19 C^{4}D oder Wie ich lernte, mit Code Clones zu leben

Udo Borkowski
IT Consultant, Noppiusstr. 3, 52062 Aachen
udo.borkowski@gmx.de

19.1 Einleitung

Das Auftreten von Code-Duplikaten (auch „Code Clones“ oder „Copy/Paste Code“ genannt) in Softwaresystemen wird allgemein als ein Qualitätsproblem betrachtet [1][3], das insbesondere auch bei der Wartung von großen Systemen an Bedeutung gewinnt. Das übliche Vorgehen zum Lösen diese Problems ist das (automatische) Finden der Code-Duplikate („Clone Detection“) und das anschließende „Herausfaktorieren“ des gemeinsamen Codes („Clone Removal“).

Verschiedene Techniken und Werkzeugen sind bekannt, die der Clone Detection dienen (eine Übersicht gibt z.B. [2]) und dabei auf unterschiedlichsten Konzepten beruhen. Zum Clone Removal werden im wesentlichen Makros [1] sowie Refactoring [5] eingesetzt [6].

Im folgenden werden einige Probleme aufgezeigt, die sich aus dem oben skizzierten Vorgehen, insbesondere durch den Clone Removal Schritt, ergeben. Es wird ein neues Verfahren „Code Clone Change Conflict Detection" (CCCCD oder C^4D) beschrieben, das diese Probleme vermeidet und weitere Vorteile aufweist.

19.2 Probleme bei Clone Detection/Removal

Ein wesentliches Problem von Code-Duplikaten liegt darin, dass i.A. nicht sichergestellt werden kann, dass während der Wartung eines Codefragmentes die entsprechenden Änderungen in allen eventuell vorhandenen Duplikaten durchgeführt werden. Hauptgrund hierfür ist, dass im Normalfall der zu ändernde Code keinen Hinweis auf Duplikate enthält. So hängt es von dem Hintergrundwissen des Entwicklers bzw. seiner „Weitsicht" ab, ob auch Duplikate geändert werden.

Durch den Einsatz von Makros bzw. durch Extrahieren von gemeinsamen Code in Funktionen (etwa durch das Refactoring „Extract Method" [4]) während des Code Removals wird diese Gefahr gebannt, da der Entwickler jetzt nur eine Codestelle ändern muss.

Allerdings hat dieses Code Removal Vorgehen auch Nachteile, von denen einige kurz aufgezeigt werden sollen:

Falsche Duplikate: die durch Clone Detection gefundenen und automatisch entfernten Duplikate können falsch in dem Sinne sein, dass sie nur zufällig aus gleichem Code bestehen. Dies kann zwei Folgen haben:

- wird die Inkonsistenz erkannt, so muss vor der Änderung des falschen Duplikates die Herausfaktorisierung rückgängig gemacht werden (z.B. durch „Inline Method" [4]).
- wird die Inkonsistenz nicht erkannt, führt der Entwickler wahrscheinlich einen Bug ein, da die Änderungen auch für die „Aufrufe" der falschen Duplikate durchgeführt werden.

Code-Duplikate-Auflösung eingeschränkt: nicht alle Code-Duplikate können mit diesem Vorgehen aufgelöst werden. Beispielsweise gibt es Schwierigkeiten bei überlappenden Codefragmenten.

Generierter Code: Das Auflösen von Codeklonen, die durch Generatoren entstanden sind, kann zu Schwierigkeiten beim späteren, inkrementellen Einsatz der Generatoren führen.

Hoher Aufwand beim Clone Removal: eine systematische, manuelle Entfernung von allen erkannten Code-Duplikaten ist in einem typischen Legacy-System in der Regel zu aufwendig, abgesehen von möglichen Fehlern, die ein manuelles Verfahren mit sich bringen würde.

