# Wartung von automatisch generiertem Java-Code nach einer Software-Migration

Christian Becker, Uwe Kaiser
pro et con Innovative Informatikanwendungen GmbH, Dittesstraße 15, 09126 Chemnitz
{christian.becker, uwe.kaiser}@proetcon.de

## Abstract

Im Zeitraum vom September 2014 bis zum August 2016 realisierte pro et con eine Software-Migration von BS2000 nach Linux [1]. Eine Kernkomponente des Projektes war die Konvertierung von ca. 360 COBOL-Programmen nach Java unter Nutzung des firmeneigenen Tools *COBOL to Java*-Converter *CoJaC*. Im Zielsystem fungiert Eclipse als Entwicklungsumgebung. Seit dem erfolgreichen Abschluss des Projektes unterstützt pro et con den Kunden bei der Wartung und Weiterentwicklung. Dieser Beitrag beschreibt dabei gesammelte Erfahrungen unter dem Kontext, dass immer noch Vorbehalte bezüglich der Wartbarkeit von automatisch generiertem Code existieren.

## 1 Programmverstehen

Das Projekt entsprach einer typischen Legacy-Migration, bei der neben der Konvertierung von COBOL-Programmen nach Java die folgenden Migrationspfade toolgestützt durchlaufen wurden:

- *ISAM*- bzw. *LEASY*-Files in *Oracle*-Datenbanktabellen,
- *IFG*-Bildschirmmasken in Weboberflächen und
- *SDF*-Jobs in *Perl*-Programme.

Im Ergebnis sollte das Anwendungssystem in Arbeitsteilung zwischen den ehemaligen COBOL-Programmierern des Kunden und den Mitarbeitern von pro et con gewartet und weiterentwickelt werden. Voraussetzung dafür ist das Programmverstehen. Beim Programmverstehen geht es um die fachlichen ("Warum") und die technischen ("Wie") Zusammenhänge in den Programmen und in der gesamten Anwendung. Der Kunde steuerte dazu das fachliche Wissen über das Anwendungsgebiet bei. Das Wissen über Abläufe, fachliche Hintergründe und Strukturierung von Daten der ursprünglichen COBOL-Entwickler war unverzichtbar. Der Wissenstransfer, bereits vorhandene Dokumentationen sowie die Befragung der COBOL-Entwickler zu den verwendeten Programmiermustern führte zu einem notwendigen Verständnis der Programme.

Auf der anderen Seite sahen sich die COBOL-Programmierer des Kunden nach Projektende und dem Beginn der Wartung mit neuen Programmiersprachen und einer neuen Entwicklungsumgebung konfrontiert. pro et con steuerte das Wissen über die neuen Sprachen und die Entwicklungsumgebung der Zielplattform bei, insbesondere auch zu den Eigenschaften des generierten Java-Codes, der auf einer 1:1-Konvertierung der COBOL-Programme basiert. Dabei erfolgte ein ständiger Wissenstransfer in beide Richtungen. Diese Aufgabenteilung hat sich bewährt, befördert das Programmverstehen auf beiden Seiten und garantiert eine kurzfristige Fehlerkorrektur und eine zügige Realiserung von Weiterentwicklungsaufträgen.

## 2 Einsatz moderner Tools

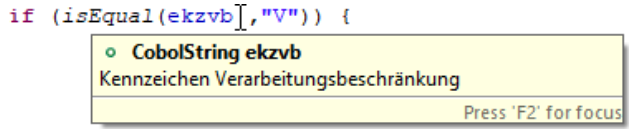
Ein Vorteil der neuen Entwicklungsumgebung wurde bereits kurz nach der Migration deutlich: Der Einsatz moderner Entwicklungswerkzeuge verbessert die Möglichkeiten von Wartung und Weiterentwicklung. Legacy-Werkzeuge vergleichbarer Funktionalität besitzen im Allgemeinen nicht die ausgereifte Funktionalität und diesen Komfort und sind zusätzlich kostenpflichtig zu nicht unerheblichen Konditionen. Im Folgenden werden ausgewählte Werkzeuge und ihr Einsatz in der Wartung skizziert:

