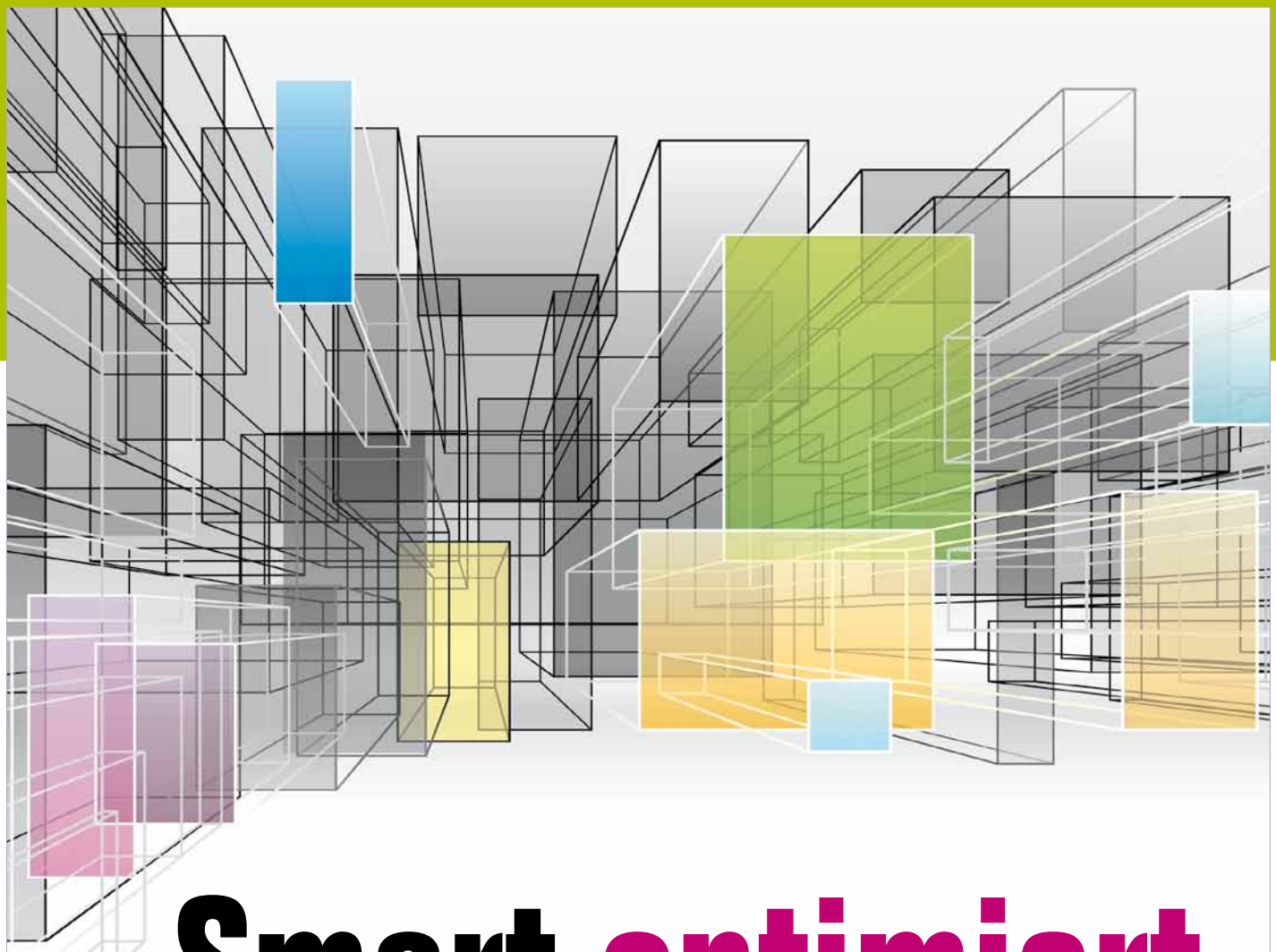
Business Technology

www.bt-magazin.de 3.2014 Heft 18

€ 4,90 S&S

Business Technology

Lean Innovation & IT Leadership



Smart optimiert

Home Automation im Enterprise

Gebaut für den Wandel

Industrie 4.0 durch IoT

Softwaremigrationsprojekte erfolgreich planen und realisieren

Aus alt mach neu – automatisch!

