



FACHBEREICH INFORMATIK  
Abteilung Software Engineering

Proposal einer Diplomarbeit

## **Clone-based Reengineering für Java auf der Eclipse-Plattform**

21. März 2003

**Bearbeitet von:** Simon Giesecke  
Schützenweg 42  
26129 Oldenburg  
simon.giesecke@acm.org  
geboren am 18. April 1980 in Herne

**Betreuer:** Dipl.-Inf. Tammo Freese

**Erstgutachter:** Prof. Dr. Wilhelm Hasselbring

**Zweitgutachter:** Dr. Ralf Reussner

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung, Aufgabenstellung</b>	<b>2</b>
<b>2</b>	<b>Grundlagen und verwandte Arbeiten</b>	<b>3</b>
2.1	Verfahren zur Entdeckung von dupliziertem Code . . . . .	3
2.2	Computergestützte Restrukturierung von Quelltexten . . . . .	6
2.3	Computergestütztes Refactoring . . . . .	6
<b>3</b>	<b>Konzeption</b>	<b>7</b>
3.1	Anwendungsfälle . . . . .	7
3.2	Überlegungen zur Bedienschnittstelle . . . . .	8
3.3	Meilensteine . . . . .	8
3.4	Verwendete Komponenten . . . . .	8
3.5	Auswertung . . . . .	9
<b>4</b>	<b>Organisation</b>	<b>10</b>
4.1	Gutachter . . . . .	10
4.2	Arbeitsumgebung . . . . .	10
4.3	Zeitplan . . . . .	10
4.4	Gliederung . . . . .	10
<b>5</b>	<b>Produkte</b>	<b>12</b>
	<b>Literatur</b>	<b>12</b>

## 1 Einleitung, Aufgabenstellung

Das zentrale Element des Eclipse Projekts [OTI03] ist eine Plattform, die die Entwicklung integrierter Entwicklungsumgebungen (IDEs) für beliebige Anwendungen ermöglicht. Die Aufgabe der Plattform besteht darin, Werkzeuge unterschiedlicher Anbieter nahtlos zu integrieren (Plug-In-Konzept).

Mit dem Java Development Tooling (JDT) wird in Eclipse beispielhaft eine leistungsfähige Entwicklungsumgebung für Softwareprojekte in Java entwickelt, die die Erweiterungsmöglichkeiten der Plattform demonstriert und selbst wieder erweiterbar ist.

Refactoring-Operationen sind Umstrukturierungen von objektorientierten Programmen, die das extern beobachtbare Verhalten nicht verändern [DDNJ02, Fow99]. Die manuelle Durchführung von Refactoring-Operationen ist für die Entwicklerin ermüdend und führt leicht zu Fehlern. Mit dem JDT ist es bereits möglich, oft verwendete Refactoring-Operationen automatisiert anzuwenden.

Ziel des Refactoring ist es, die interne Struktur eines Programmes zu verbessern, um die Verständlichkeit des Programmes zu erhöhen oder Änderungen zu erleichtern. Aus diesem Grund ist Refactoring eine wichtige Teilaktivität beim Reengineering objektorientierter Programme. Bei agilen Prozessen wie Extreme Programming [Bec00] ist ein kontinuierliches Reengineering (und insbesondere Refactoring) Teil des Entwicklungsprozesses selbst.

Um Refactoring-Operationen durchführen zu können, müssen Refactoring-Ziele – also Codefragmente, auf die ein Refactoring angewendet werden kann – identifiziert worden sein. Um

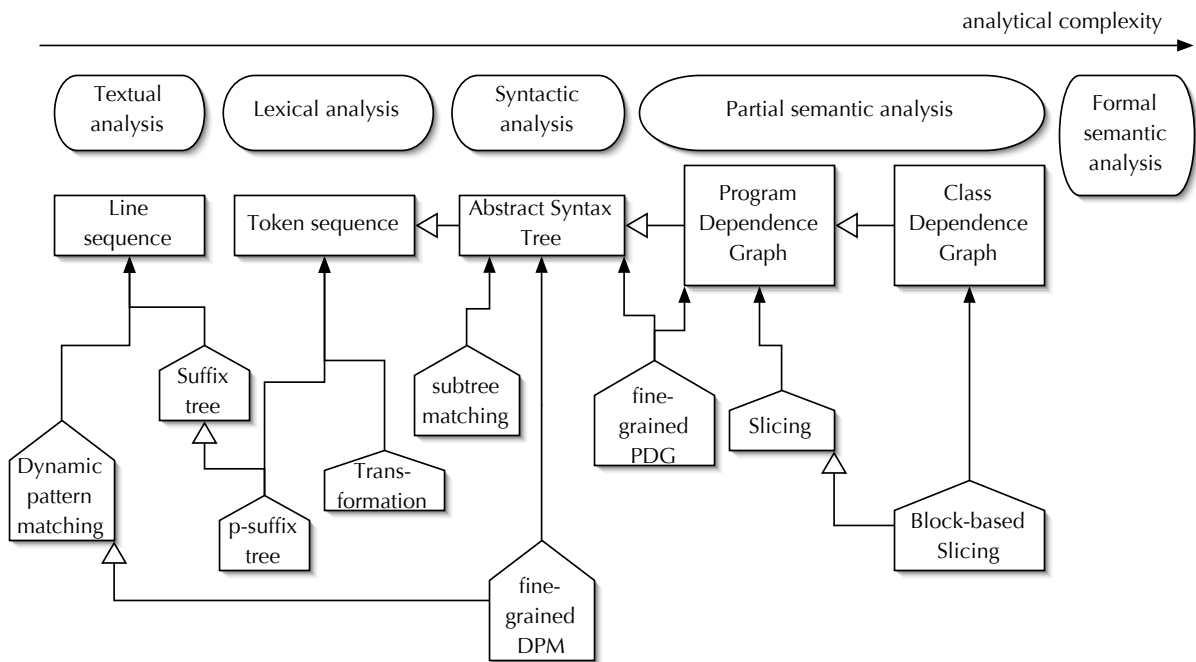


Abbildung 1: Duplikationserkennungsalgorithmen

einen Anhaltspunkt zu haben, wann und wo Refactoring-Operationen angewendet werden sollten, benennen Beck und Fowler [BF00] verschiedene „Bad Smells in Code“. Der wichtigste „Bad Smell“ ist duplizierter Code.

