

Reverse Engineering

Möglichkeiten und Grenzen verfügbarer Tools

Rund 60% aller Softwareprojekte sind Wartungs- oder Weiterentwicklungsarbeiten in denen der Entwickler meist mit fremdem, historisch gewachsenem Code und nicht selten mit nicht ausreichender oder fehlender Dokumentation konfrontiert wird. Reverse Engineering verspricht, in diesen Fällen ein gangbarer Lösungsweg zu sein. In wie weit diese Methode wirklich helfen kann, hängt wesentlich von der Funktionalität entsprechender Werkzeuge ab. THOMAS RÖDIGER

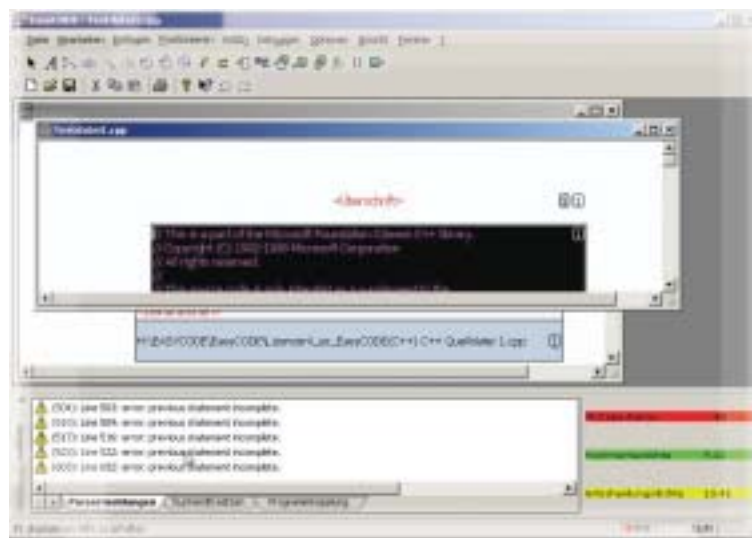


Abb. 1: Fehlermeldungen eines Parsers wie sie beim Reverse-Engineering auftauchen können. Die Metriken sind weit außerhalb der Toleranzgrenzen.



Thomas Rödiger ist Technischer Leiter der Easycode GmbH

Muss sich der Softwareentwickler wegen unzureichender Dokumentation in den vorliegenden Code Zeile für Zeile einarbeiten, wird aufgrund des Umfangs der Aufgabe Hilfe jeglicher Art als rettender Strohhalm verstanden. Entsprechende Werkzeuge werden von vielen Reverse-Engineering-Tool-Herstellern als solche angepriesen. Aber was versteckt sich hinter dem Begriff Reverse Engineering und wie kann diese Technik einem Entwickler in der geschilderten Situation helfen? Der folgende Beitrag liefert einen Überblick über Funktionen und Grenzen des Verfahrens und der verfügbaren Werkzeuge.

Grundprinzip des Reverse Engineering

Das Reverse-Engineering-Tool wandelt den reinen Quellcode in eine verständlichere Sichtweise z.B. in Form eines Struktogramms um. In der Regel wird der Quellcode dazu interpretiert und in eine für das Reverse-Engineering-Werk-

zeug verwendbare Form umgewandelt. In den meisten Fällen wird mit den ermittelten Informationen ein Repository aufgebaut. Solche Repositories müssen auf einem Netzlaufwerk liegen, damit sie von allen am Projekt beteiligten Entwicklern benutzt werden können. Eine Überwachung der Historie ist, wenn diese Funktion das Engineering-Tool nicht selbst bereitstellt, eingeschränkt möglich, da Repositories nur über das verwendete Werkzeug gelesen werden können mit dem sie erzeugt wurden. Differenzen zwischen verschiedenen Versionsständen sind daher nur schwer sichtbar zu machen. Nur wenige Werkzeuge können wie ‚Easycode‘ direkt mit der Quelldatei arbeiten und benötigen kein Repository.

