M-System NT - Ein flexibles, datenbank-basiertes Mess- und Analyse-System

Jürgen Münch, Axel Wickenkamp

Fraunhofer-Institut für Experimentelles Software Engineering

juergen.muench@iese.fraunhofer.de, axel.wickenkamp@iese.fraunhofer.de

Zusammenfassung:

Wesentliche Anforderungen an Messansätze bei der Entwicklung von Software und software-intensiven Systemen sind Zielorientierung, Zweckgebundenheit und Kontextabhängigkeit. Die Bedeutung dieser Anforderungen wird insbesondere im Zusammenhang mit kontinuierlichern Verbesserungsansätzen (wie z.B. dem Quality Improvement Paradigm, Profes, Six Sigma) deutlich, deren Modellunabhängigkeit ein besonderes Maßschneidern von Messansätzen erfordert. Auf methodischer Ebene werden diese Anforderungen durch den Goal-Question-Metric-Ansatz (GQM) abgedeckt. Auf der Ebene von Messwerkzeugen beschränkt sich die Unterstützung bisher oft auf die Erfassung vordefinierter Maße mit geringen Anpassungsmöglichkeiten. Gerade auf Kodeebene ist es vielfach notwendig, unter Ausnutzung syntaktischer und semantischer Spracheigenschaften maßgeschneiderte Metriken zu definieren, werkzeuggestützt zu erfassen und zu analysieren. Im Rahmen des Beitrags werden das methodisch gestützte Werkzeug M-System NT zur Analyse und Messung statischer Eigenschaften von Quelltexten, dessen initiale Erprobung sowie Erfahrungen mit dem System vorgestellt.

Schlüsselbegriffe

Zielorientiertes Messen, Goal/Question/Metric-Ansatz (GQM), Kode-Metriken, Parsen, syntaktische und semantische Analyse von Quelltexten, relationales Kode-Modell

1 Einleitung

Mit der allgegenwärtigen Verbreitung von Software wachsen die Qualitätsanforderungen, die an software-basierte Systeme und deren Entwicklungsprozess gestellt werden. Im Spannungsfeld zwischen Qualität, Time-to-Market und Kosten bekommt die durchgängigen Kontrolle und Bewertung von Produkten und Prozessen einen hohen Stellenwert, insbesondere im Hinblick auf die Reduktion von Risiken. Die quantitative Erfassung spezifischer Attribute von Prozessen, Produkten und Ressourcen mittels Metriken ist ein wichtiger Ansatz, um die Kontrolle über Prozesse und Produkte zu erlangen und auch zu behalten. Darüber hinaus ist Messen eine wesentliche Voraussetzung für die kontinuierlicher Verbesserung von Produkten und Prozessen.

Messobjekt einer Software-Metrik sind Entitäten, die in einem Software-Entwicklungsprozess eine Rolle spielen – Prozesse, Produkte und Ressourcen. Weiterhin kann zwischen internen und externen Merkmalen des Messgegendstands unterschieden werden [10].

Externe Merkmale beziehen sich auf die Wechselwirkungen zwischen einem Untersuchungsgegenstand und seiner Umwelt. Es werden nicht unmittelbar Merkmale des Gegenstandes vermessen, sondern Merkmale der Umgebung, deren Ausprägungen aber durch den Untersuchungsgegenstand bestimmt sind.

Interne Merkmale sind solche, die sich unmittelbar auf den Untersuchungsgegenstand beziehen. Beispiele für interne Merkmale von Software sind Größe und Komplexität.

In diesem Artikel stellen wir ein Werkzeug für die statische Analyse von Quelltexten vor. Messobjekt einer solchen Analyse ist ein Programmtext. Im Mittelpunkt der Überlegungen zur Realisierung des Kode-Analysewerkzeugs stand die Idee, ein Messwerkzeug zu entwickeln, dass ein Modell des zu analysierenden Objekts erstellt, dass wiederum durch einen geeigneten Formalismus einfach zur Definition von Metriken und Erfassung von Messwerten genutzt werden kann. Das vorgestellte Werkzeug M-System NT erzeugt eine analysierbare Zwischendarstellung eines Quelltextes in einer relationalen Datenbank. Messwerte können so durch SQL-Anfragen auf dem Datenmodell eines Quelltextes gewonnen werden.