Seit Beginn der 1990er Jahre wurden verschiedene Ansätze zur Erkennung von duplizierten Codefragmenten, sogenannten Klonen (clones), entwickelt.

**Ziele** Im Rahmen dieser Diplomarbeit soll ein Werkzeug konzipiert und prototypisch implementiert werden, das mittels der Plug-In-Schnittstellen der Eclipse-Plattform in das Eclipse JDT integriert wird. Das Werkzeug soll die Entwicklerin beim Clone-based Reengineering [BMD<sup>+</sup>99a] unterstützen, indem es

1. bekannte Verfahren zur Erkennung von Klonen nutzt,
2. Vorschläge für die Anwendung von Refactoring-Operationen unterbreitet,
3. die von der Entwicklerin ausgewählten Refactoring-Operationen automatisiert durchführt.

## 2 Grundlagen und verwandte Arbeiten

### 2.1 Verfahren zur Entdeckung von dupliziertem Code

In Abbildung 1 sind grundlegende Verfahren zur Programmanalyse, Datenstrukturen zur Darstellung der Analyseergebnisse und Algorithmen zur Duplikationserkennung, sowie deren Beziehungen dargestellt.

Die Verfahren zur Programmanalyse sind im oberen Bereich nach ihrem Aufwand von links nach rechts angeordnet, beginnend mit der rein textuellen Analyse, über die lexikalische und syntaktische Analyse zu Verfahren, die eine teilweise semantische Analyse zum Gegenstand haben.

Im Unterschied zur formalen semantischen Analyse beziehen sich hier eingeordnete Datenstrukturen und Algorithmen auf Programmiersprachen, für die keine formale Semantik definiert ist. Dies ist bei den meisten in der Praxis eingesetzten Programmiersprachen, insbesondere Java, der Fall.

Unter den Analyseverfahren zeigt Abbildung 1 in den rechteckigen Kästen Datenstrukturen, die zur Repräsentation des analysierten Programmcodes eingesetzt werden. Im unteren Bereich sind die Duplikationserkennungsalgorithmen in den fünfeckigen Kästen dargestellt, deren Beziehungen untereinander und mit den Datenstrukturen durch Pfeile gekennzeichnet sind.

Die Algorithmen, die lediglich eine textuelle Analyse durchführen und somit auf der Zeilenfolge der Quelldateien operieren, sind vollständig unabhängig von der verwendeten Programmiersprache. Im wesentlichen gilt dies auch für die Algorithmen, die eine lexikalische Analyse durchführen und auf der Tokensequenz operieren. Auf dieser Ebene sind die meisten Programmiersprachen einander so ähnlich, dass zwar eine geringfügige Anpassung des Lexers selbst notwendig ist, die resultierende Tokensequenz jedoch ohne Berücksichtigung der zu Grunde liegenden Programmiersprache analysiert werden kann.

Bei den Algorithmen, denen eine syntaktische oder semantische Analyse zu Grunde liegt, ist die Sprachunabhängigkeit in dieser Form nicht mehr gegeben. Natürlich lassen sich Verfahren für eine Programmiersprache auch auf ähnliche Sprachen übertragen, allerdings ist hier die Programmanalyse enger mit der Duplikationserkennung verknüpft.

Der Vorteil sprachunabhängiger Verfahren liegt natürlich in ihrer Universalität, die von [DRD99, DRG99, RD98] betont wird. Für die Duplikationserkennung in alten Legacy-Systemen ist es in der Tat schwierig, sprachspezifische Ansätze, die Parser benötigen, zu verfolgen, da Sprachen wie COBOL und C in mannigfaltigen Dialekten eingesetzt worden sind und für jeden Dialekt ein angepasster Parser notwendig ist.

Die spezifischen Eigenschaften jeder Programmiersprache, die typischerweise auftretende Duplikationsarten einerseits und mögliche Lösungen andererseits beeinflussen, können bei sprachunabhängigen Ansätzen nicht oder nur mit erhöhtem Aufwand berücksichtigt werden. Beispielsweise schlagen Ducasse et al. [DRG99] die Verwendung des sprachunabhängigen objektorientierten Metamodells FAMIX [DTS99] vor, das die Klassen eines Programmes mit ihren Vererbungsbeziehungen, Methoden und Attributen repräsentiert. Um auch C++-Programme darstellen zu können, unterstützt das Metamodell das Konzept der Mehrfachvererbung. Für Java-Programme stellt dies einen erhöhten Aufwand dar, da hier nur eine eingeschränkte Form der Mehrfachvererbung mittels Interfaces unterstützt wird.

Mit der analytischen Komplexität der Verfahren steigt auch die Berechnungskomplexität der Programmdarstellungen. Eine Analyse des Berechnungsaufwandes einiger Duplikationserkennungsalgorithmen findet sich in [Bel02].

Im folgenden werden nun einige konkrete sprachunabhängige und sprachspezifische Algorithmen zur Duplikationserkennung und die Werkzeuge, in denen diese umgesetzt wurden, vorgestellt.

### 2.1.1 Sprachunabhängige Verfahren

**Parametrisierte Suffixbäume (p-suffix tree)** Das älteste Verfahren dieser Kategorie wird in [Bak95, Bak92] beschrieben. Es ist in Baker's Werkzeug dup implementiert worden und arbeitet zeilenbasiert. Es führt jedoch auch eine teilweise lexikalische Analyse durch, um Bezeichner zu identifizieren, die beim Vergleich nicht berücksichtigt werden. Die so parametrisierten Codezeilen werden mit Hilfe einer angepassten Form von Suffixbäumen verglichen.

