

Statische Codeanalyse mit NDepend 3.0

Die Metrik sehen

Programmzeilen per Hand zählen ist offensichtlich nicht der Weisheit letzter Schluss. Ohne maschinelles Erfassen sind Softwametriken nicht in den Griff zu bekommen. NDepend ist ein Werkzeug, das dem Entwickler dank Visual-Studio-Integration diese Arbeit abnimmt – auch ohne großen Einarbeitungsaufwand.

Auf einen Blick



Dipl.-Inform. **Andreas Heil** ist Gründer und Geschäftsführer der Rhein-Spree Software Engineering GmbH. Als Berater rund um .NET-Technologien unterstützt er derzeit das Microsoft Research Computational Science Lab in Cambridge. Zuvor arbeitete er für die Microsoft Deutschland GmbH im akademischen Umfeld. Sie erreichen ihn unter www.rhein-spree.com.

Inhalt

- › NDepend analysiert Code anhand gängiger Maßzahlen (Metriken).
- › Dazu bietet es dem Anwender drei grundlegende Werkzeuge: das Erheben der Maßzahlen selbst, ihre Umsetzung in Grafiken und eine eigene an SQL angelehnte Abfragesprache.

dnpCode

A1009NDepend

Softwametriken sind schön und gut, lassen sich aber – so wird oft vermutet – im eigenen Projekt kaum umsetzen. Zu groß sei der Aufwand, die Daten zu erfassen, die Werte zu berechnen oder auszuwerten. Doch weit gefehlt, denn mit den entsprechenden Mitteln geht selbst die Erfassung komplexer Metriken im Handumdrehen.

Für .NET-Entwickler stellt NDepend von Patrick Smacchia [1, 2] ein Werkzeug dar, um Code zu analysieren und eine Vielzahl verschiedener Maße von .NET-Quellcode zu erfassen und auszuwerten. Dabei legt das Tool viel Wert auf eine eingängige Visualisierung, damit der Anwender die Auswertungen schnell erfassen kann. Drei grundlegende Elemente bietet NDepend hierzu: die Metriken selbst, Abhängigkeitsgraphen und eine Abfragesprache mit dem Namen Code Query Language (CQL).

Die Mächtigkeit von NDepend kann abschrecken. Schließlich gilt es, neben den mannigfaltigen Funktionen des Tools auch noch die Grundlagen zu beherrschen. Doch der Aufwand lohnt sich. Zudem ist das Analyse-Tool mit Version 2 auch in Visual Studio 2010 benutzbar.

Dieser Artikel zeigt sowohl die Grundlagen als auch den praktischen Umgang mit NDepend und führt anhand der Codebasis der Microsoft Enterprise Library 5.0 [3] in die Geheimnisse der statischen Codeanalyse ein. Der Fokus liegt dabei auf der Auswertung der von NDepend unterstützten Metriken. Darüber hinaus lassen sich mittels NDepend wesentlich mehr Informationen auswerten, als sich an dieser Stelle darstellen ließen – angefangen von Entwurfsregeln bis hin zu Namenskonventionen.

Versionen und Installation

Bei NDepend handelt es sich um ein kommerzielles Analysewerkzeug, das mit 300 Euro für eine Einzelplatzlizenz zu Buche schlägt. Wer sich angesichts dieses Preises nicht ganz sicher ist, muss nicht gleich den Sprung ins kalte Wasser wagen, sondern kann sich mit einer zeitlich beschränkten Testversion seine Meinung bilden. Wer sich zum Kauf entscheidet, erhält neben einer Lizenz-ID zum Download der Vollversion eine Lizenzdatei.

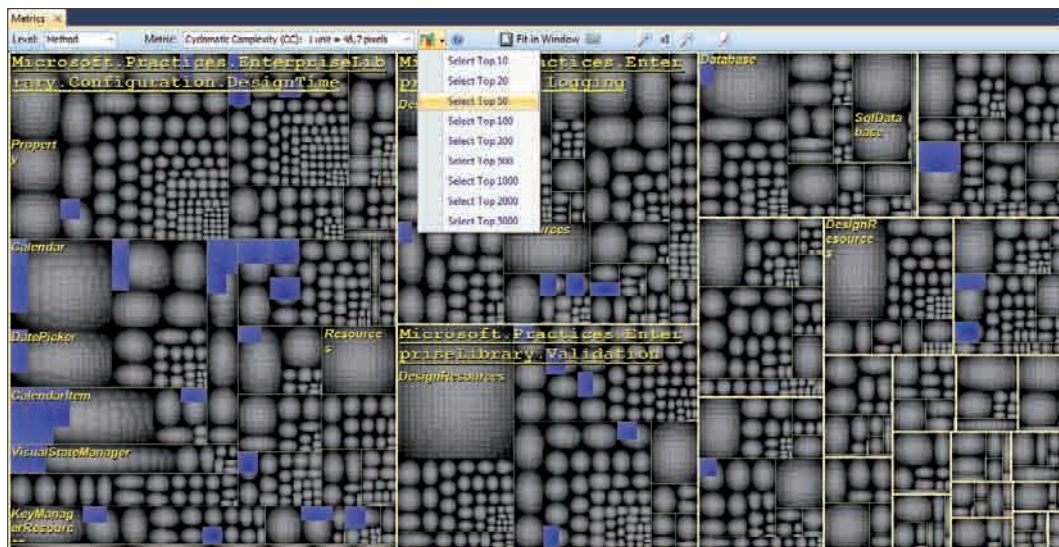
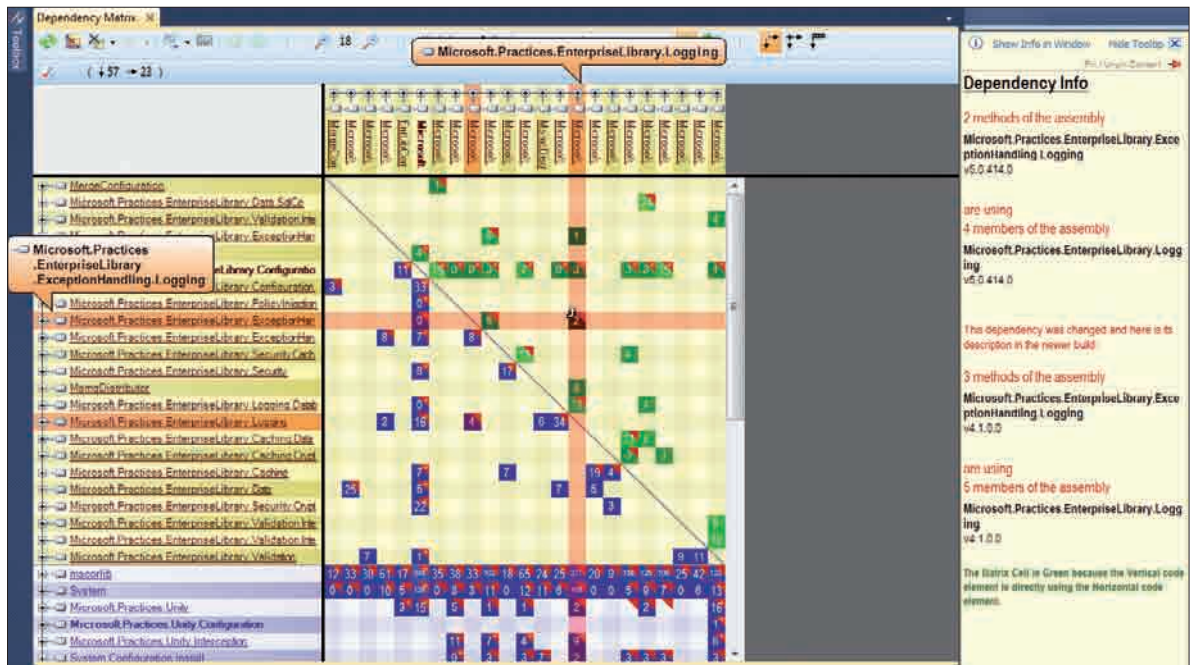
Schon fast untypisch für eine Software dieses Umfangs kommt NDepend in einer schlichten ZIP-Datei daher. Ähnlich einfach verhält es sich mit der Installation: Nach dem Entpacken des Archivs in einem beliebigen Ordner gilt es lediglich noch, die Lizenzdatei in denselbigen zu kopieren. Insgesamt drei Dateien sind in diesem Ordner von besonderem Interesse: Neben *VisualNDepend.exe*, der grafischen Oberfläche, und der Kommandozeilenanwendung *NDepend.Console.exe* ist zunächst die Datei *NDepend.Install.VisualStudio.AddIn.exe* von Bedeutung; sie sorgt für die Einbindung von NDepend in Visual Studio. Ähnlich unspektakulär wie zuvor verläuft auch diese Installation. Neben der gewünschten Version von Visual Studio ist lediglich der Installationsumfang auszuwählen.