Bei der Ablösung von Legacy-Software gibt es drei Lösungen: die Einführung von Standardsoftware, die Neuentwicklung oder die Softwaremigration. Der nachfolgende Artikel beschreibt grundsätzliche Voraussetzungen, Technologien und Softwarewerkzeuge, die in ihrem abgestimmten Zusammenspiel zu einem erfolgreichen Softwaremigrationsprojekt führen.

AUTOR: DR.-ING. HABIL. UWE KAISER

Legacy-System (Altsystem) bezeichnet in der Informatik ein etabliertes, über Jahrzehnte historisch gewachsenes Individualsoftwaresystem. Sie sind bei Banken und Versicherungen, aber auch in anderen Wirtschaftsbereichen immer noch weit verbreitet und besitzen ein großes, wirtschaftliches Potenzial. Legacy-Systeme beinhalten das gesamte, unternehmensspezifische Wissen und die individuelle Geschäftslogik von Unternehmen. Damit stellen sie für die Firmen ein Alleinstellungsmerkmal im Wettbewerb dar. Weiterhin sind diese Systeme unternehmenskritisch, besitzen eine ausgereifte Kernfunktionalität und arbeiten zuverlässig im laufenden Betrieb.

Auf der anderen Seite ist Legacy-Software häufig charakterisiert durch eine antiquierte Datenhaltung, veraltete Benutzeroberflächen und historische Programmiersprachen, die modernen Anforderungen nicht genügen.

DOPPELCHARAKTER VON LEGACY-SYSTEMEN

Die Übersetzung des englischen Begriffs „Legacy“ spiegelt den deutlich gewordenen Doppelcharakter wider. Einerseits lässt sich „Legacy“ mit „Altlast“ oder „Hinterlassenschaft“ interpretieren. In dieser Interpretation ist der Begriff negativ besetzt. Eine alternative Interpretation lautet aber auch „Erbe“ und „Vermächtnis“ mit dem Fokus auf durchaus Erhaltenswertes. Legacy-Software wird zumeist auf Mainframes betrieben, die hohe Betriebs- und Lizenzkosten verursachen, in vielen Fällen jedoch durch preiswertere Architekturen ersetzt werden könnten. Durch jahrzehntelange Anpassung an sich ständig ändernde Geschäftsprozesse erodiert die Software, und die Wartungskosten steigen. Die ursprünglichen Entwickler der Software sind mit ihr gealtert, stehen kurz vor dem Ruhestand oder sind bereits Rentner. Nachwuchs mit fachlichem Interesse an diesen historischen Technologien ist schwer verfügbar. Im Rahmen dieses „Spannungsfelds“ entschließen sich Unternehmen zunehmend, ihre Legacy-Software abzulösen. Laut Experton Group (Tabelle 1) sehen immerhin 63,5 Prozent ausgewählter Unternehmen mit mehr als 1 000 Mitarbeitern die Modernisie-

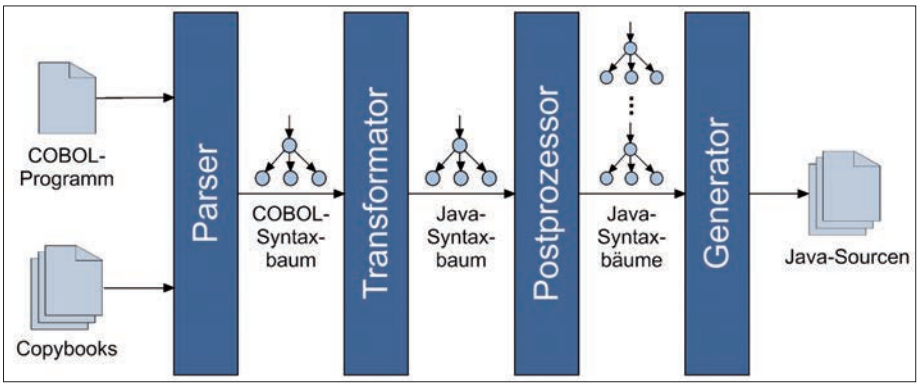


Abb. 1: Translatorarchitektur für COBOL-Java-Konvertierung, Quelle: pro et con GmbH

rung von IT-Systemen bzw. -Anwendungen als einen Schwerpunkt an.

WIE LEGACY ABLÖSEN?