**Tokenstrom-Transformation** CCFinder [KKI02] verwendet ein Verfahren, bei dem zunächst eine lexikalische Analyse des Eingabeprogramms durchgeführt wird. Auf den dabei generierten Tokenstrom werden sprachspezifische Transformationsregeln angewendet, mit dem Ziel, syntaktische Alternativen zu vereinheitlichen. Zum Beispiel werden Schleifenkörper u.ä., die aus einzelnen Anweisungen bestehen, stets in einen Block transformiert. Anschließend werden die Duplikationen im Tokenstrom mittels Suffixbäumen gefunden.

**Dynamischer Mustervergleich (Dynamic Pattern Matching)** Duploc [DRD99,DRG99,RD98] ist ein Werkzeug, das einen zeilenweisen Vergleich der Quelldateien durchführt. Es ist in Smalltalk implementiert. Duploc arbeitet weitgehend sprachunabhängig, die Eingabe wird lediglich bezüglich Whitespaces und Kommentaren gefiltert, was die Kenntnis der Programmiersprache erfordert. Die Ausgabe kann als textueller Bericht über die mittels DPM gefundenen Muster oder als Scatter Plot [Hel95] erfolgen.

### 2.1.2 Sprachspezifische Verfahren

**Teilbaum-Vergleich (Subtree matching)** Das kommerzielle Werkzeug CloneDR [BYM<sup>+</sup>98] von Semantic Designs kann Programme in verschiedenen Programmiersprachen verarbeiten, arbeitet allerdings auf dem abstrakten Syntaxbaum und benötigt somit für jede unterstützte Sprache einen Parser. Auf alle Teilbäume des Syntaxbaums wird eine Hashfunktion angewendet, die die Eigenschaft hat, Unterschiede an den Blättern zu vernachlässigen. Dadurch werden insbesondere unterschiedliche Bezeichner beim Vergleich ignoriert.

**Feingranularer dynamischer Mustervergleich (fine-grained DPM)** Das Ziel des Werkzeugs SMC (Similar Methods Classifier) [BMD<sup>+</sup>99a,BMD<sup>+</sup>00,BMD<sup>+</sup>99b] ist es, duplizierten Code nicht nur zu erkennen, sondern auch Unterschiede zwischen zwei Klonen zu klassifizieren. Als Vergleichseinheit werden hier Methoden vorgeschlagen. Die Klassifikation erfolgt anhand des Typs der geänderten Knoten des Syntaxbaumes. Der Vergleichsalgorithmus hat zwei Phasen: Zunächst werden in einem relativ ungenauen, aber schnellen, auf Metriken basierten Verfahren Klon-Kandidaten ermittelt. Auf die Tokenfolgen dieser Kandidaten wird anschließend ein Dynamic Pattern Matching Algorithmus angewendet. Die Ergebnisse der zweiten Phase werden für die Klassifizierung benutzt.

**Feingranularer Programmabhängigkeitsgraph (fine-grained PDG)** Ein Programmabhängigkeitsgraph stellt eine Abstraktion des zu Grunde liegenden abstrakten Syntaxbaumes dar, der die Zuweisungen und Kontrollanweisungen eines (prozeduralen) Programms mit ihren Daten- und Kontrollflussabhängigkeiten darstellt. Duplix [Kri01] verwendet eine feingranulare Variante des Programmabhängigkeitsgraphen zur Darstellung des Programms. In dieser werden mehr Informationen aus dem AST übernommen und insbesondere die Berechnung von Ausdrücken in einzelne Schritte zerlegt. Im Programmabhängigkeitsgraph sucht Duplix nach ähnlichen, nicht zwingend isomorphen, Teilgraphen.

**Slicing** Ein ähnlicher Ansatz, der auf traditionellen PDGs aufbaut, wurde an der University of Wisconsin-Madison entwickelt [KH01a,KH01b]. Hier werden im PDG ähnliche, baumförmige Teilgraphen gesucht, um Klone zu identifizieren.

Die beiden auf Programmabhängigkeitsgraphen aufbauenden Verfahren können nicht ohne weiteres auf objektorientierte Sprachen übertragen werden, da das Konzept des PDG auf Grund der Polymorphie hier wesentlich erweitert werden muss, siehe hierzu [CX01, LH96, Mar01].

## 2.2 Computergestützte Restrukturierung von Quelltexten

Bei der computergestützten Restrukturierung von Quelltexten in einem objektorientierten Kontext handelt es sich vorwiegend um die automatisierte Durchführung von Refactoring-Operationen. Die erste einsatzfähige Implementierung eines solchen Systems war der Refactoring Browser [RBJ97] für Smalltalk.

Für die Programmiersprache Java gibt es mittlerweile ebenfalls verschiedene Werkzeuge, die ein automatisiertes Refactoring unterstützen. Zum Teil sind diese in integrierten Entwicklungsumgebungen enthalten oder als Plug-Ins von Drittanbietern verfügbar. Eine Übersicht findet sich in [DW02, IV.4].

**Objektorientierte Metamodelle** Als Grundlage für Werkzeuge, die Restrukturierung von Quelltexten implementieren, können objektorientierte Metamodelle dienen.

Hier gibt es einerseits einen sprachunabhängigen Ansatz, das FAMIX-Metamodell [TDDN00, DTS99].

Ein anderer Ansatz ist Recoder [LH]. Hierbei handelt es sich um ein Framework zur Metaprogrammierung und Transformation von Java-Programmen. Interessant ist, dass eine inkrementelle Aktualisierung des Metamodells möglich ist. Dies dürfte die Einbettung in eine Umgebung, in der externe Änderungen am Programm möglich sind, erheblich erleichtern.