Die vollständige Installation bietet neben der Abhängigkeitsmatrix, den entsprechenden Graphen und den Darstellungen der Metriken den CQL-Editor und das automatische Aktualisieren der Analyseergebnisse bei der erneuten Kompilierung des Projekts. Die Minimalinstallation hingegen gestattet zwar NDepend-Funktionen wie CQL-Abfragen, bietet jedoch keine Möglichkeit, die visuellen Darstellungen von NDepend von Visual Studio aus zu nutzen. Am parallelen Einsatz von VisualNDepend führt hierbei kein Weg vorbei. Wer daher den vollständigen Leistungsumfang von NDepend aus Visual Studio heraus nutzen möchte, sollte sich von vornherein für die vollständige Installation entscheiden.

Bevor es überhaupt möglich ist, die verschiedenen Metriken in NDepend zu nutzen, ist zunächst das zu analysierende Projekt zu kompilieren. NDepend nutzt sowohl die .NET-Assemblies als auch die PDB-Dateien eines Projekts als Informationsquelle. Zum Erfassen der Metriken ist es also nötig, das betreffende Projekt zumindest einmal zu übersetzen. Über den Menübefehl *NDepend | Attach new NDepend Project to current VS Solution* und danach *NDepend | Run Analysis* lässt sich dieser erste Analysedurchgang starten.

Der Menübefehl *NDepend | NDepend Project | View Report* fasst den Analysebericht zusammen und öffnet ihn im Webbrowser. Das HTML-Format eignet sich hervorragend für Ausdrücke und für den Vergleich mit zuvor erstellten Analysen –

[Abb. 1] Die Enterprise Libraries 5.0 und 4.1, verglichen mit NDepend.



[Abb. 2] NDepend stellt Metriken mit Treemaps dar.

eine Möglichkeit, die gerade für die Kommunikation über die Metriken nicht zu unterschätzen ist. Der Bericht gliedert sich in mehrere Abschnitte, die unterschiedliche Messungen behandeln. Die aufgeführten Metriken sind dabei nach Assemblies, Namensräumen und Typen wie auch Methoden gegliedert (siehe auch den Kasten *Nichts wegwerfen*).

Ein Bild sagt mehr

Natürlich können Sie auf die Information aus NDepend oder Visual Studio zugreifen. In der Entwicklungsumgebung ist die entsprechende Ansicht über *NDepend | Metrics* einblendbar. Hierbei handelt es sich um eine sogenannte Treemap, die die einzelnen Elemente im Verhältnis der betrachteten Eigenschaften gegenüberstellt. Die Größe der

Elemente spiegelt dabei den Wert der betrachteten Metrik wider (Abbildung 2).

In der Werkzeugleiste am oberen Rand der Ansicht kann der Nutzer zwischen Methoden, Feldern, Typen, Namensräumen und Assemblies umschalten, die gewünschte Metrik auswählen und zusätzlich die Top-Kandidaten hervorheben, die aufgrund ihrer Werte besonders aus der Reihe tanzen. Die Auswahl eines Treemap-Felds zeigt die Werte der verfügbaren Metriken in einem Infofenster an.

Sollte einmal nicht klar sein, was der einzelne Eintrag in dieser Ansicht bedeutet, hilft ein Klick darauf weiter. Eine Internetverbindung vorausgesetzt, öffnet NDepend dann ein Browser-Fenster und zeigt die Beschreibung der Metrik von der NDepend-Webseite an (Abbildung 3).

