

Technische Universität München

Fakultät für Informatik

Forschungs- und Lehrereinheit Informatik IV

Implementierung eines Analysewerkzeugs für Quellcode-Komplexität

Systementwicklungsprojekt

Thomas Straßer

Aufgabensteller: Prof. Dr. Dr. h.c. Manfred Broy

Betreuer: Stefan Wagner

Abgabetermin: 13. Mai 2004

Inhaltsverzeichnis

1.	Einleitung	3
2.	Metriken	4
2.1.	Halstead	4
2.2.	McCabe	4
2.3.	OO-Metriken von Chidamber und Kemerer	5
2.3.1.	Weighted Methods per Class (WMC)	5
2.3.2.	Depth of Inheritance Tree (DIT)	5
2.3.3.	Number of children (NOC)	5
2.3.4.	Coupling between Objects (CBO).....	5
2.3.5.	Response for a Class (RFC)	6
2.3.6.	Lack of Cohesion in Methods (LCOM)	6
3.	Eclipse	7
3.1.	Überblick	7
3.2.	PlugIn Entwicklung	7
4.	Metrik PlugIn	10
4.1.	Analyse des Quellcodes	10
4.2.	Anwendung der Metriken.....	11
4.3.	Einbindung in Eclipse	11
4.3.1.	Integration in die PopUp-Menüs	11
4.3.2.	Integration des Views	12
4.3.3.	Integration der Export-Funktion.....	13
5.	Zusammenfassung	14

1. Einleitung

Softwaresysteme werden heute in allen Bereichen des täglichen Lebens eingesetzt und erfüllen teilweise existenzielle Aufgaben. Auf Grund dessen wird auf die Zuverlässigkeit und auf die Sicherheit von Software sehr großer Wert gelegt. Um diese beiden Faktoren sicherzustellen, stehen verschiedenste Möglichkeiten zur Verfügung. Zur Sicherstellung der Zuverlässigkeit bietet sich die Analyse der Quellcode-Komplexität an. Diese wird durch Metriken ermittelt, die Kenngrößen über den Quellcode ermitteln.

Um diese Metriken komfortabel und leicht auf vorhanden Java-Code anwenden zu können, wurde eine Open-Source-PlugIn für die ebenfalls unter einer Open-Source-Lizenz stehende Entwicklungsumgebung Eclipse entwickelt. Das PlugIn umfasst die Metriken Halstead und McCabe sowie die OO-Metriken von Chidamber und Kemerer.

2. Metriken

Metriken sind in der Softwareentwicklung eine verbreitete Methode zur Ermittlungen von Kenngrößen eines Produktes. Es existieren eine ganze Reihe von Metriken, die verschiedenste Werte für bereits existierenden Code ermitteln [RUMPE]. Durch den Einsatz von Metriken lässt sich die Qualität eines fertigen Produkts anhand von greifbaren Werten ermitteln. Die im folgenden vorgestellten Metriken beschäftigen sich in erster Linie mit der Komplexität von Quellcode. Die Komplexität ermöglicht Rückschlüsse auf die Fehleranfälligkeit eines Produktes und ermöglicht eine Aufwandsschätzung für die durchzuführenden Tests. Der Einsatz von Metriken ist jedoch nicht unumstritten, da sie isoliert betrachtet zu Fehleinschätzungen führen können. Ein weiteres Problem stellt der Mensch dar, da es für Programmierer relativ einfach ist, den Code gemäß der verwendeten Metriken zu optimieren.

2.1. Halstead

Basierend auf der Zahl der Operatoren und Operanden lassen sich mit der Metrik von Maurice H. Halstead [HALSTEAD] eine Reihe von Aussagen über ein Softwaremodul treffen. Dabei wird der Code zunächst nach Operatoren (Keywords der Programmiersprache und arithmetische Operatoren, Vergleichsoperatoren, Klammern) und Operanden (Numerische Konstanten, Strings, Variablen) durchsucht und daraus die jeweilige Gesamtzahl ($N1$ und $N2$) sowie die Anzahl der jeweils verschiedenen Elemente ($n1$ und $n2$) ermittelt. Daraus ergibt sich die Größe des Vokabulars ($n = n1 + n2$) und die Länge der Implementierung ($N = N1 + N2$).