Für die Ablösung von Legacy-Systemen existieren abstrahierend von Mischformen drei Möglichkeiten: Der Ersatz durch Standardsoftware, die Neuentwicklung oder die Softwaremigration. Unabhängig von der gewählten Methode ist die Ablösung von Legacy-Software immer ein komplexes Projekt, das mit entsprechendem Budget und Arbeitskräftevolumen geplant werden muss.

Beim Ersatz durch Standardsoftware verlässt sich der Anwender darauf, dass das gelieferte Produkt „fertig" und ausgreift ist und nur noch an seine speziellen Belange angepasst werden muss. Er spekuliert damit auf kurze Einführungszeiten und Kostenersparnis. Auch die Einheitlichkeit der Abläufe erscheint verlockend. Häufig ist Standardsoftware jedoch nicht so flexibel, dass sie die spezielle Geschäftslogik vollständig abbilden könnten. Die Folge ist, dass im Projekt überlagernde Prozesse aus der Nachbildung von Geschäftslogik in der Standardsoftware und Anpassen der Geschäftslogik an die Standardsoftware auftreten. Beim Ersatz durch Standardsoftware sollte deshalb unbedingt vorab geklärt werden, ob die innerbetrieblichen Prozesse durch die Standardsoftware abgebildet werden können bzw. welcher Aufwand getätigt werden muss, um diese Abbildung vorzunehmen.

Eine Neuentwicklung erscheint auf den ersten Blick als die optimale Lösung: Alle Komponenten der alten Software werden in moderner Hard- und Softwarearchitektur mit aktuellen Entwicklungsmethoden und -umgebungen neu aufgesetzt und zukunftssicher in der modernen Welt etabliert. Allerdings darf der Umfang der zu leistenden Arbeit nicht unterschätzt werden. Komplexe Systeme mit mehreren Millionen Lines of Code (LOC), die in Jahrzehnten entwickelt wurden, erfordern eine langjährige Projektlaufzeit mit hohen Kosten.

Eine weitere Alternative stellt die Softwaremigration dar. Softwaremigration beinhaltet neben der automatischen Konvertierung von Programmen aus antiquierten Programmiersprachen wie z. B. COBOL oder PL/I in moderne Sprachen wie C++ oder Java unter anderem auch die Integration in neue Betriebssysteme, die Modernisierung der Datenhaltung, der Benutzeroberflächen und der Softwarearchitektur unter Nutzung von ausgereiften Technologien und Migrationswerkzeugen.

Sowohl der Aufwand für eine Neuentwicklung als auch für die Softwaremigration eines Legacy-Systems lässt sich näherungsweise quantifizieren. Eine Möglichkeit dazu bietet die COCOMO-Methode (Constructive Cost Model) von Boehm [1], mit der ursprünglich der Aufwand für die Neuentwicklung von Softwaresystemen berechnet wurde und die

Welche der nachfolgenden Aktivitäten beurteilen Sie derzeit für Ihr Unternehmen als besonders wichtig?	
Top-10-Themen	**Häufigkeit der Nennungen**
Bessere Unterstützung von Geschäftsprozessen	77,8 Prozent
Optimierung der IT-Prozesse	74,6 Prozent
Automatisierung in der Bereitstellung von IT-Services	69,8 Prozent
Virtualisierung von Servern, Desktops und Storage	69,8 Prozent
Standardisierung von IT-Systemen bzw. Anwendungen	66,7 Prozent
Optimierung der IT-Organisation	66,7 Prozent
Modernisierung von IT-Systemen bzw. Anwendungen	63,5 Prozent
Modernisierung von IT-Komponenten im Rechenzentrum	49,2 Prozent
Green IT	44,4 Prozent
Zentralisierung der IT-Budgets	44,4 Prozent