Der Nutzen dieses Konzepts ergibt sich vor allem daraus, dass Metriken individuell mittels SQL-Anfragen formuliert werden können. Dadurch, dass das Analysemodell in einer relationalen Datenbank abgelegt wird, können auch sehr große Softwaresysteme effizient analysiert und bemessen werden. Hinzu kommt, dass auch Prozess- und Ressourcen-Messdaten in der Datenbank abgelegt und so im Zusammenhang mit den Kode-Metriken analysiert werden können. Ein weiterer Vorteil des Konzepts ist die offene Architektur. Andere Anwendungen (z.B. Reportgeneratoren, statistische Analyse-Werkzeuge), die über eine JDBC/ODBC Schnittstelle verfügen, können direkt auf das Datenbank-Analysemodell des Quelltextes zugreifen.

Im Folgenden schildern wir den Kontext der Entwicklung des neuen Mess-Systems (Kapitel 2). Kapitel 3 gibt einen Überblick über Architektur und Umsetzung des Mess-Systems. In Kapitel 4 wird kurz die methodische Einbettung in den GQM-Ansatz skizziert. Abschließend wird eine Zusammenfassung und ein Ausblick gegeben.

2 Kontext

Auf dem Markt existiert eine Reihe von Werkzeugen, die statische Analysen für verschiedenste Programmiersprachen durchführen. Ausgangpunkt für die Ent-

wicklung von M-System NT war ein bestehendes Analysewerkzeug, das M-System (M für Measurement, NT für New Technology), dass seit 1999 am Fraunhofer IESE entwickelt wird. M-System erlaubt die Berechnung von Größen- und Komplexitätsmaßen sowie einer Vielzahl objektorientierter Metriken für Quelltexte in den Programmiersprachen C, C++ und Java. M-System wurde in Industrieprojekten für die Umsetzung von Messprogrammen eingesetzt. Eine Überarbeitung bzw. Neuentwicklung des Werkzeugs wurde aus verschiedenen Gründen notwendig:

- Erweiterbarkeit: Weder der Satz an Metriken, die das „alte“ M-System berechnet noch die Menge analysierbaren Programmiersprachen war erweiterbar. Das Hinzufügen neuer Metriken erfordert Änderungen am Quelltext des Systems.
- Benutzerschnittstelle: Das „alte“ M-System musste über eine Kommandozeileneingabe bedient werden. Das erschwerte dessen Nutzung für Nicht-Informatiker.
- Plattformabhängigkeit: M-System wurde unter Solaris entwickelt. Eine Portierung auf andere Plattformen war aufgrund der Verwendung kommerzieller (binärer) Bibliotheken schwierig.

Ein zentraler Aspekt für die Konzeption des neuen Werkzeugs M-System NT war, die Extraktion von Informationen aus dem Quelltext, also die mittels Parsen realisierte Quelltextanalyse, von der nachfolgenden Nutzung dieser Information, der Definition und Erfassung von Metriken und anderer Analysen, zu entkoppeln. Das entbindet die späteren Nutzer des Systems von der Notwendigkeit, sich mit den Details der sprachlichen Analyse auseinanderzusetzen, um neue Metriken zu realisieren. Dies wurde dadurch erreicht, dass das Werkzeug eine auswertbare Zwischendarstellung des Quelltextes in einer relationalen Datenbank erstellt. Metriken können so durch SQL-Anfragen über dem in der Datenbank gespeicherten Datenmodell gewonnen werden.

3 Architektur und Umsetzung

Wir beschreiben im Folgenden die Komponenten des Mess-Systems und skizzieren den Verlauf des Analyseprozesses. Danach stellen wir das relationale Modell für Java-Quelltexte vor und zeigen, wie Metriken durch SQL-Anfragen dargestellt werden können. In einem weiteren Abschnitt stellen wir einen Ansatz zur Anfragebasierten Visualisierung von Graphen vor und gehen auf die Erprobung des Systems ein.

3.1 Komponenten

Abbildung 1 zeigt die Komponenten des M-System NT. Das System ist in Java geschrieben. Für das Parsen von Quelltexten setzen wir den ANTLR [1] Parser Generator ein. ANTLR erzeugt aus einer Grammatik (bestehend aus Lexer- und Parser Definition) so genannte Recursive-Descent-Parser. Bei diesem Verfahren werden Non-Terminale in Parser-Regeln auf Funktionsaufrufe abgebildet. Formal gesehen handelt es sich um LL(k)-Parser (Lesen der Eingabe von **L**inks nach Rechts, **L**inke Ableitung zuerst, **k** Token Lookahead). ANTLR ist wie alle zur Entwicklung des M-System NT eingesetzten Komponenten Open-Source und bietet Grammatiken für viele formale Sprachen, unter anderem für Java, C und C++.

