Messung und Nachdokumentation eines uralten COBOL-Systems zwecks der Migration zu Java

von Harry M. Sneed
ANECON GmbH, Vienna Austria
Institut für Wirtschaftsinformatik, Universität Regensburg, Bayern
Email: Harry.Sneed@T-Online.de

Abstrakt: Der folgende Beitrag beschreibt die Analyse einer uralten COBOL Applikation als Voraussetzung für eine Migration zu Java. Zunächst wurde der Code gemessen um Basisdaten für die Aufwandsschätzung und Risikoanalyse zu gewinnen. Anschließend wurde der Code nochmals zwecks der Nachdokumentation bearbeitet. Aus den COBOL-Sourcen wurden sämtliche Verweise auf externe Objekte – Calls, IO-Operationen und DB-Zugriffe, sowie alle interne Verzweigungen, alle Regel und alle Datenreferenzen – abgeleitet und in ein Software-Repository überführt, aus dem es möglich war Modulaufrufe, Datenflüsse, Datenbankzugriffspfade und Datenquerverweise abzufragen und graphisch darzustellen. Darüber hinaus wurden einzelne Programme und Dateien prototypweise automatisch transformiert. Die COBOL Anweisungen wurden 1:1 in Java-Methoden, die VSAM-Dateien 1:n in relationale Tabellen umgesetzt. Zum Schluss wurden die Migrationsaufwände geschätzt und eine Risikoanalyse durchgeführt.
Keywords*: Code-Analyse, Reverse Engineering, Repositories, Reengineering, COBOL to Java Transformation, VSAM to SQL Transformation, Aufwandschätzung, Risikoanalyse*

Hintergrund des Migrationsvorhabens

Der Anwender, um den es hier geht, ist eine Versicherungsanstalt in Wien. Die Anstalt ist sowohl für die Alters- als auch für die Krankenversicherung der Eisenbahner und Bergwerker verantwortlich. Das IT-System zur Verwaltung der Versicherungen wurde in den 70er Jahren mit der Sprache COBOL-68, dem Dateisystem VSAM und einem eigenen TP-Monitor in einer IBM DOS-VSE Umgebung implementiert. Es besteht aus 2084 COBOL-Modulen und 986 Copy-Members mit zusammen 1.407.127 Codezeilen. Dazu kommen 196 VSAM-Dateien und 1.139 JCL-Prozeduren. Das System wird von 14 Entwicklern gewartet und weiterentwickelt. Das sind knapp über 100.000 Codezeilen pro Wartungsmitarbeiter.

Der Plan war, die 196 VSAM-Dateien in SQL-Tabellen einer relationalen POSTGRES-DB zu versetzen, die 1.139 JCL-Prozeduren in Linux Skript neuzuschreiben, den eigenen TP-Monitor durch das JBOSS Framework zu ersetzen und den COBOL-Code in circa 3.000 Java-Klassen zu transformieren. Im Hinblick auf die beschränkten Mittel soll die ganze Migration nicht mehr als eine Million Euro an externen Kosten verursachen.

Migrationsstudie

Als Vorbereitung zu dieser Migration wurde eine Migrationsstudie bestellt. Die Studie hatte die Aufgabe zu klären:

a) ob die Migration im Rahmen der geplanten Mittel durchführbar ist
b) ob das Altsystem sich dokumentieren lässt
c) ob die COBOL-Programme automatisch nach Java transformierbar sind
d) ob die VSAM-Dateien sich automatisch in SQL-Tabellen überführen lassen
e) wie hoch die Kosten sein könnten
f) welche Risiken mit der Migration verbunden sind

Jedes angestrebte Studienergebnis sollte dokumentiert und dem Projektsteuerungsausschuss zwecks der Entscheidungsfindung präsentiert werden. Eine derartige Studie sei unverzichtbar um eine Migration auf dem richtigen Wege zu bringen [1]. Allerdings dürfte diese Studie nicht mehr als 20 Personentage in Anspruch nehmen.

Um die erste Aufgabe zu erfüllen wurde entschieden den Code mit dem *CodeAuditor* Messungswerkzeug zu messen. Das dürfte nicht mehr als zwei Tage dauern. Um zu demonstrieren, dass das System sich nachdokumentieren lässt, wurde entschieden, das Ganze gleich automatisch nach zu dokumentieren und daraus ein Repository aufzubauen. Diese Reverse-Engineering-Aktion dürfte nicht mehr als vier Personentage kosten. Für die dritte Aufgabe sollten mit einem bestehenden Transformator drei Beispielprogramme nach Java transformiert werden. Die daraus resultierenden