**Profiler:** Einige konvertierte Java-Programme besaßen nach der Migration eine lange Laufzeit. Es war eine Wartungsaufgabe, die Ursachen zu analysieren und Verbesserungen zu realisieren. Die Analyse erfolgte mit dem Profiler *JVisualVM* als Teil des *Java Development Kit* (JDK). Die Analyseergebnisse zeigten bspw. bei einem Programm eine sehr hohe Datenbanklast. Ursache war, dass dieses Programm aufgrund seiner noch in COBOL definierten Arbeitsweise auch bei der Änderung nur eines Datensatzes unnötigerweise immer die vollständige Datenbanktabelle auslas (Im Original war dies ein COBOL-File). Und das mehrfach im Programm. Schreiboperationen existierten nur wenige. Auf die Antwort der Datenbank wartete das Java-Programm dadurch ca. 85% der gesamten Laufzeit. Die Lösung bestand hier in der Einführung einer Caching-Bibliothek. Geänderte Datensätze werden direkt in die Datenbank geschrieben und im Cache aktualisiert. Die Verarbeitung von Lesezugriffen erfolgt über den Cache. Dies führte zu einer drastischen Laufzeitreduktion, da das Programm nach der Optimierung nur noch 27% seiner Gesamtlaufzeit auf den Datenbankresponse „wartet".

**Javadoc:** Auch wenn, wie im vorliegenden Projekt, die Programmdokumentation schon vor der Migration aktuell und detailliert war, ergab sich nach dem Anschluss ein Bedarf für die Nachdokumenation. Die Ursache dafür ist, dass neben den ursprünglichen COBOL-Entwicklern nun auch Java-Entwickler den Code warten. Für letztere erschließen sich ohne zusätzliche Informationen die in COBOL üblichen Programmiermuster, Namensschemata etc. nicht. *Eclipse* bietet die Möglichkeit zur Anzeige von *Javadoc*-Kommentaren. Damit können bspw. die für Java-Entwickler ungewohnten Variablennamen zusätzlich kommentiert werden:

```
/** Kennzeichen Verarbeitungsbeschränkung */
public CobolString ekzvb = createString(1);
```

In diesem Beispiel wurde die Variable `ekzvb` im Quellcode um einen Javadoc-Kommentar ergänzt.




In Eclipse werden solche Kommentare direkt im Quell-

code angezeigt und tragen so zum Programmverständnis bei. Mit dieser Methode wurden z.B auch Programmabläufe nachdokumentiert, deren COBOL-Original aus mit `GO TO` realisierten Schleifen bestand und deren Programmiermuster im Zielsystem für Java-Entwickler ungewohnt sind.
Zusätzlich existieren in der IDE nützliche Funktionalitäten wie z.B. die Anzeige der *Call-Hierarchie* oder das Nachvollziehen der Programmlogik mit Hilfe von Debugging.

**Oracle SQL Developer:** Migrationsentscheidungen beeinflussen die Wartbarkeit des Zielcodes nach der Migration. Bei der Datenmigration von ISAM- bzw. LEASY-Files in Datenbanktabellen wurden z.B. gepackte COBOL-Felder (`COMP-3`) in ein lesbares, Datenbank-konformes Oracle-Format überführt. Diese Entwurfsentscheidung verursachte im Migrationsprojekt zunächst einen erhöhten Aufwand, welcher sich aber in der Wartung bezahlt macht. Damit ist es möglich, mit Standarddatenbanktools wie z.B. dem *Oracle SQL Developer* SQL-Abfragen auf dem Datenbestand auszuführen. Die Recherche nach bestimmten Datensatzkonstellationen wird dadurch vereinfacht. Im Legacy-System war dies auf Fileebene aufwendiger, da je nach Komplexität der Abfragen Jobs oder kleinere COBOL-Programme geschrieben werden mussten.

## 3 Wartbarkeit und Weiterentwicklung

**Programmstruktur:** Grundsätzlich handelt es sich bei Migrationsprojekten von pro et con um *1:1-Migrationen*. Programmstrukturen, Anweisungen und Datenfelder sind in den generierten Programmen strukturell und semantisch identisch zu den Originalen. Dadurch finden sich die ursprünglichen Entwickler auch im neuen, generierten Code zurecht. Diese Eigenschaft hat sich als sehr nützlich in der Wartung herausgestellt. Vor allem in den ersten Monaten nach einer Migration vergleichen die Entwickler oft den originalen COBOL-Code mit dem generierten Java-Code, um sich an die neue Syntax zu gewöhnen.