Der Umfang des Programms (Volume) lässt sich schließlich nach folgender Formel berechnen $V = N \log_2 n$. Der Anspruch (Level) eines Programms kann durch die Formel $L = (2/n1) * (n2/N2)$ berechnet werden. Die Schwierigkeit ein Programm zu verstehen (Difficulty) ergibt sich dann mit $D = 1/L$. Der Aufwand (Effort) um einen Algorithmus in Programmcode umzusetzen beträgt nach Halstead $E = V / L$. Unter zu Hilfenahme der Stroud Number lässt sich dieses Ergebnis auch in Zeiteinheiten ausdrücken. Diese geht auf den Psychologen John Stroud zurück, der damit die Anzahl der atomaren Entscheidungen eines Menschen pro Sekunde angibt. Außerdem kann der Intelligence Content $I = L * V$ sowie der Language Level $\lambda = L2 * V$ berechnet werden.

2.2. McCabe

Diese Metrik geht auf Thomas J. McCabe zurück und dient zur Ermittlung der Cyclomatic Complexity [MCCABE1], die auf dem Kontrollflussgraphen der Software basiert. Sie misst die Menge der Entscheidungslogik in einem Softwaremodul und wird in zwei Phasen des Entwicklungsprozesses eingesetzt. Zum einen gibt sie darüber Aufschluss, wie viele Testfälle für das Softwaremodul nötig sind. Zum anderen ist sie während des Entwicklungsprozesses von Bedeutung und gibt Aufschluss über Zuverlässigkeit, Testbarkeit und Wartbarkeit.

Die Cyclomatic Complexity $v(G)$ [NIST235] wird aus der Menge der Knoten (n) und aus der Menge der Kanten (e) des Kontrollflussgraphen berechnet. Für $v(G)$ gilt $e - n - 2$. Eine einfachere Zählmethode, die die Ermittlung der Cyclomatic Complexity ohne den Kontrollflussgraphen ermöglicht, trägt den Namen Simplified Complexity Calculation

[NIST235]. Demnach ist die Formel $v(G) = p + 1$, wobei mit p die binären Entscheidungspunkte (if, while, usw.) bezeichnet werden.

2.3. OO-Metriken von Chidamber und Kemerer

Um dem zunehmenden Bedeutung der Objektorientierung Rechnung zu tragen und die Eigenheiten dieses Paradigmas mit Metriken messbar zu machen, stellten Shyam Chidamber und Chris Kemerer eine Metric Suite [CHIDAMBER1] zusammen.

2.3.1. Weighted Methods per Class (WMC)

Bei dieser Metrik wird die Summe der Komplexitäten der Methoden einer Klasse gebildet. Für die Komplexität einer Methode sind verschiedene Berechnungsmodi möglich. Sofern keine spezifiziert wurde, ist die Komplexität einfach mit eins anzusetzen und die Metrik liefert schlichtweg die Anzahl der Methoden einer Klasse.

Durch die WMC Metrik lassen sich Rückschlüsse auf den Entwicklungs- und Wartungsaufwand einer Klasse ziehen. Des weiteren haben Klasse mit vielen Methoden größere Auswirkungen auf mögliche Kinder, da diese sämtliche Methoden erben. Schließlich kann angenommen werden, dass Klassen mit vielen Methoden applikationsspezifisch sind und tendenziell schlechter wiederverwendbar sind

2.3.2. Depth of Inheritance Tree (DIT)

Diese Metrik liefert die Anzahl der Oberklassen einer Klasse und liefert somit ihre Tiefe im Vererbungsbaum. Chidamber und Kemerer [CHIDAMBER1] treffen auf Basis der Metrik Aussagen über Komplexität und Wiederverwendbarkeit. Demnach sind Klassen mit vielen Oberklassen komplexer und schwerer handhabbar, da sie mehrere Methoden erben. Des weiteren sind große Vererbungshierarchien schwer zu entwerfen, da viele Methoden und Klassen beteiligt sind.

2.3.3. Number of children (NOC)

Die Anzahl der Klassen, die von der betreffenden Klasse erben, wird mit dieser Metrik berechnet. Das Ergebnis gibt darüber Aufschluss, wie hoch der Wiederverwendungsgrad der Klasse ist. Sofern die Klasse viele Kinder hat, so ist sie von großer Bedeutung im Projekt und sollte intensiven Tests unterzogen werden. Bei vielen Kindern sollte weiterhin überprüft werden, ob die Kinder fachlich korrekt von der Klasse erben, um fehlerhaftes Anwenden der Vererbung zu verhindern.