Inject/J [GKS<sup>+</sup>] setzt auf Recoder auf und stellt eine Skriptsprache für die automatisierte Transformation von Java Programmen bereit. Inject/J soll in Kürze auch als Plug-In für Eclipse zur Verfügung stehen.

**Automatisierte Extraktion von Codeblöcken** Komondoor und Horwitz [KH00] beschäftigen sich mit der automatisierten Extraktion von Prozeduren mittels Slicing.

Maruyama [Mar01] erweitert dieses Konzept in einem objektorientierten Kontext und erläutert somit eine Automatisierung der Refactoring-Operation „Method Extraction“. Er beschreibt auch ein Werkzeug, das Vorschläge für mögliche Extraktionen aus einer Methode macht und diese durchführt.

## 2.3 Computergestütztes Refactoring

Computergestütztes Refactoring geht insoweit über reine Restrukturierung von Quelltexten hinaus, als es dem Benutzer automatisch Codefragmente vorschlägt, die sich für ein Refactoring anbieten.

Es gibt bereits Werkzeuge, die verschiedene „Bad Smells“, darunter auch duplizierten Code, entdecken und teilweise der Benutzerin auch bei der Behebung helfen:

J/Art [DW02]	Refactoring Plug-In für Netbeans
JCosmo [vEM]	Code Smell Browser
RevJava [Flo]	Design Critic Framework
Package Structure Analysis Tool [Com]	Add-on für OptimalJ

Weiterhin gibt es einige kommerzielle Werkzeuge, die ähnliche Aufgaben übernehmen. Nicht berücksichtigt sind hier Werkzeuge, die im wesentlichen nicht strukturelevante Fehler erkennen, wie z.B. JLint.

Ideen für Werkzeuge zur Behandlung von dupliziertem Code finden sich in [DRG99, DRT98].

Bisher gibt es noch keine Werkzeuge die die Prozessaktivität des Clone-based Reengineering sinnvoll in den Entwicklungsprozess integrieren. Wie bereits Roberts [Rob99, 6.2.2] feststellte, ist die Integration von Restrukturierungswerkzeugen essenziell für ihren Anwendungserfolg: „Simply having [the refactorings] at your fingertips made all the difference.“ Nicht oder nur lose angekoppelte Werkzeuge werden nicht (freiwillig) benutzt. Dies ist bereits bei der Konzeption zu berücksichtigen, um nicht nur eine technische Integration in die Entwicklungsumgebung zu erhalten, sondern tatsächlich eine nahtlose Integration in den Entwicklungsprozess.

## 3 Konzeption

### 3.1 Anwendungsfälle

Unabhängig davon, ob das Clone-based Reengineering in den Entwicklungsprozess oder in einen gesonderten Reengineering-Prozess eingebettet ist, sind zwei Szenarien des Clone-based Reengineering vorstellbar, die sich hinsichtlich des Subjekts, das die Codeduplikation initial feststellt, unterscheiden.

Zunächst handelt es sich um das Vorgehen der meisten beschriebenen Werkzeuge, die den kompletten Quelltext eines Programmes analysieren und dann eine Liste (oder eine andere Darstellungsform) aller gefundenen Klone bzw. Klonkandidaten ausgeben. Dieses Werkzeugorientierte Clone-based Reengineering wird systemweit durchgeführt.

Dies ist jedoch für eine interaktive, in die Entwicklungsumgebung eingebettete Sitzung ein eher ungeeignetes Vorgehen. Ich stelle im folgenden einen Entwickler-orientierten Ansatz für das Clone-based Reengineering vor.

Diese Herangehensweise orientiert sich direkter an der Idee der „Bad Smells“ [BF00]. Wenn ein Programmkonstrukt von der Entwicklerin als mögliche Duplikation empfunden wird – möglicherweise, weil sie ein ähnliches Programmkonstrukt bereits an einer anderen Codestelle gesehen hat – kann dies als Indiz dafür gewertet werden, dass dieses oder ein ähnliches Konstrukt an weiteren Stellen im Quelltext verwendet wurde. Aus diesem Grund wird von der Entwicklerin eine Clone-based Reengineering Sitzung initiiert.

Diese Herangehensweise verläßt sich mehr auf die Erfahrung der Programmiererin, mögliche Duplikationen zu erkennen. Andererseits wird so ggf. der Reengineering-Aufwand verringert, da nicht Duplikationen betrachtet werden, die die Entwicklung des Programmes gar nicht behindern.

Es muss natürlich berücksichtigt werden, dass Duplikationen im Quelltext existieren können, die subjektiv nicht als störend empfunden werden, sich jedoch objektiv negativ auf die Systemstruktur auswirken und diese negative Wirkung bei einer Weiterentwicklung des Systems ohne Behebung der Duplikation verstärkt wird. Wegen des hohen Aufwands eines systemweiten Clone-based Reengineerings ist es jedoch zweckmäßig, beide Ansätze zu verfolgen und häufiger interaktive Clone-based Reengineering Sitzungen durchzuführen und seltener ein systemweites Clone-based Reengineering.

## 3.2 Überlegungen zur Bedienschnittstelle

**Prozessübersicht** In der Prozessübersicht wird eine Gesamtsicht auf die Prozessaktivität des Clone-based Reengineering angeboten. Hierzu sollen aggregierte Daten und Berichte über identifizierte und entfernte Klone gehören. Außerdem ist in dieser Sicht die Verwaltung der Duplikationsdaten eingeordnet:

- Die Parameter für die Klonerkennung sowie Vorgaben für die Klonentfernung können eingestellt werden.
- Die Duplikationsdaten können für ein Projekt neu generiert werden.
- Einstellungen für die inkrementelle Aktualisierung der Duplikationsdaten können vorgenommen werden.

