Metadaten basiertes, teilautomatisiertes Software-Reengineering

Andres Koch (akoch@objeng.ch), Remo Koch (rkoch@objeng.ch)
Object Engineering GmbH, CH-8142 Uitikon-Waldegg

Abstract

Eine bestehende Applikation ist eigentlich selbstbeschreibend, das heisst, wenn man Zugang zu deren Metadaten bekommt, hat man eine genaue Beschreibung davon. In der Praxis kann eine teilautomatisierte Analyse- und Generierungs-Methode zielführender sein als eine vollständig automatisierte Transformation der bestehenden Applikation. Der Grund liegt darin, dass man ein unzulängliches Design eigentlich nicht in ein neues System transformieren möchte. Die auf Metadaten basierte Reengineering-Methode entstand über die vergangenen Jahre bei der Modernisierung von Applikationen und wurde laufend weiterentwickelt.

Metadaten-Modell

Als Grundlage für das System wurde ein Metadaten-Modell definiert, mit welchem die typisch angetroffenen Applikations-Welten modelliert und auf sinnvolle Granularität abgebildet werden können. Dabei werden verschiedene Detailierungs-Ebenen berücksichtigt.

In dieses "normierte" Metamodell werden Metadaten bestehender Applikations-Artefakte abgelegt. Dabei spielt hauptsächlich, aber nicht ausschliesslich, Programm- und Datenbank-Code eine zentrale Rolle. Es werden aber auch Metadaten aus Konfigurationen, Datenbankmodellen und bei Bedarf dynamische Interprozess-Kommunikationsdaten zwischen Systemen als Datenquellen berücksichtigt.

Die automatisch erstellten Metadefinitionen eines realen Systems sind sehr viel umfangreicher und genauer als zum Beispiel eine manuell erstellte Dokumentation. Dabei werden auch Abweichungen von verwendeten Programmier-Mustern (Anomalien) erfasst, welche oft die grössten Barrieren eines automatisierten Reengineerings darstellen. Sollen diese Daten dem Software-Designer für die Architektur des neuen Systems oder dem Anforderungs-Engineer dienen, muss der Detailierungsgrad für die Zielgruppe angemessen sein. Will man aus diesen Metadaten jedoch neue Artefakte generieren, muss anderseits ein sehr hoher Detailierungsgrad und eine breite Abdeckung vorliegen.

Vorgehen

Ein in der Praxis bewährtes Vorgehen umfasst mehrere Stufen und sollte als ein Muster und nicht roboterhaft angewendet werden. Die Methode sollte flexibel und mit gewissem Pragmatismus angewendet werden. Die Schritte werden oft iterativ durchgeführt. Wenn z.B. in der Designphase des neuen Systems eine zusätzliche Fragestellung auftritt, können Schritt 1.-6. speziell für diesen Aspekt repetiert werden.

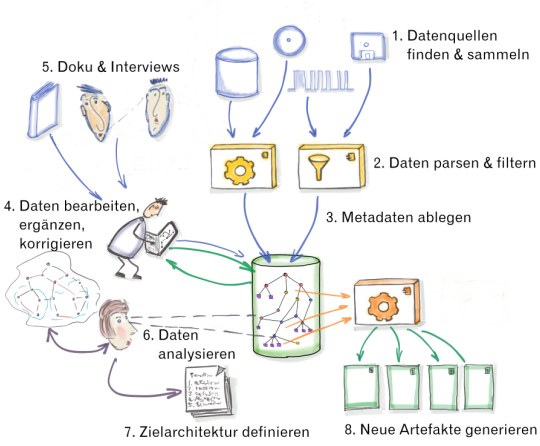

Abbildung 1: Praxis-Vorgehen

Finden von geeigneten Applikations-Artefakten

Das Finden der verfügbaren Applikations-Artefakte scheint zwar trivial zu sein, kann aber gerade bei älteren, grossen und "vergessenen" Applikationen die erste Herausforderung darstellen.

Feststellen der Artefakt-Typen

Systeme sind zwar oft nach gängigen Mustern aufgebaut, doch werden diese meistens durch die lokale Entwicklungskultur geprägt. Diese Ausprägungen bestimmen die Heuristiken, welche bereits für die automatisierte Erfassung der Metadaten berücksichtigt werden. Diese findet man durch manuelle Sichtung, oft iterativ nach den ersten automatischen Erfassungen. Man geht meistens vom vorgegebenen Metamodell aus und ergänzt dieses aber mit Attributen, welche den spezialisierten Bedarf abdecken.

Automatisches Erfassen von Metadaten

Durch ein flexibles Parser-Framework [1] und teils speziell für die untersuchte Applikation erstellte Filter werden die interessierenden Daten ermittelt und ins Metadaten-Repository [2] gesichert. Ob es sich um traditionelle Sprachen wie COBOL, Pascal, C etc. oder objektorientierte Sprachen wie C++, C#, Java etc. handelt, macht keinen grossen Unterschied. Wichtig ist die Abbildung dieser Artefakte auf das erwähnte generische Metamodell.

Damit können Programmteile mit Datenbank-Zugriffen und Zugriffe auf Benutzer-Schnittstellen-

Elemente (Masken, Felder) erkannt und kategorisiert werden. Daraus können System-übergreifende Beziehungen aufgezeigt werden.

