COBOL-Java Strategie

```
id division.
program-id. jembedded.

environment division.
configuration section.
repository.
class java java.util.Map end-java
class java java.lang.System end-java.

data division.
working-storage section.
01  ee-c2j-string      pic x(20) value "cobol string".
01  ee-c2j-int         pic 9(10) value 1234567890.

01  ee-j2c-string      pic x(20).
01  ee-j2c-int         pic 9(10).

procedure division.
main section.
main-st.

    display "start - embedded java"
```

Nur in der IT ist der Begriff Legacy bis dato „negativ“ belegt. Dieses Dokument wird erklären, warum diese Annahme falsch ist und was aus Legacy wachsen kann.

Ausgangsszenarien

In vielen Unternehmen lassen sich die Anwendungslandschaften, welche noch COBOL-zentrierte Anwendungen unterhalten, grob in drei Kategorien unterteilen. Alle diese Kategorien werden durch die am Ende aufgezeigte Strategie adressiert. Lediglich die jeweiligen Einstiegspunkte unterscheiden sich dabei.

Mainframe- und Legacy fokussierte Anwendungslandschaft

In dieser Landschaft spielen Großrechnersysteme, wie z.B. der IBM-Mainframe oder die FUJITSU BS2000, die zentrale Rolle bei der geschäftskritischen Datenverarbeitung. Lediglich bestimmte Front Ends sind in modernen Technologien wie z.B. Java realisiert und nutzen über verschiedene Schnittstellen die mainframeseitig implementierte Geschäftslogik. Neue fachliche Anforderungen und Subsysteme werden mit Ausnahme der Erweiterung von Front Ends auf dem Mainframe implementiert.

Hybridlandschaft

In der typischen Hybridlandschaft läuft ein bedeutender Teil der geschäftskritischen Anwendungen auf einem Mainframe oder auf Legacy-Basis. Die Front Ends sind in modernen Technologien im LUW-Umfeld realisiert und besitzen bereits teilweise eigene Anteile an Geschäftslogik. Fachliche Anforderungen werden auf dem Mainframe oder, wenn möglich und sinnvoll, bereits im LUW-Umfeld in Java oder anderen modernen Programmiersprachen implementiert. Ebenso werden neuere Subsysteme bereits in modernen Sprachen implementiert.

EasiRun Europa GmbH
Stockheimer Weg 9
61250 Usingen

Tel. +49 6081 9160-30
Fax +49 6081 9160-49
info@easirun.de
www.easirun.de

LUW-Landschaft

Die reine LUW-Landschaft nutzt keine Großrechnersysteme mehr, sondern basiert auf modernen Server-technologien. Die Landschaften sind in den meisten Fällen keine reinen COBOL-Landschaften mehr sondern umfassen auch Komponenten, die auf anderen Programmiersprachen wie z.B. C/C++, VisualBasic, C# oder eben auch Java basieren. Die Interaktion zwischen diesen Komponenten ist über verschiedene API's oder aber auch mit netzwerkbasierten Technologien realisiert, z.B. TCP/IP oder darauf aufbauend über Web Services. Teilweise sind solche gemischten Anwendungen das Ergebnis bereits begonnener Umstellungen von COBOL weg in eine andere Programmiersprache.

Eine Besonderheit, gerade im Versicherungsbereich, die in allen drei Kategorien gelegentlich anzutreffen ist, ist ein COBOL-Rechenkern, der vom Großrechner oder einem Serversystem geklont, das Herzstück dezentraler Einzelplatzanwendungen im Windows-Umfeld ist (z.B. Maklerprogramme).

Lösungsansatz „Einmalige Codetransformation"

Ein naheliegender Ansatz, die oben beschriebenen Herausforderungen anzugehen, ist die einmalige, direkte Transformation des COBOL-Codes nach Java. Die Technologien, um eine solche Transformation umzusetzen, existieren und werden von einigen Anbietern im Markt bereits zur Verfügung gestellt. Um die Zukunftsfähigkeit einer solchen durch Transformation entstandenen Lösung zu beurteilen, müssen mehrere Aspekte betrachtet werden.

