HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

# MASTER THESIS

B.Sc.
**Nicole Hartmann**

## Analyse von Software-Testergebnissen mit Data Mining-Methoden

2011

Fakultät **Elektro- und Informationstechnik**

# MASTER THESIS

## Analyse von Software-Testergebnissen mit Data Mining-Methoden

Autor:
**Nicole Hartmann**

Studiengang:
Industrial Management (M.Sc.)

Seminargruppe:
ZM08w1

Erstprüfer:
Prof. Dr.-Ing. Wilfried Schubert,
Fakultät Mathematik/Naturwissenschaften/Informatik

Zweitprüfer:
Dr. rer. nat. Christoph Lingenfelder,
IBM Deutschland Research & Development GmbH

Mittweida, Juli 2011

**Bibliografische Angaben**

Hartmann, Nicole: Analyse von Software-Testergebnissen mit Data Mining-Methoden, 95 Seiten, 37 Abbildungen, Hochschule Mittweida, Fakultät Elektro- und Informationstechnik

Master Thesis, 2011

**Referat**

Beim Software-Test können automatisierte Testtools heute eine große Anzahl von Tests durchführen, bei denen viele Kombinationen von Daten und Parametern verwendet werden. Dementsprechend schwierig ist es, diese Tests systematisch auszuwerten und zu Erkenntnissen über Stärken und Schwächen eines Software-Produktes zu kommen. Aus diesem Grund soll untersucht werden, in welcher Form Data Mining-Methoden eingesetzt werden können, um die Schwachpunkte einer Software aufzudecken. Des Weiteren sollen die Möglichkeiten einer mittels Data Mining erfolgenden systematischen und automatischen Priorisierung und Auswahl einer reduzierten Menge von Testfällen für den Regressionstest beleuchtet werden.

**Markenrechtlicher Hinweis**

# I. Inhaltsverzeichnis

# II. Abbildungsverzeichnis

# III. Tabellenverzeichnis

# IV. Abkürzungsverzeichnis

| | |
|---|---|
| BS | Betriebssystem |
| bzw. | beziehungsweise |
| CRISP-DM | Cross Industry Standard Process for Data Mining |
| CSV | comma-separated values |
| DBMS | Datenbankmanagementsystem (engl. database management system) |
| DBS | Datenbanksystem |
| DB | Datenbank |
| DM | Data Mining |
| DS | Design Studio |
| EM | Easy Mining |
| ERD | Entity-Relationship-Diagramm |
| ERM | Entity-Relationship-Modell |
| ETL | extract-load-transform |
| FCM | factors criterion metrics |
| FMEA | Fehlermöglichkeits- und Einflussanalyse (engl. Failure Mode and Effects Analysis) |
| HW | Hardware |
| IBM | International Business Machines |
| IDE | Integrierte Entwicklungsumgebung (engl. integrated development environment) |
| IM | Intelligent Miner |
| ISW | InfoSphere Warehouse |
| IT | Informationstechnik (engl. information technology) |
| JDBC | Java Database Connectivity |
| KDD | Knowledge Discovery in Databases |
| ODC | Orthogonal Defect Classification |
| OLAP | Online Analytical Processing |
| PMML | Predictive Model Markup Language |
| PoC | Point of Control |
| PoO | Point of Observation |
| SQL/MM | SQL Multimedia and Application Packages |
| SQL | Structured Query Language |
| SW | Software |
| UDF | user-defined function |
| UDM | user-defined method |
| UDT | user-defined type |
| XML | Extensible Markup Language |
| z.B. | zum Beispiel |

# V. Vorwort

Die vorliegende Arbeit wurde in Zusammenarbeit mit dem Unternehmen IBM Deutschland Research & Development GmbH [1] erstellt, insbesondere mit den Mitarbeitern der Abteilungen Data Mining und Quality Assurance, beide Teil des Bereichs Information Management der IBM Software Group [2].

Hiermit möchte ich mich bei all denen ganz herzlich bedanken, die mir während der praktischen Durchführung dieser Master Thesis und bei der Erstellung des schriftlichen Teils mit ihrem Wissen und Rat zur Seite standen. Insbesondere danken möchte ich Herrn Dr. Christoph Lingenfelder, der mich während der gesamten Bearbeitungsphase fachlich betreute und bei der Umsetzung der gesteckten Ziele unterstützte. Auch Herrn Prof. Dr.-Ing. Wilfried Schubert von der Hochschule Mittweida möchte ich an dieser Stelle für seine Unterstützung bei fachlichen und auch bei Formfragen danken. Des Weiteren sei den Mitarbeitern von IBM, vor allem der beiden Abteilungen Data Mining und Quality Assurance gedankt, die mir gern Hilfestellung bei auftauchenden Problemen gaben. Meinem Freund und meiner Familie möchte ich für die Ratschläge und die Unterstützung in stressreichen Momenten danken.

# 1 Einleitung

## 1.1 Motivation

In einem Unternehmen, welches Software entwickelt, sollte die Zufriedenheit seiner Kunden an erster Stelle stehen, denn diese ist eine notwendige Voraussetzung für den langfristigen Erfolg des Unternehmens. Die Kunden werden so lange zufrieden mit dem entwickelten Produkt sein, wie es ihren Qualitätsanforderungen genügt oder diese sogar übertrifft. Ein Unternehmen kann die Erfüllung der Qualitätsanforderungen durch seine Produkte nur dann garantieren, wenn es zum einen die Anforderungen seiner Kunden kennt und sich zum anderen der qualitativen Stärken und Schwächen seiner entwickelten Software-Produkte bewusst ist, denn nur wenn Schwächen bekannt sind, können Maßnahmen zur Verbesserung der Qualität eingeleitet werden. Zur Bestimmung der Stärken und Schwächen müssen, getreu dem Motto *„You cannot control what you cannot measure“* (Tom DeMarco), im Rahmen der Qualitätssicherung geeignete Methoden zur Messung der Qualität der entwickelten Software eingeführt werden. Entsprechende Qualitätssicherungsmethoden sind in der Literatur bereits zu finden, jedoch ist die Anwendung dieser (meist manuellen) Methoden im Unternehmen oft mit einem hohen Aufwand verbunden. Dadurch kann die Anwendung in Konflikt mit den beschränkten Ressourcen des Unternehmens geraten und wird in vielen Fällen, zu Lasten der Software-Qualität, nur beschränkt durchgeführt. Aus diesem Grund ist die Entwicklung von effizienten und wirtschaftlichen Messmethoden von enormer Wichtigkeit. Im Rahmen der wissenschaftlichen Forschung konnten bereits kontinuierliche Verbesserungen der Qualitätssicherungstechniken erwirkt werden, jedoch besteht weiterhin ein Verbesserungsbedarf der Lösungen. Die vorliegende Arbeit soll einen Beitrag zur Effizienzsteigerung im Bereich der Software-Qualitätssicherung leisten, indem eine Lösung zur systematischen und automatischen Analyse der Software-Qualität mittels der Analyseverfahren des Data Mining entwickelt wird.

## 1.2 Zielstellungen

Das Ziel der vorliegenden Arbeit ist daher, ein Verfahren aufzuzeigen, welches sowohl zur systematischen Analyse der Qualität eines Software-Produktes als auch zur Effizienzsteigerung hinsichtlich des Ressourceneinsatzes in Rahmen der Software-Qualitätssicherung eingesetzt werden kann.

Mit Hilfe der Unterstützung durch die moderne Informationstechnik und die Analyseverfahren des Data Mining soll es möglich sein, fehleranfällige Bestandteile (Komponenten) einer Software zu identifizieren, um während der weiteren Prüfung der Software-Qualität den Fokus insbesondere auf diese Schwachstellen zu legen. Die Ermittlung der fehleranfälligen Komponenten basiert dabei auf der systematischen Auswertung der Ergebnisse früherer Überprüfungen der Software. Des Weiteren soll mit Hilfe des Analyseverfahrens ein effizienterer Einsatz von Testressourcen durch die Priorisierung und Selektion von Testfällen möglich sein.

## 1.3 Thematische Abgrenzung

Diese Arbeit beinhaltet die Vorstellung eines Ansatzes zur Analyse der Qualität von Software, Hardware wird von den Betrachtungen ausgeschlossen. Ferner dient das präsentierte Verfahren der Auswertung von dynamischen Regressionstests, die der Prüfung der korrekten Funktionsweise einer Software dienen und für die eine sehr große Anzahl von Daten zu Testparametern und Testresultaten anfällt, welche eine computergestützte Analyse erforderlich machen. Für Tests der strukturellen Eigenschaften einer Software wird die Anwendbarkeit des zu entwickelnden Verfahrens nicht geprüft und bei Testverfahren mit nur wenigen anfallenden Daten ist der Einsatz von Data Mining-Verfahren nicht erforderlich, da diese mit konventionellen statistischen Methoden von Hand ausgewertet werden können.

## 1.4 Aufbau der Arbeit

In den folgenden Kapiteln wird das im Rahmen dieser Arbeit entwickelte Verfahren zur Aufdeckung von Schwächen einer Software sowie zur Priorisierung und Selektion von Testfällen vorgestellt.

Für ein besseres Verständnis der in der Arbeit verwendeten Begriffe und Methoden des Data Mining werden diese im nachfolgenden, zweiten Kapitel „Einführung in Data Mining“ erläutert. Dies wird für die Grundlagen der in der Arbeit relevanten Teilgebiete des Software-Qualitätsmanagement im dritten Kapitel „Aspekte des Software-Qualitätsmanagements“ wiederholt. Anschließend wird der theoretische Hintergrund des Verfahrens im vierten Kapitel „Analyse der Software-Qualität und Testfallselektion mittels Data Mining“ vorgestellt und die praktische Anwendbarkeit des vorgestellten Verfahrens für ein reales Entwicklungsprojekt in Form einer Fallstudie im fünften Kapitel „Fallstudie ”Entwicklungsprojekt für Data Mining-Software”“ nachvollzogen.

# 2 Einführung in Data Mining

Da seit etwa 50 Jahren die Leistungsfähigkeit von IT-Systemen[1], vor allem im Hinblick auf die Geschwindigkeit der Datenverarbeitung und die Kapazität der Speichermedien, rasant gestiegen ist, können heute riesige Mengen von Daten kostengünstig gespeichert[2] werden. Dies hat zu Folge, dass in allen Bereichen des gesellschaftlichen Lebens, vor allem im wissenschaftlichen, militärischen und kommerziellen Bereich, mit Unterstützung der heutigen IT enorme Mengen von Daten aufgenommen und zu verschiedensten Zwecken ausgewertet werden können. Waren in der Vergangenheit noch Spezialisten damit beschäftigt, Daten von Hand auszuwerten um nützliches Wissen aus ihnen zu gewinnen, ist dies bei den heutigen Datenmengen nicht mehr so einfach möglich. Die Spezialisten mussten sich intensiv mit den Daten vertraut machten und diese anschließend manuell analysieren und interpretieren. Aufgrund des enormen Zeit- und Kostenaufwandes, der mit einer manuellen Analyse von Millionen von Daten verbunden wäre, und der meist hochkomplexen Beziehungen zwischen den Daten, kann eine Datenanalyse auf diese Weise heute nicht mehr von Einzelpersonen bewältigt werden und muss, wenn auch teilweise, mit Unterstützung der IT automatisiert werden. Um dieses Problem zu lösen, wurden in den vergangenen Jahrzehnten auf Basis der Mathematik und Informatik verschiedene Analyseverfahren entwickelt. All diese Verfahren werden eingebettet in einen Gesamtprozess, der alle notwendigen Schritte von der Auswahl der zu untersuchenden Daten und deren Vorbereitung über die eigentliche Informationsgewinnung mit Hilfe der entwickelten Analyseverfahren bis hin zur abschließenden Interpretation der Analyseergebnisse umfasst. Für diesen Prozess können in der Literatur verschiedene Bezeichnungen gefunden werden. Unter Statistikern und Datenanalysten wurde überwiegend der Begriff *Data Mining (DM)* verwendet, während in den Bereichen der künstlichen Intelligenz (KI)[3] und des maschinellen Lernens der Ausdruck *Knowledge Discovery in Databases(KDD)* Verbreitung fand [3]. Bei letzterem wird Data Mining lediglich als Teilabschnitt in den Gesamtprozesses eingeordnet und zwar als der zentrale, in dem verschiedene Analysemethoden auf einen Datenbestand angewendet werden. Da jedoch in der letzten Zeit beide Ausdrücke synonym verwendet wurden, soll Data Mining auch in dieser Arbeit als Bezeichnung für den Gesamtprozess dienen.