Verständlichkeit des Ergebnisses

Das Ergebnis, d.h. die Verständlichkeit des Reverse-Engineering-Ergebnisses, kann wesentlich verbessert werden, wenn alle zum Projekt gehörende Dateien in das Werkzeug importiert

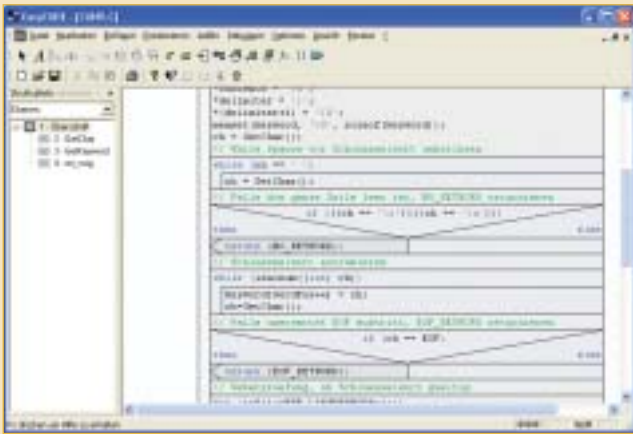


Abb. 2:
Die strukturierte
Darstellung verbes-
sert die Verständ-
lichkeit des
Quellcodes

werden. Dieses kann dann alle Module und deren Beziehungen untereinander feststellen. Nur dann erhält der Entwickler eine Übersicht über das gesamte Projekt.

Ein Reverse-Engineering-Werkzeug kann aber auch die Verständlichkeit des Sourcecodes selbst erhöhen und den Entwickler bei der Analyse und Dokumentation der zum Teil komplexen Ablauflogiken unterstützen. Viele der verfügbaren Werkzeuge decken aber nur eine dieser Funktionen ab.

Darstellungsform des Ergebnisses

„Ein Bild sagt mehr als tausend Worte“. Dieses Sprichwort könnte den Erfindern des Struktogramms, Nassi und Shneidermann, Pate gestanden sein. Tatsächlich ist diese Darstellungsform wesentlich einfacher zu verstehen als der reine textuelle Quellcode. Struktogramme verfügen über nur sieben Elemente, die in der DIN 66261 und in der ISO/IEC 8631 beschrieben werden. Auch andere Darstellungsarten wie Program Flowcharts (PF), Program Structure Diagrams (PSD), Design Structure Diagramms (DSD), Structured Programming Diagramms (SPD), Hierarchical and Compact Description Chart (HCD) erleichtern die Lesbarkeit des Programmcodes.

Werkzeuge, die mit diesen Darstellungsformen umgehen können, eignen sich nicht nur zur Quellcodegenerierung, sondern auch für das Reverse Engineering. Vor allem zeichnen sie sich durch sehr kurze Einarbeitungszeit, die in aller Regel bei ein paar Stunden liegt, und einen vergleichsweise günstigen Anschaffungspreis aus. Wenn die Analyse auf Codeebene durchgeführt werden soll, genügen diese Werkzeuge den Ansprüchen.

Grenzen einfacher Werkzeuge

In manchen Fällen reicht aber die einfache Übersicht über eine Quellcodedatei nicht aus,

um sich einen umfassenden Überblick über ein ganzes System zu verschaffen. Während bei strukturierten Systemen das Reverse Engineering Prozedurzusammenhänge noch sehr erfolgreich darstellen kann, haben objektorientierte UML-Werkzeuge damit ihre Probleme. UML ist eine Darstellungsform für die objektorientierte Analyse (OOA) und dem objektorientierten Design (OOD). Während Diagrammtypen aus dem OOD re-engineert werden können, bleiben die Diagrammtypen der OOA unberührt und müssen gegebenenfalls mühsam per Hand nachgezogen werden. Daher beschränken die meisten UML-Tools ihre Reverse-Engineering-Fähigkeiten auf die Generierung von Klassen, Typen und Beziehungen zwischen den einzelnen Klassen, sofern möglich. Sie werden keine Interaktionsdiagramme oder gar Use-Cases als Ergebnis des Reverse Engineerings liefern.

In vielen Fällen haben Tool-Hersteller das Reverse Engineering nur zur Verfügung gestellt, damit der Entwickler den vom Werkzeug generierten Quellcode wieder zurückführen kann. Hierbei werden oft zusätzliche Strukturinformationen, die die Rückführung erleichtern sollen, in Form von Kommentaren in den Code geschrieben. Ändert der Entwickler absichtlich oder aus Versehen bestimmte Bereiche, kann es passieren, dass der Code nicht mehr zurückgeführt werden kann. In diesem Fall muss der Entwickler auf die Sicherung zurückgreifen, die Änderungen eines ganzen Arbeitstages können verloren sein. Gleicht man nun nicht aufwändig den Quellicodestand mit dem Modell ab, entstehen irreparable Inkonsistenzen, die den Vorteil des Reengineering-Vorgangs zunichte machen können.