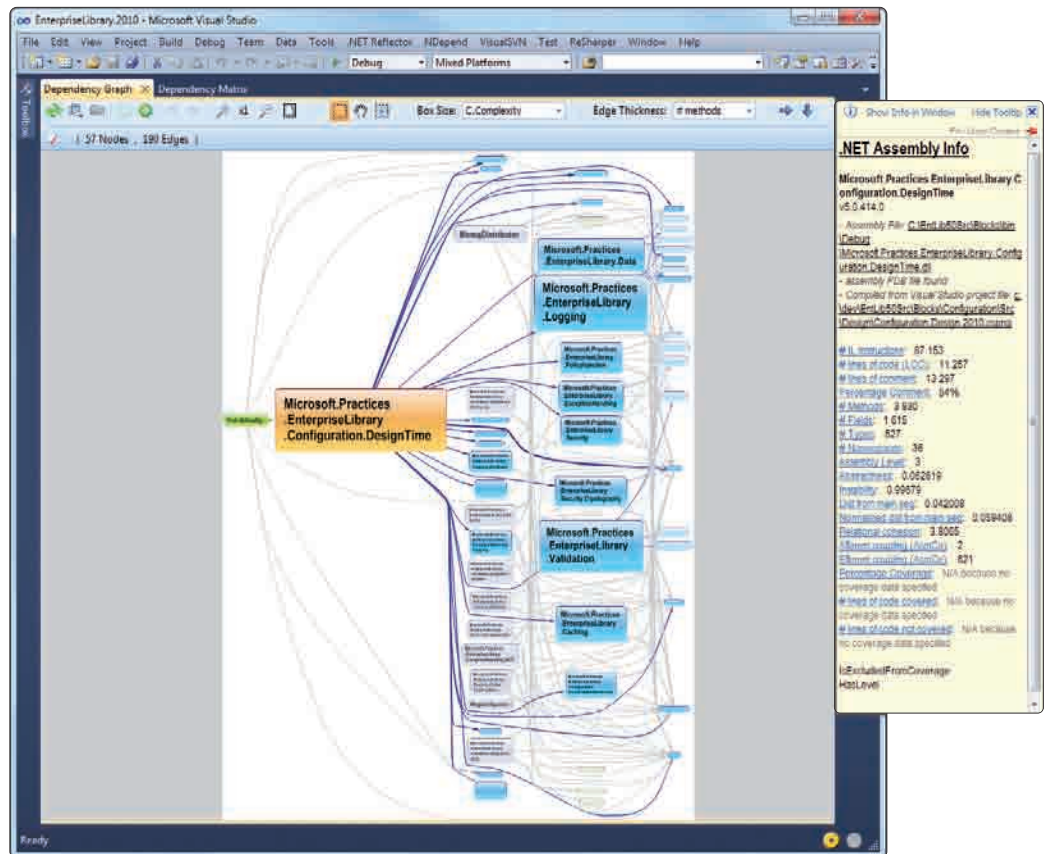
Abhängigkeiten in der Matrix

Wie NDepend Abhängigkeiten darstellt, ist anhand der „Dependency Matrix“ zu sehen, die Abbildung 1 zeigt. Diese Ansicht bringt die verschiedenen Elemente des Quellcodes in direkten Zusammenhang. Jede gefüllte Zelle beschreibt eine Abhängigkeit. Blaue Zellen beschreiben die Nutzung des horizontalen Elements durch das jeweilige vertikale, während grüne Zellen zeigen, dass das vertikale Element eines der horizontalen Leiste verwendet. Zusätzlich ist die Anzeige zwischen direkter und indirekter Nutzung umschaltbar, um die Abhängigkeiten zwischen einzelnen Elementen noch genauer zu analysieren.

Die Gewichtung der Abhängigkeit zeigt sich ebenfalls an den verschiedenen Werten. So lässt sich die Anzahl der Methoden,



[Abb. 3] Detailangaben im Infofenster.



[Abb. 4] Der Abhängigkeitsgraph, den NDepend zur Enterprise Library 5.0 erzeugt, und weitere visuelle Hilfen dazu.

der Typen oder der genutzten Namensräume betrachten.

Auch hier liefert die Auswahl einer Zelle einen Infotext mit zusätzlichen Details, wie es schon in Abbildung 1 zu sehen war; die Abhängigkeiten sind dadurch recht einfach zu erfassen. Hier ist dann in Klartext zu lesen, dass beispielsweise drei Klassen aus der einen Assembly fünf Klassen aus einer anderen Assembly verwenden. Beim Vergleich zweier Berichte wird zusätzlich der beobachtete Unterschied beschrieben. Ein Klick auf die entsprechende Zelle liefert daraufhin einen Graphen, der die Abhängigkeiten in ein Diagramm umsetzt.

Von der Matrix zum Graphen

Der Graph nutzt einen anderen Visualisierungsansatz als die Matrix. Er hat zwar mit dem typischen Skalierungsproblem zu kämpfen, das sich bei einer großen Anzahl von Komponenten ergibt, ist jedoch intuitiv zu verstehen. Abbildung 4 zeigt zum Beispiel alle Abhängigkeiten innerhalb der Enterprise Library 5.0 in einem solchen Graphen an; für die Größe der Boxen wurde hier die Metrik zyklomatische Komplexität (McCabe-Metrik [4]) gewählt. Welche der Komponenten am ehesten eine genauere Prüfung benötigt, dürfte sich hier schnell erschließen. Das Kontextmenü eines Ele-

ments erlaubt es, nicht nur durch die Struktur des Graphen zu navigieren, sondern auch zahlreiche visuelle Hilfen ein- und auszublenden, die die Analyse erleichtern.

Auch hier ist es dem Benutzer nahezu freigestellt, welche Daten er visualisieren will. So kann er in dieser Ansicht für die Größe der Boxen aus den unterstützten Metriken wie zum Beispiel Codezeilen oder zyklomatischer Komplexität wählen, wohingegen die Pfeildicke die Anzahl der ab-

Nichts wegwerfen

Die Berichte, die NDepend erzeugt, befinden sich im Unterverzeichnis *NDependOut* des jeweils analysierten Projekts. Ältere Berichte überschreibt das Tool nicht, sondern archiviert sie für die spätere Verwendung und den Vergleich verschiedener Berichte. Daher empfiehlt es sich durchaus, diese Ordner in das Code-Repository aufzunehmen, um auch später auf die Daten zugreifen zu können. Mittels des Menübefehls *NDepend | NDepend Project | Load a previous Analysis Report of the Project [...]* lassen sich die Analysen zu Vergleichszwecken wieder laden. Abbildung 1 zeigt den direkten Vergleich zwischen den Enterprise Libraries Version 5.0 und 4.1 von Microsoft.

hängigen Methoden, Klassen oder Felder widerspiegeln kann.