Im Verlauf dieses Kapitels wird als Grundlage für ein Verständnis der weiteren Untersuchungen und Betrachtungen vermittelt, was genau unter Data Mining zu verstehen ist und wie ein Data Mining-Prozess einschließlich der durchzuführenden Aufgaben im Groben abläuft. Außerdem soll eine Übersicht der verschiedenen Data Mining-Analysemethoden mit ihren Anwendungsmöglichkeiten, sowie den im Hintergrund ablaufenden mathematischen und statistischen Berechnungsverfahren gegeben werden. Am Schluss des Kapitels werden das IBM InfoSphere Warehouse und die darin enthaltene Data Mining-Lösung, der Intelligent Miner, sowie das Design Studio als Werkzeug zur grafischen Unterstützung eines Data Mining-Projektes vorgestellt.

[1] IT = Informationstechnik (Oberbegriff für die Informations- und Datenverarbeitung)

[2] Speichermedien, wie Festplatten, können heute drei Terabyte (mehr als drei Billionen Byte) an Daten aufnehmen, was einer Menge von etwa anderthalb Milliarden DIN-A4-Blättern und einem Papierstapel von 160 km Höhe entspricht.

[3] engl. artificial intelligence (AI)

## 2.1 Data Mining

Das englische Wort „Mining“ wird übersetzt mit „Abbau“ und bezieht sich auf die Förderung von wertvollen Rohstoffen, wie Gold oder Silber, aus deren natürlichen, unterirdischen Lagerstätten. Entsprechend dieser Analogie können wertvolle Informationen als *nuggets* (Goldklumpen) bezeichnet werden, die in großen Datenmengen verborgen liegen und im Rahmen eines Data-Mining-Prozesses anhand von Mining-Verfahren zu Tage gefördert werden sollen.

> *„[Data Mining bezeichnet den] Prozess des Entdeckens bedeutsamer neuer Zusammenhänge, Muster und Trends durch die Analyse großer Datensätze mittels Mustererkennung sowie statistischer und mathematischer Verfahren“. (Erick Brethenoux, Gartner Group)*

Data Mining kann zur Lösung vieler sich in den verschiedensten gesellschaftlichen Bereichen ergebenden Problemstellungen beitragen, die eine Gewinnung von neuen Erkenntnissen aus großen Datenbeständen zum Inhalt haben. Ein Beispiel für eine solche Problemstellung ist die Vorhersage der Staugefahr auf Straßen, bei der mit Hilfe von Daten aus der Vergangenheit Abhängigkeiten zwischen den örtlichen Auswirkungen von Wetterlagen oder anderen Ereignissen, wie Veranstaltungen, und dem gleichzeitigen Auftreten von Staus bestimmt werden [4]. Auch im Bereich des Handels wird Data Mining eingesetzt, um Muster im Kaufverhalten der Kunden zu entdecken und dementsprechend das Waren- oder Dienstleistungsangebot anzupassen. Ein weiteres Beispiel findet sich in der Geologie, wo Data Mining das Auffinden von Erdöllagerstätten unterstützt, indem aus großen Mengen unterschiedlicher Daten zur mineralischen Zusammensetzung, geologischen Struktur und dem flüssigen Inhalt von Gestein sowie aus seismischen Messungen die für unterirdische Erdöllagerstätten charakteristischen geologischen Eigenschaften herausgefiltert werden. Basierend auf den Data-Mining-Ergebnissen sollen unter anderem präzisere Simulationen der genauen Lage von Ölfeldern durchgeführt werden können [5]. Viele dieser Problemstellungen sind ähnlicher Natur und können zwei wesentlichen Problemtypen zugeordnet werden, den *Beschreibungsproblemen (description)* und den *Prognoseproblemen (prediction)* [3, 6].

Bei Beschreibungsproblemen werden Muster und Zusammenhänge in den Daten von Objekten der realen Welt - Individuen, Gegenständen, Ereignissen oder abstrakte Konzepten - aufgedeckt, um das Verhalten oder den Zustand dieser Objekte in einer leicht auswertbaren Form zu *beschreiben*. Zum Beispiel kann damit das Konsumverhalten von verschiedenen Kundengruppen anhand von Warenkorbdaten und demografischen Daten (z.B. Alter, Einkommen) beschrieben werden. Ein Objekt wird dabei repräsentiert durch einen Datensatz (Zeile einer Tabelle/Datei), der zusammengesetzt ist aus einzelnen Datenfeldern, die zu jedem charakteristischen Attribut (Spalte einer Tabelle/Datei) dieser Art von Objekt den entsprechenden individuellen Attributwert des Objektes enthalten.

Bei Prognoseproblemen werden ebenfalls Zusammenhänge in den Datensätzen erforscht, jedoch um darauf basierend den Zustand oder das Verhalten von Objekten in der Zukunft *vorherzusagen* (zu schätzen). Ein Beispiel dafür ist die eben erwähnte Vorhersage der Staugefahr auf Straßen basierend auf verschiedenen Ereignissen, bei denen in der Vergangenheit bereits ein Zusammenhang mit dem Auftreten von Staus festgestellt werden konnte.

## 2.2 Methoden des Data Mining

Zur Lösung von Beschreibungs- oder Prognoseproblemen stehen in Data-Mining-Werkzeugen verschiedene Data-Mining-Methoden zur Verfügung. Die Auswahl von geeigneten Methoden zur Lösung einer konkreten Problemstellung stellt eine der wichtigsten Aufgaben eines Analysten im Rahmen eines realen Data-Mining-Projektes dar, da hiervon der Erfolg des gesamten Data-Mining-Projektes abhängen kann. Zur Systematisierung der existierenden DM-Methoden sind in der Literatur verschiedene Ansätze zu finden. Aufgrund der Unterscheidung nach den grundlegenden Problemtypen einer Data-Mining-Analyse, wird der von Ballard et al. [7] erarbeitete Ansatz gewählt. Dieser nimmt eine Einteilung in zwei Gruppen vor, die *discovery methods*, frei übersetzt mit „Entdeckungsmethoden“, und die *predictive methods*, zu Deutsch „Prognosemethoden“.

Die **Entdeckungsmethoden** dienen der Lösung von Beschreibungsproblemen, indem sie zuvor unbekannte Muster und Zusammenhänge in den vorliegenden Daten erforschen und zu deren Präsentation in einem Modell abbilden. Zu den Entdeckungsmethoden zählen die Cluster-Analyse und die Assoziationsanalyse.

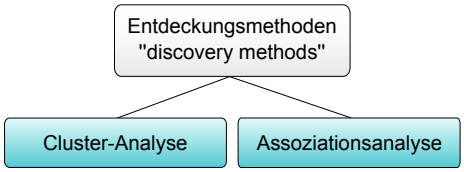



**Abbildung 2-1: Entdeckungsmethoden des Data Mining**

**Prognosemethoden** werden dagegen eingesetzt bei Prognoseproblemen, um Korrelationen in Daten der Vergangenheit zu finden und anhand eines, aus diesen Informationen erstellten, Modells den Zustand oder das Verhalten eines Objektes in der Zukunft vorherzusagen. In die Gruppe der Prognosemethoden können die Klassifikationsanalyse und die Regressionsanalyse eingeordnet werden.

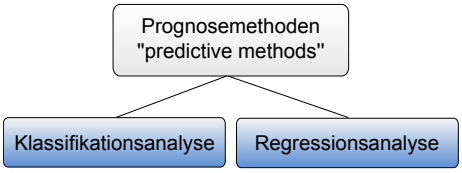



**Abbildung 2-2: Prognosemethoden des Data Mining**

Um ein Verständnis für die Eignung der einzelnen Data-Mining-Methoden bezüglich des Einsatzes bei verschiedenartigen Problemstellungen herzustellen, wird nachfolgend eine Einblick in diese gegeben.

#### Cluster-Analyse

Mit Hilfe der Cluster-Bildung, auch als Segmentierung oder Clustering bezeichnet, können Objekte anhand ihrer Ähnlichkeit zu Gruppen (Cluster) zusammengefasst werden. Die Objekte innerhalb eines Clusters weisen dann, bezogen auf die Ausprägungen ihrer Attribute, eine

möglichst große Ähnlichkeit auf, während sich die Objekte eines Clusters von denen der anderen Cluster in ihren Attributwerten möglichst stark unterscheiden. Die Cluster sind vor ihrer Bildung nicht bekannt, sondern entstehen während des Analyseprozesses mit Hilfe von Methoden aus dem Bereich der multivariaten Statistik [8]. Das Ziel der Cluster-Analyse ist die Gewinnung von nützlichem Wissen über ähnliche Objekte. Mit diesem Wissen können zum Beispiel im Rahmen von Marketingaktionen Kundengruppen mit ähnlichen Produktvorlieben ausgemacht und gezielt angesprochen werden.

#### Assoziationsanalyse

Mit der Assoziationsanalyse können interessante Zusammenhänge (Assoziationen) im Auftreten von Attributwerten bei Objekten aufgedeckt werden. Die gefundenen Abhängigkeiten werden durch Assoziationsregeln der Form $A \rightarrow B$ dargestellt, wobei $A$ und $B$ zwei Mengen von Elementen (Attributwerten) darstellen. Eine solche Assoziationsregel besagt, dass wenn bei einem Objekt eine Menge von bestimmten Elementen vorhanden ist, mit hoher Wahrscheinlichkeit noch eine Menge mit bestimmten anderen Elementen vorliegt. Anzumerken ist, dass es sich bei den beschriebenen Zusammenhängen um Korrelationen und nicht zwingend um kausale Ursache-Wirkungs-Zusammenhänge handelt [8]. Über mögliche Zusammenhänge werden im Vorfeld der Analyse keine Annahmen getroffen, diese werden analog zur Cluster-Analyse im Laufe des Analyseprozesses mit Hilfe der Berechnungsverfahren gefunden. Ein typisches Beispiel für eine Assoziationsanalyse ist die Warenkorbanalyse, bei der ermittelt wird, welche Waren in einem realen beziehungsweise einem Online-Geschäft besonders häufig zusammen gekauft werden, um daran zum Beispiel die Platzierung der Waren oder Rabattaktionen auszurichten [8]. Erkennbar sind im Hintergrund stattfindende Warenkorbanalysen auf Webseiten von Online-Shops zum Beispiel durch Aussagen der Art „Kunden, die dieses Produkt gekauft haben, kauften auch folgende Produkte“.