Klassen sollten den Entwicklern des Kunden vorgelegt werden um zu prüfen, ob sie mit dem Java-Code zurecht kommen. Die vierte Aufgabe sollte auf ähnlicher Weise erledigt werden. Anhand von drei VSAM-Dateien sollte gezeigt werden wie die Daten in relationale Tabellen umgesetzt werden. Für diese beiden Aufgaben wurden insgesamt 10 Personentage vorgesehen. Die Schätzung der Migrationsaufwände – Aufgabe sollte mit dem Werkzeug *SoftCalc* bewältigt werden. Die Größenmaße des Codes aus der Codemessung waren in die Schätzdatenbank zu übernehmen und die Produktivitätswerte der Prototyp-transformationen zu verwenden. Das Ergebnis soll eine Schätzbandbreite mit einer minimalen und maximalen Aufwandsgrenze geben. Hierfür waren zwei Tage vorgesehen. Die letzte Aufgabe sollte innerhalb der verbleibenden zwei Tage erfüllt werden. Ein Analytiker sollte die Risiken zusammentragen und nach Wahrscheinlichkeit und Schadensausmaß bewerten. Zum Schluss sollten die Ergebnisse im Rahmen einer Halbtagespräsentation vorgestellt werden.

Codemessung

Zur Codemessung wurden die 2.024 COBOL-Module und 986 Copy Members auf 26 Verzeichnisse verteilt, ein Verzeichnis für jedes der 25 Teilsysteme plus ein Verzeichnis für die gemeinsamen Datenstrukturen bzw. Copy Members. Für die Assembler-Module und JCL-Prozeduren wurden getrennte Verzeichnisse aufgebaut. Die Hauptarbeit – einen halben Tag – ging darauf die Source-Verzeichnisse aufzubauen. Die Messung selbst dauerte nur eine Stunde. Die Ergebnisse waren für jedes Modul, jedes Teilsystem und für die Software insgesamt ein Metrikbericht mit 52 Quantitätsmaßen, 8 Komplexitätsmaßen und 8 Qualitätsmaßen. Dazu kamen die Größen in Anweisungen, Function-Points und Data-Points. Eine Metrik-Exportdatei wurde auch für jedes Teilsystem erstellt. Diese dienten als Verbindung zu dem Schätzwerkzeug. Am Ende blieb sogar ein Tag übrig um die Messungsergebnisse auszuwerten und für den Anwender zu protokollieren [2].

Nachdokumentation

Für die Nachdokumentation war der Source-Code von der Messung her bereits aufbereitet. Es blieb nur übrig die Dokumentationswerkzeuge – *COBRedoc*, *ASMRedoc* und *JCLRedoc* – anzustoßen. *COBRedoc* und *ASMRedoc* erstellen pro Modul fünf Dokumente – eine Tabelle der IO und Call-Schnittstellen, ein Modulinhalts-verzeichnis, sprich einen Baum der internen Prozeduren – Paragraphen und Sections, ein Struktogramm der Ablauflogik, eine Datenflusstabelle mit den Eingaben und Ausgaben von jedem Codeblock und ein Datenverzeichnis der verwendeten Daten. Außerdem wird pro Teilsystem eine CSV-Datei sämtlicher interner und externer Relationen erstellt. Mittels dieser Querverweise ist es möglich zu erkennen welche Code-Entitäten welche anderen Code-Entitäten besitzen oder benutzen. Nach der Codeanalyse wurden die Programmstrukturen automatisch in Klassen-strukturen umgesetzt. Für jede Datengruppe bzw. für jede abgeschlossene Prozedur wird eine Klasse definiert. Die Datenklassen und die Subroutinenklassen werden in der Klassen-hierarchie zu Superklassen ummodelliert, die von den anderen Klassen geerbt werden. Das Verfahren und das Werkzeug dazu wurde auf der ICSM-Konferenz 2002 vorgestellt [3].

Da die Nachdokumentation, einschließlich der Transformation des prozeduralen Architektur-modells in ein OO-Architekturmodell, voll auto-matisiert ist, konnte diese Nachdokumentations-arbeit in einem Tag abgeschlossen werden [4]. Am folgenden Tag wurde das Repository mit den Exportdateien der 25 Teilsysteme plus den globalen Datenstrukturen bevölkert. Anschließend wurden die beispielhaften Darstellungen – Klassendiagramme und Sequenzdiagramme – aus dem Repository erstellt.

Probetransformation der COBOL-Programme

Die Probetransformation der COBOL Programme fand mit dem COBOL2Java Konvertierungs-programm der Firma Shark statt. Drei Programme wurden ausgewählt – ein großes, ein kleines und ein mittleres. Das Konvertierungswerkzeug setzt die COBOL-Anweisungen 1:1 in Java-Methoden um. Für jeden COBOL-Anweisungstyp, einschließlich der GOTO-Anweisung, gibt es eine entsprechende Java-Methode. Die COBOL-Daten werden in entsprechende Java-Datentypen umgewandelt. Jedes COBOL-Modul ergibt eine statische Klasse. Vererbt werden nur die globalen Datenstrukturen, bzw. die COPY-Strukturen. Die generierten Java-Klassen sind um etwa 50% grösser als die ursprünglichen COBOL-Module, aber sie sind eine korrekte, lauffähige Abbildung des COBOL-Codes. Die ausgewählten Module konnten ein einem Tag umgesetzt werden. Dennoch wurden vier weitere Tage gebraucht um das Werkzeug anzupassen. Denn das Transformationswerkzeug wurde für strukturiertes COBOL-85 konzipiert und dieser Code war unstrukturiertes COBOL-68. Fast jede vierte prozedurale Anweisung ist ein GO TO. Hinzu kommen alte COBOL Konstrukte wie

GOTO DEPENDING ON und PERFORM THRU, die nur umständlich in Java abzubilden sind. Dennoch ist es gelungen alle drei Programme so umzusetzen, dass der Java-Compiler sie umwandeln konnte.