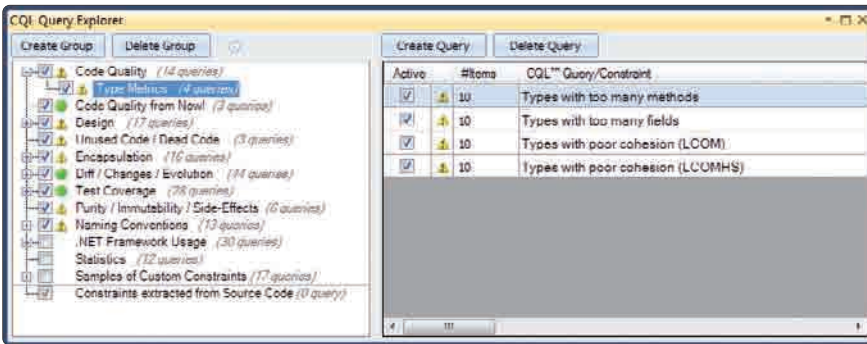
Das Frage-Antwort-Spiel

Eine weitere große Hilfe ist die Abfragesprache CQL. Die Sprache ist auf der einen Seite relativ einfach zu nutzen, auf der anderen Seite so mächtig, dass sich nahezu jegliche Information über die Codebasis abfragen lässt. Für Interessierte ist eine Spezifikation unter [5] verfügbar, die zum Erlernen jedoch nicht unbedingt erforderlich ist.

Aus Visual Studio heraus lässt sich CQL gleich mit zwei Tools nutzen: mit dem CQL Query Explorer und dem CQL Query Editor. Der Explorer (Abbildung 5) ermöglicht den direkten Zugriff auf vorgefertigte Abfragen. Um auch bei umfangreichen Auswertungen den Überblick zu behalten, lassen sich diese zu Gruppen zusammenfassen. Jede Gruppe zeigt dabei durch ein typisches Icon ihren jeweiligen Status der letzten Analyse an.

Der CQL Query Editor hingegen ermöglicht das Erstellen neuer und das Bearbeiten vorhandener Abfragen. Die Auswahl einer Abfrage im Explorer öffnet diese automatisch im Editor. Der Editor selbst besteht aus zwei Komponenten: Im oberen Bereich kann die Abfrage eingegeben werden, der untere Bereich zeigt die Ergebnisse an (Ab-

UI Test leicht gemacht



[Abb. 5] Der CQL Query Explorer.

bildung 6). Mittels der *Export to*-Schaltfläche kann der Nutzer die Abfrage und deren Ergebnisse in unterschiedlichen Formaten exportieren. Neben dem Export in HTML (Abbildung 7), Excel, XML und Text lassen sich die Ergebnisse auch im Tool selbst als Graph oder Matrix anzeigen.

Metriken

Die ausgewerteten Metriken geben eine Übersicht über den Aufbau der analysierten Visual-Studio-Lösungsmappen. Hier findet sich zunächst der Klassiker der Softwaremetriken in zwei Versionen wieder: als Anzahl der Programmzeilen (LOC, Lines of Code) und als Anzahl der Kommentarzeilen (CLOC, Comment Lines of Code). Hervorzuheben ist die Zählweise von NDepend: Als Maß für die Programmzeilen stützt sich die Software auf die logischen

Programmzeilen (LLOC, Logical Lines of Code) auf Basis der im IL-Code befindlichen Sequenzpunkte. Das macht die Metrik sprachunabhängig und unempfindlich gegenüber unterschiedlichen Programmierstilen mehrerer Entwickler. Auf der anderen Seite werden Namensräume, Klassen-, Felder- und Methodendeklarationen nicht berücksichtigt, da diese im IL-Code nicht als Sequenzpunkte erscheinen. Die Zahl der von NDepend ermittelten Codezeilen kann somit von den Ergebnissen anderer Tools abweichen, weshalb ein Vergleich nur auf Basis der durch von NDepend ermittelten Werte sinnvoll ist.

Als weitere Werte finden sich die Anzahl analysierter Assemblies, abstrakter und konkreter Klassen und Interfaces. Als besonders interessant erweist sich die Anzahl der generischen Klassen und Methodendefinitionen und der öffentlichen Klassen und Methoden. Mit ihnen lässt sich ein erster Eindruck vom Aufbau der Software gewinnen, ohne in den Code zu schauen.

Mit CQL ins Detail

Richtig spannend wird es freilich erst mit der Möglichkeit, die gelieferten Metriken nach eigenen Wünschen auszuwerten. Hier spielt die Abfragesprache CQL ihre Stärken aus. Das Beispiel der Microsoft Enterprise Library 5.0 soll zeigen, wie CQL hilft, die Software zu analysieren.

CQL ist recht einfach zu erlernen. Die Sprache ist stark an die Syntax von SQL angelehnt und folgt mit leichten Variationen generell deren Muster:

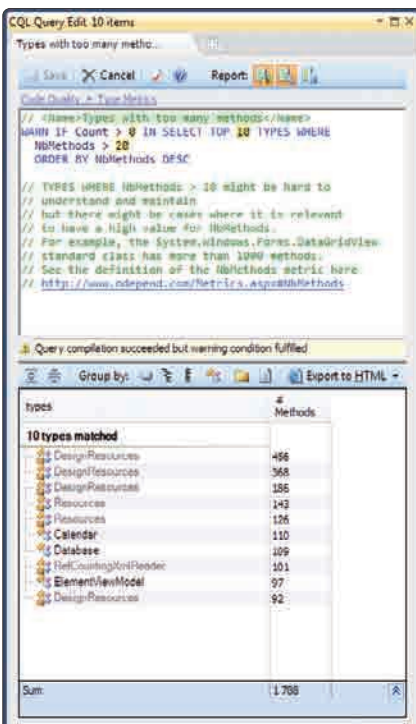
```
SELECT TOP n FROM x WHERE y ORDER BY z
```

Wer direkt loslegen möchte, erfährt durch NDepend die bestmögliche Unterstützung. Das beginnt im CQL Editor mit einem sehr guten IntelliSense (Abbildung 8).

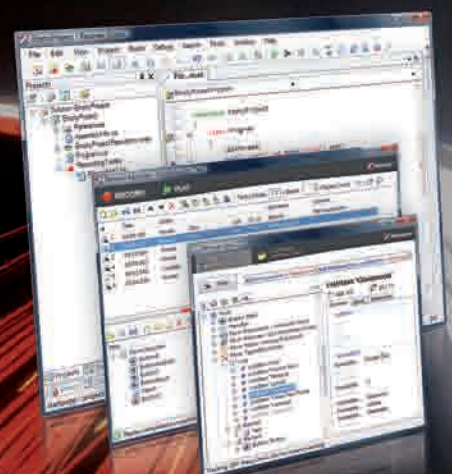
Zunächst ist der Typ zu bestimmen, der analysiert werden soll. Die zur Verfügung stehenden Elemente führt Tabelle 1 auf.