2.3.4. Coupling between Objects (CBO)

Die CBO Metrik ermittelt die Anzahl der Klassen zu denen die Klasse in Beziehung steht. Als Beziehungen wird die Nutzung von Methoden oder Instanzvariablen anderer Klassen gewertet. Laut Chidamber und Kemerer [CHIDAMBER1] verhindert eine intensive Kopplung einer Klasse an andere Klassen die Wiederverwendung. Außerdem wird die Komplexität eines Projektes dadurch in die Höhe getrieben, da Änderungen in einer Klasse weitreichende Einflüsse haben kann. Deshalb sollten Beziehungen zwischen Klassen so gering wie möglich

gehalten werden. Eine intensive Kopplung zwischen Klasse hat des weiteren Einfluss negativen Einfluss auf das Testen, da zusätzliche und komplexere Testfälle betrachtet werden müssen.

2.3.5. Response for a Class (RFC)

In dieser Metrik werden die Methoden der Klasse und die von den Methoden aufgerufenen Methoden anderer Klassen gezählt. Je größer die Anzahl, desto größer ist die Komplexität und die Fehleranfälligkeit der Klasse, da eine durch eine Nachricht mehrere Methoden aufgerufen werden. Dies erschwert das Testen und das Debuggen der Klasse.

2.3.6. Lack of Cohesion in Methods (LCOM)

Diese Metrik analysiert die Nutzung der Instanzvariablen durch die Methoden der Klasse. Dabei wird zunächst überprüft, welche Instanzvariablen $\{I_j\}$ von den Methoden benutzt werden. Anschließend werden Schnittmengen (I_i, I_j) gebildet und die Anzahl der Methoden-Paar gezählt bei denen die Menge leer (P) und nicht leer (Q) ist. Sofern P größer als Q ist, wird als LCOM die Differenz $P - Q$ angegeben. Andernfalls liefert die Metrik das Ergebnis 0.

Die Kohäsion innerhalb einer Klasse ist wünschenswert, da sie ein Indiz für gute Kapselung ist. Sofern es einer Klasse an Kohäsion mangelt, so sollte in Betracht gezogen werden, diese in zwei oder mehrere Klassen aufzuteilen. Ein niedrige Kohäsion erhöht die Komplexität und damit auch die Fehleranfälligkeit der Software.

3. Eclipse

3.1. Überblick

Die Open-Source-Entwicklungsumgebung Eclipse bietet eine Plattform zur integrierten Web- und Anwendungsentwicklung [EDD]. Die Kernfunktionalität der Umgebung ist äußerst spartanisch, bietet aber die Möglichkeit durch PlugIns das Funktionspaket zu erweitern.

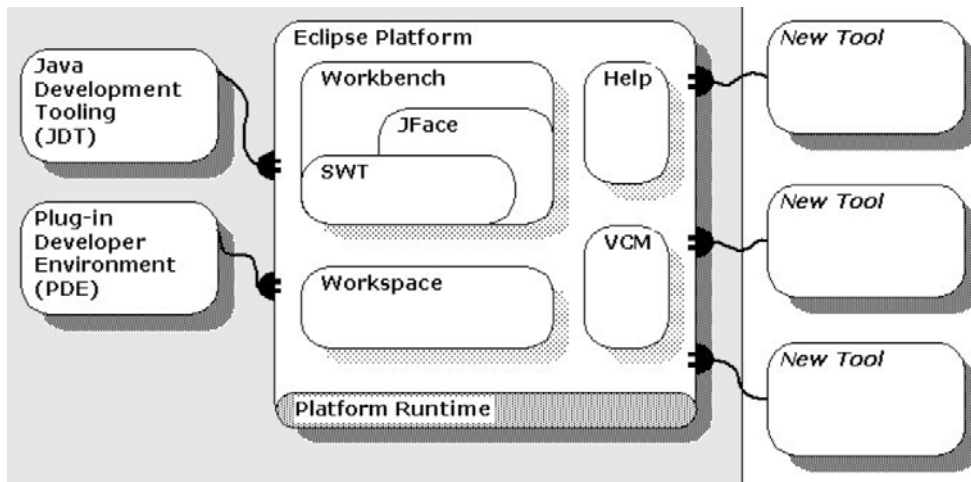


Abbildung 1: Eclipse Architektur, Quelle [EDD]

Im Zentrum von Eclipse steht die „Eclipse Platform“, die zahlreiche Schnittstellen (Extension Points) für PlugIns zur Verfügung stellt. Die „Platform Runtime“ übernimmt das Management der PlugIns und erkennt diese beim Start von Eclipse. Der „Workspace“ übernimmt das Management von Ressourcen (Projekte, Dateien und Ordner) im Dateisystem. Die Anwendungsoberfläche und Bibliotheken zur Erstellung neuer Komponenten werden von der „Workbench“ bereitgestellt. Zusätzlich sind ein Hilfesystem und ein „Version and Configuration Management“ (VCM) integriert.