Tabelle 1: Umfrage zur Modernisierung von Altsystemen, 63 Unternehmen in Deutschland mit mehr als 1 000 Mitarbeitern (Mehrfachantworten möglich), Quelle: Experton Group 2010–2011

nun auch für die Aufwandschätzung bei der Ablösung von Legacy-Software adaptiert wird. Basis ist das Mengengerüst des abzulösenden Softwaresystems. Es wird ermittelt, wie viele Anweisungen ein Programmierer bei einer Neuentwicklung pro Zeiteinheit formulieren kann. Berücksichtigt werden dabei auch solche Einflussfaktoren wie die Programmiersprache, die Entwicklungsumgebung und die Komplexität der Aufgabenstellung. Gegenübergestellt wird die Zeit, die benötigt wird, diese Menge von Anweisungen unter Nutzung von Migrationswerkzeugen im Rahmen eines Migrationsprojekts zu transformieren. Projekterfahrungen des Autors belegen ein Verhältnis von 1:8 zwischen Migration und Neuentwicklung. Umfasst ein Softwaremigrationsprojekt z. B. fünfzehn Mannjahre (eine durchaus übliche Größe), dann ist bei einer Neuentwicklung mit ca. 120 Mannjahren zu rechnen.

SOFTWAREMIGRATION MIT GEEIGNETEN WERKZEUGEN

Softwaremigration ist nur mit Migrationswerkzeugen sinnvoll. Viele existierende Werkzeuge unterstützen die automatisierte Migration der Datenhaltung (SAM- und ISAM-Files, proprietäre Datenbanksysteme), der Masken und eine Dialektanpassung der Programme beim Compilerwechsel. Diese Werkzeuge besitzen in Bezug auf die Technologie der Konvertierung unterschiedliche Qualität.

Weit verbreitet ist die Technologie der Musterkennung und -ersetzung auf Basis regulärer Ausdrücke im Sourcecode [2]. Das reicht aus, um z. B. eine Dialektanpassung in den Programmen beim Übergang zu einer anderen Compilerversion vorzunehmen oder Datenzugriffe in den Programmen anzupassen.

Eine wesentliche Entscheidung in Migrationsprojekten ist, ob die Programmiersprache auf dem Zielsystem erhalten bleibt oder nicht. COBOL ist die am weitesten verbreitete Programmiersprache in Legacy-Systemen. Die Praxis zeigt, dass COBOL trotz aller Kritiken an der Sprache meist auf dem Zielsystem erhalten bleiben soll. In diesem Fall muss lediglich eine Dialektanpassung erfolgen. Hier reichen die einfachen Operationen der Codemusterersetzung aus.

Die Beibehaltung von COBOL bei einer Migration lässt sich nur durch das „Beharrungsvermögen“ der COBOL-Entwickler erklären. Es existieren allerdings Alternativen der automatischen Konvertierung von COBOL bzw. anderen historischen Programmiersprachen in moderne Sprachen und Entwicklungsumgebungen. Bei dieser fachlich anspruchsvollen und kreativen Aufgabe versagen Codemusterersetzungen. Für derart komplexe Übersetzungsvorgänge wird der Einsatz spezieller Konvertierungswerkzeuge, so genannter Translatoren, erforderlich. Translatoren arbeiten in Analogie zu einem Compiler, der Quellprogramme über verschiedene Zwischenstufen in ausführbaren Zielcode übersetzt. Für ihre Entwicklung ist deshalb vertieftes Wissen der Compilertechnik notwendig. Da antiquiertes COBOL Legacy-Systeme dominiert, soll in **Abbildung 1** die Architektur eines Translators am Beispiel einer COBOL-Java-Konvertierung diskutiert werden.

Ein so genannter Parser liest das COBOL-Programm einschließlich der dazugehörigen Copybooks ein und erzeugt daraus einen internen Syntaxbaum, der das vollständige COBOL-Altprogramm repräsentiert. Da der Parser eine wesentliche Komponente in dieser Architektur darstellt, spricht man auch von „parserbasierten“ Werkzeugen. In dem Syntaxbaum sind z. B. auch alle Kommentare vermerkt, um sie bei Bedarf in den Zielcode wieder einfügen zu können. Hier und in weiteren Teilaufgaben unterscheidet sich der Translator vom Compiler, der Kommentare in seinem Übersetzungsprozess verwirft. Der nächste Konvertierungsschritt besteht in der Überführung des COBOL-Syntaxbaums in einen äquivalenten Java-Syntaxbaum durch einen Transformator. Er arbeitet auf Basis einer Abbildungsvorschrift, in der definiert ist, wie einzelne COBOL-Konstrukte (etwa Definitionen und Anweisungen) in Java abgebildet werden. Eine solche Vorschrift zu erarbeiten, ist eine der anspruchsvollsten Aufgaben in der Translatorentwicklung. Je genauer die Abbildung, desto größer ist die semantische Äquivalenz zwischen Quell- und Zielcode und umso einfacher lässt sich der Zielcode zukünftig warten.