#### Klassifikationsanalyse

Bei der Klassifikationsanalyse wird die Zugehörigkeit von Objekten anhand ihrer Merkmale zu einer von mehreren alternativen, zuvor definierten Klassen bestimmt. Die abstrakte Beschreibung der charakteristischen Merkmale einer jeden Klasse erfolgt durch ein Klassifikationsmodell, welches mit Hilfe eine Untersuchung der verschiedenen Merkmale von Objekten die bereits den Klassen zugeordnet sind, also anhand von Daten aus der Vergangenheit, erstellt wurde. Die Merkmale von Objekten mit bisher unbekannter Klassenzugehörigkeit werden mit dem Modell abgeglichen und diese Objekte dann der Klasse mit der größten Übereinstimmung zugeordnet. Zum Beispiel werden die Daten aus vergangenen EEG[4]-Messungen von Epilepsie-Patienten mit Hilfe von Klassifikationsalgorithmen analysiert, um anhand charakteristischer Merkmale in den Messergebnissen den aktuellen Zustand von Patienten hinsichtlich eines epileptischen Anfalls in „normal“, „Anfallvorstufe“ und „Anfallnachstufe“ einzuordnen (zum Zweck der Diagnose oder Therapie) [9].

[4] Elektroenzephalografie, ein medizinisches Messverfahren zur Bestimmung der elektrischen Aktivität des Gehirns

#### Regressionsanalyse

Die Regressionsanalyse stimmt in ihrer Zielsetzung, der Prognose zukünftiger Attributwerte von Objekten, mit der Klassifikationsanalyse überein, jedoch handelt es sich bei den vorhergesagten Werten nicht um bereits bekannte kategorische Werte (Klassen) sondern um unbekannte numerische. Der unbekannte Wert eines Objekt-Attributes („abhängige Variable“) wird basierend auf der Abhängigkeit zu mindestens einem weiteren Attribut mit bekannten Werten („unabhängige Variable“) bestimmt [6, 8]. Bei der Zeitreihenanalyse, einem Spezialfall der Regressionsanalyse, wird die zukünftige Entwicklung der Werte eines Objekt-Attributes im Zeitverlauf zusätzlich auf Basis von deren bekannter historischer Entwicklung bestimmt [8]. Praktisch angewendet wird die Zeitreihenanalyse zum Beispiel zur Prognose der künftigen Entwicklung von Aktienkursen am Finanzmarkt [10] anhand einer mathematisch-statistischen Analyse der zeitlichen Aktienkurs-Verläufe der Unternehmen in der Vergangenheit und der zu den Aktien gesammelten historischen Daten.

#### Kombination der Methoden

Es ist ebenfalls möglich mehrere der Methoden zur Lösung einer Aufgabe zu kombinieren, zum Beispiel die Verwendung einer Kundensegmentierung in Kombination mit einer Warenkorbanalyse, um sowohl die geeigneten Kunden (Segmentierung) als auch die zum Kunden passenden Werbeartikel (Warenkorbanalyse) für eine bestimmte Marketing-Kampagne zu wählen. Ferner kann es notwendig sein, verschiedene der Methoden getrennt auf die vorhandenen Daten anzuwenden, um auszuloten welches, mit Hilfe der Methoden, erstellte Modell der Aufgabe am besten gerecht wird [7, S.49].

## 2.3 Vergleich mit bisherigen Verfahren

Zur Analyse der Daten bedienen sich die Data-Mining-Methoden der traditionellen Verfahren aus verschiedenen wissenschaftlichen Feldern, wie der Statistik, dem maschinellen Lernen und der Mustererkennung [3, S.39] und weist daher Ähnlichkeiten zu bereits bekannten Analyseverfahren, wie der Multidimensionalen Datenanalyse (MDA), dem Online Analytical Processing (OLAP), modernen Visualisierungstechniken oder traditionellen statistischen Verfahren auf. Man wird sich an dieser Stelle vielleicht fragen, was Data Mining von den bisherigen Analyseverfahren unterscheidet und es zu etwas Besonderem macht, das einen eigenen Ausdruck verdient. Die unvoreingenommene Erkenntnissuche der Data-Mining-Verfahren kann als ein erster Grund gesehen werden. Viele der bisherigen Verfahren sind „hypothesengetrieben“ und dienen der Auswertung von Daten aus der Vergangenheit zur Verifikation oder Falsifizierung einer im Vorfeld der Analyse aufgestellten Hypothese. Im Gegensatz dazu dienen Data-Mining-Verfahren der „datengetriebenen“ Analyse und suchen „eigenständig“ nach Mustern und Zusammenhängen in den Daten um Hypothesen zu generieren. Aufgrund der fehlenden Beeinflussung durch eine vorab definierte Hypothese können auch weitere, vom Analysten bisher nicht bedachte Aspekte in den Daten einen Erkenntnisgewinn bewirken. Ein anderer wichtiger Grund ist die Anwendbarkeit der Data-Mining-Verfahren auf sehr große Datenmengen, bei denen andere Analyseverfahren, wie diejenigen aus dem Bereich der traditionellen Statistik, nicht oder nur unter einem enormen zeitlichen Aufwand einsetzbar wären.

## 2.4 Prozessmodell

Um den grundlegenden Ablauf eines Data-Mining-Prozesses kennenzulernen, kann man sich mit verschiedenen, in der wissenschaftlichen Literatur existierenden Prozessmodellen zu dessen Strukturierung und Beschreibung beschäftigen und wird feststellen können, dass diese sich hinsichtlich der Inhalte der durchzuführenden Aufgaben sehr ähnlich sind und sich lediglich anhand der Einordnung der Aufgabe in Prozessschritte sowie des Detailgrades der Aufgabenbeschreibung unterscheiden. Ein Vertreter dieser Prozessmodelle ist der *Cross Industry Standard Process for Data Mining (CRISP-DM)*, dessen Methodologie im Gegensatz zu anderen Modellen, neben den allgemeinen auch recht detaillierte, praxisnahe Beschreibungen der durchzuführenden Aufgaben enthält, weshalb dieses Prozessmodell im weiteren Verlauf der Arbeit für die Datenanalyse mittels Data Mining verwendet wird.

Das *CRISP-DM-Prozessmodell* wurde in Zusammenarbeit der drei Unternehmen DaimlerChrysler (heute Daimler AG), SPSS und NCR sowie der von ihnen gegründeten Vereinigung von Data-Mining-Experten, der CRISP-DM Special Interest Group (SIG), entwickelt. Es stellt die Grundlage einer standardisierten und industrieunabhängigen Methodologie zur Unterstützung bei der Durchführung eines Data Mining-Projektes dar, in welche die praktischen Data-Mining-Erfahrungen der beteiligten Experten eingeflossen sind [11]. Die vollständige Version des Prozessmodells wurde im Jahr 2000 als *CRISP-DM 1.0* beschrieben im Handbuch „CRISP-DM 1.0 Step-by-step data mining guide“ [12].

Die *CRISP-DM-Methodologie* beschreibt den Data-Mining-Prozess auf vier hierarchisch angeordneten Abstraktionsstufen: Phasen, allgemeine Aufgaben, spezialisierte Aufgaben und Prozessinstanzen. Auf der höchsten Ebene ist der Data-Mining-Prozess in sechs *Phasen* gegliedert, wobei jede Phase in der zweiten Stufe aus mehreren *allgemeinen Aufgaben* besteht. Diese Aufgaben sind allgemein gehalten, um für alle denkbaren Kontexte eines Data-Mining-Projektes, wie eine Kundensegmentierung im Marketing-Bereich oder eine Abhängigkeitsanalyse bei defekten Kraftfahrzeugteilen im KFZ-technischen Bereich, gleichermaßen anwendbar zu sein. Innerhalb der dritten Stufe des Modells werden *spezielle Aufgaben* beschrieben, die dazu dienen die allgemeinen Aufgaben (zum Beispiel die Bereinigung von Daten) in konkreten Projektsituationen durch individuell erforderliche Tätigkeiten (zum Beispiel die Beseitigung von fehlenden Werten) umzusetzen. Die unterste Modellebene, die der *Prozessinstanzen*, stellt eine Aufzeichnung der praktischen Handlungen, Entscheidungen und Resultate innerhalb der einzelnen Aufgabengebiete eines Data-Mining-Projektes dar [12, S.9]. Zur Erläuterung der CRISP-DM-Methodologie dienen zwei Beschreibungsformen mit unterschiedlichem Abstraktionsgrad, zum einen ein Referenzmodell mit einem kurzen Überblick über die Phasen und Aufgaben inklusive deren Ergebnisse und zum anderen ein User-Guide mit detaillierteren Hinweisen und Tipps für jede Phase und die einzelnen Aufgaben. Das Referenzmodell beschreibt demnach allgemein, *was* in einem Data-Mining-Projekt zu tun ist, und der User-Guide, *wie* ein DM-Projekt konkret durchzuführen ist [12, S.10].

Abbildung 2-3 zeigt den im Referenzmodell beschriebenen *Lebenszyklus* eines Data-Mining-Projektes mit seinen sechs Phasen. Diese laufen nicht streng linear ab, denn in Abhängigkeit vom Ergebnis einer Phase können Vor- und Rücksprünge zwischen den einzelnen Phasen und deren Aufgaben notwendig sein. Der äußere Kreis in Abbildung 2-3 symbolisiert den iterativen Charakter des gesamten Data-Mining-Prozesses, denn auch nach dem Abschluss eines Projektes ist die

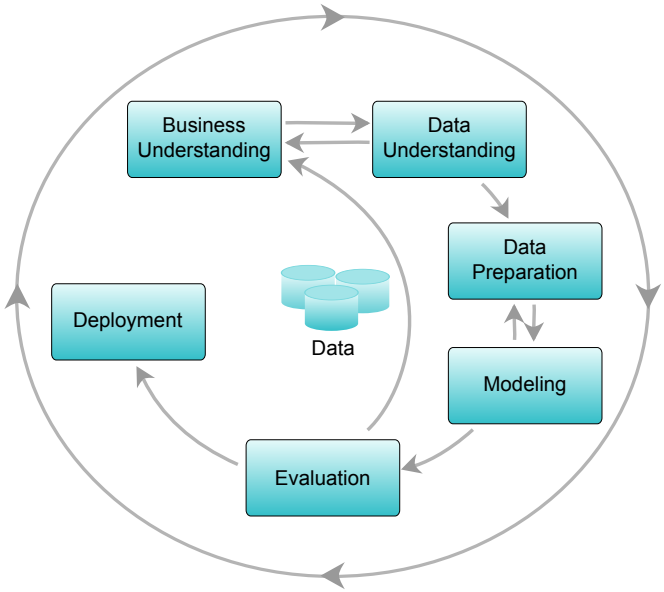



**Abbildung 2-3: Phasen des CRISP-DM-Referenzmodells, in Anlehnung an [11]**

Datenanalyse noch nicht beendet. Die während des Projektes gewonnenen Erkenntnisse können Anstoß für neue, oft präzisere Fragestellungen an das Data Mining geben.

Die Phasen des Data-Mining-Lebenszyklus und die ihnen zugeordneten Aufgaben werden, in Anlehnung an [12], im Folgenden näher vorgestellt.