Die beiden Komponenten JDT und PDE sind bereits als PlugIns in Eclipse integriert. Das „Java Development Tooling“ (JDT) bietet spezielle Features zum Editieren, Compilieren, Debuggen und Ausführen von Java Code. Das „PlugIn Development Environment“ (PDE) bietet sämtliche Funktionalitäten zum Erstellen, Ändern, Debuggen und Deployen von PlugIns.

3.2. PlugIn Entwicklung

Die Entwicklung und Einbindung von PlugIns ist bei der Entwicklungsumgebung Eclipse von zentraler Bedeutung. Bis auf die oben dargelegten Kernfunktionalitäten sind sämtliche Features von Eclipse als PlugIns realisiert und an den Anwendungskern angedockt. Somit können Entwickler mit geringem Aufwand Anpassungen vornehmen und neue Features hinzufügen.

Das Einbinden von PlugIns funktioniert über Extension Points, die vom Anwendungskern aber auch von bereits existierenden PlugIns bereitgestellt werden. Die Benutzerschnittstelle

von Eclipse ist aus verschiedenen Komponenten aufgebaut, die alle durch PlugIns ergänzt und erweitert werden können.

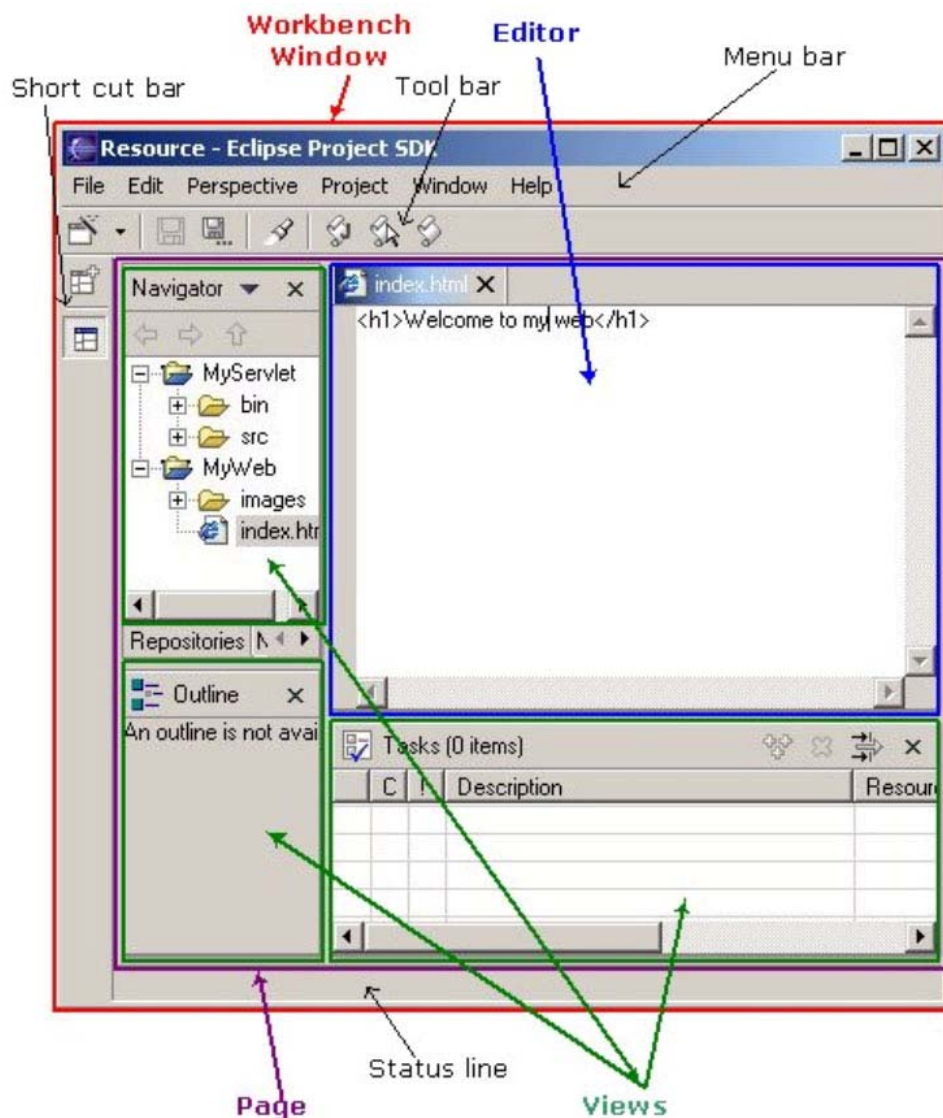


Abbildung 2: Eclipse Workbench, Quelle [EDD]