Im dritten Schritt wird der komplexe Java-Syntaxbaum durch einen Postprozessor in einzelne Syntaxbäume zerteilt, die die Programmstruktur (Java-Klassen und -Packages) des zukünftigen Java-Programms repräsentieren. Auch dieser Konvertierungsschritt ist neu gegenüber dem allgemeinen Compilermodell. In dieser Phase erfolgt auch die Integration der Kommentare, sie werden als Knoten an die entsprechenden Stellen im Java-Syntaxbaum platziert.

Der letzte Konvertierungsschritt besteht in der Generierung von Java-Klassen und -Packages aus den einzelnen, internen Java-Syntaxbäumen durch einen Generator. Die eigentliche (Sprach-)Konvertierung vollzieht sich im Translator auf der Ebene von Syntaxbäumen und nicht auf Quelltextniveau. Die Praxistauglichkeit von Translatoren konnte bereits in mehreren Migrationsprojekten nachgewiesen werden. Die oben beschriebene Architektur wurde z. B. in einem konkreten Translator „CoJaC“ (COBOL to Java Converter) [3] realisiert, der u. a. in einem BS2000-

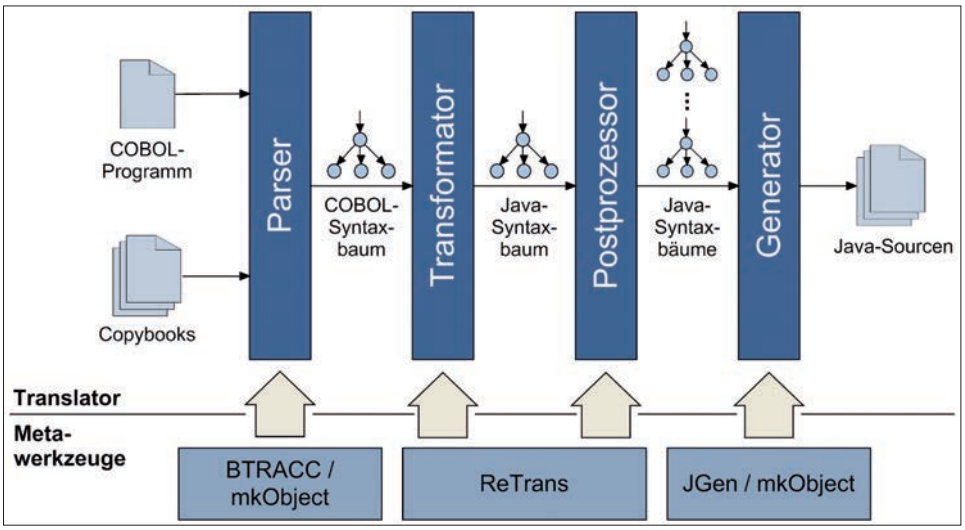


Abb. 2: Toolbox für die Softwaremigration, Quelle: pro et con GmbH

Migrationsprojekt bei ZIVIT [4] eingesetzt wird. Als weiteres Beispiel dient das BS2000-Migrationsprojekt ARNO (Application Relocation to New Operating System) der Firma Amadeus Germany [5]. Auf der Basis der oben vorgestellten Architektur wurde ein Translator S2C (SPL to C++) für die BS2000-Programmiersprache SPL (System Programming Language) und ein Translator S2P (SDF to Perl) für die Jobsteuersprache SDF (System Dialog Facility) entwickelt. Mit beiden Translatoren wurden Millionen LOC im ARNO-Projekt nach C++ bzw. Perl konvertiert. Diese Konvertierung erfolgte fehlerfrei mit hohem Automatisierungsgrad. Nach Aussagen der Amadeus-Entwickler ist der generierte Zielcode wartbar. Die Notwendigkeit von Iterationen im Projekt ergab sich daraus, dass parallel zur Migration das System von Amadeus weiterentwickelt wurde, sodass bereits konvertierte Sourcen erneut konvertiert werden mussten. Das war mit S2C fehlerfrei in beliebigen Iterationen möglich.