- ✓ Aufzeichnen & Abspielen von Benutzeraktionen
- ✓ Automatische .NET Codegenerierung
- ✓ Testen zahlreicher Technologien

*Winforms / C# / VB.NET
Flash / Flex / Win32 / MFC
WPF / Silverlight / Web 2.0 / AJAX*

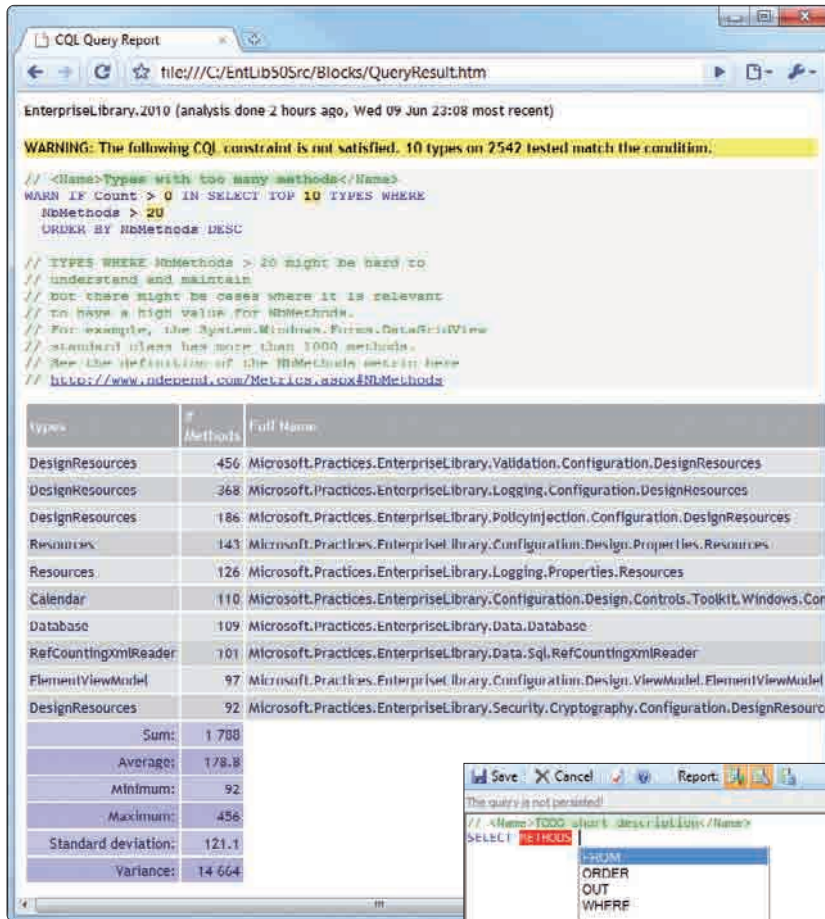


[Abb. 6] Der CQL Query Editor.



Jetzt testen!
➔ Auf der Heft-CD

www.ranorex.de
Ranorex GmbH | Graz, Österreich
+43 316 28 13 28 | info@ranorex.de



[Abb. 7] CQL-Berichte im Browser.

Nun ist die Abfrage genauer zu spezifizieren. Das geschieht mittels der in Tabelle 2 aufgelisteten Einschränkungen. Zwar gelten sowohl *ORDER BY* als auch *WHERE* als optional, doch muss eine Abfrage mindestens eines der beiden Schlüsselwörter enthalten.

Query oder Constraint

Jede Abfrage in CQL lässt sich mittels des folgenden Konstrukts in eine sogenannte Constraint, also eine Bedingung oder Einschränkung, umwandeln:

```
WARN IF (BEDINGUNG) IN SELECT ...
```

Hierbei unterscheidet NDepend zwei unterschiedliche Arten von Constraints:

Tabelle 1

Die wichtigsten Elemente von CQL-Abfragen.

CQL	Umfasst
ASSEMBLIES	Assemblies
FIELDS	sämtliche Felder einschließlich Konstanten und nur lesbarer Felder
METHODS	abstrakte, virtuelle und nicht virtuelle Methoden
NAMESPACES	.NET-Namensräume
TOP	auszuwählende Anzahl
TYPES	Klassen, abstrakte Klassen, Strukturen, Enumerationen, Delegationen und Schnittstellen

Tabelle 2

Abfrageelemente in CQL.

CQL	Bedeutung
FROM	Suche einschränken
ORDER BY	Ergebnisse sortieren
OUT OF	Suche einschränken
WHERE	Bedingungen
TOP	anzuweisende Anzahl

wert überschreiben dürfen; denn vielleicht ist nicht alles in kurzen Methoden zu implementieren. Die Abfrage lautet somit:

```
WARN IF Count > 50 IN SELECT METHODS
WHERE NLinesOfCode > 25
```

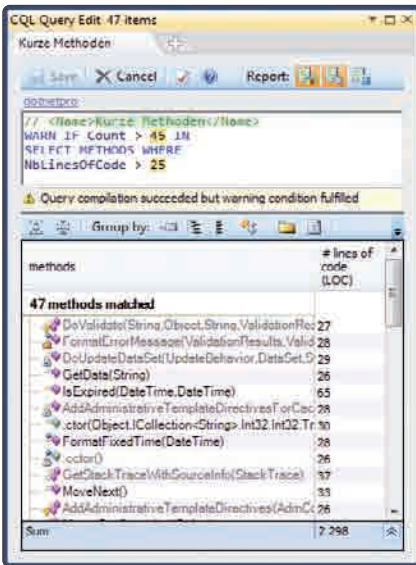
Im CQL Query Editor ist sofort zu sehen, dass 47 Methoden diese Vorgabe verletzen (Abbildung 9). Auch der CQL Query Explorer merkt dies sofort und macht mittels des kleinen gelben Warndreiecks darauf aufmerksam (Abbildung 10).

Nun erscheint eine willkürlich gewählte Anzahl von 45 Methoden bei einem Umfang von nahezu 30 000 Codezeilen nicht unbedingt geeignet, um die Qualität der Enterprise Library zu beurteilen. Soll die Bewertung anhand relativer Werte erfolgen, lässt sich nun festlegen, dass fünf Prozent der Methoden gegen die Regel kurzer Methoden verstoßen dürfen. Damit sieht das Ergebnis der Auswertung ganz anders aus (Abbildung 11).

Zahlenwerte lassen sich im CQL Query Editor mittels eines Reglers verändern. Der jeweilige Maximalwert hängt dabei von den durch die Metrik ermittelten Daten ab. Gerade bei der Suche nach absoluten Werten erhält der Nutzer hierdurch zumindest ein Gefühl dafür, in welchen Bereichen sich die Abfrage bewegt.