Die reine Darstellung von Inhalten wie zum Beispiel der Klassenstruktur des Projekts oder Suchergebnissen sind durch Views realisiert. Zum Bearbeiten von Dateien stehen Editoren zur Verfügung. In Abhängigkeit der Funktionalität des PlugIns muss sich der Entwickler für eine dieser Komponenten entscheiden. Das Aufrufen des PlugIns kann auf verschiedene Arten angestoßen werden. Es kann in das Menu, die Toolbar oder als Short Cut eingebaut werden.

Die Erstellung neuer und die Bearbeitung existierender PlugIns wird durch das integrierte „PlugIn Development Environment“ (PDE) unterstützt. Mit dieser Funktionalität kann der Entwickler gezielt ein PlugIn-Projekt anlegen und mit Hilfe der dadurch generierten Klassen und Dateien arbeiten. Jedes PlugIn benötigt zumindest eine Hauptklasse, die von der Klasse `AbstractUIPlugin` abgeleitet sein muss. Des weiteren muss eine Datei namens `plugin.xml` erzeugt werden, die Metainformationen über das PlugIn enthält und die beteiligten Komponenten deklariert.

Das Deployment des fertigen PlugIns erfolgt durch bloßes kopieren der `plugin.xml` und der als jar gepackten Klassen in ein Verzeichnis im Eclipse-PlugIn-Verzeichnis. Der Name dieses

Verzeichnisses muss per Konvention dem Package-Namen des PlugIns entsprechen. Beim Neustart von Eclipse wird das neue PlugIn dann automatisch erkannt und in die Entwicklungsumgebung integriert.

4. Metrik PlugIn

Das Metrik PlugIn für Eclipse wurde nicht als reines PlugIn entwickelt, sondern kann auch als java Anwendung auf der Kommandozeile eingesetzt werden. Auf Grund dessen stützt sich der Programmkern nur auf Klassen des Java Runtime Environment (JRE 1.4.2). Die PlugIn-Fähigkeit gewährleistet nur die bequeme Nutzung des Tools in Eclipse und die Visualisierung der Ergebnisse. Des weiteren wurde eine Export-Funktion integriert, die es ermöglicht die Daten in eine CSV-Datei zu exportieren und in einer anderen Anwendung weiterzuverarbeiten.

4.1. Analyse des Quellcodes

Zu Analyse des Quellcodes wird ein eigens entwickelter Parser eingesetzt, der den Code in einen Baum umwandelt. Die Wurzel des Baues ist leer. Die Knoten sind entweder Klassen, Interfaces, Konstruktoren, Deklarationen oder Methoden.