Häufig wird bei Sprachkonvertierungen ein Automatisierungsgrad von 100 Prozent versprochen. Erfahrungen des Autors können das nicht bestätigen. Realistisch betrachtet gilt hier eher eine 80:20-Regel, d. h., 80 Prozent der Programme können automatisch konvertiert werden, während 20 Prozent manuell nachbearbeitet werden müssen. Der Grund besteht nicht darin, dass es keine automatische Lösung gibt, sondern liegt vielmehr in einer Aufwand-Nutzen-Betrachtung: Wenn ein Konstrukt nur drei- oder viermal in mehreren Millionen LOC vorkommt, dann ist dieses schneller manuell umgestellt, als den Translator für die Behandlung dieses Konstrukts zu erweitern.

Die vorgestellte Translatorarchitektur lässt sich auf die Entwicklung von Migrationswerkzeugen allgemein übertragen, unabhängig davon, ob es sich um Sprach-, Masken- oder Datenmigration handelt. Voraussetzung ist immer, dass eine formale Beschreibung von Basis und Ziel existiert. Als Beispiel dient MaTriX (Maskenmigration in serverbasierten Systemen), ein Werkzeug für die Migration von ASCII-orientierten Maskensystemen in moderne HTML- und JavaScript-basierte Oberflächen, das entsprechend dieser Architektur arbeitet. In [6] wird eine erfolgreiche Maskenmigration bei MAN Nutzfahrzeuge München mit dem System MaTriX beschrieben. MaTriX ist konfigurierbar und kann für die Migration unterschiedlicher Maskensysteme (z. B. SCREEN COBOL [HP NonStop] oder IFG [BS2000]) eingesetzt werden.

Zusammenfassend kann gesagt werden, dass immer dann, wenn eine formalisierte Beschreibung von Basis- und Zielsystem existiert, parserbasierte Migrationswerkzeuge auf Grundlage der vorgestellten Architektur entwickelt werden können. Der Vorteil dieser Werkzeuge gegenüber der Codemusterersetzung besteht in der vollständigen, redundanzfreien Abbildung von Basis- und Zielsystem in Form von Syntaxbäumen. Erst damit besitzt man Zugriff auf die vollständigen Informationen von Basis- und Zielsystem während der Transformation. Das ermöglicht eine fehlerfreie, automatische Konvertierung in beliebigen Iterationen unter Beachtung semantischer Äquivalenz zwischen Basis- und Zielsystem.

WERKZEUGE UND METAWERKZEUGE

Parserbasierte Migrationswerkzeuge sind komplex. Die Beherrschung der Komplexität im Entwicklungsprozess besteht in der Nutzung von Metawerkzeugen. Mit Metawerkzeugen werden Werkzeuge für die Softwaremigration generiert. Ein Beispiel für ein Metawerkzeug ist das seit Jahrzehnten im UNIX-Umfeld verbreitete „Yacc“ (yet another compiler compiler) [7], mit dem Informatikstudenten seit Generationen in diversen Praktika konfrontiert wurden. „Yacc“ ist ein Parsergenerator, der eine „Yacc“-typische Grammatikbeschreibung einer Sprache einliest und daraus einen Parser für die Sprache generiert.

Folgende Anforderungen existieren bei der produktiven Werkzeugentwicklung an einen Parsergenerator:

Beherrschung der Komplexität gelingt durch den Einsatz von Metawerkzeugen.

- Verarbeitung komplexer Grammatiken (insbesondere bei antiquierten Quellsprachen wie z. B. COBOL und PL1)
- Verwendung der in Nutzerdokumentationen vorgegebenen Grammatiksyntax ohne aufwändige Umstellung
- Verarbeitung mehrerer Dialekte in einem Werkzeug
- Einfache Integration neuer, syntaktischer Konstruktionen ohne Umbau der existierenden Grammatik

Die als Open Source verfügbaren Parsergeneratoren („Yacc“, „COCO/R“, „PCCTS“ ...) werden diesen Anforderungen nicht vollständig gerecht. Insbesondere die Forderung, die in Dokumentationen verfügbare Syntax ohne Umstellungen zu verwenden und Erweiterungen komfortabel integrieren zu können, ist nicht erfüllt. Eine Erweiterung einer komplexen „Yacc“-Grammatik z. B. führt in der Regel zu einer Reihe von Konflikten, die durch aufwändige Umstellungen beseitigt werden müssen. Diese Argumente waren die Motivation für die Entwicklung eines neuen Parsergenerators „BTRACC“ (Backtracking Compiler Compiler), der den oben genannten Anforderungen an einen Parsergenerator genügt, und mit dem alle in diesem Artikel genannten Migrationswerkzeuge generiert wurden. In **Abbildung 2** werden weitere, firmeneigene Metawerkzeuge dokumentiert, die in einer „Toolbox für die Softwaremigration“ zusammengefasst wurden [8].