Paradigmenwechsel

- Programmparadigmen COBOL / Java unterschiedlich – beeinflusst Umsetzung Transformation
- Kompromisse im Design der Java-Lösung erforderlich um funktionale Aspekte von COBOL in Java umzusetzen

Datentypen

- COBOL-Datentypen nicht direkt in Java-Datentypen umsetzbar aufgrund von interner Byte-Repräsentation der Datenstruktur im Speicher
- Emulierung der Speicherbereiche für die Daten im Hintergrund bei Java-Umsetzung

Lesbarkeit, Erweiterbarkeit und Wartung

- Jede automatisierte Transformation von COBOL-Programmen nach Java stellt eine mehr oder weniger lesbare Umsetzung der COBOL-Syntax nach Java dar
- Welcher Typ Entwickler soll die transformierte Anwendung warten?
- Ehemalige COBOL-Entwickler müssen zunächst in Java geschult werden
- Schwierig aufgrund der Unterschiede in Sprachparadigmen und der höheren Komplexität von Java

- Java Entwickler können mit transformiertem COBOL-Code wenig anfangen, Java Strukturelemente sind nur bedingt zu finden
- Trainingsaufwand und Spezialwissen für Fortentwicklung der Anwendung erforderlich
- Bei Integration größerer fachlicher Anforderungen in die bestehende Anwendung muss nach der Transformation umfangreich Code geändert werden
- Tool-Unterstützung möglich, Erhöhung der Entwicklungskomplexität der transformierten Anwendung

Performance

- Sehr gute Performance von COBOL-Anwendungen ist mit reinen Java-Mitteln schwer erreichbar
- Gerade bei automatisierter Transformation von COBOL nach Java ergibt sich ein Spannungsfeld zwischen les- und wartbarem Java-Code und hohen Performanceanforderungen
- Gut lesbarer, wartbarer Java-Code zeigt eine schlechtere Performance, während performanterer Java-Code jegliche Java-Paradigmen und Lesbarkeit aufgeben muss

Architektur

- Blick auf gesamte Anwendungslandschaft von der die Anwendung entstammt
- Wurzeln liegen meist auf dem Mainframe / Anwendung war Bestandteil eines größeren Systems, eng verknüpft mit anderen Systemen wie z.B. CICS, VSAM und IMS/DC
- Auf neuen Serversystemen (Linux/Unix/Windows) wurden diese Systeme meistens durch vergleichbare Produkte ersetzt (z.B. TX Series, Fujitsu UTM, ORACLE Tuxedo o.ä.)
- Frage klären, wie mit diesen Systemen umzugehen ist
    - Systeme und Schnittstellen weiterhin nutzen bedeutet halb moderner Ansatz Richtung Java, da Systeme nach wie vor zur Legacy Umgebung gehören
    - Neu-Implementierung der Funktionalitäten der Legacy-Technologien auf Basis von Java-Technologien (z.B. JEE)
    - Der damit verbundene Entwicklungs- und Testaufwand sind quantitativ mindestens genauso aufwändig wie die eigentliche Codetransformation einzustufen
- Projekt mit einmaliger Codetransformation ist ein „ganz oder gar nicht“-Projekt, produktionstaugliche Zwischenstände sind nicht zu realisieren
- Big-Bang-Lösungen sind technisch riskanter und ressourcentechnisch immer schwerer kalkulierbar als kleinere überschaubare Modernisierungsschritte

Allein die hier aufgezählten Aspekte lassen erahnen, dass die „einfache“ Lösung „einmalige Codetransformation von COBOL nach Java“ möglicherweise am Ende mehr neue Probleme und Herausforderungen schafft als löst.

Der EasiRun-Weg

Überblick

Kerngedanke des EasiRun-Weges zum Übergang von der COBOL- in die Java-Technologie ist die frühzeitige, solide Integration der COBOL-Anwendung in das Java-Umfeld und davon ausgehend eine sukzessive Ablösung von COBOL-Subsystemen und Geschäftsvorfällen durch native, neu entwickelte Java-Programme. Zentrales Werkzeug dafür sind COBOL-Java Crosscompiler.