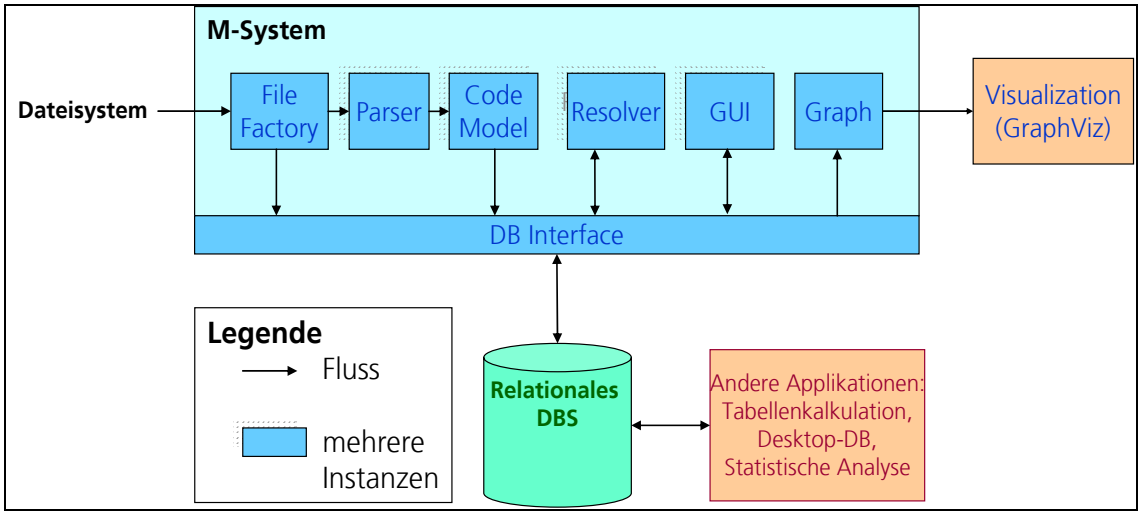

Abbildung 1: M-System NT - Architektur

Der Parser schreibt das Ergebnis der syntaktischen Analyse nicht direkt in die Datenbank. Hierzu bedient er sich eines *Code-Models*, das einen abstrakten Syntaxbaum im Hauptspeicher darstellt. Für Zugriff auf die Datenbank nutzen wir eine objekt-relationale Mapping-Technik. Das Werkzeug DbGen [2] erlaubt es, für eine gegebene Datenbanktabelle Datenzugriffsobjekte zu erzeugen (Java-Klassen), die für das Schreiben und Lesen von Datenbankinhalten verwendet werden können. Die erzeugten Java-Klassen haben Setter- und Getter-Methoden für die in einer Datenbanktabelle definierten Attribute und können über eine API in die Datenbank geschrieben bzw. aus ihr gelesen werden. Das *Code-Model* realisiert einen Syntaxbaum im Hauptspeicher, dessen Knoten aus den automatisch erzeugten Datenzugriffsobjekten bestehen. Mit dieser Technik entfällt die Notwendigkeit für den Entwickler, SQL-Anfragen für das Lesen und Schreiben von Datenbankinhalten zu verwenden; diese Aufgabe wird von der DbGen-API übernommen. Die aktuelle Implementierung verwendet die MySQL relationale Datenbank [4].

Der Parser liefert eine strukturelle Dekomposition eines Quelltextes (Syntaxbaum). Nach diesem Schritt ist die Struktur einer einzelnen Eingabedatei bekannt, nicht aber deren Beziehungen zu anderen Dateien bzw. Software-Komponenten.

Für die Bestimmung dieser Beziehungen ist ein weiterer Schritt notwendig, die Namensauflösung bzw. das *Resolving*. So erfordern die Berechnung von kopplungs- und objektorientierten Kode-Metriken (z.B. für Vererbung) die Auflösung von Namen und somit eine semantische Analyse. M-System NT führt die Namensauflösung nicht während des Parsens, sondern erst nach dem Einlesen aller zu einem System gehörenden Quellen auf der Datenbank durch.