Abbildung 12 zeigt, dass die Abfrage nun im grünen Bereich liegt. Das heißt: Weniger als fünf Prozent der Methoden überschreiten eine Länge von 25 Zeilen. Dies ist ein gutes Zeichen, denn anscheinend haben die Microsoft-Programmierer das Prinzip von kurzen Methoden überwiegend befolgt. Die Qualität der Enterprise Library 5.0 stellt sich hinsichtlich der Methodenlängen im Vergleich zur vorherigen Auswertung somit anders dar.

Das Beispiel verdeutlicht aber auch die Gefahr, dass ein fahrlässiger Umgang mit Metriken schnell zu überstürzten oder fehlerhaften Beurteilungen führen kann. Mit

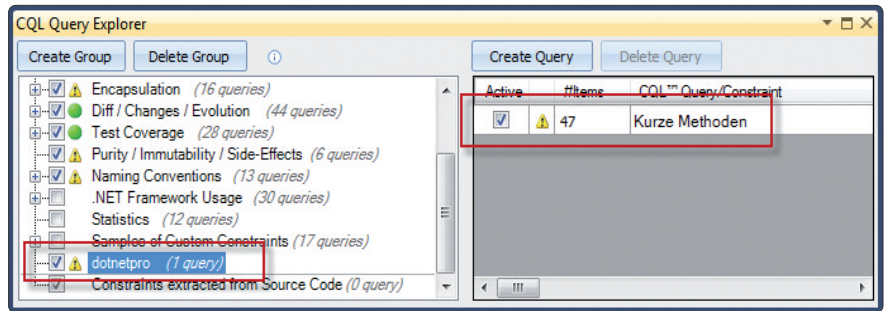


[Abb. 9] Eine CQL-Abfrage zum Ermitteln der Codequalität.

einem Werkzeug wie NDepend lassen sich derart viele Informationen ermitteln, dass genau zu überlegen ist, welche Auswertung welchen Zweck hat und was die Zahlen bedeuten – und was nicht.

Tabelle 3 gibt eine Übersicht über eine Reihe ausgewählter Softwaremetriken und Messzahlen in NDepend. Die bereits zuvor besprochene Metrik der Codezeilen lässt sich mittels *NbLinesOfCode* oder *NbLinesOfComment* auswerten. Auf den Anteil von Kommentarzeilen wiederum kann in einer CQL-Abfrage direkt mittels *PercentageComment* zugegriffen werden.

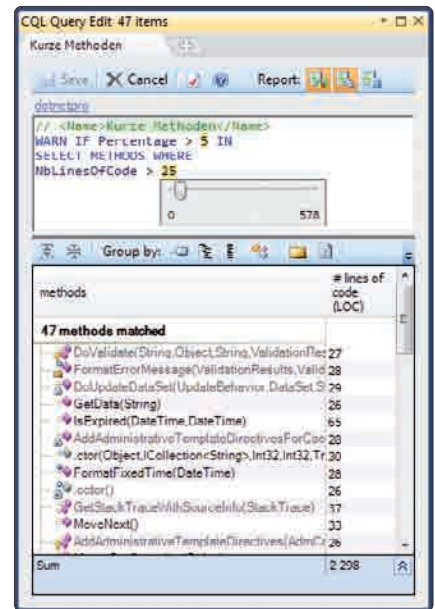
Die zyklomatische Komplexität ist ein spezielles Maß, das die Komplexität von Me-



[Abb. 10] Warnhinweise im CQL Query Explorer.

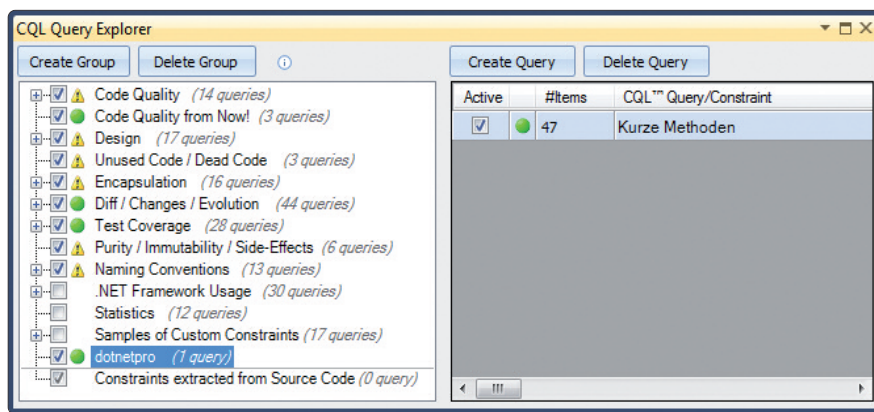
thoden ausdrücken soll – mehr dazu können Sie im Grundlagenbeitrag *1024 KByte guten Code, bitte! lesen* [6].

DepthOfInheritance spiegelt ein traditionelles Maß wider, nämlich die Vererbungstiefe in objektorientierten Systemen. Hier empfiehlt es sich jedoch, .NET-spezifische Besonderheiten zu beachten: So gilt allgemein im Software Engineering ein Grenzwert von 7 oft als traditionelles Maß. Im .NET Framework erbt jeder statische Typ automatisch von *System.Object*, was die Vererbungstiefe bereits geringfügig beeinflusst. Tatsächlich weisen nach Aussagen der NDepend-Entwickler Klassen, die beispielsweise von *System.Windows.Forms.Control* abgeleitet werden, im Schnitt eine Vererbungstiefe von 5,3 auf. Bei dem gängigen Richtwert von 7 bliebe hier nur wenig Spielraum für weitere Vererbung. Die Auswertung einer Reihe von WPF-Projekten für diesen Artikel hat sogar eine durchschnittliche Vererbungstiefe von 9 ergeben. Grenzwerte für dieses Maß scheinen daher in Bezug auf das jeweilige Projekt von Vorteil.



[Abb. 11] Die neu konfigurierte Abfrage im CQL Query Editor.

Mittels *NbChildren* lässt sich eine weitere traditionelle Softwaremetrik ermitteln: die Anzahl der direkten Kinder eines Typs (NOC, Number of Children). Ein übertrie-



[Abb. 12] Alles im grünen Bereich im CQL Query Explorer.

ben hoher Wert hierfür weist oft auf einen Missbrauch der Vererbung hin und somit einen Designfehler. Übermäßig hohe Werte sollten daher vermieden werden. Allerdings ist bei der Auswertung darauf zu achten, dass Unterklassen außerhalb des analysierten Codes nicht mitgezählt werden – die Maßzahl sagt somit nur etwas über den internen Entwurf des Codes aus.