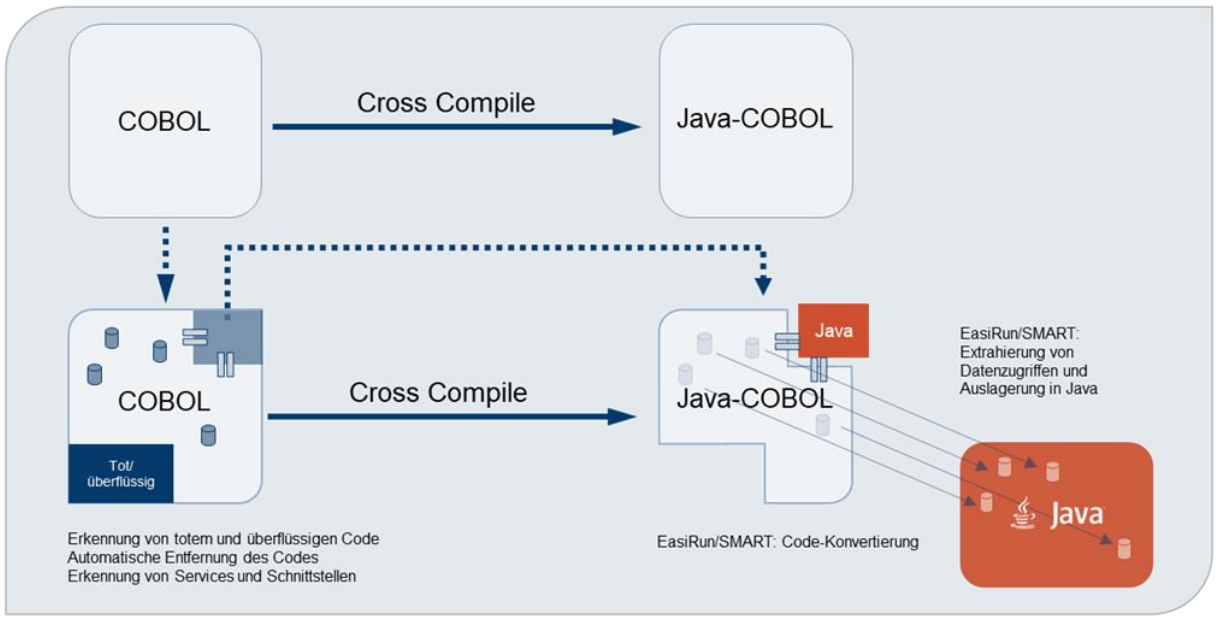

Abbildung 1: Entwicklungsprozess (Wartung des Codes auf COBOL-Ebene)

Dieser Ansatz birgt mehrere Vorteile:

- Fachliche Änderungen können weiterhin einfach in der vertrauten COBOL-Syntax umgesetzt werden. Spezialkenntnisse oder zusätzliche Schulungen sind dazu nicht erforderlich. Das vorhandene COBOL-Know-how kann weiter genutzt werden.
- An den Java-Zwischencode werden keine hohen Anforderungen bezüglich Lesbarkeit und Wartbarkeit gestellt. Dies ermöglicht eine hohe Optimierung bezüglich Funktionsidentität und Performance der Anwendung.
- Durch vorhandene Schnittstellenfunktionalitäten und einfachen Sprachübergang zwischen COBOL und Java (und umgekehrt) möglich, Teilfunktionalitäten oder Subsysteme in Java neu zu implementieren und in bestehende COBOL-Anwendung zu integrieren
- Umgekehrt ist es natürlich auf diese Weise auch möglich, wertvolle COBOL-Logik anderen Java-Anwendungen auf einfache Weise verfügbar zu machen.

Da es mit einem einfachen Tausch des Compilers aber nicht getan ist, möchten wir in den folgenden Abschnitten Lösungsansätze für unterschiedliche Ausgangsszenarien aufzeigen, die einzeln oder unterneh-

mensspezifisch kombiniert eine aus unserer Sicht sinnvolle Strategie auf dem Weg aus der COBOL-Welt in die reine Java-Welt bilden.

Rehosting

Der erste logische Schritt auf dem Weg in die reine Java-Welt für Organisationen mit einer mainframe-fokussierten oder hybriden Anwendungslandschaft ist die Migration auf eine reine LUW-Plattform (Linux/Unix/Windows).