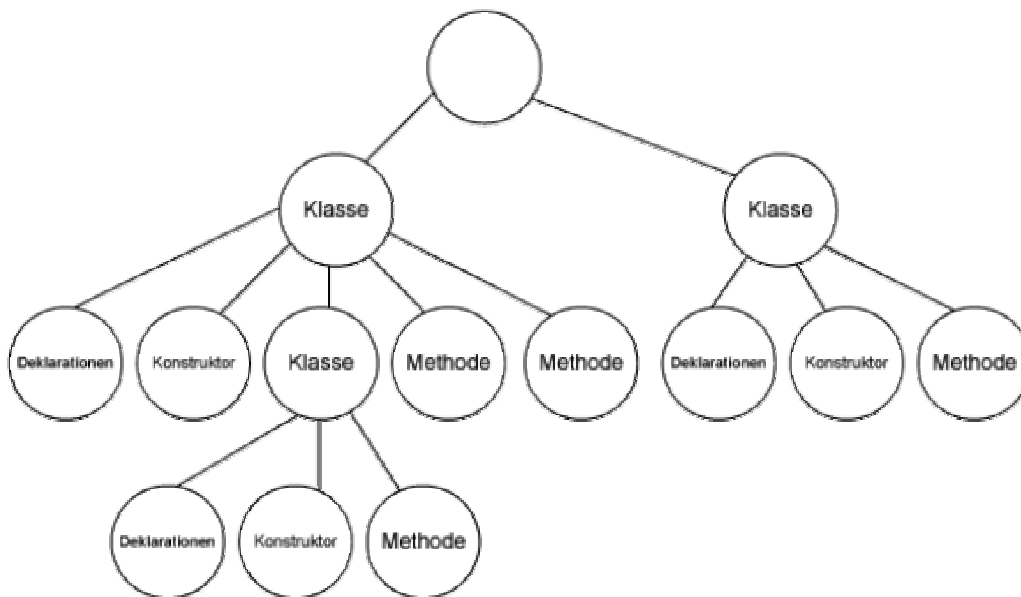


Abbildung 3: SourceCodeTree

Jeder Knoten ist vom Typ `SourceCodeElement`, das den Typ, den Namen und den Quellcode des Knotens enthält. Die Knoten vom Typ `Klasse` und `Interface` enthalten keinerlei Quellcode. Instanzvariablen einer Klasse werden als Typ `Deklaration` gespeichert.

Der Quellcode selbst wird im `SourceCodeElement`, nicht als `String` abgelegt, sondern als Liste. Diese Liste enthält das Resultat des Parsers, der Schlüsselwörter der Programmiersprache, Operatoren, Symbole und Literale maskiert, so dass nur noch Methodennamen, Klassennamen und Variablenamen in der Liste im Klartext vorliegen.

4.2. Anwendung der Metriken

Für jede Metrik ist ein `Analyser` implementiert, der den übergebenen Quellcode entsprechend der Metrik auswertet. Das Anwenden der Metriken auf den geparsen Quellcode funktioniert schließlich durch das Übergeben des `Analysers` an die Wurzel des Quellcode-Baums. Alle Knoten des Baums werden dann rekursiv abgearbeitet und analysiert. Die `Analysers` wissen selbst welche Knoten-Typen für sie relevant sind und werten entsprechend nur die relevanten aus.

Bei den `Analysern` gibt es die beiden Spezialfälle „Depth of Inheritance Tree“ und „Number of Children“. Beide interessieren sich nicht für den Quellcode, sondern für die Tiefe der Vererbungshierarchie der Klasse bzw. die Anzahl der Kinder einer Klasse. Für derartige Analysen sind nur der Klassename, der Packagename und der Classpath relevant. Für die Metrik „Depth of Inheritance Tree“ muss gewährleistet sein, dass die gewählte Klasse im Classpath hängt. Nur dann kann sie mit Hilfe des `ClassLoader` geladen werden, um ihre Superklasse zu erfragen. Bei der Metrik „Number of Children“ werden alle im Classpath befindlichen Klassen im `ClassLoader` geladen und deren Superklasse mit der zu analysierenden Klasse verglichen.

Die Ergebnisse der Metrik werden als Liste von metrik-spezifischen `ResultObjekten` zurückgeliefert, die alle gegen das Interface `IMetricResult` implementiert sind. Dadurch können die Ergebnisse für die Kommandozeilenausgabe sowie für die Darstellung in Eclipse einheitlich abgegriffen werden. Einzig die Metrik „Halstead“ wird in Eclipse als Sonderfall behandelt, da sie im Gegensatz zu den anderen Metriken, die nur jeweils ein Ergebnis liefern, eine ganze Reihe von Werten zurückliefert.

4.3. Einbindung in Eclipse

Die Integration des Metrik PlugIns in Eclipse erfolgt an zwei Schnittstellen. Zum einen wird die Darstellung der Ergebnisse als `View` eingebunden und zum anderen wird der Aufruf der Metriken in die `PopUp-Menus` integriert.

4.3.1. Integration in die PopUp-Menus

Die Integration der Metrik PlugIns in die `PopUp-Menus` von Eclipse erfolgt in der Datei `plugin.xml`. Hier wird als `Extension Point` die Klasse `org.eclipse.ui.popupMenus` verwendet, bei der eine `ObjectContribution` durchgeführt wird. Dadurch kann genau gesteuert werden, für welche Objekte die Metriken in die `PopUp-Menus` integriert werden.