Ergänzend kann man Konfigurationen und sogar aufgezeichnete, dynamische Kommunikations-Daten einlesen. Bei der Art der eingelesenen Artefakte sind eigentlich keine Grenzen gesetzt. Natürlich sollten diese Metainformationen möglichst präzise und aktuell sein. Das ist bei manuell erstellten Dokumentationen wie Handbüchern und den verbreitet verwendeten Excel-Tabellen nicht immer der Fall.

Der Export in ein Graphen-Datenbank-Repository [3] erweist sich für die Sondierung der erfassten Daten und später für die System-Analyse als sehr hilfreich.

Korrektur und Ergänzung der Metadaten

Eingelesene Daten werden, wo nötig und möglich, ergänzt und unter Umständen manuell oder automatisiert korrigiert.

Automatische Analyse-Läufe

Sobald eine relevante Menge an Applikations-Artefakten im Metadaten-Repository vorhanden ist, können übergreifende Abhängigkeiten und Zusammenhänge ermittelt werden, welche dann wiederum im Repository abgelegt werden. Durch automatisierte Analyse-Läufe werden Abhängigkeiten gesucht und andere, in früheren Schritten definierte Heuristiken, angewendet.

Analyse-Berichte und -Dokumentation

Das Metadaten-Repository wächst mit jedem Schritt zur vollausgewachsenen Beschreibung der Applikation oder des Systems heran.

Durch Generierung von Analyse-Berichten, individueller Nachdokumentation und Graphen wird der Modernisierungs-Prozess unterstützt.

Erstellen der Ziel-Architektur und des Migration-Plans

Durch Verwendung der erwähnten Graphen-Datenbank als Metadaten-Repository eröffnen sich Analyse-Möglichkeiten, welche bei einem herkömmlichen Datenbanksystem nur beschränkt vorhanden wären. Dies ist für die Definition des Ziel-Systems von höchster Wichtigkeit und Wert. Es können wertvolle Erkenntnisse für die Zielarchitektur und/oder für die Migration eines zu modernisierenden oder eines neu zu erstellenden Systems gewonnen werden.

Generierung neuer Artefakte

Es liegt auf der Hand, dass an dieser Stelle nicht das Ende erreicht ist. Metadaten können und sollten auch für die Generierung von neuen Komponenten des neuen Systems oder Teilen davon verwendet werden. Speziell häufig auftretende, ähnliche Komponenten wie Benutzer-Oberflächen, Meldungs-Strukturen, Datenbank-Zugriffe und vor allem strukturierte Daten (z.B. aus 4GL-Sprachen) eignen sich besonders, da man hier den repetitiven Charakter nutzen und damit die Qualität des neuen Codes erhöhen kann. Dass generierter Code nicht gepflegt werden kann und nicht lesbar sei, ist ein Mythos, der von Software-Entwicklern gerne als Einwand aufgebracht wird. Wenn eine fähige Person ein gut pflegbares Programm schreiben kann, dann kann diese auch den entsprechenden Generator dafür programmieren. Eine umfangreichere und detailliertere Informations-Sammlung als das Metadaten-Repository bekommt man am einfachsten aus der bestehenden Applikation.

Erfahrungen im praktischen Einsatz

Ein interessanter Aspekt ist, dass heute bereits Systeme in sogenannten "moderneren" Sprachen wie C++, Java und C# einem Reengineering unterzogen werden müssen. Das war vor einigen Jahren fast ausschliesslich für Systeme der Fall, die in COBOL, Pascal, PL/1 und ähnlichen Sprachen implementiert waren.

Es gibt eine Vielzahl von Code-Analyse- und Reengineering-Werkzeugen, welche entweder den Programm- resp. den Datenbank-Code oder das Entwicklerverhalten minutiös untersuchen und dokumentieren. Die hier vorgestellte Methode konzentriert sich auf die übergreifenden Zusammenhänge und verzichtet darauf jede Code-Zeile zu erfassen, wenn es der Fragestellung nicht dient.

Das Verfahren wurde in verschiedenen Migrations- oder Analyse-Projekten eingesetzt. Bei einer Migration einer in C++ geschriebenen Fat-Client-Applikation auf eine Single Page Web-Applikation konnten 50% des für die manuelle Migration geschätzten Aufwandes eingespart werden [4]. Darin waren alle Aufwände einschliesslich Architektur- und System-Anpassungen enthalten.

Für eine AS400-Batch-Applikation mit rund 100 Tabellen und 1630 DB-Queries konnten die Abhängigkeiten für eine Neuimplementation aufgezeigt und insbesondere rund 4000 unnötige Artefakte ausgeschieden werden.

Referenzen

[1] ANTLR, ANother Tool for Language Recognition, `http://www.antlr.org/`

[2] Metasafe-Repository, Metadaten Modellierung und Repository, `http://www.metasafe-repository.com/`

[3] Neo4J, Graphen-Datenbank, `https://neo4j.com/`

[4] Dr. Reinhold Thurner, Andres Koch, Remo Koch. *Modellbasiertes Software-Reengineering: Teilautomatisierte Migration eines GUI in eine Web-Umgebung.* Java Spektrum 06/2014, 2014.