- Für Anwendungen vom IBM-Host empfiehlt sich für folgende Modernisierungsschritte und Kosten-reduzierung von MIPS Migration auf LINUX for Z
- Grundgedanke ist, die Anwendung weitestgehend unverändert auf LUW-Plattform zu portieren
- Mainframetypische Subsysteme wie z.B. Transaktionsmonitore, Scheduling, JobControl und andere werden durch Lösungen mit identischen Funktionalitäten und Schnittstellen ersetzt
- Rehosting ist ein großer technischer und organisatorischer Sprung, aber durch Know-how und der Erfahrungen im Markt inzwischen gut kalkulierbar

Eine Vorstellung davon, wie eine einfache Migration von einem Mainframe System hin zu einem LUW System aussehen kann, ist der folgenden Abbildung zu entnehmen:

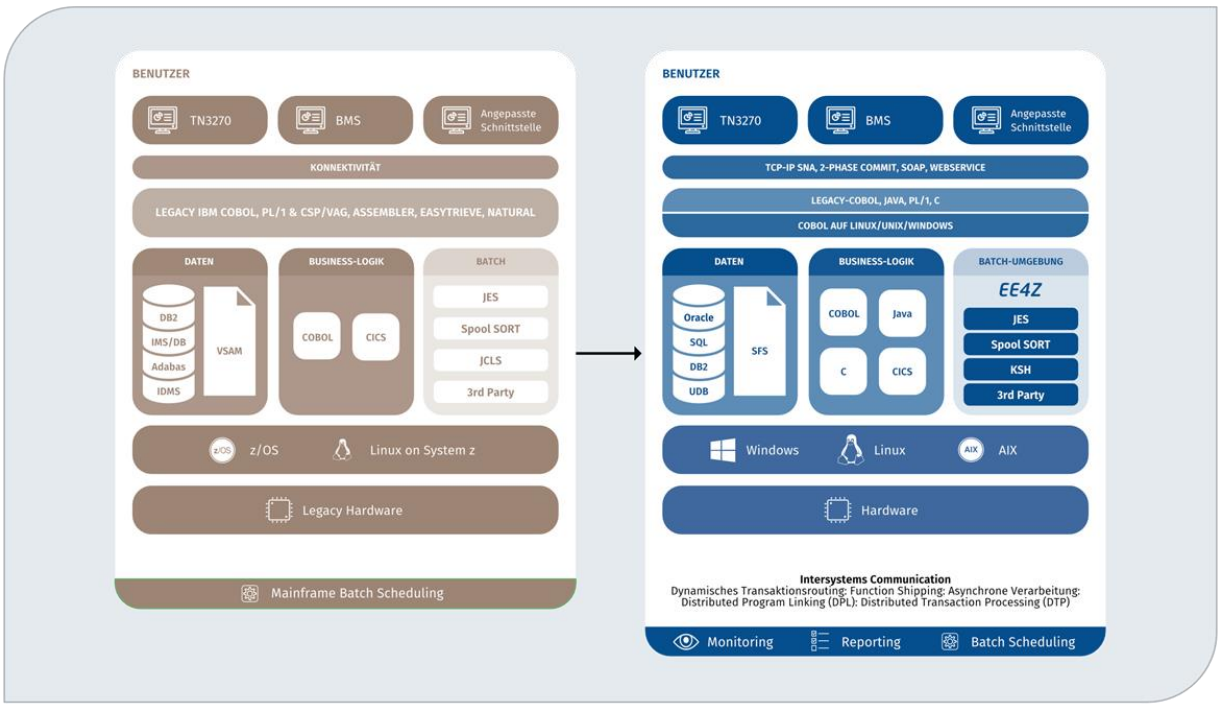

Abbildung 2: Rehosting (Beispiel)

- Im Zuge des Rehostings ist der Einsatz eines COBOL-Java-Crosscompilers sinnvoll
- Vorteil 100%-ige Kompatibilität mit gängigen Mainframe-Compilern (z/OS, BS2000)
- Vorteil Schnittstellen von und in Richtung Java

- Erste Verbindung in Java-Welt mit wenig Aufwand und kleinen Modernisierungsschritten
- Bereitstellung von Web-Services bestimmter Funktionalitäten der COBOL-Anwendung
- Möglichkeit jüngere Mitarbeiter einfacher in vorhandenes System einzuarbeiten
- Teilauslagerung MIPS-intensiver Anwendungen auf LUW-Systeme / z/LINUX-Partition Mainframes
- Realisierung als Kostensenkungsmaßnahme oder Zwischenschritt zum vollständigen Rehosting
- COBOL-Anwendung unter z/Linux kann mit Java-Crosscompiler vorhandene ZIIP/ZAAP Java-Prozessoren / HIPERSOCKETS DB2-Zugriff effektiv nutzen / MIPS-Verbrauch signifikant reduzieren