3.2 Analyseprozess

Eingabe für den Analyseprozess ist eine Verzeichnisstruktur, welche die zu analysierenden Dateien enthält. Die Analyse dieser Verzeichnisstruktur und der in ihr enthaltenen Quelltexte erfolgt in drei aufeinander folgenden Schritten:

1. In einem ersten Schritt wird ein Modell des Dateisystems erstellt (Komponente *File Factory*). Für jedes analysierte Verzeichnissystem erfolgt ein Eintrag in der Tabelle *SubSystem* (siehe Abbildung 2). Der Verzeichnisbaum wird in der Tabelle *Package*, die vorgefundenen Dateien in der Tabelle *Artifact* eingetragen. Das in diesem Schritt erstellte Modell erlaubt bereits die Berechnung einfacher Größenmaße (Datei- und Verzeichnisgrößen).
2. In einer zweiten Phase werden Dateien, für die ein Parser vorhanden ist, syntaktisch analysiert. Ergebnis dieses Schritts ist ein *Code-Model*, das über eine Datenbank-Zugriffsschicht (*DB-Interface*) in die Datenbank geschrieben wird. Die Schritte eins und zwei werden für jeden in unterschiedlichen Verzeichnisstrukturen abgelegten Teil des Gesamtsystems wiederholt durchgeführt.
3. Im dritten Schritt, nachdem alle SubSysteme eingelesen wurden, erfolgt eine semantische Analyse, deren Ergebnis die Namensauflösung sowie die Auflösung von Vererbungsbeziehungen ist.

3.3 Ein relationales Modell für Java-Kode

Abbildung 2 zeigt das relationale Datenmodell, dass für die Analyse von Java-Quelltexten eingesetzt wird. Im Bild sind die Tabellen zu erkennen, die Informationen der Elemente eines Java-Programms aufnehmen. Ein Pfeil in diesem Bild kennzeichnet eine 1:n Beziehnung (ein *SubSystem* besteht aus mehreren *Packages*).

Im Bild sind nur die direkten Beziehungen zwischen Tabellen dargestellt. Das Datenmodell wurde bewusst nicht normalisiert, sondern für die effiziente Auswertung und Metrikberechnung optimiert. Die Motivation für die Normalisierung von relationalen Modellen ist die Vermeidung von Redundanz und so genannter Anomalien, die bei der Änderung von Datenbankeinträgen auftreten. Das Analysemodell für Quelltexte ist als Snapshot eines Systems zu verstehen, das Modell wird nach seiner Erzeugung nicht mehr verändert.

Unser Datenschema enthält z.B. redundante Fremdschlüssel, um die Berechnung von Metriken zu erleichtern. Ein Eintrag in der Tabelle *ControlFlow* enthält nicht nur einen Fremdschlüssel auf die Tabelle *Feature*, sondern Fremdschlüssel für alle übergeordneten Elemente, namentlich *Feature, Class, Component, Artifact, Package* und *SubSystem*. Für eine Metrik, die sich auf Kontrollfluss-Konstrukte eines bestimmten SubSystems beziehen („Wie viele If-Anweisungen enthält das System?“) entfallen dadurch die ansonsten notwendigen SQL-Joins über die gesamte Hierarchie.

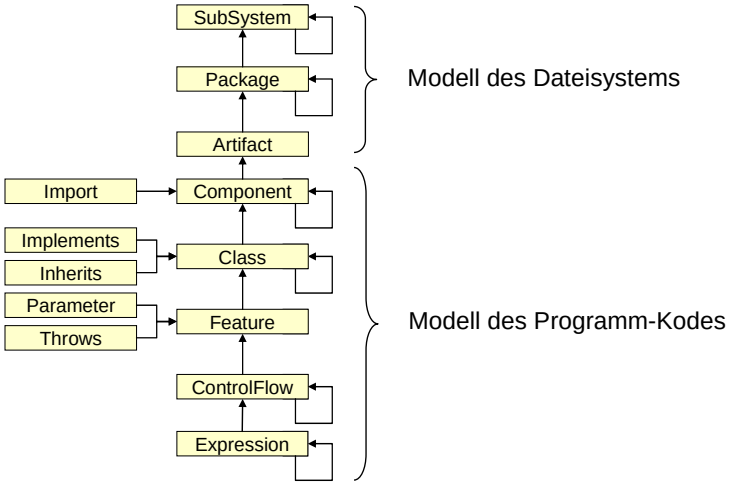

Abbildung 2: Relationales Modell für OO-Programme (Java)

3.4 Definition von Metriken

Wir zeigen an Beispielen, wie in unserem Ansatz Metriken mit SQL-Anfragen dargestellt werden.