### Business Understanding

In dieser ersten Phase des Data-Mining-Prozesses sollen die Ziele und Rahmenbedingungen der Data-Mining-Analyse genau geklärt werden. Der Analyst verschafft sich einen umfassenden Überblick über die bestehende Situation im Projektumfeld sowie über die geforderten Projektziele und andere wichtige Faktoren, die das Ergebnis des Projektes beeinflussen können, wie eventuell zu berücksichtigende Einschränkungen (zum Beispiel gesetzlicher Art) oder die Verfügbarkeit von Ressourcen. Aus all diesen Informationen leitet er die Fragestellungen und Ziele für das Data Mining ab. Außerdem erarbeitet er sorgfältig einen Projektplan, in dem festgelegt ist, anhand welcher nachfolgenden Schritte die geforderten Projektziele erreicht werden sollen. Am Schluss dieser Phase wird aus den am Markt verfügbaren Data-Mining-Werkzeugen ein für das Projekt geeignetes ausgesucht, zum Beispiel eines das diverse Funktionen zur Unterstützung der späteren Prozessschritte oder eine für die Data-Mining-Ziele geeignete Auswahl von Data-Mining-Verfahren bereithält.

### Data Understanding

Dieser Prozessschritt beinhaltet die Erfassung und das nähere Kennenlernen der Daten. Die Daten (oder deren Zugriff) sind zu Beginn entweder aus verschiedensten Quellen zu beschaffen (falls sie bereits vorliegen) oder neu zu generieren und anschließend in Dateien oder Datenbanktabellen abzuspeichern. Falls eine Software zum Verstehen der Daten verwendet wird, sind die Daten in diese hinein zu laden. Danach können die direkt sichtbaren Eigenschaften der

Daten, wie deren Format oder Größe (Anzahl der Datensätze und Datenfelder), untersucht und beschrieben werden. Darüber hinaus kann eine Erforschung der verborgenen Eigenschaften der Daten, wie deren Werteverteilung oder Korrelationen zwischen einzelnen Datenattributen, mittels statistischer Verfahren geeignet sein, um einige Data-Mining- Fragen bereits zu beantworten. Außerdem sollten in dieser Phase qualitative Probleme der Daten eingeschätzt werden, zum Beispiel anhand der Feststellung fehlender Datenwerte und Anomalien, sowie die Methoden zu deren Behandlung festgelegt werden.

**Data Preparation**

Diese Phase schließt alle Aktivitäten zur Vorbereitung der vorhandenen Rohdaten auf die nachfolgende Analyse mittels Data-Mining-Verfahren ein. Die Aktivitäten sind mehrmals durchzuführen, sobald verschiedene Data-Mining-Verfahren mit individuellen Anforderungen an die Daten eingesetzt werden sollen.

Eine *Selektion* wird vorgenommen, um eine Teilmenge von Datensätzen und Datenattributen aus den Rohdaten auszuwählen, die relevant für die Fragestellungen des Data Mining ist oder die bestimmten Qualitätsanforderungen und technischen Beschränkungen (zum Beispiel Datenvolumen, Datentypen) genügt.

Im *Bereinigungsschritt* werden Daten in ihrer Qualität soweit verbessert, wie es die eingesetzten Data-Mining-Verfahren erfordern. Dies kann zum Beispiel realisiert werden, indem Teilmengen aus den Rohdaten selektiert werden, welche die benötigte Qualität aufweisen, oder indem geeignete Standardwerte für fehlende Werte eingesetzt werden. Bei der Bereinigung kann man sich an den Einschätzungen aus der vorhergehenden Phase orientieren.

Außerdem kann es zu einer effektiveren Lösung der Data-Mining-Fragestellung beitragen, wenn im Rahmen einer *Transformation* aus den vorhandenen Datenattributen neue Attribute (zum Beispiel Gewinn = Umsatz * Kosten) oder umgeformte Attribute (zum Beispiel durch Diskretisieren oder Verdichten der Attributwerte) abgeleitet werden.

Weiterhin kann eine *Integration* mehrerer Tabellen, die verschiedene Informationen zu denselben Datenobjekten enthalten, zu einer einzigen Tabelle, aufgrund der besseren Übersicht, von Vorteil bei der Auswertung der Daten sein. Auch kann dies die Voraussetzung eines Data-Mining-Algorithmus sein, welches es vorab zu erfüllen gilt. Die Ergebnistabelle enthält nach der Zusammenführung eine Kombination der einzelnen relevanten Spalten aus den Ursprungstabellen.

Ferner kann aufgrund der Eigenschaften des Analysewerkzeugs eine *Formatierung* der Daten angebracht sein, also eine syntaktische Umformung der Daten (Umwandlung von Zeichen oder der Reihenfolge von Attributen) bei gleichzeitiger Erhaltung ihrer Bedeutung. Zum Beispiel kann es notwendig sein, Kommata innerhalb von Datenfeldern in CSV[5]-Dateien zu entfernen oder durch ein anderes Zeichen zu ersetzen, damit das Analysewerkzeug diese fehlerfrei erkennt.

[5] comma-separated values, ein Dateiformat zur Speicherung oder zum Austausch von Daten in Form von strukturiert aufgebauten Textdateien, bei dem die einzelnen Datenfelder jeweils durch ein Trennzeichen, wie das Komma oder ein anderes Sonderzeichen, getrennt werden

### Modeling

In dieser Phase der Modellbildung wird die eigentliche Datenanalyse durchgeführt, indem mit Hilfe von Data-Mining-Verfahren Informationen aus den Daten gezogen und Modelle erstellt werden, die diese Informationen aufnehmen und repräsentieren. Dazu sind, den Zielen der Analyse entsprechend, geeignete Data-Mining-Verfahren zu wählen. Vor Durchführung der Analyse ist es notwendig einen Mechanismus einzuführen, mit dessen Hilfe die Qualität der erstellten Modelle später nach technischen Kriterien, wie der Vorhersagegenauigkeit oder der Anwendbarkeit des Modell auf neue Objekte zur Prognose ihres Zustandes oder Verhaltens, beurteilt werden kann. Ein Mechanismus zu Ermittlung letzterer wäre zum Beispiel die (zufällige) Aufteilung der Daten in eine Trainings- und eine Testmenge, wobei das Modell auf Basis der Trainingsmenge erstellt und anschließend auf die Testmenge angewendet wird. Wurde ein Mechanismus etabliert, können die ausgewählten DM-Verfahren eingesetzt werden, um die vorbereiteten Datensätze zu analysieren. Für die Verfahren existieren dabei meist spezielle Parameter, die in einem iterativen Prozess auf optimale Werte eingestellt werden sollten, um hinsichtlich der Data-Mining-Ziele qualitativ hochwertige Modelle mit bestmöglichem Informationsgehalt zu gewinnen. Die Modelle können im Anschluss an die Analyse im PMML[6]-Format [13, 14] in eine Datei exportiert und zu deren Interpretation, oder Anwendung auf neue Daten, zwischen verschiedenen Data-Mining-Werkzeugen ausgetauscht werden.

### Evaluation

Nachdem verschiedene Modelle erzeugt wurden, ist in der Evaluierungsphase zu beurteilen, welches davon das nach technischen Kriterien qualitativ beste darstellt und inwieweit dieses Modell auch inhaltlich zur Erfüllung der gestellten Projektziele beiträgt. Zusätzlich kann an diesem Punkt anhand einer Testanwendung geprüft werden, ob das Modell in der späteren realen Umgebung den gestellten Zeit- und Kostenanforderungen entsprechen wird. Überdies ist eine kritische Prüfung des gesamten Prozesses ratsam, um Fehler bei der Modellbildung aufzudecken, zum Beispiel die Nutzung von Datenattributen, die in zukünftigen Analysen nicht mehr zur Verfügung stehen. Ebenfalls wichtig ist eine Untersuchung, ob bei der Durchführung des Data-Mining-Projektes wichtige Faktoren oder Aufgaben übersehen wurden, die eventuell eine Wiederholung einiger Prozessschritte notwendig machen. Die Entscheidung, wie nach dieser Phase weiter verfahren wird, das heißt ob das Modell eingesetzt werden kann oder ob Prozessschritte zu wiederholen sind, ist am Schluss dieser Phase zu fällen.

### Deployment

Im Rahmen der Verfügbarmachung des Ergebnisses der Data-Mining-Analyse wird das Modell in eine Software-Anwendung der realen Umgebung integriert. Dies kann zum Beispiel ein Empfehlungsdienst (Recommendation Engine) oder ein Bewertungsverfahren (Scoring) sein. Nach der Einführung des Modells ist ein Wartungsplan für das produktiv eingesetzte Modell (z.B. zur Kontrolle seiner weiteren Gültigkeit) sowie ein abschließender Bericht zum durchgeführten Data-Mining-Projekt einschließlich einer Dokumentation der gesammelten Erfahrungen zu erstellen.

[6] Predictive Model Markup Language, eine standardisierte Sprache zur Speicherung und zum Austausch von statistischen und Data-Mining-Modellen zwischen verschiedenen kompatiblen Anwendungen

## 2.5 Data Mining im IBM InfoSphere Warehouse

Das InfoSphere Warehouse stellt eine integrierte Data-Warehouse-Lösung der Firma IBM dar, die unter anderem Werkzeuge zur Unterstützung des Data-Mining-Prozesses bereithält.

Ein *Data Warehouse* [15] dient im Allgemeinen der langfristigen und vom laufenden Unternehmensgeschäft unabhängigen Speicherung mitunter großer Mengen von Daten und deren Auswertung mit Hilfe mathematischer und statistischer Analysewerkzeuge zur Gewinnung nützlicher Informationen aus den Daten. Zum Beispiel können die im Zuge der betrieblichen Tätigkeiten eines Unternehmens anfallenden Daten von Kunden, Angestellten oder Lieferanten in ein Data Warehouse übertragen und dort ausgewertet werden, um strategische und operative sowie organisatorische Entscheidungen im Unternehmen mit aussagekräftigen Statistiken und Kennzahlen zu unterlegen.

InfoSphere Warehouse, als ein Vertreter von Data-Warehouse-Lösungen, enthält Werkzeuge zum Extrahieren von Daten aus verschiedenen Datenquellen, zum Laden der Daten in das Data Warehouse und zum Transformieren (Vorverarbeiten) der Daten (zusammen bezeichnet als *ETL* für extract-load-transform). Des Weiteren beinhaltet es Tools zur Analyse der vorverarbeiteten Daten mit Hilfe von Methoden wie dem Online Analytical Processing (OLAP), den In-line Analytics oder dem Data Mining. Es ist zum aktuellen Zeitpunkt als InfoSphere Warehouse in Version 9.7 verfügbar [16], wurde jedoch vor einiger Zeit (in Version 9.5) noch mit *DB2 Warehouse* bezeichnet [7, S.51].

#### Übersicht der Komponenten

InfoSphere Warehouse (ISW) besteht, wie Abbildung 2-4 zu entnehmen ist, aus einer komponentenbasierten Architektur mit drei Komponentengruppen, dem Datenbankserver (database server, rechts oben im Bild), dem Anwendungsserver (application server, rechts unten im Bild) und dem Client (links unten im Bild). Diese drei Komponentengruppen erfüllen folgende Aufgaben:

Ein **Datenbankserver** nimmt im Allgemeinen ein oder mehrere Datenbanksysteme (DBS) auf, welche wiederum aus drei Komponenten bestehen. Die erste Komponente umfasst Datenbanken (engl. databases) zur elektronischen Speicherung von Daten. Die Daten werden dabei durch Objekte repräsentiert (zum Beispiel Produkte, Kunden), zwischen denen Beziehungen bestehen (zum Beispiel Welcher Kunde kauft welches Produkt?) und die in den meisten Fällen in Tabellen abgelegt sind. Die zweite Komponente, der Datenkatalog (engl. data dictionary) dient zur Speicherung von Metadaten, also Informationen zur Struktur, den Eigenschaften und der Verwendung der in den Datenbanken vorhandenen Daten. Den Aufbau, die Verwaltung, die Kontrolle und Manipulation der Daten übernimmt die dritte Komponente, das Datenbankmanagementsystem (engl. database management system, DBMS) [18, S.8f]. Im konkreten Fall des InfoSphere Warehouse wird auf dem Datenbankserver das Datenbanksystem *DB2 Enterprise Server Edition* [19] eingesetzt.