**Restrukturierungsansicht** Nach der Auswahl einer Methode stellt die Restrukturierungsansicht mögliche Klone dieser Methode dar. Die Benutzerin kann nun geeignete Klone auswählen. Anschließend schlägt das Werkzeug Möglichkeiten für die Extraktion gemeinsamen Codes vor. Wenn die Benutzerin eine Möglichkeit auswählt, erfolgt eine Vorschau in einer zweispaltigen Ansicht, in der die Situation vor und nach der Restrukturierung gegenüber gestellt werden.

**Eclipse Tasks** Codefragmente, die als wahrscheinliche Klone identifiziert wurden, werden zu der Menge der Eclipse Tasks hinzugefügt. Dadurch werden die Klone direkt in den Editorfenstern markiert. Dies kann einerseits für ein interaktives Clone-based Reengineering hilfreich sein, da der Entwicklerin signalisiert wird, an welchen Stellen dem Werkzeug mögliche Klone bekannt sind und somit der Start einer Sitzung möglich ist.

## 3.3 Meilensteine

In Abbildung 2 sind die geplanten Meilensteine für die Umsetzung des Lösungskonzeptes, sowie die dazwischen bestehenden Abhängigkeiten, dargestellt.

## 3.4 Verwendete Komponenten

Wesentliche Grundlage für den Entwurf des Werkzeugs wird die Architektur der Eclipse Platform und des Eclipse JDT sein.

Zur Duplikationserkennung wird ein zweistufiges Verfahren eingesetzt: Zunächst werden mittels eines schnellen, Token-basierenden Verfahrens Kandidaten ermittelt.

Anschließend werden genauere Klonkandidaten mit einem Subtree-Matching auf dem AST ermittelt. Falls die Möglichkeiten, die Eclipse selbst hierzu bietet, nicht ausreichen, soll das Recoder Framework eingesetzt werden.

Ein zweistufiges Vorgehen wird in [BMD<sup>+</sup>99a] vorgeschlagen und ist in dem Werkzeug SMC (Similar Methods Classifier) eingesetzt. SMC setzt allerdings in der ersten Phase Metriken auf Methodenebene ein, was dazu führt, dass nur ähnliche Methoden erkannt werden können, nicht aber Methoden, die nur teilweise ähnliche oder gleiche Codefragmente enthalten.

Zur Auswahl der Codefragmente für die Method Extraction wird das von Maruyama [Mar01] vorgeschlagene Verfahren verwendet. Grundlage für die in dem Verfahren benötigten Program Dependence Graphs und Class Dependence Graphs ist der AST. Zur Implementierung soll ebenfalls ggf. das Recoder Framework verwendet werden.

Zum Testen wird das JUnit-Framework eingesetzt.

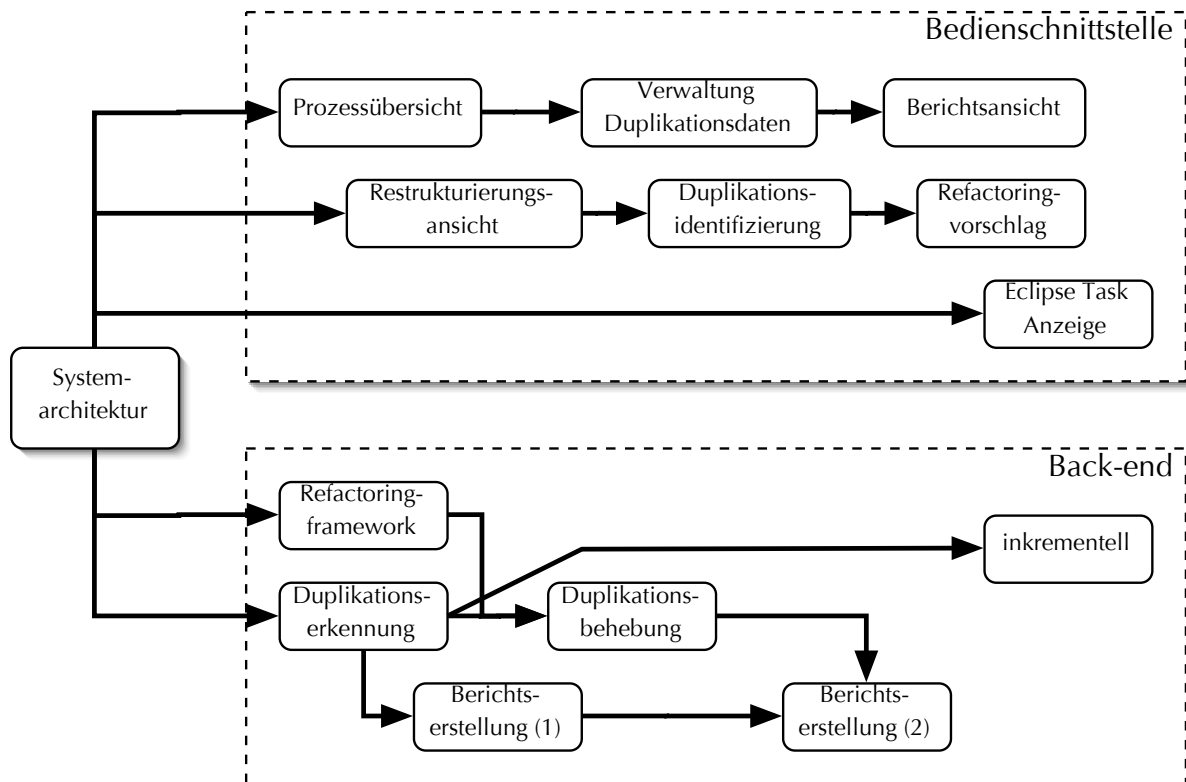


Abbildung 2: Meilensteine

### 3.5 Auswertung

Die Diplomarbeit ist schwerpunktmäßig auf die Konzeption und Entwicklung einer interaktiven Erkennung und Entfernung von Klonen ausgerichtet ist. Daher ist im Rahmen der Diplomarbeit die Auswertung eines vollständigen Clone-based Reengineering eines größeren Legacy-Systems zeitlich nicht möglich. Zur Bewertung der Möglichkeiten, die das Clone-based Reengineering grundsätzlich bietet, wird auf die Literatur verwiesen. Eine Auswertung der Qualität der verschiedenen Algorithmen zur Duplikationserkennung ist ebenfalls nicht Gegenstand dieser Arbeit (siehe hierzu [Bel02]).