Beispiel 1: Datei-basiertes Größenmaß, ohne syntaktische Analyse

```
SELECT SubSystem.Name, Artifact.Type,
count(*) as numFiles, sum(size) / 1024 as KBType
FROM SubSystem JOIN Artifact
ON SubSystem.Id = Artifact.SubSystemId
WHERE SubSystem.Id = ?
GROUP BY SubSystem.Id, Artifact.Type
```

Die Anfrage berechnet ein einfaches Größenmaß für ein System, sie listet Anzahl und Gesamtgröße der in einem System bzw. in einer Verzeichnisstruktur enthaltenen Dateitypen. Es handelt sich hier nicht um ein Größenmaß für den Kode einer Anwendung, da nur Datei-Informationen für die Berechnung herangezogen werden. Die Anfrage liefert jedoch einen groben Überblick bezüglich der Systemgröße, ohne dass Dateien syntaktisch analysiert werden müssen.

Beispiel 2: Komplexitätsmaß (Zyklomatische Komplexität [7]), nach syntaktischer Analyse und Erstellung des relationalen Kode-Modells

```
SELECT Class.Package, Class.Name, Feature.Name, count(*) as CC
FROM SubSystem
```

```
JOIN Class ON SubSystem.Id = Class.SubSystemId
JOIN Feature ON Class.Id = Feature.ClassId
JOIN ControlFlow ON Feature.Id = ControlFlow.FeatureId
WHERE FlowType IN
('METHOD','IF','FOR','WHILE','UNTIL','CASEGROUP','DEFAULT')
AND SubSystem.Id = ? AND Feature.IsOperation
GROUP BY Class.Id, Feature.Id
```

Die Anfrage berechnet die zyklomatische Komplexität für alle Methoden eines SubSystems. Die Berechnung findet nicht auf dem Kontrollflussgraphen statt; stattdessen werden Entscheidungspunkte im Kontrollfluss von Methoden gezählt.

Beispiel 3: Objektorientiertes Maß (Number of Children“ [6]), die Berechnung erfordert Namensauflösung

```
SELECT Class.Package, Class.Name, count(*) AS NOC
FROM Inherits JOIN Class ON Inherits.TypeId = Class.Id
WHERE Class.SubSystemId = ?
GROUP BY Class.Id
```

Die Tabelle *Inherits* enthält nach der Namensauflösung für jede Klasse C eine Liste der Klassen, von denen die Klasse C erbt. Die Metrik Number Of Children ergibt sich in diesem Zusammenhang aus einer Zählung der Inherits-Einträge für eine bestimmte Klasse.

3.5 Anfrage-basierte Visualisierung

Für das Verständnis von Software kann eine Visualisierung von Zusammenhängen (z.B. eine graphische Darstellung des Kontrollflusses oder von Kopplungsbeziehungen) sehr hilfreich sein. Für M-System NT wurde ein Mechanismus entwickelt, Graphen mittels SQL-Anfragen zu definieren. Zu diesem Zweck nutzen wir eine tabellarische Darstellung von Graphen; die den Graph definierende Tabelle wird durch eine SQL-Anfrage erzeugt.

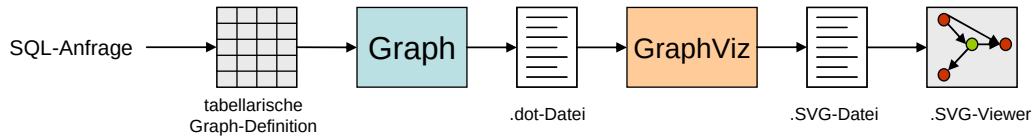

Abbildung 3: Visualisierung von Graphen

Die Tabelle muss ein bestimmtes Namensschema für Spaltennamen aufweisen, damit sie als Graph visualisiert werden kann. Beispielsweise muss mindestens eine Spalte mit Namen *node1* enthalten sein. Ergebnis der Visualisierung einer solchen Tabelle wäre ein Graph, der lediglich Knoten (keine Kanten) enthält; die Knoten werden mit den in der node1-Spalte enthaltenen Werten beschriftet.