Hoher Aufwand bei Wartung: Eine automatische Entfernung von Code-Duplikaten führt u.a. zu dem bereits oben beschriebenen Problem der falschen Klone. Um dies zu lösen, müssten die Entwickler bei Änderungen in Funktionen/Makros stets alle Aufrufkontexte überprüfen, da sie nicht davon ausgehen dürften, dass Funktionen entsprechend ihrer Semantik aufgerufen werden (sondern eventuell nur aufgrund ihres gemeinsamen Codes).

Verständlichkeit: die Verständlichkeit/Lesbarkeit von Code mit Makro- bzw. Funktionsaufrufen, insbesondere von automatisch generierten, kann sich gegenüber dem Original verschlechtern.

19.3 Code Clone Change Conflict Detection (C^4D)

Das Verfahren „Code Clone Change Conflict Detection" (CCCCD oder C^4D) vermeidet die oben beschriebenen Schwierigkeiten, indem es die während der Clone Detection gewonnenen Informationen nicht benutzt, um die Duplikate zu entfernen, sondern um zukünftige Änderungen potentieller Klone zu erkennen. Für diese Änderungen wird dann geprüft, ob sie konsistent in allen zugehörigen Duplikaten durchgeführt sind. Im Einzelnen wird dabei wie folgt vorgegangen:

1. Mit Hilfe von Clone Detection Werkzeugen wird eine Liste von potentiellen Klonpaaren erzeugt. Ein Klonpaar enthält dabei zwei Listen von Codefragmenten, die (exakte oder ähnliche) Kopien voneinander sind, sowie ggf. Parameterisierungsinformationen.
2. Bei nachfolgenden Änderungen am Code wird geprüft, ob sich diese Änderungen auf Codestellen beziehen, die in einem Codefragment eines potentiellen Klonpaares enthalten sind.
3. Wird eine solche Änderung entdeckt, folgt nun die Überprüfung, ob die Änderung auch entsprechend im Gegenstück-Codefragment des Klonpaares durchgeführt wurde. Sollte dies nicht der Fall sein („Change Conflict"), wird dem Entwickler die Situation mitgeteilt und er kann folgendermaßen darauf reagieren:
   (a) Der Entwickler entscheidet, dass die Änderung beide Codestellen betrifft und zieht die Änderung im Gegenstück ebenfalls nach. Hierbei wird er sich ggf. entscheiden, Makros bzw. Refactorings wie „Extract Method" einzusetzen, um so im folgenden die parallele Wartung mehrerer Codestellen zu vermeiden. An dieser Stelle liegt es in der Hand des Entwicklers, andere Nachteile von Code-Duplikaten (wie etwa Codegröße, mangelnde Abstraktion,...) zu beheben. Die Change Conflict Erkennung dient hier sozusagen als Auslöser.
   (b) Der Entwickler entscheidet, dass die Änderung nur die von ihm geänderte Codestelle betrifft.

Dies kann prinzipiell zwei Gründe haben:

i. Es handelt sich um einen falschen Klon (s.o). In diesem Fall wird das potentielle Klonpaar als „falsch“ markiert und bei den weiteren Prüfungen nicht mehr beachtet.
ii. Es handelt sich um eine Modifikation eines echten Klons, die aber nur für einen Teil des Klonpaares gilt. In diesem Fall wird die Modifikation als weiterer Parameter dem Klonpaar hinzugefügt, wobei der Parameterwert des Gegenstück-Codefragmentes leer ist.

Anmerkungen

Das Verfahren ist weitgehend unabhängig von den in 1. erwähnten Clone Detection Werkzeugen. Es ist darüberhinaus der parallele Einsatz mehrerer Werkzeuge möglich, so dass auf der Vereinigungsmenge der erkannten potentiellen Klonpaare gearbeitet werden kann. Überlappende Codefragmente sind dabei kein Problem. Auch ist das Verfahren weitgehend von der Programmiersprache unabhängig, soweit für diese ein passendes Clone Detection Werkzeug existiert.