Es wird jedoch beispielhaft betrachtet werden können, ob bzw. inwieweit die Einbettung eines Werkzeugs für das Clone-based Reengineering in eine integrierte Entwicklungsumgebung wie Eclipse die Prozessaktivität des Clone-based Reengineering vereinfacht und möglicherweise neue Möglichkeiten eröffnet.

Es ist geplant, zu diesem Zweck die Anwendbarkeit auf folgende Anwendungen bzw. Komponenten zu untersuchen:

- Sun Java 2 SDK 1.4.1 Standard Edition javax.swing.\* (<http://java.sun.com/j2se/1.4.1/download.html>)
- Apache Jakarta Tomcat 4.1.18 (<http://www.apache.de/dist/jakarta/tomcat-4/source/>)
- Projektgruppe ORISS
- Projektgruppe Maja

Die Java Runtime Library wurde bereits in anderen Untersuchungen [Bel02, KN01] zur Duplikationsanalyse verwendet.

Bei ORISS (Oldenburger RIdESharing System) handelt es sich um eine Entwicklung einer studentischen Projektgruppe an der Universität Oldenburg. Nach meiner Erfahrung und einer kurzen Stichprobe mittels Duploc sind dort vielfältige Duplikationen vorhanden.

## 4 Organisation

### 4.1 Gutachter

**Betreuer** Tammo Freese

**Erstgutachter** Prof. Dr. Hasselbring

**Zweitgutachter** Dr. Ralf Reussner

### 4.2 Arbeitsumgebung

**Softwareentwicklung** Die Systemarchitektur des Werkzeugs soll in der UML modelliert werden; als Werkzeuge kommen hierfür Poseidon for UML Community Edition der Firma Gentleware AG, Together ControlCenter von Borland oder EclipseUML von Omondo in Betracht.

Als Entwicklungsumgebung wird Eclipse verwendet, das für die Entwicklung von Plug-Ins für Eclipse selbst Unterstützung bietet. Es wird die momentan aktuelle Version 2.1RC3a – bzw. gegebenenfalls eine aktuellere – eingesetzt.

Maßgebliche Testplattform ist MacOS X in Verbindung mit dem Java 2 SDK 1.4.1 von Apple. Eclipse ist in Java implementiert und verwendet die teilweise plattformspezifische GUI-Bibliothek SWT (Standard Widget Toolkit). Hier wird davon ausgegangen, dass das entwickelte Werkzeug auf allen anderen von Eclipse unterstützten Plattformen ebenfalls einsetzbar sein wird.

**Ausarbeitung** Die Ausarbeitung wird in  $\LaTeX$  gesetzt. Für die Erstellung von PDF-Vektorgrafiken wird OmniGraffle von OmniGroup, für die Erstellung der Folien des Diplomvortrages Keynote von Apple verwendet.

### 4.3 Zeitplan

Die Diplomarbeit ist gemäß DPO3 auf sechs Monate – entsprechend 26 Wochen – angelegt. In Abbildung 3 ist der geplante zeitliche Ablauf der Aktivitäten dargestellt.

Gemäß dem Leitfaden zur Durchführung von Diplomarbeiten finden in der Regel wöchentliche Besprechungen mit dem Betreuer statt.

### 4.4 Gliederung

Die schriftliche Arbeit wird sich an der vorläufigen Gliederung in Abbildung 4 orientieren.

Aktivität	Woche
Literaturrecherche	1-3
Vorstellung im SE-Café	5 (?)
Einarbeitung Eclipse	2-5
Einarbeitung Slicing	3-6
Entwurf Bedienschnittstelle	4-7
Entwurf Systemarchitektur	4-8
Erstschrift der Ausarbeitung	6-9
Implementierung & Test	10-19
Auswertung	20-23
Überarbeitung der Ausarbeitung	23-24
Abschlusskorrektur	25
Druck/Bindung	26
Abgabe	26
Präsentation der Ergebnisse im D&D-Seminar	27+x

Abbildung 3: Zeitplan

1. Einleitung
2. Grundlagen und Verwandte Arbeiten
  - 2.1. Refactoring
    - 2.1.1. Definition
    - 2.1.2. Automatisierung
  - 2.2. Duplikationen
    - 2.2.1. Entstehung
    - 2.2.2. Klassifizierung
    - 2.2.3. Algorithmen zur Erkennung
    - 2.2.4. Verfahren zur Behebung
    - 2.2.5. Clone-based Reengineering im Entwicklungsprozess
  - 2.3. Eclipse Plattform
3. Konzeption eines Werkzeugs für das Clone-based Reengineering
4. Systemarchitektur des Werkzeugs für das Eclipse JDT
5. Anmerkungen zur Implementierung
6. Auswertung
  - 6.1. Anwendung auf Beispielprojekte
  - 6.2. Schlussfolgerungen
  - 6.3. Möglichkeiten für weiterführende Arbeiten
7. Literaturverzeichnis

Abbildung 4: Vorläufige Gliederung

## 5 Produkte

Im Rahmen der Diplomarbeit werden folgende Produkte erstellt:

- Diplomarbeit in gedruckter Form und als PDF-Datei,
- Folien des Diplomvortrags als PDF,
- Prototyp des Werkzeugs in Binärformat und Quelltext

Die Quelltexte des Werkzeugs werden nach Bewertung der Diplomarbeit unter der LGPL (Lesser GNU Public License) oder der CPL (Common Public License) veröffentlicht.