Durch den Einsatz falscher Werkzeuge wird für die höhere Verständlichkeit von Projekten häufig ein sehr hoher Preis bezahlt, der sich aus den reinen Produktkosten, Schulungskosten und zusätzlichem Arbeitsaufwand errechnet. Die Beratung durch einen unabhängigen Experten oder auch die Erfahrungen anderer Unternehmen können hier sehr wertvoll sein. ►

Anzeige

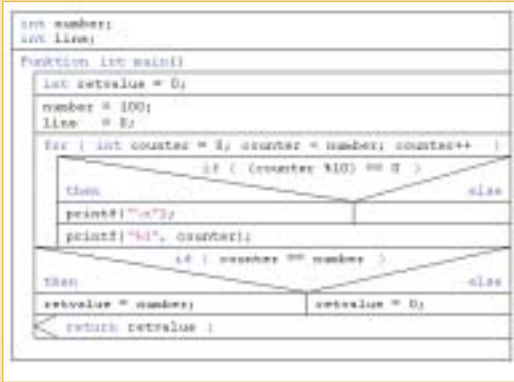


Abb. 3: Darstellung eines einfachen Codeausschnittes in Editorform und in ‚Easycode‘. Schon die Verwendung von graphischen Elementen erhöht die Verständlichkeit der Ablauflogik.

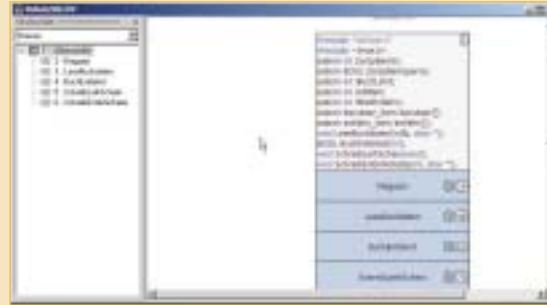


Abb. 4: Eine lückenlose ISO-konforme Dokumentation, die durch die Verwendung eines Struktogrammeditors jederzeit konsistent zum Code ist.

Grenzen des Reverse Engineerings

Wie schon weiter oben erwähnt hat das Reverse Engineering seine Grenzen. Soll ein Code fehlerfrei interpretiert werden, sollten im Quellcode verwendete Preprozessoranweisungen Programmstrukturen nicht zerschneiden. Der Entwickler muss sich darüber im Klaren sein, dass #if-, #ifdef- oder #ifndef-Anweisungen eine weitere Strukturebene über die vorhandene Programmstruktur stützen. Werden in dieser Programmierweise z.B. Funktionsdeklarationen unterschiedlich ausgewiesen, so hat das Reverse-Engineering-Tool Probleme diese Informationen entsprechend zuzuweisen.

Das Reverse Engineering von ganzen Projekten kann nur bedingt UML-Diagrammtypen erzeugen. Die Grenze liegt hier bei den Interaktionsdiagrammen, Use-Case-Diagrammen, Aktivitätsdiagrammen und teilweise auch bei State Charts.

Der Reverse-Engineering-Vorgang ganzer Projekte kann unter Umständen sehr lange dauern und sollte in einem Zeitraum durchgeführt werden, in dem nicht an den Quelldateien gearbeitet wird.

Kriterien für die Tool-Auswahl

Kosten versus Verständlichkeit

Zunächst sollte eine klare Definition über das Ziel des Reverse Engineering erstellt werden. Will man nur die Lesbarkeit eines Quellcodes erreichen, empfiehlt sich der Einsatz eines leistungsfähigen Struktogrammeditors wie ‚Easycode‘. In ca. 80% der Fälle ist das Ergebnis für den Entwickler ausreichend. Besonders die Möglichkeiten nicht nur eine Darstellung der Sourcen zu erreichen, sondern auch eine aktuelle, ISO-konforme Dokumentation (Abb. 4) erzeugen oder gar Modifikationen durchführen zu können, erweitern die Funktionalitäten dieser Werkzeuge. Die Kosten für einen Struktogrammeditor liegen im Bereich 1.500 bis 3.000 Euro und die Einarbeitungszeit liegt im Stundenbereich. Hilfreich ist, dass der Grossteil der Entwickler bereits in der Ausbildung

Struktogramme kennen gelernt hat und daher die Verwendung der Werkzeuge sehr schnell erlernbar ist.