Mit dieser Toolbox ist es möglich, die Teilkomponenten einer Translatorentwicklung durch Metawerkzeuge zu unterstützen:

- mkObject: Mit mkObject werden aus deklarativen Beschreibungen von Syntaxbäumen automatisch Programme generiert, mit denen Syntaxbäume aufgebaut und manipuliert werden können.
- ReTrans: Ist ein Metawerkzeug, das auf Basis einer deklarativen Abbildungsbeschreibung Transformatoren generiert, welche die Überführung von Syntaxbäumen aus dem Basis- in das Zielsystem vornehmen. In der Grafik handelt es sich um einen Transformator für die Abbildung von COBOL-Syntaxbäumen in Java-Syntaxbäume. Ändert man z. B. die Abbildungsbeschreibung für die Zielsyntaxbäume in ReTrans in C++, dann generiert ReTrans einen Transformator für eine COBOL-C++-Abbildung.
- JGen: Ist ein Werkzeug, welches aus Java-Syntaxbäumen Java-Quellcode generiert.

Die Metawerkzeuge kommen bei der Entwicklung von neuen Migrationswerkzeugen bzw. bei der Anpassung von existierenden Werkzeugen an neue Anforderungen in Migrationsprojekten zum Einsatz.

Erst Metawerkzeuge erlauben eine Entwicklung neuer Migrationswerkzeuge in einem Zeitraum, der den Projektrahmen nicht sprengt. Anpassung von existierenden Werkzeugen an konkrete Projektanforderungen können mit Metawerkzeugen kurzfristig und ohne Risiko vorgenommen werden. Es müssen dazu lediglich die deklarativen Beschreibungen angepasst werden, aus denen dann die neuen Werkzeugversionen generiert werden.

Obwohl die wissenschaftlichen Grundlagen zur Compilerentwicklung theoretisch fundiert und seit Jahren bekannt sind, wurden nicht alle am Markt verfügbaren Migrationswerkzeuge nach diesen Prinzipien entwickelt. Die Konsequenz kann dann sein, dass die Konvertierungsergebnisse nicht semantisch äquivalent zu den Quellprogrammen arbeiten und unkorrekte Ergebnisse liefern. Expertise im Compilerbau ist somit eine wichtige Voraussetzung für den Erfolg jedes werkzeugbasierten Migrationsprojekts [8].

ERFAHRUNGSWERTE

Aus den Erfahrungen erfolgreicher Migrationsprojekte resultieren Grundsätze für den Ablauf eines erfolgreichen Projekts:

1. Studie: Zum Start jedes Migrationsprojekts ist das Anfertigen einer Studie nützlich. In ihr wird das Zielsystem detailliert dokumentiert, und die Migrationspfade zwischen Basis- und Zielsystem werden beschrieben. Kein Migrationsprojekt gleicht einem anderen. Deshalb existieren fertige Werkzeuge im Allgemeinen nicht. Es wird der Aufwand abgeschätzt, um existierende Werkzeuge an die konkreten Projektanforderungen anzupassen bzw. Werkzeuge neu zu entwickeln. Die Arbeitsteilung zwischen Auftraggeber und Auftragnehmer wird abgestimmt. Mit

diesen Informationen kann der Aufwand (Zeitrahmen, Budget) für das Migrationsprojekt abgeschätzt werden.