Probetransformation der VSAM-Dateien

Die Probetransformation der VSAM-Dateien sollte ebenfalls mit einem Werkzeug der Firma Shark durchgeführt werden. Da es im Gegensatz zu Datenbanken kein Schema für Dateien gibt, muss die Inhaltsbeschreibung der Dateien aus den Satzstrukturen des Programmcodes entnommen werden. In diesem System gab es ein getrenntes Copy Member für jede VSAM-Datei. Das vereinfachte die Sache etwas, aber Schwierigkeiten gab es genug mit überlagerten Datendefinitionen, wiederholten Datengruppen und unverträglichen Datentypen. Es stellte sich daher heraus, dass es nicht möglich war das Datenkonvertierungs-werkzeug in der vorgesehen Zeit auf den erforderlichen Stand zu bringen. Die Aufgabe wurde nicht erledigt. Es blieb bei einem Prototypkonzept der Datenkonversion.

Schätzung der Migrationsaufwände

Um zu kontrollieren, ob der Aufwand für die Migration in dem vorgesehenen finanziellen Rahmen bleiben würde, wurde er vorsichtshalber mit dem Tool *SoftCalc* geschätzt. Geschätzt wurde er nach COCOMO-II, Function-Point und Data-Point. Die Schätzung samt Übernahme der Metrik aus dem Tool SoftAudit und die Einstellung der Produktivität und Einflussfaktoren nahmen nur zwei Tage in Anspruch. Das Schätzergebnis war ein Schock für alle Beteiligten. Bei jeder Methode lag der erforderlichen Aufwand weit über dem geplanten Budget. Das lag vor allem daran, dass bei der Aufstellung des geplanten Budgets die Systemtestkosten nicht berücksichtigt wurden. Außerdem wurde davon ausgegangen, dass die automatisierte Transformation reibungslos abläuft. Die Datenmigration wurde auch grob unterschätzt. Demzufolge lagen die geschätzten Kosten 90 bis 125% über dem geplanten Budget. Laut der methodischen Schätzung war das Projekt weder zu den geplanten Kosten noch in der geplanten Zeit machbar.

Migrationsrisikoanalyse

Die letzte Aufgabe der Migrationsstudie war die Analyse der Risiken, die mit dem Projekt verbunden waren. Es wurden 16 Risiken identifiziert und aus ihrer Wahrscheinlichkeit und Auswirkung ein Risikofaktor errechnet [5]. Die Summe der einzelnen Risikofaktoren ergab einen Gesamtrisikofaktor von 2,42. Das Hauptrisiko war, wie oben geschildert, die Budget- und Zeitüberschreitung. Weitere beträchtliche Risiken waren die Performanz des 1:1 umgesetzten Codes, die Fehlerhaftigkeit der Datenkonversion, die mangelnde Erfahrung mit Open-Source Produkten und die Einarbeitung neuer Mitarbeiter in den COBOL ähnlichen Java-Code. Die Risikoanalyse bewies eindeutig, dass das Projekt unter den geplanten Bedingungen nicht zu dem gewünschten Erfolg führen konnte.

Schlussfolgerung

Die Migrationsstudie führte zur Aufgabe des originalen Planes und zur Verschiebung des Migrationsprojektes. Es ist bis heute noch keine Entscheidung gefallen, wie es weitergehen soll. Diese Fallstudie belegt wie schwierig es ist alte Softwaresysteme in eine moderne Umgebung zu überführen. Die optimale Lösung wäre, ein neues System mit einer neuen Datenhaltung zu entwickeln. Dafür fehlen jedoch, wie häufig der Fall ist, die nötigen Mittel.

Referenzen

[1] Brodie, M./Stonebraker, M.: Migrating Legacy Systems, Morgan Kaufmann Pub., San Francisco, 1995

[2] Kozlov/Koskinen/Sakkinen/Markkula: “Assessing Maintainability”, Journal of Software Maintenance & Evolution, Vol.20., Nr.1, Jan. 2008, S. 31

[3] Sneed, H.: “Tranforming procedural programs to OO-Class Structures”, Proc. of ICSM-2002, IEEE Computer Society Press, Montreal, Oct. 2002, S. 213

[4] Sneed, H.: Objektorientierte Softwaremigration, Addison-Wesley Verlag, Bonn, 1999

[5] Sneed, H./Winter, A./Ackermann, E.: Softwaremigration, dpunkt Verlag, Heidelberg, 2009