Zwei Metriken, die Robert C. Martin eingeführt hat [7], beschreiben die Kopplung innerhalb einer Assembly: *Ca* und *Ce* stehen für „afferent coupling“ (eintreffende Kopplung) und „efferent coupling“ (ausgehende Kopplung). Mit *Ca* lässt sich demnach die Anzahl von Typen außerhalb einer Assembly ermitteln, die von Typen innerhalb der Assembly abhängig sind; *Ce* nimmt die andere Richtung ins Visier, die Abhängigkeit von Typen innerhalb einer

Table 3

Eine Auswahl von Methoden, die NDepend unterstützt.

Metrik	Bedeutung
NbLinesOfCode	Anzahl Codezeilen
NbLinesOfComment	Anzahl Kommentarzeilen
PercentageComment	Anteil Kommentarzeilen
CyclomaticComplexity	Zyklomatische Komplexität
ILNestingDepth M	Verschachtelungstiefe
DepthOfInheritance	Vererbungstiefe
NbChildren	Anzahl direkter Kinder
Ca	Eingehende Kopplung
Ce	Ausgehende Kopplung
ABT	Assoziation zwischen Typen
LCOM	Lack of Cohesion
LCOMHS	Lack of Cohesion of Methods Henderson-Sellers
Abstractness	Abstraktheit
Instability	Instabilität
NormDistFromMainSeq	Distanz von der Ideallinie: Abstraktheit + Instabilität = 1
MethodRank	Googles PageRank Algorithmus

Assembly von Typen außerhalb der Assembly. Diese Maßzahl steht mittels *NameSpaceCa/NameSpaceCe*, *MethodCa/MethodCe* und *FieldCa* auf verschiedenen Ebenen des Codes zur Verfügung.

Zwei hieraus abgeleitete Maßzahlen von der Ebene der Assemblies sind die Abstraktheit (*Abstractness*) und die Instabilität (*Instability*). Erstere gibt das Verhältnis interner abstrakter Klassen zur Anzahl interner Klassen im Bereich von 0 bis 1 wieder, wobei 1 eine rein abstrakte Assembly beschreibt. Die Instabilität berechnet sich dabei aus dem Verhältnis von ausgehender Kopplung zur gesamten Kopplung einer Assembly. Hierbei deutet ein größerer Wert im Bereich von 0 bis 1 auf eine Assembly hin, die eher dazu neigt, sich zu verändern, als auf eine stabile Assembly, die sich durch einen kleinen Wert auszeichnet. Mit *NormDistFromMainSeq* lassen sich diese Werte ins Verhältnis setzen und die Balance zwischen Abstraktheit und Instabilität beschreiben. Während abstrakt und stabil mit 0 bewertet wird, steht 1 für konkret und instabil. Als Richtwert gilt: Weisen mehr als 15 Prozent der Assemblies einen Wert von 0,7 auf, sollte der Entwurf geprüft werden. In CQL lässt sich dies in folgender Abfrage ausdrücken:

```
WARN IF Percentage > 15 IN SELECT
ASSEMBLIES WHERE
NormDistFromMainSeq > 0.7
```

Hier gilt es demnach den goldenen Mittelweg zu finden. Die über den NDepend-Bericht bereitgestellte Grafik bereitet die Informationen besonders anschaulich auf: Je weiter sich eine Assembly außerhalb des grünen Bereichs befindet, desto argwöhnischer sollte ihr Entwurf überprüft werden.

Ein weiteres Maß zur Beschreibung der Abhängigkeiten, in diesem Fall von Klassen zueinander (Association Between Classes),

ist über *ABC* nutzbar und liefert die Menge von Methoden, Eigenschaften und Feldern anderer Klassen, die direkt innerhalb des analysierten Typs Verwendung finden.

LCOM hingegen ermittelt den fehlenden Zusammenhang zwischen Methoden (Lack of Cohesion of Methods). Dieses Maß wird vornehmlich zur Analyse mangelnder Kapselung herangezogen. Der fehlende Zusammenhang an Methoden kann ein Indikator dafür sein, dass eine Klasse geteilt werden kann, da sie mehrere voneinander unabhängige Zuständigkeiten besitzt und somit vermutlich ein Entwurfsfehler vorliegt. Allerdings lässt sich statistisch kein eindeutiger Zusammenhang nachweisen, weshalb die Metrik in der Fachliteratur kontrovers ist. *LCOMHS* ermittelt eine normalisierte Version der vorherigen Maßzahl, wodurch die Werte immer zwischen 0 (bester Wert) und 1 (schlechtester Wert) liegen.

Als Letztes ist eine eher ungewöhnliche, aber auch interessante Messzahl zu erwähnen: *MethodRank*. Sie basiert auf dem Prinzip des PageRank-Algorithmus von Google und beschreibt die Abhängigkeiten zwischen Methoden. Größere Werte zeugen von hoher Abhängigkeit und somit auch von größeren Auswirkungen bei Fehlern in diesen Methoden. Als Faustregel gilt daher, dass Methoden mit dem höchstem *MethodRank*-Wert am sorgfältigsten getestet werden sollten.

Fazit

Was sich mit NDepend über den eigenen – und fremden – Code herausfinden lässt, konnte auch dotnetpro-Chefredakteur Tilmann Börner schon am eigenen Leib erfahren [8]. Dabei hilft NDepend nicht nur, den Code zu verstehen, sondern auch objektiv zu bewerten und die Entwicklung quantitativ zu erfassen und zu beobachten. NDepend stellt ein Werkzeug zur statischen Codeanalyse dar, das genau diesem Zweck dient. Konsequenterweise eingesetzt, lassen sich bereits mit wenigen Handgriffen Schwächen im eigenen Code aufdecken.

Überrascht hat die NDepend-Integration überdies in Sachen Leistungsverhalten: Nur selten ist auf einer durchschnittlichen Entwicklermaschine offensichtlich, dass die Analyse läuft. Lediglich beim Erzeugen von Berichten können einige wenige Sekunden verstreichen, die der Benutzer warten muss.

Ärgerlich ist einzig die Tatsache, dass angelegte NDepend-Fenster beim erneuten Start von Visual Studio nicht mehr am zuvor festgelegten Platz erscheinen, sondern

Gespräch mit Patrick Smacchia, Lead Developer von NDepend

Patrick Smacchia ist Visual C# MVP und entwickelt seit mehr als 15 Jahren Software. Nach einem Abschluss in Mathematik und Informatik arbeitete er an zahlreichen Projekten, darunter am Börsensystem der Société Générale, am Amadeus Ticketreservierungssystem für Fluggesellschaften und an einer Satellitenbasisstation der Firma Alcatel. Patrick ist Autor des 2006 erschienen Buches *Practical .NET 2 and C# 2* und arbeitet seit dieser Zeit an NDepend. dotnetpro-Autor Andreas Heil konnte mit Patrick Smacchia über NDepend sprechen.



dotnetpro NDepend ist in Version 3.0 erhältlich, und der Funktionsumfang ist sehr beeindruckend. Was war der Grund, eine solche komplexe Software zu entwickeln?