Ein **Anwendungsserver** ist ein System, welches über definierte Schnittstellen einen Zugriff auf bestimmte Anwendungen gewährt. Bei InfoSphere Warehouse sind dies Anwendungen für die manuelle oder zeitlich geplante Ausführung und Überwachung der zur Analyse realer

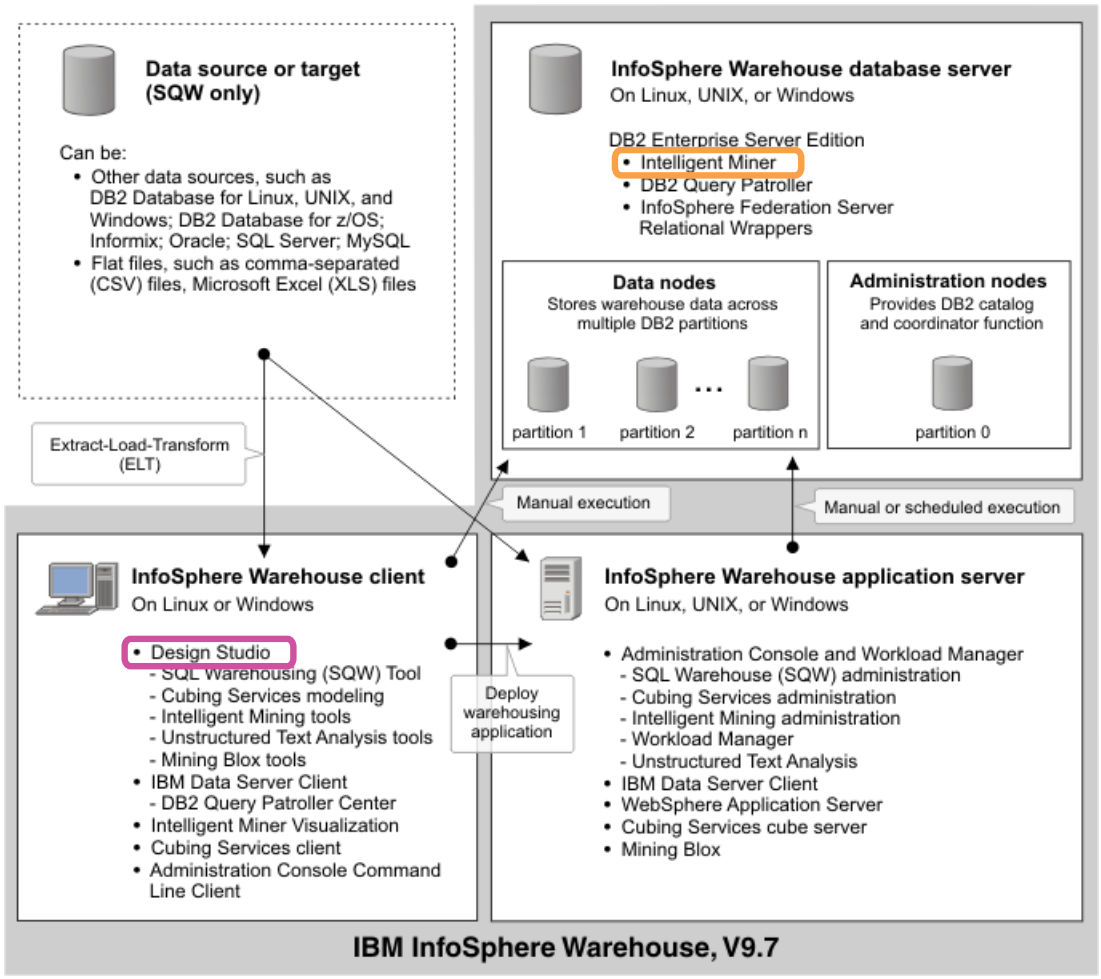



**Abbildung 2-4: Komponenten des IBM InfoSphere Warehouse, entnommen aus [17]**

Geschäftsdaten eingerichteten Warehousing-Anwendungen.

Ein **Client** ist ein Software-Programm, das zu einem anderen Software-Programm, dem Server, eine Verbindung herstellt, um von diesem einen Dienst zur Lösung einer bestimmten Aufgabe in Anspruch zu nehmen. Der Datenbankserver des ISW stellt den Fernzugriff auf die von ihm gehosteten Datenbanksysteme als einen Dienst bereit und die Client-Anwendungen greifen darauf zu, um auf diese Weise Daten in eine Datenbank zu laden, sie zu ändern, abzufragen oder verschiedene Analysen mit ihnen durchzuführen. Die Client-Anwendungen dienen darüber hinaus der Einrichtung von Warehousing-Anwendungen für reale Geschäftsdaten, die auf dem Anwendungsserver ausgeführt werden.

Diese komponentenbasierte Architektur ermöglicht es, bei großen, produktiv eingesetzten Rechnersystemen die Client- und Serverkomponenten auf getrennten, jedoch untereinander vernetzten Rechnern zu betreiben. Dies ist notwendig, sobald mehrere Personen mit jeweils einem eigenen Client auf das Datenbanksystem zugreifen wollen.

**Intelligent Miner als Data Mining-Komponente**

Zur Durchführung einer Datenanalyse mittels Data-Mining-Verfahren stellt InfoSphere Warehouse auf dem Datenbankserver eine Komponente namens *Intelligent Miner* bereit (in Abbildung 2-4 orange markiert). Der Intelligent Miner besteht aus einer Funktionsbibliothek, die bei der Installation des InfoSphere Warehouse bereitgestellt wird. Diese Funktionsbibliothek enthält eine Sammlung von Programmfunktionen in denen die Data Mining-Verfahren des Intelligent Miner implementiert sind und auf die vom Datenbankmanagementsystem aus über zwei verschiedene APIs[7] zugegriffen werden kann, um die Verfahren in der jeweiligen Datenbank („in-database“) auf die zu analysierenden Daten anzuwenden.

Bei den Schnittstellen zur Funktionsbibliothek handelt es sich zum einen um das standardisierte SQL/MM[8]-API und zum anderen um die so genannten Easy-Mining-Prozeduren.

Das **SQL/MM-API** basiert auf dem Standard SQL:1999[9], welcher die SQL-Sprache um objektorientierte Funktionalitäten erweitert [21]. Dieser SQL-Standard führt ein Konzept mit speziellen benutzerdefinierten Datentypen (engl. user-defined types, UDT), benutzerdefinierten Methoden (engl. user-defined methods, UDM) und benutzerdefinierten Funktionen (engl. user-defined functions, UDF) ein, damit Objekte in den Spalten einer Tabelle der (objektrelationalen) Datenbank abgelegt und deren Attribute abgefragt oder geändert werden können. Beim Intelligent Miner werden diese Datentypen, Funktionen und Methoden mit Hilfe einer ausführbaren Datei (engl. executable) in einem speziellen Bereich der betreffenden DB2-Datenbank eingerichtet[10], um diese für die Data-Mining-Analyse vorzubereiten.

Die *benutzerdefinierten Datentypen* können unterteilt werden in den einfachen Typ (engl. distinct UDT) und den strukturierten Typ (engl. structured UDT). Sie unterscheiden sich dadurch, dass einfache Typen lediglich eine Wiederverwendung eines einzelnen, bereits definierten Datentyps unter einem neuen (prägnanten) Namen darstellen [21] und strukturierte Typen eine formale Struktur zur Kapselung von Objektattributen und zur Zuweisung der sich auf das Objekt beziehenden *benutzerdefinierten Methoden*. Die Attribute werden dabei mit einem zuvor definierten, einfachen Datentyp deklariert und die benutzerdefinierten Methoden dienen dem externen Zugriff auf diese gekapselten Attribute, um sie mit einem Wert zu belegen, ihren Wert abzufragen oder zu ändern. Strukturierte UDTs, beziehungsweise deren Instanzen, werden im Intelligent Miner eingesetzt zur Speicherung der verschiedenen Einstellungen im Rahmen der Bildung und Überprüfung von Data-Mining-Modellen, wobei die Einstellungen den Instanzen mittels der zugehörigen UDMs übergeben werden. Das Pendant zu UDMs bilden die *benutzerdefinierte Funktionen*, mit deren Hilfe Objekte eines einfachen Datentyps erstellt, deren Werte abgefragt und weiterverarbeitet oder geändert werden können. Beim Intelligent Miner dienen die Instanzen von einfachen UDTs der Aufnahme und Speicherung der erstellten Data-Mining-Modelle [22]. Mit Hilfe der UDFs kann dann auf die Modelle zugegriffen und deren Eigenschaften, wie die Modellqualität, abgefragt werden.

[7] application programming interface, Schnittstelle einer Software oder Funktionsbibliothek, über welche die enthaltenen Funktionalitäten zur Weiterverwendung anderen Anwendungen zur Verfügung gestellt werden

[8] SQL Multimedia and Application Packages (Part 6: Data mining), standardisiert in ISO/IEC 13249-6:2006 [20]

[9] SQL = Structured Query Language, eine Standard-Abfragesprache mit Befehlen zum Erstellen, Ändern, Löschen und Verwalten von Datenbankinhalten

[10] Beispiele für die Definition von benutzerdefinierten Typen, Funktionen und Methoden gibt [21]

Zusätzlich zu UDMs und UDFs existieren beim Intelligent Miner *Gespeicherte Prozeduren* (engl. stored procedures), die in allgemeiner Form im Standard SQL92 beschrieben sind [21]. Mit ihnen können häufig verwendete Abfolgen von einzelnen Befehlen verschiedener Sprachen (z.B. SQL, Java, C++) in einer Prozedur vordefiniert und anhand von deren Aufruf auf einem Client (mittels CALL oder EXECUTE) über dessen Verbindung mit einer bestimmten Datenbank ausgeführt werden. Mit Gespeicherten Prozeduren kann beim Intelligent Miner die Bildung und Überprüfung der Data-Mining-Modelle durchgeführt sowie der gesamte Data-Mining-Prozess durch das Setzen und Ändern von Umgebungsvariablen gesteuert werden. Sie werden analog den UDTs, UDFs und UDMs zur Durchführung des Data Mining in der betreffenden Datenbank eingerichtet.

Auch die **Easy-Mining-Prozeduren** gehören dem Typ der Gespeicherten Prozeduren an. Sie vereinfachen den Datenanalyseprozess für unerfahrene Mining-Anwender, indem sie für verschiedene, häufig verwendete Data-Mining-Aufgaben, wie die Prognose des Zustandes oder Verhaltens von Objekten oder die Auffindung von Abhängigkeiten zwischen den Attributen von Objekten, die soeben vorgestellten Funktionen und Methoden des SQL/MM-API sowie deren Parameter vordefinieren und somit die komplexen API-Funktionen vor dem Mining-Anwender verbergen.

#### Design Studio zur grafischen Unterstützung

Es existieren verschiedene Möglichkeiten, um die SQL/MM-API-Funktionen oder die Easy-Mining-Prozeduren zur Durchführung des Data Mining in der jeweiligen Datenbank aufzurufen. Zum Beispiel können die entsprechenden Befehle textbasiert über ein Kommandozeilenwerkzeug abgesetzt werden, welches zusammen mit DB2 installiert wurde.