## Literatur

- [Bak92] BAKER, BRENDA S.: *A Program for Identifying Duplicated Code*. Computing Science and Statistics, 24:49–57, 1992. Available from World Wide Web: [citeseer.nj.nec.com/baker92program.html](http://citeseer.nj.nec.com/baker92program.html).
- [Bak95] BAKER, BRENDA S.: *On Finding Duplication and Near-Duplication in Large Software Systems*. In: WILLS, L., P. NEWCOMB und E. CHIKOFKY (Herausgeber): *Second Working Conference on Reverse Engineering*, Seiten 86–95, Los Alamitos, California, 1995. IEEE Computer Society Press. Available from World Wide Web: [citeseer.nj.nec.com/baker95finding.html](http://citeseer.nj.nec.com/baker95finding.html).
- [Bec00] BECK, KENT (Herausgeber): *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2000.
- [Bel02] BELLON, STEFAN: *Vergleich von Techniken zur Erkennung duplizierten Quellcodes*. Diplomarbeit, Universität Stuttgart, September 2002.
- [BF00] BECK, KENT und MARTIN FOWLER: *Extreme Programming Explained: Embrace Change*, Kapitel Bad Smells in Code. In: BECK, KENT [Bec00], 2000.
- [BGKS02] BAUER, MARKUS, THOMAS GENSSLER, VOLKER KUTTRUFF und OLAF SENG: *Werkzeugunterstützung für evolutionäre Softwareentwicklung*. In: *Proceedings of the fourth German Workshop on Software-Reengineering*, Bad Honnef, April 2002.
- [BMD<sup>+</sup>99a] BALAZINSKA, MAGDALENA, ETTORRE MERLO, MICHEL DAGENAIS, BRUNO LAGUE und KOSTAS KONTOGIANNIS: *Measuring Clone Based Reengineering Opportunities*. In: *Sixth International Symposium on Software Metrics*, Seiten 292–303. IEEE, IEEE Computer Society Press, 1999.
- [BMD<sup>+</sup>99b] BALAZINSKA, MAGDALENA, ETTORRE MERLO, MICHEL DAGENAIS, BRUNO LAGUE und KOSTAS KONTOGIANNIS: *Partial Redesign of Java Software Systems Based on Clone Analysis*. In: *Sixth Working Conference on Reverse Engineering*, Seiten 326–336, Atlanta, Georgia, Oktober 1999. IEEE, IEEE Computer Society Press.

- [BMD<sup>+</sup>00] BALAZINSKA, MAGDALENA, ETTORE MERLO, MICHEL DAGENAI, BRUNO LAGUE und KOSTAS KONTOGIANNIS: *Advanced Clone-Analysis to Support Object-Oriented System Refactoring*. In: *Seventh Working Conference on Reverse Engineering*, Seiten 98–107, Brisbane, Australia, Nov 2000. IEEE, IEEE Computer Society Press.
- [BYM<sup>+</sup>98] BAXTER, IRA D., ANDREW YAHIN, LEONARDO M. DE MOURA, MARCELO SANT'ANNA und LORRAINE BIER: *Clone Detection Using Abstract Syntax Trees*. In: *Proceedings of International Conference on Software Maintenance*, Seiten 368–377. IEEE Computer Society Press, 1998. Available from World Wide Web: [citeseer.nj.nec.com/baxter98clone.html](http://citeseer.nj.nec.com/baxter98clone.html).
- [Com] COMPUWARE. *OptimalJ 2.2 Package Structure Analysis Tool* [online]. Available from World Wide Web: <http://javacentral.compuware.com/pasta/>.
- [CX01] CHEN, ZHENQIANG und BAOWEN XU: *Slicing object-oriented Java programs*. ACM SIGPLAN Notices, 36(4):33–40, 2001.
- [DDNJ02] DEMEYER, SERGE, STÉPHANE DUCASSE, OSCAR NIERSTRASZ und RALPH E. JOHNSON: *Object Oriented Reengineering Patterns*. Morgan Kaufmann Publishers Inc., 2002.
- [DRD99] DUCASSE, STÉPHANE, MATTHIAS RIEGER und SERGE DEMEYER: *A Language Independent Approach for Detecting Duplicated Code*. In: YANG, HONGJI und LEE WHITE (Herausgeber): *Proceedings of International Conference on Software Maintenance*, Seiten 109–118. IEEE, 1999. Available from World Wide Web: [citeseer.nj.nec.com/ducasse99language.html](http://citeseer.nj.nec.com/ducasse99language.html).
- [DRG99] DUCASSE, STÉPHANE, MATTHIAS RIEGER und GEORGES GOLOMINGI: *Tool Support for Refactoring Duplicated OO Code*. In: DUCASSE, STÉPHANE und OLIVER CIUPKE (Herausgeber): *Proceedings of the ECOOP'99 Workshop on Experiences in Object-Oriented Re-Engineering*. Forschungszentrum Informatik, Karlsruhe, Juni 1999. Available from World Wide Web: [citeseer.nj.nec.com/ducasse99tool.html](http://citeseer.nj.nec.com/ducasse99tool.html).
- [DRT98] DEMEYER, SERGE, MATTHIAS RIEGER und SANDER TICHELAAAR: *Three Reverse Engineering Patterns*, April 1998. Available from World Wide Web: <http://www.iam.unibe.ch/~scg/Archive/Papers/Deme98pThreeRevEngPatterns.pdf>. Writing Workshop at EuroPLOP'98.
- [DTS99] DEMEYER, SERGE, SANDER TICHELAAAR und PATRICK STEYAERT: *FAMIX 2.0 – The FAMOOS Information Exchange Model*. Technischer Bericht, University of Bern, August 1999. Available from World Wide Web: <http://www.iam.unibe.ch/~demyer/Deme99y/>.
- [DW02] DUDZIAK, THOMAS und JAN WLOKA: *Tool-Supported Discovery and Refactoring of Structural Weaknesses in Code*. Diplomarbeit, TU Berlin, [tomdz@cs.tu-berlin.de](mailto:tomdz@cs.tu-berlin.de), Februar 2002.
- [Flo] FLORIJN, GERT. *RevJava – a review assistant for Java programs* [online]. Available from World Wide Web: <http://www.serc.nl/people/florijn/work/designchecking/RevJava.htm>.