Smacchia Die Intention war, ein einfaches Tool zu entwickeln, das große und komplexe Programme entmystifiziert. Unser Ziel ist, die meisten Aspekte aus dem breiten Gebiet der Softwarekomplexität in einem einzelnen Tool bereitzustellen. Wir möchten eine einheitliche Lösung, die die verschiedenen Mängel zeigt, wie verworrene Architekturen, zu große Klassen und Methoden, geringe Codeabdeckung durch Tests oder unkontrollierte Evolution.

dotnetpro Eine Testversion von NDepend lässt sich über einen langen Zeitraum kostenfrei nutzen. Welche Einschränkungen gibt es hier für den Benutzer?

Smacchia Das Ziel der freien Testversion ist zweierlei: Zunächst soll der potenzielle Benutzer die Zeit haben, sich mit einem innovativen Tool auseinanderzusetzen, das in keine typische Kategorie passt. Sodann soll er ein freies Tool erhalten, das zwar weniger Features als die professionelle Version bietet, aber dennoch nützlich für Studenten, Forscher und Open-Source-Software-Teams ist. Tatsächlich vergeben wir oftmals kostenfreie Pro-Lizenzen an Akademiker und OSS-Teams, die sie wirklich benötigen. Wir sehen hier die Möglichkeit, sowohl den nicht kommerziellen Communities zu helfen als auch unser Produkt in ganz verschiedenen Sphären der Entwicklung zu fördern.

dotnetpro NDepend unterstützt verschiedene Continuous-Integration-Systeme wie beispielsweise CruiseControl.NET oder TeamCity. Benötigen Entwickler hierfür eine spezielle Lizenz?

Smacchia Ja. Es gibt hier zwei Arten von Lizenzen: für Entwickler und für Build-Maschinen. Mit NDepend 3.0 ist das Produkt jedoch vollständig in Visual Studio 2010, 2008 und 2005 integriert. Wir haben enormen Aufwand betrieben, um zu vermeiden, dass es Visual Studio ausbremst. Unser primäres Ziel war es, den Benutzer von allen Möglichkeiten des Tools profitieren zu lassen – live, in der Visual-Studio-Umgebung. Wir glauben, das ist eine wesentlich bessere Erfahrung, als nur Berichte zu lesen, die der Build-Server generiert.

dotnetpro NDepend wird mittels XCOPY-Deployment installiert. Lässt sich VisualNDepend vom USB-Stick ausführen? Das wäre speziell für freiberufliche Berater sehr interessant.

Smacchia Wir ziehen dies gerade für ein mittelfristiges Release in Betracht. Wenn jedoch NDepend in Visual Studio als Add-in installiert ist, ist sowieso die Minimalinstallation auf der Maschine nötig.

dotnetpro Unterstützte Metriken in NDepend behandeln Assemblies, Namensräume, Typen und Methoden – da fragt sich sicher der eine oder andere Leser, ob Eigenschaften in .NET nicht berücksichtigt werden.

Smacchia Die primäre Datenquelle für NDepend ist der IL-Code in einer Assembly. Eigen-

schaften sind ein reines C#- und VB.NET-Konstrukt. IL weiß gar nichts über Eigenschaften, es kennt lediglich Getter- und Setter-Methoden. Demzufolge erkennt NDepend Eigenschaften mittels dieser *get_*- und *set_*-Methoden. Diese speziellen Methoden werden mit *IsProperty-Getter*- und *IsPropertySetter*-Flags markiert, um es dem Benutzer zu ermöglichen, seine eigene Logik auf Eigenschaften anzuwenden.

dotnetpro Wie geht es weiter mit NDepend?

Smacchia Von meiner Perspektive aus war es ein hochgestecktes Ziel, alle NDepend-Funktionen in Visual Studio zu integrieren, ohne dabei einen für den Benutzer merklichen Performance-Verlust zu verursachen. Eingedenk der Tatsache, dass NDepend zur Laufzeit mit einer sehr großen Codebasis von bis zu mehreren Millionen Zeilen umgehen kann – was für uns vor zwei Jahren noch reine Science-Fiction war –, haben wir jetzt noch ambitioniertere Pläne für die nächsten zwei Jahre, die wir wiederum heute noch als Science-Fiction betrachten. Das ist es, was ich am NDepend-Abenteuer so liebe: die Herausforderung, immense Erwartungen in echte Merkmale zu verwandeln. Auf diesem Weg leben wir eine überaus interessante Programmiererfahrung und liefern beständig neue Funktionen, die bisher in keinem Entwickler-Tool angeboten wurden. Für den Moment halten wir die zukünftigen Pläne aber noch geheim. Wir können aber sagen, dass wir den Zeitplan für NDepend 4 an künftige Visual-Studio-Versionen anpassen. Ich würde sagen, in 18 Monaten tut sich in dieser Hinsicht etwas. Aber wer weiß das schon?

oft nicht mehr angedockt sind. Nach Aussagen der NDepend-Entwickler handelt es sich dabei jedoch um ein bekanntes Problem, das auf das Visual-Studio-API zurückzuführen ist. **[jpl]**

[1] NDepend Home Page, www.ndepend.com
[2] Patrick Smacchia, www.smacchia.com

[3] MSDN, Microsoft Enterprise Library 5.0, www.dotnetpro.de/SL1009Ndepend1

[4] Wikipedia, McCabe-Metrik, www.dotnetpro.de/SL1009Ndepend2

[5] Code Query Language 1.8 Specification, <http://ndepend.com/CQL.htm>

[6] Andreas Heil, 1024 KByte guten Code, bitte! Quelltext mittels Software-Metriken messen

und bearbeiten, dotnetpro 9/2010, Seite 20ff., www.dotnetpro.de/A1009Grundlagen

[7] Robert C. Martin, Agile Principles, Patterns, and Practices in C#, ISBN 978-0-13-185725-4

[8] Ralf Westphal, Die Gegend auskundschaften, Refaktorisierung einer Brownfield-Anwendung, dotnetpro 5/2010, Seite 124 ff., www.dotnetpro.de/A1005Kolumne