Definition der Zielarchitektur

Ein wichtiger Schritt, der vor allen weiteren Schritten in Richtung Java vollzogen werden sollte, ist die Definition der Java-Zielarchitektur. Leitlinie sollten dabei die Standards sein, die den heutigen Java-Business-Architekturen zugrunde liegen. Im Ergebnis des Definitionsprozesses sollte eine Architektur vorliegen, die mindestens die folgenden Aspekte berücksichtigt:

- Graphical User Interface, Datenhaltung, Transaktions-, Druck- und Output-Management
- Schnittstellen zu anderen, auch externen, Systemen
- Skalierbarkeit (z.B. Micro Services, Container, Orchestrierung)
- Generelle Produktions-Optimierung inklusive Scheduling (Business-IT Automation)

Ablösung von Subsystemen

Nachdem die Zielarchitektur festgelegt ist, kann der Weg in die Zukunft beginnen. Hierbei werden nun die vorhandenen Nicht-COBOL-Subsysteme in die zukünftige Architektur überführt. Es bieten sich zwei grundsätzlich verschiedene Wege an:

- Sukzessive Ablösung von Subsystemen begleitet vom Aufbau der Zielarchitektur
- Aufbau der kompletten Zielarchitektur mit allen benötigten Subsystemen und anschließende Integration der COBOL-Anwendung in das System

a) Sukzessive Ablösung von Subsystemen begleitet vom Aufbau der Zielarchitektur

Bei diesem Ansatz werden in kleinen Modernisierungsschritten einzelne Subsysteme sukzessiv abgelöst.

Im Zuge dieser Ablösung werden die für dieses Subsystem benötigten Komponenten der Zielarchitektur erstellt und integriert. Diese Subsysteme können beispielsweise die in der Zieldefinition genannten Aspekte Datenhaltung, Druckmanagement bzw. Output Management, Schnittstellen zu anderen Systemen oder Transaktions-Management und grafische Benutzeroberfläche sein. Jeder einzelne dieser Modernisierungsschritte ist in sich abgeschlossen und führt nach seiner Umsetzung wieder in einen produktionsreifen Zustand und wird auch produktiv eingesetzt.