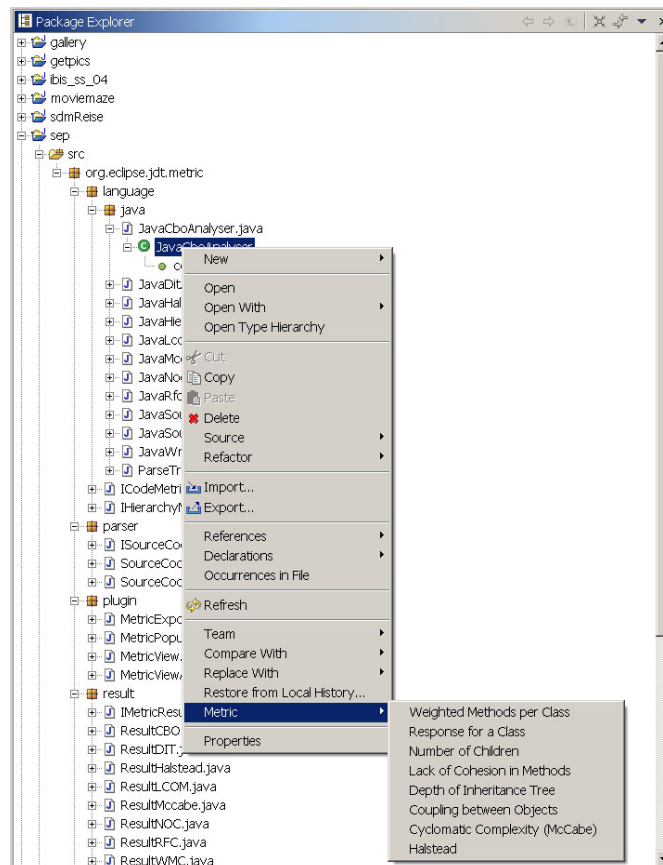


Abbildung 4: Eclipse PopUp-Menu

Für das Metrik PlugIn sind die drei Objekttypen `ICompilationUnit`, `IType` und `IMethod` von Bedeutung. Die `ICompilationUnit` repräsentiert eine Java-Datei, der `IType` steht für eine Klasse und die `IMethod` repräsentiert eine Methode. Durch die Nutzung der `ObjectContribution` taucht das Metrik PlugIn in allen PopUp-Menus auf, die für eines der drei Objekte geöffnet wurden. Somit ist das PlugIn auf einen Schlag in den Package Explorer (siehe Abbildung 3) und den Outliner integriert.

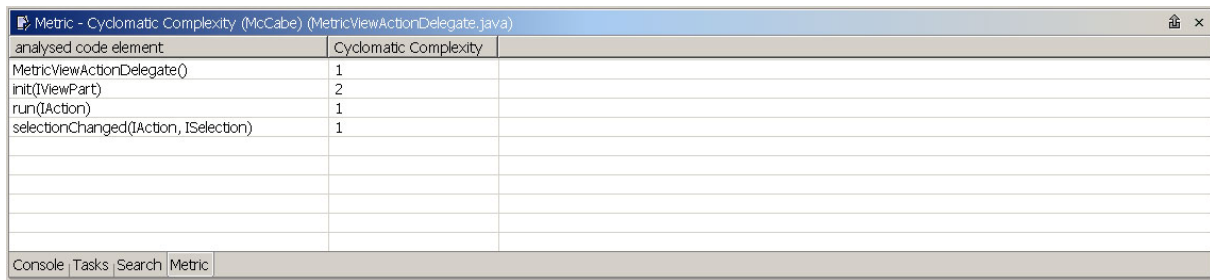
Die Anwendung der Metriken Halstead und McCabe ist für alle drei Objekttypen möglich. Die Metriken der Metric Suite von Chidamber und Kemerer sind für Methoden nicht ausgelegt und deswegen auch nicht in deren PopUp-Menus integriert.

Um das Ausführen des PlugIns auf Grund der Auswahl im PopUp-Menu anzustoßen ist in der Datei `plugin.xml` auf eine Klasse zu verweisen, die von der Klasse `IObjectActionDelegate` erbt. Diese hat Zugriff auf das Objekt, für das das PopUp-Menu geöffnet wurde, und kann damit direkt die Auswertung des Objekts durch die Metrik anstoßen. Anschließend wird die Liste der ResultObjekte an den `MetricView` übergeben, der sie zur Anzeige bringt.

4.3.2. Integration des Views

Um einen neuen View in Eclipse einzubinden, ist dieser wiederum in der Datei `plugin.xml` zu deklarieren. Hierzu wird der Extension Point `org.eclipse.ui.views` benutzt. Gleichzeitig muss eine Klasse implementiert werden, die von `ViewPart` erbt. Im Falle des

Metrik PlugIns besteht der View nur aus einer Tabelle, die für die Darstellung der Ergebnisse benutzt wird.



analysed code element	Cyclomatic Complexity
MetricViewActionDelegate()	1
init(IViewPart)	2
run(IAction)	1
selectionChanged(IAction, ISelection)	1

Abbildung 5: MetricView

4.3.3. Integration der Export-Funktion

Die Exportfunktionalität ist als Symbol in die Toolbar des MetricView integriert. Um dies zu erreichen, muss an dem Extension Point `org.eclipse.ui.viewActions` angedockt werden. Im Gegensatz zur ObjectContribution bei den PopUp-Menus ist nur eine ViewContribution durchzuführen. Damit kann genau gesteuert werden, für welches Anzeigeelement, die Exportfunktion in der Toolbar erscheint. In unserem Fall der MetricView. Des weiteren muss auf eine Klasse verwiesen werden, die einen Klick auf das Symbol weiterverarbeitet. Diese muss von der Klasse `IViewActionDelegate` erben. Dies fällt im Falle der Export-Funktion sehr spärlich aus und ruft direkt den `MetricExportDialog` auf.

