brenda S. Baker. On Finding Duplication and Near-Duplication in Large Software Systems. In L. Wills, P. Newcomb und E. Chikofsky, Hrsg., *Second Working Conference on Reverse Engineering*, Seiten 86–95, Los Alamitos, California, July 1995. IEEE Computer Society Press.

[Cor03] J.R. Cordy. Comprehending Reality: Practical Challenges to Software Maintenance Automation. In *International Workshop on Program Comprehension*, Seiten 196–206. IEEE Computer Society Press, 2003.

[CYC+01] A. Chou, J. Yang, B. Chelf, S. Hallem und D. R. Engler. An empirical study of operating system errors. In *In Symposium on Operating Systems Principles*, Seiten 73–88, 2001.

[DRD99] Stéphane Ducasse, Matthias Rieger und Serge Demeyer. A Language Independent Approach for Detecting Duplicated Code. In *International Conference on Software Maintenance*, Seiten 109–118, 1999.

[Fow99] Martin Fowler. *Refactoring: improving the design of existing code*. Addison Wesley, 1999.

[Gie03] Simon Giesecke. Clone-based Reengineering für Java auf der Eclipse-Plattform. Diplomarbeit, Carl von Ossietzky Universität Oldenburg, Department für Informatik, Abteilung Software Engineering, Germany, 2003.

[Gie07] Simon Giesecke. Generic modeling of code clones. In Rainer Koschke, Ettore Merlo und Andrew Walenstein, Hrsg., *Duplication, Redundancy, and Similarity in Software*, number 06301 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2007. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany.

[KBLN04] M. Kim, L. Bergman, T. Lau und D. Notkin. An Ethnographic Study of Copy and Paste Programming Practices in OOPL. In *International Symposium on Empirical Software Engineering*, Seiten 83–92. IEEE Computer Society Press, 2004.

[KG06] Cory Kapser und Michael W. Godfrey. "Clones considered harmful"considered harmful. In *Working Conference on Reverse Engineering*, Seiten 19–28, 2006.

[KMM+96] K. Kontogiannis, R. De Mori, E. Merlo, M. Galler und M. Bernstein. Pattern Matching for clone and Concept Detection. *Automated Software Engineering*, 3(1/2):79–108, June 1996.

[Kon07] Kostas Kontogiannis. Managing Known Clones: Issues and Open Questions. In Rainer Koschke, Ettore Merlo und Andrew Walenstein, Hrsg., *Duplication, Redundancy, and Similarity in Software*, number 06301 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2007. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany.

[Kos07] Rainer Koschke. Survey of Research on Software Clones. In Rainer Koschke, Ettore Merlo und Andrew Walenstein, Hrsg., *Duplication, Redundancy, and Similarity in Software*, number 06301 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2007. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany.

[KSNM05] Miryung Kim, Vibha Sazawal, David Notkin und Gail C. Murphy. An Empirical Study of Code Clone Genealogies. In *European Software Engineering Conference and Foundations of Software Engineering (ESEC/FSE*, 2005.

[LLMZ06] Z Li, S Lu, S. Myagmar und Y. Zhou. Copy-Paste and Related Bugs in Large-Scale Software Code. *IEEE Computer Society Transactions on Software Engineering*, 32(3):176–192, Marz 2006.

[LPM+97] B. Lague, D. Proulx, J. Mayrand, E.M. Merlo und J. Hudepohl. Assessing the benefits of incorporating function clone detection in a development process. In *International Conference on Software Maintenance*, Seiten 314–321, 1997.

[MNK+02] A. Monden, D. Nakae, T. Kamiya, S. Sato und K. Matsumoto. Software quality analysis by code clones in industrial legacy software. In *IEEE Symposium on Software Metrics*, Seiten 87–94, 2002.