Ebenso möglich ist die Durchführung des gesamten Data-Mining-Analyseprozesses mittels Unterstützung durch eine graphische Entwicklungsumgebung, das *Design Studio*. Das Design Studio (in Abbildung 2-4 violett markiert) ist Teil der Client-Komponentengruppe des InfoSphere Warehouse und unterstützt neben den Data-Mining-Verfahren auch andere Datenanalysetechniken, wie OLAP. Es verfügt über eine Eclipse[11]-basierte graphische Benutzeroberfläche, die im Anhang auf Seite 82 dargestellt ist. Diese Oberfläche besteht aus einem Arbeitsbereich zur Durchführung des Data-Mining-Projektes, einer Palette mit verschiedenen Werkzeugen („Operatoren“) zur Ausführung der Funktionen und Methoden der SQL/MM-API sowie der Easy-Mining-Prozeduren, einem Projekt-Explorer zur Speicherung der erstellten Data-Mining-Projektinhalte, einem Datenbank-Explorer zur Verwaltung der Datenbankverbindungen und weiteren Fenstern, zum Beispiel zur Anzeige der von den Data-Mining-Verfahren benötigten Einstellungen.

Eine dritte Möglichkeit ist der Aufruf der Funktionen und Prozeduren mittels SQL-Anweisungen aus einer eigenen Software-Anwendung heraus, zum Beispiel einer Java-Anwendung unter Einsatz von JDBC[12]. Das Design Studio stellt dazu eine Codegenerierungskomponente [23] zur Verfügung, welche die für die Funktionsaufrufe benötigten SQL-Anweisungen in einem SQL-Skript ausgibt, dessen Inhalt in die eigene Anwendung eingebettet werden kann.

[11] Integrierte Entwicklungsumgebung (IDE) zur Unterstützung der Entwicklung von Anwendungen

[12] Java Database Connectivity, ein standardisiertes API für den SQL-basierten Zugriff auf verschiedene Typen von Datenbanken und tabellarischen Datenquellen aus einer Java-Anwendung heraus

# 3 Aspekte des Software-Qualitätsmanagements

Für eine langfristige Bindung von Kunden an Produkte im Software-Bereich ist, abgesehen von regionalen und emotionalen Kaufkriterien, die hohe Qualität einer Software heute meist von entscheidender Bedeutung. Auch für das Software-Unternehmen selbst spielt die Qualität der eigenen Produkte eine beachtenswerte Rolle, da sie einen unmittelbaren Einfluss auf die nicht zu unterschätzenden Kosten einer späteren Maintenance[13] der verkauften Produkte nimmt. Daher und aufgrund einer Gefährdung des Unternehmensgeschäfts, verursacht durch den Verkauf von Produkten schlechter Qualität und den damit verbundenen potentiellen Image-Schaden, sollte sich ein Unternehmen intensiv mit dem Thema Software-Qualität und deren Sicherung auseinander setzen.

## 3.1 Software-Qualität

Die Qualität von Software-Produkten kann entsprechend dem Standard ISO 9126 definiert werden als *„Gesamtheit der Merkmale und Merkmalswerte eines Software-Produkts, die sich auf dessen Eignung beziehen, festgelegte oder vorausgesetzte Erfordernisse zu erfüllen“* [24].

Um diesen abstrakten Begriff der Software-Qualität in der Praxis zu operationalisieren, wurden in der Vergangenheit verschiedene Qualitätsmodelle entwickelt, die grundsätzlich aus einem Baum oder Netz abgeleiteter Begriffe und Unterbegriffe bestehen [25, S.10], sich jedoch in Bezug auf deren Anordnung und zum Teil in den Bezeichnungen für Begriffe mit inhaltlich gleicher Bedeutung unterscheiden.

Die Modelle sind generell aus mehreren Begriffsebenen aufgebaut, zum Beispiel im Falle von FCM[14]-Modellen aus vier Ebenen. FCM-Modelle enthalten in der höchsten Ebene die, aus den Qualitätsanforderungen (*requirements*) abgeleiteten, notwendigen Qualitätsmerkmale (*factors*) einer Software. In der darunter liegenden Ebene werden Software-Attribute (*criterion*) festgelegt, die aufzeigen wie die erforderlichen Merkmale der Software konkret umgesetzt werden. In den Blättern des Modellbaumes werden Indikatoren (*metrics*) definiert, welche die darüber liegenden Attribute messbar machen [25, S.10]. Ein Beispiel dazu:

| | |
|---|---|
| Anforderung | Das System soll gut wartbar sein. |
| Merkmal | Lesbarkeit der Software |
| Attribut | gute Kommentierung |
| Indikator | Anzahl Kommentarzeilen pro 100 Zeilen Programmcode |

**Tabelle 3-1: Beispiel für einen Ausschnitt aus einem FCM-Modell, entnommen aus [25, S.10]**

[13] Maintenance bezieht sich im Bereich von Software-Produkten auf notwendige Fehlerbehebungen, Aktualisierungen und Verbesserungsmaßnahmen nach Auslieferung der Software

[14] factors criterion metrics

Ein Beispiel für ein Qualitätsmodell ist das standardisierte Modell nach DIN ISO 9126 [24], welches für Software die folgenden Qualitätsanforderungen und Qualitätsmerkmale definiert:

| Anforderung | Merkmale |
|---|---|
| Funktionalität | Angemessenheit, Richtigkeit, Interoperabilität, Ordnungsmäßigkeit, Sicherheit |
| Zuverlässigkeit | Reife, Fehlertoleranz, Wiederherstellbarkeit |
| Benutzbarkeit | Verständlichkeit, Erlernbarkeit, Bedienbarkeit |
| Effizienz | Zeitverhalten, Verbrauchsverhalten |
| Änderbarkeit | Analysierbarkeit, Modifizierbarkeit, Stabilität, Prüfbarkeit |
| Übertragbarkeit | Anpassbarkeit, Installierbarkeit, Konformität, Austauschbarkeit |

**Tabelle 3-2: Qualitätsanforderungen und Qualitätsmerkmale nach DIN ISO 9126 [24]**

Die funktionale Anforderung beschreibt, was ein Software oder Software-Teile im Sinne des Kunden oder Anwenders leisten sollen. Die Umsetzung dieser geforderten Funktionalität entscheidet darüber, ob die Software überhaupt für ihren Zweck einsetzbar ist [26, S.69]. Die anderen, nicht-funktionalen Anforderungen bestimmen Attribute des funktionalen Verhaltens, das heißt in welchem qualitativen Maße die Software ihre Funktion erbringen soll. Die Zuverlässigkeit, Benutzbarkeit und Effizienz der Software haben einen starken, unmittelbaren Einfluss auf die Zufriedenheit des Kunden, während die Kriterien Änderbarkeit und Übertragbarkeit in erster Linie wichtig für den Hersteller der Software sind, da hiervon die späteren Wartungskosten abhängen. Jedoch können letztere die Kundenzufriedenheit indirekt beeinflussen, da sie zum Beispiel davon abhängt, wie schnell und einfach sich eine Software an geänderte Anforderungen des Kunden anpassen lässt [26, S.71f].

Die Anforderungen an eine Software oder Software-Teile müssen vor deren Entwicklung in einer Spezifikation[15] schriftlich festgehalten werden, um den Entwicklern hinsichtlich der zu realisierenden Software-Merkmale eine Orientierung zu geben.

## 3.2 Software-Fehler

Abgesehen von einer Vorgabe der zu realisierenden Software-Merkmale, kann anhand der Spezifikation später nachgewiesen werden, ob die Merkmale der entwickelten Software oder Software-Teile den gestellten Anforderungen entsprechen. Im Falle der Nichterfüllung einer Qualitätsanforderung liegt ein *Fehler* vor [26]. Da der Fehlerbegriff in der Literatur nicht einheitlich gehandhabt wird und für verschiedene Sachverhalte stehen kann, wird dessen Bedeutung nachfolgend geklärt.

[15] formale Beschreibung der geforderten Merkmale einer zu erbringenden Sache oder Dienstleistung, in diesem Fall der Software, zur Herstellung eines einheitlichen Verständnisses bei allen Beteiligten

**Terminologie**

Unterschieden werden kann die *Fehlerursache* (engl. error), der *Fehler* an sich (engl. fault) und das *Fehlverhalten* (engl. failure) eines Systems [27]. Unter einer Fehlerursache wird eine menschliche Handlung verstanden, die einen Fehler zur Folge hat. Beispiele dafür sind „Schnitzer“ (z.B. Tippfehler) oder Irrtümer (Denkfehler) eines Programmierers, wobei letztere aufgrund eines unzureichenden Hintergrundwissens („überindividuelle Irrtümer“) oder bei unzureichender Ausbildung („individuelle Irrtümer“) auftreten können [28]. Bei Hardware-Systemen kann auch eine Veränderung (Störung) in deren Umfeld einen Fehler verursachen [26]. Der Fehler, auch bezeichnet als Fehlerzustand oder Defekt (engl. defect), steht für einen *„Merkmalswert, der die vorgegebenen Forderungen nicht erfüllt“* [29], zum Beispiel eine fehlerhafte Anweisung im Programmcode oder eine nicht mit der Kundenanforderung übereinstimmende Beschreibung in der Spezifikation. Fehler befinden sich grundsätzlich seit dem Zeitpunkt der Entwicklung in einer Software und entstehen nicht danach durch Alterung oder Verschleiß, wie dies bei Defekten in Hardware-Systemen der Fall sein kann. Kommt eine fehlerhafte Programmzeile bei der Benutzung der Software durch einen Tester oder den späteren Anwender zur Ausführung, macht sich der Fehler in Form eines Fehlverhaltens (z.B. eines Programmabsturzes), auch als Fehlfunktion oder Fehlerwirkung bezeichnet, bemerkbar. Ein Fehlverhalten bezeichnet daher die *„Diskrepanz zwischen dem berechneten, beobachteten oder gemessenen [(IST-) Wert oder Verhalten] und dem spezifizierten oder theoretisch korrekten [(SOLL-) Wert oder Verhalten]“* [25, S.41]. Zu beachten ist, dass ein Fehler nicht unmittelbar zu einer Fehlerwirkung führen muss. Zum Beispiel kann er durch andere Fehler maskiert (verdeckt) oder kompensiert werden und erst nach deren Korrektur in Erscheinung treten. In einem solchen Fall hat die Korrektur einen so genannten *Seiteneffekt* zur Folge [26]. Abgesehen von der beschriebenen Fehlermaskierung kann es vorkommen, dass die Fehlerwirkung eines Defektes gar nicht oder erst zu einem späteren Zeitpunkt und in einem anderen Programmteil eintritt, sodass diesem Programmteil möglicherweise zu Unrecht ein Defekt zugeschrieben wird. Den ursprünglichen Defekt als Ursache für eine Fehlerwirkung zu lokalisieren und zu korrigieren, ist die Aufgabe eines Software-Entwicklers und wird als *Debugging* bezeichnet [26].

Auch wenn eine Software den gesetzten Qualitätsanforderungen gerecht wird, ist es darüber hinaus möglich, dass sie eine nicht explizit kommunizierte, jedoch berechtigte Erwartung nicht erfüllt oder eine gestellte Anforderung nicht in angemessenem Maße erfüllt [26]. In diesem Fall spricht man von einem *Mangel*.