Für die Definition von Kanten müssen in der erzeugten Tabelle weitere Spalten *node2*, *node3*, ... , *node<n>* enthalten sein. Eine Zeile repräsentiert dann einen Teilgraphen, der für jede Spalte *node<i>* einen Knoten enthält, der mit dem Inhalt

der entsprechenden Tabellenzelle beschriftet ist. Der Teilgraph enthält dann Kanten von *node1* nach *node2*, *node2* nach *node3* und so fort.

Die *Graph*-Komponente des M-System NT generiert aus eine solchen Tabelle eine Textdatei, die eine Beschreibung eines Graphen in Dot-Syntax enthält. Diese Datei kann von den Layout-Programmen des GraphViz-Frameworks [3] eingelesen und der errechnete Graph in verschiedenen Grafikformaten ausgegeben werden. Die derzeitige Realisierung nutzt das SVG-Format [6] (Scalable Vector Graphic) und den freien ZGRViewer [11] zur Anzeige der durch GraphViz erzeugten SVG-Datei.

Wir zeigen die Funktionsweise an einem Beispiel: Die Verzeichnisstruktur eines Software-Systems (Package-Diagramm) kann mit der folgenden SQL-Anfrage visualisiert werden:

SQL-Anfrage

```
SELECT
Package1.Id as node1,Package1.Name as label1,
Package2.Id as node2,Package2.Name as label2
FROM Package as Package1
JOIN Package as Package2
ON Package1.Id = Package2.ParentId
WHERE Package1.SubSystemId = ?
```

Resultierende Tabelle

node1	label1	node2	label2
1	<default>	2	.settings
1	<default>	3	msystem
3	msystem	4	db
4	db	5	script
4	db	6	tree
…	…	…	…

Die Anfrage wählt Paare Vater/Sohn Verzeichnisse aus der Tabelle *Package* aus. Es entsteht ein Graph, dessen Konten Verzeichnisse repräsentieren und dessen Kanten von Verzeichnissen zu enthaltenen Unterverzeichnissen verlaufen.

Für Knoten und Kanten können weitere Attribute spezifiziert werden. Im obigen Beispiel sind die Spalten *label1* und *label2* im Abfrageergebnis enthalten. Die Attributwerte dieser Spalten liefern eine Zeichenkette für die Beschriftung von Knoten (*label1* ist die Beschriftung für *node1*). Auf diesem Wege können auch Farbe, Form, Schriftart und weitere Attribute für Knoten und Kanten des Graphs festgelegt werden.

3.6 Erprobung

M-System NT wurde an verschiedenen Open-Source-Projekten getestet. Die Erstellung des Analyse-Modells für den Quelltext des Java Development Kit (Version 1.4.2, 274 Verzeichnisse, 4142 Dateien, 6684 Klassen, 54850 Methoden, 22602 Attribute) benötigt z.B. inklusive Namesauflösung ca. 12 Minuten auf einem modernen PC.

Neben zur Definition von Metriken und der Messdatenerfassung wird M-System NT auch zur Programmdokumentation und als Hilfsmittel bei Re-Engineering-Projekten verwendet. Die Anfrage-basierte Visualisierung ermöglicht hierbei eine gezielte, visuell gestützte Analyse der Beziehungen zwischen Programmteilen, da die zu visualisierenden Sachverhalte flexibel definiert werden können.

4 Einbettung des Werkzeugs in die GQM-Methodik

Eine wesentliche Anforderung an Messansätze sind Zielorientierung, Zweckgebundenheit und Kontextorientierung. Zielorientierung erfordert, dass Maße aus Zielen (z.B. Geschäftsziele, Messziele) abgeleitet werden und dementsprechend Daten erfasst werden. Hierdurch wird sichergestellt, dass einerseits keine überflüssigen Daten erhoben werden und sich andererseits die Datenerfassung auf das notwendige Minimum beschränkt. Abhängig vom Zweck der Datenerfassung (Kontrolle, Vorhersage, Bewertung etc.) sind unter Umständen jeweils andere Metriken notwendig. Beispielsweise erfordert die Messung zum Zweck der Vorhersage in der Regel die Erfassung von unabhängigen Variablen. Da die Software-Entwicklung kontextspezifisch ist, müssen Maße auch an den jeweiligen Entwicklungskontext (der durch Attribute wie Anwendungsdomäne, Erfahrung der Entwickler oder Kritikalität charakterisiert werden kann) angepasst werden können. So gibt es beispielsweise in unterschiedlichen Umgebungen völlig verschiedene Anforderungen an Größen wie Dependability, Zuverlässigkeit oder Sicherheit.