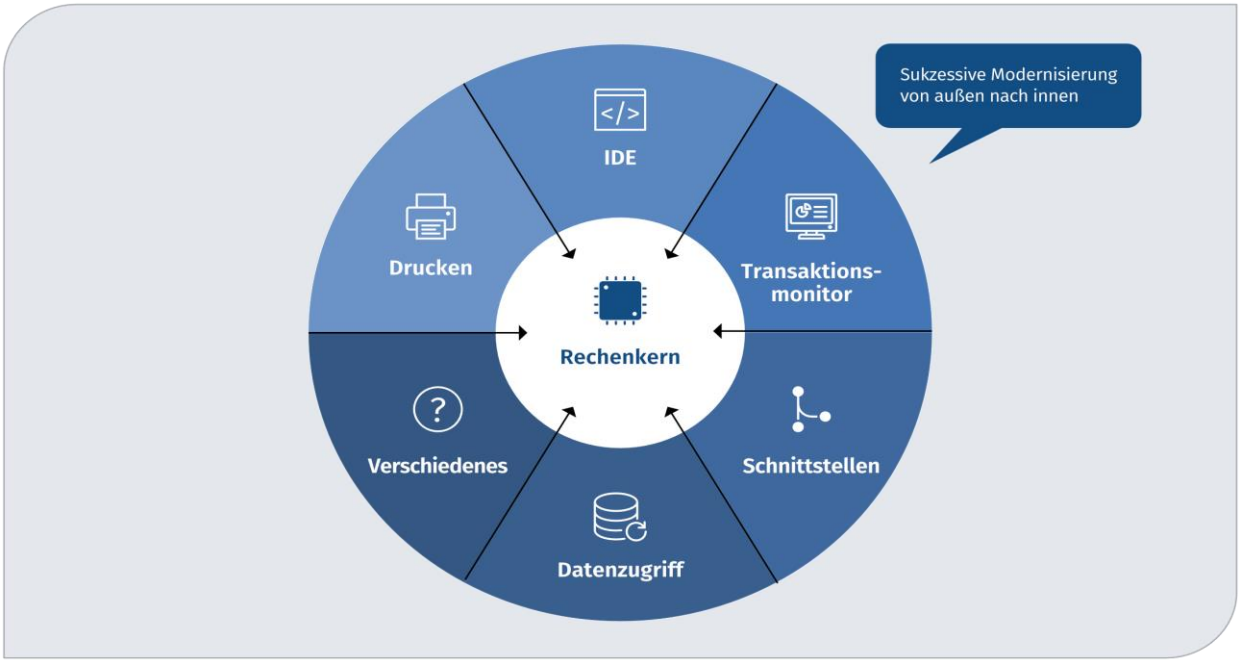

Abbildung 3: Sukzessive Ablösung von Subsystemen von außen nach innen (Prinzip)

Vorteile

- Kleine überschaubare Schritte, reduziertes Risiko
- Ermittlung von Anforderungen an Leistung und Erfahrung mit erforderlichen Prozessen
- Schnelle Überführung modernisierter Subsysteme in die Produktion

Nachteil

- Funktionierende Zwischen- bzw. Übergangslösungen können zu permanenten Lösungen werden

b) Aufbau der Zielarchitektur und anschließende Integration

Dieser Ansatz sieht den Aufbau der kompletten Zielarchitektur oder Benutzung bereits bestehender Java-Anwendungsframeworks mit allen benötigten Subsystemen und die anschließende Integration der COBOL-Anwendung vor. Er eignet sich besonders in den Fällen, in denen bereits eine funktionierende Java-Architektur existiert.

Vorteile

- Es wird von Beginn an in einer reinen Java-Architektur gearbeitet
- Ggf. bereits vorhandene Java. Komponenten und das Wissen darum werden weiterhin verwendet
- Höherer Gestaltungsspielraum bei der Ausgestaltung der Java-Architektur

Nachteile

- Große Modernisierungsschritte von einem produktionstauglichen Zustand in den anderen
- Höhere Anforderungen in Anpassung interner Prozesse und Vermittlung des Prozess-Know-hows
- Aufwändigere Integrationstests, größerer Aufwand bei der Integration der COBOL-Businesslogik

Analyse der Kernanwendung

Nachdem die eher technisch geprägte Migration der Subsysteme abgeschlossen ist, ist es an der Zeit die Fachlogik umfassend zu analysieren, um damit die Voraussetzungen für eine sukzessive Ablösung der COBOL-Komponenten mit der Fachlogik durch neu erstellte Java Programme vorzubereiten. Ziel der Analyse ist es dabei, die Ergebnisse für verschiedene Ansätze innerhalb des Unternehmens in Form von grafischen Auswertungen, Reports und Tools nutzbar zu machen. Der Know-how Gewinn steht hier an erster Stelle, um mehr Informationen über die Zusammenhänge der Anwendung zu erhalten. Das wiederum ist die Grundlage dafür, mehr Mitarbeitern wesentliche inhaltliche Bereiche des Systems zugänglich machen zu können. Strategische Entscheidungen können dadurch leichter ermittelt und getroffen werden. Aus den Ergebnissen lassen sich zudem Service-Kandidaten für Modernisierungsansätze identifizieren, aber auch eine Einarbeitung von neuen Mitarbeitern in das Anwendungssystem lässt sich dadurch leichter umsetzen.

Das Erreichen dieser Ziele erfolgt mit Hilfe einiger Analysewerkzeuge:

- Durch eine umfangreiche statische Analyse wird eine vollständige Landkarte der untersuchten Anwendung samt aller Schnittstellen erstellt
- Diese Landkarte beinhaltet alle möglichen Optionen, die in der Anwendung verwendet werden, auch über Systemgrenzen hinweg
- Die dynamische Analyse erzielt die automatisierte Instrumentierung von Quellcode, welche zum einen für Diagnostik und zum anderen für die Erstellung von Loggings und die Protokollierung von Szenarien verwendet wird
- Es werden lediglich die Programme und Komponenten angezeigt, die abhängig von den gewählten Anwendungsfunktionen verwendet werden, sodass sich genau erkennen lässt wie die Anwendung bei der Ausführung durchlaufen wird
- Die Logging-Ergebnisse werden für die statische Analyse in Form von Metadaten importiert und mit dem Quellcode zu einer vollständigen Durchführungslandkarte ergänzt
- Aufbauend auf der statischen Analyse wird ein umfangreiches Werkzeug zur Betrachtung der Qualität, Komplexität und Sicherheit von unternehmenskritischen Anwendungen eingesetzt
- Basierend auf vorhandenen und kundenspezifisch definierbaren Regeln und Kriterien werden Unternehmensanwendungen analysiert und die Ergebnisse mittels Business-Intelligence in Form von Reports und Empfehlungen bereitgestellt