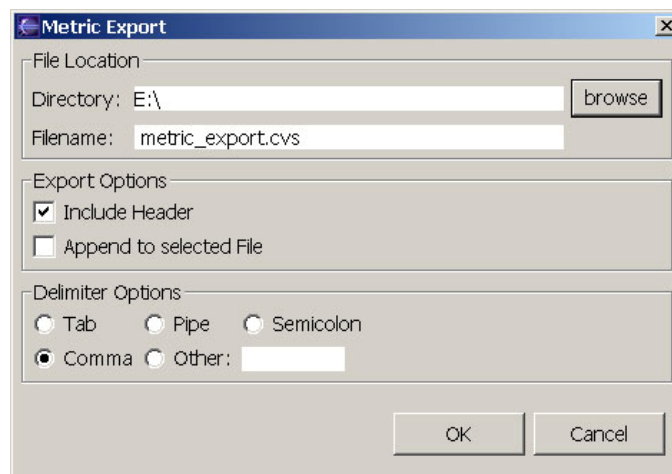


Abbildung 6: Metrik Export

Der `MetricExportDialog` ist von der Klasse `Dialog` abgeleitet und erscheint deshalb als `PopUp` innerhalb von Eclipse. Der Dialog bietet die gängigen Features einer Exportfunktionalität und erlaubt zusätzlich das anhängen der neu zu exportierenden Daten an eine bereits bestehende Export-Datei. Der Dialog arbeitet mit der Klasse `IDialogSettings`, die von Eclipse bereits gestellt wird. Diese Klasse ermöglicht es, die gewählten Optionen des Dialoges zwischenspeichern und bei erneuten Öffnen wieder anzuzeigen. Die Schnittstelle ist extrem einfach zu bedienen, da sich Eclipse um das Schreiben und Lesen der Daten auf die Festplatte selbst kümmert.

5. Zusammenfassung

In dieser Arbeit wurde eine Software entwickelt, die die gängigen Komplexitätsmetriken bündelt und diese bequem auf Java-Quellcode anwendbar macht. Die Software ist sowohl auf der Kommandozeile, als auch als PlugIn in der Entwicklungsumgebung Eclipse einsetzbar. In der aktuellen Version können die Metriken auf einzelne Klassen oder Methoden angewendet werden.

Die Entwicklung eines PlugIns für Eclipse gestaltet sich nach kurzer Einarbeitungsphase relativ unproblematisch und es sind schon nach kurzer Zeit gute Resultate erzielbar. Das PlugIn integriert die Metriken in die PopUp-Menüs, die sich beim Rechtsklick auf Klassen oder Methoden öffnen und macht sie auf diese Weise bequem wählbar. Die Resultate der Metrik werden in tabellarischer Form ausgegeben und sich durch eine Export-Funktion im CSV-Format abgespeichert werden.

Eine interessante Weiterentwicklung der Software, wäre die Metriken auf Pakete sowie ganze Projekte anwendbar zu machen. Auch die Integration weiterer Metriken und weiterer Programmiersprachen in den Funktionsumfang, wäre eine interessante Herausforderung.

Literaturverzeichnis

[CHIDAMBER1]

Chidamber, S. and Kemerer, C., “A Metrics Suite for Object Oriented Design”, M.I.T. Sloan School of Management, December 1993.

[EDD]

Eclipse Project Documentation, Eclipse Project, “Platform Plug-in Developer Guide”, <http://www.eclipse.org/documentation/main.html>

[HALSTEAD]

Halstead, M., “Elements of Software Science”, Elsevier Computer Science Library, 1977

[MCCABE1]

McCabe, T., “A Complexity Measure,” IEEE Transactions on Software Engineering, December 1976.

[NIST235]

NIST Special Publication 500-235, Watson, A. and McCabe, T., “Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric”, Computer Systems Laboratory National Institute of Standards and Technology, September 1996.

[RUMPE]

Rumpe, B., “Vorlesungsscript Softwartechnik”, TU München, Oktober 2002