Zur besseren Veranschaulichung der grundlegenden Begriffe Fehlerursache, Fehler und Fehlerwirkung, kann man diese beispielhaft auf den Automobilbereich anwenden. Ein konkretes Beispiel wäre eine undichte Kraftstoffleitung im Auto, welche die an sie gestellte Qualitätsanforderung, den verlustfreien Transport von Kraftstoff, nicht (mehr) erfüllt. Der Austritt von Kraftstoff, beziehungsweise der damit verbundene erhöhte Kraftstoffverbrauch des Autos, stellt in diesem Fall die wahrnehmbare Fehlerwirkung dar. Der Grund für den Kraftstoffaustritt ist ein Fehler oder Defekt an der Leitung, zum Beispiel ein Leitungsmaterial (Polyamid), welches in Kombination mit den Inhaltsstoffen (Chlor) des transportierten Kraftstoffs nach einem gewissen Zeitraum spröde wird [30, 31] und Risse ausbildet. Die Fehlerursache kann ein Irrtum eines Entwicklers des Autos sein, der diesen Zusammenhang nicht bedachte.

### Schwachstelle

Eine einheitliche Definition für eine *Schwachstelle* konnte in der Literatur nicht gefunden werden, da dieser Begriff je nach Fachgebiet und Problembereich eine eigene Bedeutung erhält. Zum Beispiel wird der Begriff im Bereich der Betriebswirtschaft im Rahmen der Schwachstellenanalyse verwendet als strategisch wichtige Eigenschaft eines Unternehmens, die im Vergleich zur Konkurrenz besonders schwach ausgebildet ist und sich zum Nachteil für das Unternehmen in einem bestimmten strategischen Geschäftsfeld entwickeln kann. Dagegen stellt eine Schwachstelle im Software-technischen Bereich einen Programmabschnitt einer Software dar, der besonders fehlerbehaftet ist und damit nach verschiedenen Gesichtspunkten, wie der Datensicherheit und der Zufriedenheit des Anwenders, ein Risiko in sich birgt. Aus diesem Grund ist es von besonderer Wichtigkeit Schwachstellen in einer Software zu identifizieren und zu beseitigen oder idealerweise im Vorfeld bereits zu vermeiden. Die Untersuchung der Möglichkeiten zur Identifizierung von Schwachstellen mit Hilfe des Data Mining stellt eines der beiden Hauptziele dieser Arbeit dar.

Die einzelnen Bedeutungsräume zusammenfassend, sind Schwachstellen diejenigen abgrenzbaren Bereiche eines Systems, die eine bemerkenswert geringe Qualität aufweisen. Sie sind die Folge von in hohem Maße oder wiederholt an dieser Stelle auftretenden Fehlern oder Mängeln, die zu einem fehlerhaften und unerwünschten, zum Teil (in verschiedener Hinsicht) kritischen Zustand oder Verhalten des Systems führen.

### Klassifikation von Fehlern

Fehler können nach vielerlei Kriterien klassifiziert, also bestimmten Arten zugeordnet werden. Im Folgenden werden die verschiedenen Fehlerarten nach den, für die weiteren Betrachtungen in dieser Arbeit relevanten Kriterien vorgestellt.

Eine erste grundlegende Einteilung von Fehlern kann nach ihrem Auftrittsort in *Hardware-Fehler* und *Software-Fehler* erfolgen [32]. Hardware-Fehler treten in physikalischen Hardware-Systemen auf und haben deren Abweichung vom spezifizierten Normverhalten zur Folge [32], während Software-Fehler im Bereich von Software-Systemen zu finden sind. Im Hauptteil dieser Arbeit werden ausschließlich Software-Fehler betrachtet, da das vorgestellte Verfahren im Bereich der Qualitätssicherung von Software angesiedelt ist.

Ein zweites Kriterium zur Klassifikation von Fehlern ist der Grad des Determinismus bei ihrem Auftreten. Diesem Kriterium entsprechend, können Fehler in zwei Kategorien eingeordnet werden, in die der *systematischen* (deterministischen) und die der *zufälligen* (stochastischen) Fehler [33]. Systematische Fehler entstehen in einem System grundsätzlich zu einem bestimmten Zeitpunkt, zum Beispiel durch eine falsche Spezifikation oder durch Irrtümer beim Entwurf, treten jedoch erst dann in Erscheinung, wenn bestimmte Umstände, wie die Ausführung des defekten Systemteils oder Materialverschleiß, eingetreten sind [34]. Sind die Bedingungen für das Sichtbarwerden eines Fehlers bekannt, so ist das Fehlverhalten reproduzierbar und dadurch der Fehler in den meisten Fällen lokalisierbar und korrigierbar [33]. Systematische Fehler sind sowohl in Hardware-Systemen als auch in Software-Systemen zu finden. Im Gegensatz zu systematischen, sind zufällige Fehler nicht von Beginn an in einem System vorhanden, sondern können zu jedem beliebigen, nicht vorhersehbaren Zeitpunkt, in jeder beliebigen Reihenfolge und

an jeder beliebigen Stelle in einem System auftreten [34]. Sie werden durch unkontrollierbare, zufällige Einflüsse hervorgerufen, die im Rahmen der aktuellen wissenschaftlichen Kenntnisse und technischen Möglichkeiten nicht reproduziert werden können [33]. Ein Beispiel dafür sind zufällige Bitfehler in einem Funksignal, welche durch atmosphärische Störungen bei der Funkübertragung hervorgerufen werden. Zufällige Fehler treten ausschließlich in Hardware auf, können jedoch eine Fehlerwirkung in einer Software, die auf dieser Hardware basiert, zur Folge haben.

Ein drittes Kriterium zur Einteilung von Fehlern in Klassen ist deren zeitliches Verhalten, nach dem *permanente*, *transiente*[16] und *intermittierende*[17] Fehler unterschieden werden können [35]. Permanente Fehler bleiben nach ihrer Entstehung dauerhaft bestehen, wie eine unterbrochene Leiterbahn auf einer Platine.

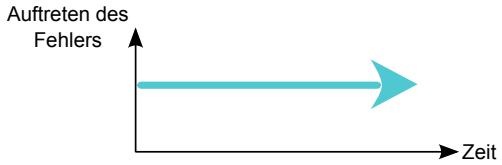



**Abbildung 3-1: Permanentes Auftreten eines Fehlers**

Dagegen ist ein transienter Fehler zeitlich nur vorübergehend existent, das heißt solange seine Ursache besteht. Auch hier trifft das Beispiel mit dem, durch eine kurzzeitige atmosphärische Störung verursachten, Fehler in einem Funksignal zu.

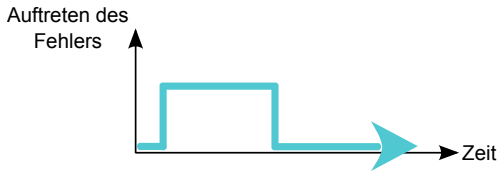



**Abbildung 3-2: Transientes Auftreten eines Fehlers**

Intermittierende Fehler sind diejenigen, die in unregelmäßigen Abständen immer wieder (sporadisch) auftreten, zum Beispiel Wackelkontakte, die durch gebrochenen Lötstellen auf einer Platine verursacht werden.

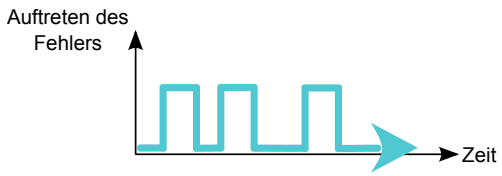



**Abbildung 3-3: Intermittierendes Auftreten eines Fehlers**

Im Gegensatz zu einigen Hardware-Fehlern sind Software-Fehler nach ihrer Entstehung stets beständig, sie verschwinden nicht zusammen mit ihrer Ursache, sondern können lediglich durch

[16] lat. transire = vorbeigehen
[17] lat. intermittere = unterbrechen, aussetzen

einen Eingriff von außen in Form einer Korrektur beseitigt werden. Software-Fehler werden daher den permanenten Fehlern zugeordnet.

Abgesehen von den Software-Fehlern an sich, die stets einen permanenten Charakter haben, kann eine Fehlerwirkung über den zeitlichen Verlauf des Software-Lebenszyklus hinweg permanent, transient oder intermittierend auftreten. Bei einer permanenten Fehlfunktion liegen stets die gleichen Laufzeitbedingungen vor und der Fehler wurde nicht korrigiert, sodass bei der Ausführung eines fehlerhaften Programmteils auch das resultierende Fehlverhalten dauerhaft beobachtet werden kann. Eine transiente Fehlfunktion ist dann gegeben, wenn sich ein Fehler nur vorübergehend bemerkbar macht, da er entweder nach seinem Sichtbarwerden behoben und nicht erneut verursacht wurde oder die Laufzeitbedingungen, die den Fehler zu Tage gefördert haben, nicht noch einmal eingetreten sind. Kann hingegen ein Fehlverhalten eines Programmteils von Zeit zu Zeit wieder erneut festgestellt werden, zum Beispiel verursacht durch wiederholt auftretende Seiteneffekte bei der Beseitigung von Fehlern, so kann von einer intermittierenden Fehlfunktion gesprochen werden.

#### Möglichkeiten der Fehlerprävention

Betrachtet man die Kosten von Software-Entwicklungen über die verschiedenen Phasen des Software-Lebenszyklus hinweg, so wird man feststellen können, dass in der Wartungsphase eines sich bereits im Markt befindenden Software-Produktes die meisten Kosten entstehen [36]. Die Ursache liegt bei Fehlern, die erst vom Anwender entdeckt werden, denn in dieser späten Phase des Software-Lebenszyklus sind Fehler oft nur mit großem Aufwand und, im Vergleich zur frühzeitigen Fehlerentdeckung, mit exponentiell gestiegenen Kosten zu beheben. Um diese hohen Kosten für eine späte Fehlerkorrektur zu reduzieren, sollte die Anwendung konstruktiver Maßnahmen zur frühzeitigen Fehlervermeidung das primäre Ziel sein [37]. Im folgenden Verlauf werden daher zwei Möglichkeiten zur frühzeitigen Prävention von Fehlern kurz vorgestellt.

Die **Fehlermöglichkeits- und Einflussanalyse (FMEA)** [38] (engl. Failure Mode and Effects Analysis) ist eine Methode zur systematischen Identifikation von potenziellen Fehlern in einem System und zur Beurteilung der Folgen, die ein solcher Fehler hätte. Die Folgen können dabei von einem geringen Schaden bis hin zur Lebensgefahr (zum Beispiel bei medizinischen Geräten) reichen. Ursprünglich wurde die FMEA für Hardware-Systeme entwickelt, kann aber ebenfalls zur Fehlerprävention bei Software angewendet werden. Das Ziel der Anwendung dieser konstruktiven Methode ist neben der Feststellung potenzieller Fehlerquellen in einer Software auch die Festlegung von Maßnahmen zur Beseitigung dieser Fehlerquellen. Anhand einer Dokumentation der aus der FMEA gewonnenen Erkenntnisse, ist es außerdem möglich in zukünftigen Software-Projekten diesen Fehlermöglichkeiten von Vornherein vorzubeugen. Bei der FMEA können allgemein zwei Arten unterschieden werden, die Produkt-FMEA und die Prozess-FMEA. Die Produkt-FMEA beschäftigt sich mit der Architektur einer Software, indem für jede Komponente untersucht wird, in welcher Form deren Fehlermöglichkeiten die Funktionsfähigkeit der Software beeinträchtigen können. Dagegen dient die Prozess-FMEA der Analyse der einzelnen Prozesse, die an der gesamten Entwicklung, der Benutzung (zum Beispiel bei Systemüberlastung oder Fehlbenutzung) und der Wartung der Software beteiligt sind, um die Auswirkungen von in den Prozessen enthaltenen Fehlern auf die Funktionsfähigkeit der Software zu ermitteln. Beide Arten der FMEA laufen, trotz ihrer unterschiedlichen Anwendungsgebiete, nach dem gleichen Schema

ab. Eine FMEA beginnt mit der Eingrenzung des Systems und der Software-Komponenten, die zu untersuchen sind und fährt fort mit der Identifikation der Fehlermöglichkeiten der Software-Komponenten oder der Prozesse. Der nächste Schritt umfasst die Feststellung der Auswirkungen der gefundenen Fehlermöglichkeiten sowie deren Ursachen. Bei dieser Ausarbeitung können Ursache-Wirkungsdiagramme Hilfestellung leisten. Im abschließenden Schritt werden die Maßnahmen zur Beseitigung der Probleme und damit zur Reduzierung des Ausfallrisikos der Software festgelegt und anschließend ausgeführt sowie eine vollständige Dokumentation des Analyseprozesses erstellt, wobei für letztere vorgefertigte Arbeitsblätter existieren.