Zu 2. ist anzumerken, dass eine Änderung einer Codestelle ggf. mehrere Klonpaare betreffen kann. Dies kann beispielsweise dann der Fall sein, wenn ein Clone Detection Werkzeug die transitive Hülle einer Klonklasse als Klonpaare ausgibt [2] oder wenn überlappende Clonefragmente vorliegen.

Bei 3. ist zu beachten, dass die Art der Prüfung von dem Klontyp [2] abhängt. Besonderes Augenmerk ist dabei auf Typ 2 und Typ 3 Klons und „ähnlichen“ Klons zu richten. Hierzu wird zuerst versucht, durch Vergleich der ursprünglichen Klonpaar-Codefragmente eine geeignete Parameterisierung des Klonpaares zu ermitteln. Die Parameterisierung geht dabei über eine reine Umbenennung von Namen hinaus, sondern deckt auch „Modifikationen“ ab. Dann wird bei den geänderten Codefragmenten geprüft, ob die Änderungen durch entsprechende Anpassungen von Code oder Parametern abdeckbar sind. Im Zweifelsfall wird eine „Ungleichheit“ gemeldet, die der Entwickler dann ggf. manuell auflösen kann.

Bei 3.a. ist erwähnenswert, dass eine Änderung im Gegenstück-Codefragment ggf. eine Änderung in einem weiteren Klonpaar darstellt, so dass sich hier der beschriebene Prozess ab 2. wiederholt. Dies ist insbesondere dann der Fall, wenn ein Klonpaar zu einer Klonklasse von mehreren Klonpaaren gehört.

Im Fall 3.b. sind zusätzlich Vorkehrungen zu treffen, dass durch die ausgewählten Maßnahmen andere Klonpaare, die eventuell mit dem betroffenen Klonpaar eine Klonklasse bilden, beachtet werden. Auch können dem Entwickler hier weitere Eingriffsmöglichkeiten angeboten werden, etwa das „Splitten“ in zwei Klonepaare u.ä.

19.4 Zusammenfassung

Es wurde eine neue Herangehensweise an das Code-Duplikat-Problem vorgestellt, die wesentliche Schwierigkeiten der klassischen Verfahren vermeidet. Darüberhinaus bietet sie weitere Vorteile, wie weitgehende Unabhängigkeit von Programmiersprachen und den verwendeten Clone Detection Werkzeugen, Behandeln von „nichtwohlgeformten“ Duplikaten, Einsetzbarkeit in großen Softwaresystemen usw.

Das neue Verfahren stellt darüber hinaus einen relativ einfachen Weg dar, den Umgang mit Code-Duplikaten in den Entwicklungsprozess zu integrieren und so „mit Code Clones zu leben“.

19.5 Ausblick

Das beschriebene Verfahren wird zur Zeit in einem Werkzeug umgesetzt. Von dessen Einsatz in produktiven Systemen werden weitere Einsichten in die Möglichkeiten und Grenzen von C^4D erwartet.

19.6 Literaturverzeichnis

[1] Baxter, Ira D., Andrew Yahin, Leonardo Moura, Marchelo Sant'Anna und Lorraine Bier: Clone Detection Using Abstract Syntax Trees. in Proc. ICSM, 1998.

[2] Bellon, Stefan: Vergleich von Techniken zur Erkennung duplizierten Quellcodes. Diplomarbeit, Universität Stuttgart, 2002.

[3] Ducasse, Stéfane, Matthias Rieger und Serge Demeyer: A Language Independent Approach for Detecting Duplicated Code. in Proc. ICSM, 1999.

[4] Fowler, Martin: Refactoring: Improving the Design of Existing Code. Addison-Wesley, 1999

[5] Opdyke, W.: Refactoring Object-Oriented Frameworks. PhD thesis, University of Illinois at Urbana-Champaign, 1992.

[6] Rysselberghe, F. Van and S. Demeyer: Evaluating Clone Detection Techniques. In proceedings of the International Workshop on Evolution of Large Scale Industrial Applications (ELISA). pages 25-36, 2003