Java-Fähigkeit für Kern- und Standardkomponenten

- Bei sukzessiver Umsetzung der Anwendung nach Java beginnt man nun zweckmäßigerweise bestimmte Kern- und Standardkomponenten neu in nativem Java zu implementieren
- Komponenten sollten zustandslos und in sich abgeschlossen sein, über definierten Einstiegspunkt verfügen und nicht von weiteren Anwendungsteilen abhängig sein
- Standardkomponenten sind zweckmäßigerweise vorher umzusetzen (Bottom-Up-Ansatz)

- Schnittstellen neuer Komponenten sollten gängigen Java-Standards genügen
- Lässt verschiedene Optionen und hohen Grad an Flexibilität für zukünftigen Einsatz zu
- Java-Komponenten können nahtlos in COBOL-Anwendung integriert werden oder in anderer Form bereitgestellt werden, z.B. als Web-Services oder Micro Services gehostet in Docker-Containern
- Kapselung einzelner Teile der COBOL-Anwendung, Bereitstellung von reinen Java-Schnittstellen in COBOL-Anwendungsteilen möglich
- z.B. für monolithischen Rechenkern, der nach Java transformiert werden oder als COBOL-Anwendungsteil Bestand haben soll

Sukzessive Umsetzung von Geschäftsvorfällen

- Schrittweise Umsetzung von Geschäftsvorfällen als anspruchsvollste Aufgabe der Strategie
- Im Gegensatz zur kompletten Neuentwicklung einer Anwendung deutlich risikoärmer durch Ansatz der kleinen Schritte und ein Geschäftsvorfall sehr viel kleiner ist als eine Komplettanwendung
- Erfahrungsgewinn durch Start mit kleineren Geschäftsvorfällen
- Umsetzung eines Geschäftsvorfalls kann auch durch neue fachliche Anforderungen getriggert werden, die auch im COBOL-Umfeld erhebliche Aufwände mit sich bringen würden
- Bei Umsetzung von Geschäftsvorfällen sollten nur in Java implementierte Standard- / Kern-komponenten oder mit Java-Schnittstellen versehene COBOL-Anwendungsteile aufgerufen werden

Zusammenfassung

Die dargestellte Strategie zur Überführung einer COBOL-Anwendung in die Java-Welt zeigt einen roten Faden auf, der natürlich auf das konkrete Anwendungssystem implementiert und angepasst werden muss. Kerngedanke hierbei ist, in kleinen, in sich abgeschlossenen Schritten vorzugehen und den Übergang nach Java nicht als adHoc-Großprojekt anzugehen. Dies widerspiegelt auch unsere Sichtweise, dass IT-Modernisierung eine ständige Aufgabe jeder IT-Organisation ist und kontinuierlicher Bestandteil der täglichen Arbeit sein muss. Eingebettet in geeignete Prozesse, in diesem Zusammenhang sei auf DevOps verwiesen, führt diese Vorgehensweise zu einer Verstetigung der Modernisierungsaktivitäten und sichert auch langfristig die Zukunftsfähigkeit der Anwendung bei gleichzeitig hoher Verfügbarkeit und Qualität.

Die Umsetzung dieser Strategie wird stark vereinfacht durch den Einsatz spezieller Werkzeuge. Im Zentrum steht hierbei die COBOL-Java-Crosscompiler-Strategie ergänzt durch entsprechende Analysewerkzeuge.

Informationen zu Mainframe & Legacy Portfolio sowie weiteren Produkten erhalten Sie auch unter www.easirun.de

© Copyright 2019 EasiRun Europa GmbH. Alle Rechte vorbehalten. Microsoft, Windows und das Windows-Logo sind Marken der Microsoft Corporation. Alle anderen Produkt- und Firmennamen sind Marken der jeweiligen Inhaber.