Ein Ansatz, der diesen Anforderungen gerecht wird, ist der Goal/Question/Metric-Ansatz [9]. Zu seinen maßgeblichen Prinzipien gehört die Festlegung expliziter Ziele, die Top-down-Definition von Maßen (unter Berücksichtigung existierender Messprogramme), die Bottom-up-Interpretation von Daten im Kontext der Ziele sowie die enge Zusammenarbeit mit den beteiligten Projektmitgliedern beim Aufsetzen und Durchführen von Messprogrammen. Werkzeugseitig erfordert der GQM-Ansatz, dass automatisch und semi-automatisch zu erfassende Metriken effizient und individuell definiert werden können. Hierauf ist M-System NT zugeschnittten. Messwerkzeuge, die einen weitgehend vordefinierten Satz an Metriken und Analysemethoden anbieten, sind in der Regel nicht für zielorientiertes Messen geeignet und führen in der Praxis häufig zu unnötigen Aufwänden bei der Erfassung und Verwaltung überflüssiger Daten.

5 Zusammenfassung und Ausblick

Das vorgestellte Werkzeug M-System NT dient der statischen Analyse von Quelltexten, insbesondere der Berechnung von Kode-Metriken. Hierbei berechnet das Messwerkzeug selbst keine Metriken, sondern es erstellt ein analysierbares Modell von Quelltexten in einer relationalen Datenbank. Messwerte können in diesem Konzept durch die Ausführung von SQL-Anfragen gewonnen werden.

Die aktuellen Arbeiten am M-System NT konzentrieren sich auf die Integration der Programmiersprachen C/C++. Eine Besonderheit dieser Sprachfamilie ist das so genannte Präprocessing, das vor der eigentlichen Übersetzung durch einen Compiler durchgeführt wird. Dadurch ergeben sich unterschiedliche Sichten auf den Quelltext eines Systems:

- Die Sicht, die der Präprozessor und somit der Entwickler auf die Struktur der Quelldateien hat.
- Die Sicht, die ein Compiler auf das System hat, nachdem die Eingabedateien präprozessiert wurden.

Für ein Kode-Mess-System sind beide Sichten von Interesse und folglich müssen beide Sichten in einem gemeinsamen Analysemodell integriert werden. In einem ersten Schritt wurde eine Komponente zur Variantenanalyse in C/C++ Quelltexten erstellt. Das erstellte Analysemodell repräsentiert die Sicht des Präprozessors auf das System und erlaubt z.B. die Berechnung von Metriken bezüglich Variabilität (wie die Anzahl der Variationspunkte #IFDEF, #IFNDEF usw.) als auch die Berechnung von Kopplungsmaßen.

Literaturhinweise

1. ANTLR Website, On-line unter http://www.antlr.org (September 2005)
2. DbGen Website, On.line unter http://dbgen.sourceforge.net/ (September 2005)
3. GraphViz Website, On-line unter http://www.graphviz.org/ (September 2005)
4. MySQL Website, On-line unter http://www.mysql.com/ (September 2005)
5. Java Software Development Kit (JDK), http://java.sun.com (September 2005)
6. SVG-Format, On-line unter http://www.w3.org/TR/SVG/ (September 2005)
7. S.R. Chidamber, C.F. Kemerer, “A Metrics Suite for Object Oriented Design”, IEEE Transactions on Software Engineering, 20 (6), pp. 476-493, 1994.
8. T.J. McCabe, “A Complexity Measure”, Proceedings of the 2nd international conference on Software engineering, 1976 , San Francisco, California, United States
9. V. Basili, G. Caldeira, and H. Rombach, “The Goal Question Metric Approach”, in J. Marciniak (ed.), Encyclopedia of Soft-ware Engineering, Wiley, 1994.
10. N.E. Fenton, S.L.O. Pfleeger, “Software Metrics: A Rigorous and Practical Approach”, published by International Thomson Computer Press, 1997.
11. ZGRViewer, On-line unter http://zvtm.sourceforge.net/ (September 2005)