Werden Funktionalitäten benötigt, die über die strukturierte Darstellung und Modifizierbarkeit des Quellcodes hinaus gehen, müssen Werkzeuge verwendet werden, die für etwa 10.000 Euro erhältlich sind. Hier die den eigenen Anforderungen entsprechende Lösung zu finden, bedarf allerdings einer umfangreicher Recherche. Bei der Selektion des passenden Werkzeugs sollten die Kosten, die für die eine oder andere Variante ganz besonders für die Einarbeitung oder Schulung entstehen nicht außer Acht gelassen werden. Gerade im Falle von UML können eventuell zwei Tools erforderlich sein, um einerseits für eine ausreichende Verständlichkeit des Quellcodes und andererseits des Gesamtprojekts sorgen zu können, da die meisten UML für die Implementation keine oder nur geringe Unterstützung anbieten.

Folgende Kriterien sollten zusätzlich bei der Evaluierung eines Reverse-Engineering-Tools einfließen:

Embedded Programmierung – Native Programmierung

In der Embedded-Programmierung fokussieren sich die meisten Entwickler auf die Darstellung von State-Charts und weiteren Diagrammen, die den dynamischen Aspekt einer Anwendung darstellen. Daher werden auch die Schwachstellen der UML in der Darstellung von Real-Time-Systemen oder paralleler Verarbeitung von vielen Herstellern durch eigene proprietäre Lösungen ausgeglichen. In der Native-Programmierung reichen die angebotenen UML-Diagramme aus. Teilweise werden diese aber auch durch zusätzliche Diagrammtypen wie z.B. Package-Diagrammen ergänzt.

Strukturierte Programmierung – Objektorientierte Programmierung

Strukturierte Programmierung erfordert entweder ein großes und mächtiges Werkzeug oder eine gut aufeinander abgestimmte Produktkette, um alle am Projekt beteiligten Kompo-

nenten abzudecken. Einarbeitungszeit und Kosten sind hier besonders hoch. Dafür zeichnen sich diese Produkte oder Tool-Ketten meist durch einen langen Lebenszyklus aus.

Da UML eine standardisierte ‚Modellierungssprache‘ ist, bietet sie hier klare Vorteile. Ist die Sprache erst einmal erlernt, ist der Umgang mit UML-Tools kein großes Problem mehr. Besonders die Einarbeitung von neuen Mitarbeitern, die bereits in UML geschult sind, wird durch den Einsatz von UML-Werkzeugen erheblich verkürzt. Nebenbei können manche dieser Tools auch strukturierten C-Code erzeugen.

Unterstützte Diagrammtypen

Nicht jedes UML-Tool unterstützt auch alle UML-Diagrammtypen. Entsprechend der Firmenphilosophie wird nur ein bestimmter Teil der möglichen Darstellungsformen unterstützt. Deshalb ist es wichtig sich für die Auswahl eines Produkts eine Liste der benötigten Diagrammtypen zusammenstellen. Mit einer genauen Definition der Anforderungen können teure Fehlinvestitionen in überdimensionierte Programmpakete, deren Funktionalität nur teilweise gebraucht wird, vermieden werden. Ebenso teuer kann aber auch eine ‚billige‘ Lösung werden, welche die Anforderungen nicht erfüllt und ergänzt oder ersetzt werden muss.

Unterstützungen in Code Generierung und Reverse Engineering

Ein wichtiger Aspekt beim Importieren bestehender Anwendungen ist die Genauigkeit des Reverse Engineerings. Ziel sollte sein, dass alle statischen und dynamischen Sichten erzeugt werden können. Dies gilt gleichermaßen für die strukturierte wie auch die objektorientierte Programmierung. Funktionshierarchien und Methodenkaskaden sind wichtige Informationen die besonders für die Optimierung von Systemen geeignet sind. Laufzeit- oder Multi-tasking-Probleme lassen sich so aber nicht lösen.