- [Fow99] FOWLER, MARTIN: *Refactoring: improving the design of existing code*. Addison-Wesley Longman Publishing Co., Inc., 1999.
- [GKS<sup>+</sup>] GENSSLER, THOMAS, VOLKER KUTTRUFF, OLAF SENG, TOBIAS GUTZMANN, SEBASTIAN MIES und SVEN LUZAR. *Inject/J Homepage* [online]. Available from World Wide Web: <http://injectj.sourceforge.net>.
- [Hel95] HELFMAN, J.: *Dotplot Patterns: A Literal Look at Pattern Languages*. TAPOS, 2(1):31–41, 1995.
- [HHL02] HEUZEROTH, DIRK, THOMAS HOLL und WELF LÖWE: *Combining Static and Dynamic Analyses to Detect Interaction Patterns*. In: *6th International Conference on Integrated Design and Process Technology*. Society for Design and Process Science, Juni 2002.
- [KH00] KOMONDOOR, RAGHAVAN und SUSAN HORWITZ: *Semantics-preserving procedure extraction*. In: *Proceedings of the 27th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, Seiten 155–169. ACM Press, 2000.
- [KH01a] KOMONDOOR, RAGHAVAN und SUSAN HORWITZ: *Tool Demonstration: Finding Duplicated Code Using Program Dependences*. In: SANDS, D. (Herausgeber): *Programming Languages and Systems*, Band 2028 der Reihe *Lecture Notes in Computer Science*, Seiten 383–386. Springer, 2001. Available from World Wide Web: [citeseer.nj.nec.com/487759.html](http://citeseer.nj.nec.com/487759.html).
- [KH01b] KOMONDOOR, RAGHAVAN und SUSAN HORWITZ: *Using Slicing to Identify Duplication in Source Code*. In: COUSOT, P. (Herausgeber): *Static Analysis (SAS)*, Band 2126 der Reihe *Lecture Notes in Computer Science*, Seiten 40–56. Springer, 2001. Available from World Wide Web: [citeseer.nj.nec.com/komondoor01using.html](http://citeseer.nj.nec.com/komondoor01using.html).
- [KKI02] KAMIYA, TOSHIHIRO, SHINJI KUSUMOTO und KATSURO INOUE: *CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code*. *IEEE Transactions on Software Engineering*, 28(7):654–670, 2002.
- [KN01] KONI-N’SAPU, GEORGES GOLOMINGI: *A Scenario Based Approach for Refactoring Duplicated Code in Object Oriented Systems*. Diploma thesis, University of Bern, Juni 2001. Available from World Wide Web: <http://www.iam.unibe.ch/~scg/Archive/Diploma/golomingi.pdf>.
- [Kri01] KRINKE, JENS: *Identifying Similar Code with Program Dependence Graphs*. In: *Proc. Eighth Working Conference on Reverse Engineering*, Seiten 301–309, 2001. Available from World Wide Web: [citeseer.nj.nec.com/krinke01identifying.html](http://citeseer.nj.nec.com/krinke01identifying.html).
- [LH] LUDWIG, ANDREAS und DIRK HEUZEROTH. *Recoder Homepage* [online]. Available from World Wide Web: <http://recoder.sourceforge.net>.
- [LH96] LARSEN, LOREN und MARY JEAN HARROLD: *Slicing object-oriented software*. In: *Proceedings of the 18th international conference on Software engineering*, Seiten 495–505. IEEE Computer Society Press, 1996.

- [Mar01] MARUYAMA, KATSUHISA: *Automated method-extraction refactoring by using block-based slicing*. In: *Proceedings of the 2001 symposium on Software reusability*, Seiten 31–40. ACM Press, 2001.
- [OTI03] OBJECT TECHNOLOGY INTERNATIONAL: *Eclipse Platform Technical Overview*, Februar 2003. Available from World Wide Web: <http://eclipse.org/whitepapers/eclipse-overview.pdf>.
- [RBJ97] ROBERTS, DON, JOHN BRANT und RALPH JOHNSON: *A refactoring tool for Smalltalk*. *Theory and Practice of Object Systems*, 3(4):253–263, 1997.
- [RD98] RIEGER, MATTHIAS und STÉPHANE DUCASSE: *Visual Detection of Duplicated Code*. In: DUCASSE, STÉPHANE und JOACHIM WEISBROD (Herausgeber): *Proceedings ECOOP Workshop on Experiences in Object-Oriented Re-Engineering*. Forschungszentrum Informatik Karlsruhe, 1998. Available from World Wide Web: [citeseer.nj.nec.com/rieger98visual.html](http://citeseer.nj.nec.com/rieger98visual.html).
- [Rob99] ROBERTS, DONALD BRADLEY: *Practical Analysis for Refactoring*. Doktorarbeit, University of Illinois at Urbana-Champaign, 1999.
- [TDDN00] TICHELAAR, SANDER, STÉPHANE DUCASSE, SERGE DEMEYER und OSCAR NIERSTRASZ: *A Meta-model for Language-Independent Refactoring*. In: *Proceedings ISPSE 2000*, Seiten 157–167. IEEE, 2000. Available from World Wide Web: [citeseer.nj.nec.com/tichelaar00metamodel.html](http://citeseer.nj.nec.com/tichelaar00metamodel.html).
- [vEM] EMDEN, EVA VAN und LEON MOONEN. *JCosmo Code Smell Browser* [online]. Available from World Wide Web: <http://www.cwi.nl/projects/renovate/javaQA/>.