**GO TO:** Aus der Wartung haben sich auch neue Erkenntnisse in Bezug auf den generierten Code ergeben. So lassen sich z.B. Verbesserungen für die Konvertierung von `GO TO`-Statements ableiten. Eine gelungene Lösung ist die Konvertierung von `GO TO`-Sprüngen zu `EXIT`-Paragraphen als `return`. Die Lesbarkeit im generierten Code ist gegeben. Komplexere Sprünge werden als `switch`-/`case`-Statement abgebildet, wobei alle Sprungmarken (Paragraphen und Sections) als `case`-Statement realisiert sind. Diese komplexen Abläufe können damit funktional korrekt in Java abgebildet werden, besitzen allerdings Verbesserungspotential. Diese Erfahrungen sollen zur Verbesserung von CoJaC durch eine Überarbeitung der Konvertierung von `GO TO` in diesen Spezialfällen genutzt werden.

**Datenbank:** Nach Abschluss der Migration werden Weiterentwicklungen zunächst in dem Stil erfolgen, in dem die migrierten Programme vorliegen. Dennoch muss im Einzelfall entschieden werden, wann der "COBOL-lastige" Stil verwendet wird und wann eine alternative Technologie sinnvoller ist. Bei der Datenmigration wurden Files (ISAM/LEASY) in Datenbanktabellen überführt, wobei die konvertierten Programme aus "ihrer Sicht" immer noch auf Files zugreifen. Befehle wie `READ` und `WRITE` werden in einer separaten Schnittstelle durch `SELECT` und `INSERT` abgebildet. Bei ausgewählten Weiterentwicklungen wurden neue Datenbanktabellen jedoch aus Sicht der Programme nicht als COBOL-Files inkl. komplexer Dateiverarbeitung eingebunden, sondern über eine separate Datenbankschnittstelle (*DAO*-Pattern), was zu einer besseren Les- und Wartbarkeit führte.

**Refactoring:** Ob bzw. wie Refactoring durchgeführt werden kann, hängt vom Programmierstil und der Code-Strukturierung ab. Wie in der Literatur empfohlen, sollten alle Refactoring-Maßnahmen innerhalb der Wartung in kleinen Schritten und nur bei entsprechender Testabdeckung erfolgen. Ein mögliches Refactoring ist z.B. die Codereduktion. So konnte z.B. ein migriertes Programm auf ca. 20% seiner ursprünglichen Größe reduziert werden. Es enthielt eine Menge Code, welcher bei früheren Erweiterungen kopiert und nur leicht angepasst wurde. Eine Verallgemeinerung der Datenstrukturen und der betroffenen Methoden (z.B. durch Einführung von Methodenparametern und einheitlicher Benennung von Variablen) führte zu dieser Reduktion. Bei solchen Fällen handelt es sich jedoch um Ausnahmen. Häufiger kamen in dem Projekt große Datenstrukturen (z.B. Satzstrukturen von Dateien) vor, welche kopiert und in leicht veränderter Form in den Programmen verwendet werden. Solche Strukturen sind keine identischen Klone. Im Zuge der Wartung müssen diese z.B. bei einer Erweiterung der originalen Strukturen ebenfalls gefunden und angepasst werden. Der "Ausbau" solcher Kopien verursacht einen höheren Wartungsaufwand (Anpassung von Bezeichnern und Zugriffen). Allerdings handelt es sich hierbei um ein lohnendes Refactoring, da in Zukunft Änderungen nur noch an einer Stelle durchgeführt werden müssen. Auch weniger komplexe Refactoring-Maßnahmen (z.B. die Reimplementierung von `GO TO`-Schleifen als `for` oder `while`) führen auf lange Sicht zu einer Verringerung der Wartungskosten.

## 4 Zusammenfassung

Die am Projekt beteiligten Mitarbeiter von pro et con warten das Projekt mit der Effizienz und Qualität „normaler" Javaprojekte. Für COBOL-Jünger ist automatisch generierter Java-Code aus Prinzip unwartbar. Für COBOL-Programmierer, die am migrierten Java-Projekt weiter beteiligt sind, ist automatisch generierter Java-Code wartbar, wenn sie willens und befähigt sind, Java zu erlernen. Für Java-Entwickler ist automatisch generierter Java-Code wartbar, wenn sie willens sind, sich die aus der COBOL-Migration begründeten Besonderheiten des Java-Codes anzueignen. Beide Personengruppen realisieren dies im weiteren Verlauf der Wartung zunehmend effizienter.

## Literaturverzeichnis

[1] Kaiser, U.; Uhlig, D.: Erfolgreiche BS2000-Migration. In: it-dayli.net, Oktober 2016.