2. „Sanieren vor Migrieren“: Das Basissystem wird im Rahmen des Migrationsprojekts aufgeräumt. Erfahrungen besagen, dass sich dabei der Migrationsumfang um ca. 10 bis 15 Prozent reduziert.
3. Pilotprojekt: Das Pilotprojekt realisiert an einer ausgewählten Teilmenge von ca. 20 Prozent des Basissystems einen vertikalen Durchstich (Maske, Server, Datenbank) zur Verifikation der gewählten Migrationstechnologie und der verwendeten Werkzeuge. Im Ergebnis werden diese Werkzeuge bei Notwendigkeit nachjustiert.
4. So verführerisch es scheint, im Rahmen einer Softwaremigration Verbesserungen an der Software vorzunehmen, so komplex ist es auch. Erfahrungen besagen, dass eine 1:1-Migration zu favorisieren ist, um die Komplexität beherrschen zu können. Veränderungen der Software werden entweder in der Phase „Sanieren vor Migrieren“ (z. B. Ausbau der „GO TO“-Anweisungen bei COBOL) oder in einem der Migration nachfolgenden Reengineering vorgenommen.

FAZIT

Ein Resümee zur Softwaremigration beinhaltet „pros“ und „cons“: Softwaremigration ist dann eine Möglichkeit der Ablösung von Legacy, wenn die in die Algorithmen eingegossene Funktionalität im Wesentlichen den aktuellen praktischen Anforderungen entspricht. Im Ergebnis der Softwaremigration finden sich diese Algorithmen mit identischer Funktionalität auch in der neuen Welt wieder.

Die Wahrscheinlichkeit für den erfolgreichen Abschluss eines Softwaremigrationsprojekts erhöht sich mit ausgereiften Technologien und Werkzeugen. Beste Resultate werden dann erzielt, wenn „parserbasierte“ Werkzeuge zum Einsatz kommen. Dies führt zur Reduktion der Fehlerrate und minimiert das Projektrisiko. Die Projektlaufzeit ist kürzer, und das Projektbudget ist gegenüber einer manuellen Migration geringer.

Der Einsatz von Metawerkzeugen ermöglicht es, die eigentlichen Migrationswerkzeuge vergleichsweise schnell zu entwickeln bzw. kurzfristig an neue Anwenderforderungen anzupassen. Letzteres ist in laufenden Projekten gängige Praxis. Der Vorteil des Einsatzes von Metawerkzeugen ist: Die neuen Anforderungen werden nicht in den Migrationswerkzeugen programmiert, sondern in den Metawerkzeugen deklarativ beschrieben. Danach werden die Werkzeuge inklusive der neuen Anforderungen quasi generiert.

Unter den beschriebenen Voraussetzungen ist die Softwaremigration eine überaus sinnvolle Alternative zu einer Neuentwicklung bzw. der Einführung von Standardsoftware bei der Ablösung von Legacy-Software.

Links & Literatur

[1] Sommerville, I. (1995): „Software Engineering“, Addison-Wesley, 1995, Fifth Edition

[2] Ullmann, J.; Sethi, R.; Aho, A.: „Compilerbau 1“, Oldenburg Wissenschaftsverlag, 1999, 2. Auflage

[3] Erdmenger, U.; Uhlig, D.: „CoJaC – Ein Translator für die COBOL to Java Migration“, Softwaretechnik-Trends, Band 31, Heft 2, Mai 2011

[4] Pressemitteilung der pro et con GmbH vom 16.06.2014: http://www.proetcon.de/de/news/news.html#news_10062014

[5] Teppe, W.; Eppig, R.: „Das ARNO Projekt: Herausforderungen und Erfahrungen in einem großen industriellen Software-Migrationsprojekt“, Lecture Notes in Informatics (LNI) Proceedings, Series of the Gesellschaft für Informatik (GI), Volume P-126, 2008, S. 99–113

[6] „Automatische und toolgestützte Migration“, COMPUTERWELT, Printausgabe 09/2014

[7] http://de.wikipedia.org/wiki/Yacc

[8] Erdmenger, U.; Kaiser, U.; Loos, A.; Uhlig, D.: „Methoden und Werkzeuge für die Software Migration“, Lecture Notes in Informatics (LNI) Proceedings, Series of the Gesellschaft für Informatik (GI), Volume P-126, 2008, S 83–99

Dr.-Ing. habil. Uwe Kaiser

forschte und lehrte von 1977 bis 1994 an der Technischen Universität Karl-Marx-Stadt/Chemnitz zu compilergenerierenden Werkzeugen und Metawerkzeugen. Seit 1994 ist er geschäftsführender Gesellschafter der pro et con GmbH, einem Unternehmen, das Technologien und parserbasierte Werkzeuge für die Softwaremigration entwickelt und diese in Softwaremigrationsprojekten zum Einsatz bringt.

uwe.kaiser@proetcon.de