Die zweite Möglichkeit zur Fehlerprävention stellt ein persönliches **Fehlerbuch** [28,39] dar. Dabei handelt es sich um einen Katalog von Programmierregeln, der von jedem Entwickler angefertigt und ständig erweitert werden sollte, da er Erkenntnisse und Lösungen zu typischen, während der Entwicklung von Software aufgetretenen Problemen und Fehlern enthält. Es sollten darin Fehler eingetragen werden, wenn die Suche nach ihnen lange gedauert hat, wenn die vom Fehler verursachten Kosten hoch waren oder wenn der Fehler lange Zeit unentdeckt geblieben ist. Mit Hilfe der im Fehlerbuch beschriebenen Gegenmaßnahmen, die das Ergebnis früherer Fehleranalysen sind, können neu auftretende, ähnliche Probleme schneller gelöst und bereits einmal gemachte Fehler in Zukunft vermieden werden.

## 3.3 Software-Test

Wie kann man nun im Rahmen der Qualitätssicherung überprüfen, ob und in welchem Maße das Software-Produkt den gestellten Qualitätsanforderungen gerecht wird?

#### Terminologie

Zu diesem Zweck wird die zu prüfende Software, allgemein als *Testobjekt* bezeichnet, verschiedenen *Software-Tests* unterzogen. Diese erfolgen entweder in Form von Analysen der zugehörigen Entwicklungsdokumente oder anhand einer stichprobenartigen Ausführung [26] des Testobjektes, wobei dessen Eigenschaften mit den in der Spezifikation geforderten verglichen werden.

Bezogen auf eine zu testende Eigenschaft des Testobjektes wird jeweils ein so genannter *Testfall (engl. test case)* ausgeführt. Ein Testfall wird in einer formalen Struktur, zum Beispiel einer Datei, definiert und besteht aus einem Set von *Vorbedingungen* (Herstellung eines bestimmten Systemzustandes vor Ausführung des Testfalls), *Eingaben* (Eingabedaten für das Testobjekt und Angabe der im Test durchzuführenden Aktionen) und *erwarteten Ergebnissen* (Reaktionen des Testobjektes auf die Eingabedaten in Form von Ausgabewerten oder Meldungen, neuer Systemzustand) [40].

Ein Software-Test kann im Allgemeinen nur das Vorhandensein von Fehlern nachweisen und nicht die Fehlerfreiheit einer Software, denn dazu müsste ein Programm unter Berücksichtigung aller Kombinationen von möglichen Situationen, Randbedingungen und Eingaben getestet werden [26]. Dies würde zu einem, mit der Anzahl der Parameter des Tests exponentiell steigenden Testaufwand führen - bezeichnet als *Testfallexplosion* [26]. Aufgrund beschränkter Ressourcen, wie Zeit und Personal, ist ein solcher vollständiger Test in der Praxis, außer vielleicht in wenigen

Ausnahmefällen, nicht durchführbar [26]. Um dennoch mit den für die Tests vorhandenen Ressourcen so zu viele Fehler wie möglich aufdecken zu können und unnötige Tests, die nicht zur Entdeckung neuer Fehler führen, zu vermeiden, müssen die Tests strukturiert und systematisch durchgeführt [26] und dazu im Vorfeld durchdacht und geplant werden. Der gesamte Prozess des Testens erfordert daher einen strukturierten Ablauf.

#### Ablauf des Testprozesses

Der Testprozess beginnt laut Spillner und Linz [26] mit einer Planung der Aufgaben und Ziele des Tests, der benötigten Ressourcen und Zeit sowie der zu verwendenden Hilfsmittel, festgehalten in einem Testkonzept. Ein zweiter Arbeitsschritt stellt die Steuerung der Testaktivitäten mit Aktualisierung des Testplans dar. Es folgen die Analyse der Spezifikation und das daraus resultierende Testdesign mit Definition des Testinhalts und des SOLL-Verhaltens der Software. Während der sich daran anschließenden Phase der Testdurchführung wird das zu prüfende Testobjekt unter definierten Rahmenbedingungen ausgeführt und das dabei beobachtete IST-Verhalten mit dem zuvor spezifizierten SOLL-Verhalten verglichen. Nach der Testdurchführung erfolgt die Dokumentation und Auswertung der erhaltenen Ergebnisse sowie die Anfertigung eines abschließenden Testberichts. Anzumerken ist, dass sich die genannten Aufgaben in der Praxis überschneiden oder parallel durchgeführt werden können.

#### Einordnung des Testens in den Gesamtprozess der Entwicklung

Software-Tests werden, je nach angewendetem Software-Entwicklungsmodell, an unterschiedlichen Zeitpunkten und mit unterschiedlichem Umfang im Entwicklungsprozess durchgeführt [26]. Entwicklungsmodelle werden verwendet, um den Gesamtprozess der Software-Entwicklung zu strukturieren, ihn dadurch planbar und steuerbar zu machen. Beispiele für Modelle dieser Art sind [26]:

- Wasserfallmodell [41]
- allgemeines V-Modell [42]
- Vorgehensmodell des Bundes und der Länder [43]
- Spiralmodell
- inkrementelle und agile Methoden (z.B. SCRUM)
- leichtgewichtige Methoden (z.B. Extreme Programming [44])

Auf das Wasserfallmodell und die agilen Methoden soll nachfolgend detaillierter eingegangen werden.

Das *Wasserfallmodell* war das erste grundlegende Modell [26]. Es sieht einen fest definierten, sequentiellen Ablauf der Entwicklungsaktivitäten vor, bei dem jede der Entwicklungsphasen erst nach Abschluss der vorherigen Phase begonnen wird und nur zwischen angrenzenden Phasen Rückkopplungen[18] vorgenommen werden. Das Testen wird bei diesem Modell einmalig und erst

[18] Wiederholung oder Nachholen von Tätigkeiten der vorhergehenden Phase, wie eine Überarbeitung des Programmcodes nach Feststellung von Fehlern

am Projektende durchgeführt, nachdem alle anderen Entwicklungsschritte abgeschlossen sind und bevor die Auslieferung der Software an den Anwender erfolgt [26].

Davon unterscheiden sich die modernen Entwicklungsmethoden, wie die des *Agile Software Development*, die für eine schlankere und flexiblere Gestaltung des Entwicklungsprozesses sorgen sollen. Umgesetzt wird dies zum Beispiel durch iterativ aufgebaute Entwicklungsphasen mit Rückkopplungen zwischen allen Phasen und einer flexiblen Dauer. Bei diesem Ansatz wird der Test am Ende jeder Iteration durchgeführt und Fehler sofort in der darauffolgenden Iteration im Zuge der (Weiter-)Entwicklung behoben. Um die Beseitigung der Fehler zu verifizieren und unerwünschte Seiteneffekte der Korrekturen aufzudecken, werden in der Testphase am Ende jeder Iteration auch die Tests der vorausgegangenen Iterationen wiederholt. Auf diese Weise kann die Software kontinuierlich von gefundenen Fehlern befreit und die Software-Qualität frühzeitig positiv beeinflusst werden, was einen wesentlichen Vorteil gegenüber dem Wasserfallmodell und anderen Entwicklungsmodellen mit nur einer Testphase am Projektende darstellt.

### Testebenen

Die Testaktivitäten können, hinsichtlich ihres Abstraktionsgrades und ihres Ausführungszeitpunktes im Entwicklungsprozess, auf unterschiedlichen Ebenen ablaufen. Die zu Beginn feingranular vorliegenden Testobjekte werden im Zeitverlauf Stück für Stück vergrößert, bis schließlich das gesamte Software-System dem Test unterzogen wird. Die folgenden vier Ebenen können unterschieden werden [26, 27]:

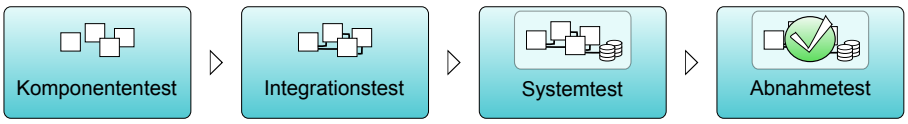



**Abbildung 3-4: Testebenen, in Anlehnung an [26]**

Bei einem **Komponententest** werden die kleinsten Einheiten einer Software, die Komponenten (je nach Programmiersprache Module, Units oder Klassen genannt), direkt nach deren Implementierung (meist vom Entwickler selbst) auf das Vorhandensein von Fehlern geprüft. Die isolierte Prüfung der einzelnen Komponenten sorgt dafür, dass aufgedeckte Fehlerwirkungen aufgrund des Fehlens äußerer Einflüsse eindeutig der getesteten Komponente zugeordnet werden können.

Auch wenn die Komponenten im Einzelnen ihre Funktionsweise korrekt erfüllen, bedeutet das nicht, dass dies auch im Verbund miteinander der Fall ist. Um dies zu überprüfen, werden im **Integrationstest** einzelne bereits getestete und korrigierte Komponenten zu größeren Teilsystemen verbunden und deren Schnittstellen anhand der Interaktion mit anderen Komponenten auf Fehler überprüft.

Beim nachfolgenden **Systemtest** werden alle Komponenten zusammengeführt, um zu testen, ob das Gesamtsystem hinsichtlich seiner funktionalen und nicht-funktionalen Eigenschaften den Benutzeranforderungen (gemäß der Spezifikation) entspricht. Die Implementierung der Software steht auf dieser Ebene, im Gegensatz zu den vorherigen, eher im Hintergrund. Beim Systemtest

sollte die Ausführung des Testobjektes auf einem Testsystem erfolgen, welches dem späteren Produktiv-System[19] so ähnlich wie möglich ist, um eine umfassende Identifizierung von Fehlern zu garantieren.

Der **Abnahmetest** ist der letzte Test vor der Auslieferung und Inbetriebnahme der Software beim zukünftigen Benutzer beziehungsweise Auftraggeber. Es handelt sich dabei um einen Systemtest, der gewöhnlich von Repräsentanten des Auftraggebers federführend geleitet und durchgeführt wird, um das Software-System hinsichtlich seiner korrekten Funktionsweise und der nicht-funktionalen Anforderungen abschließend zu überprüfen und im Erfolgsfalle die Vertragserfüllung zu bestätigen.

#### Arten von Software-Testverfahren

Es ist grundsätzlich möglich eine Klassifikation von Tests in zwei Gruppen vorzunehmen, die *statischen* und die *dynamischen* Testverfahren. Unter statischen Testverfahren versteht man die Analyse all derjenigen Dokumente, die für die Erstellung einer Software notwendig sind, mit dem Ziel frühzeitig Fehler sowie Abweichungen von der, zu Projektbeginn festgelegten, Spezifikation und von eventuell einzuhaltenden Standards festzustellen [26]. Auf die statistischen Testverfahren soll jedoch im weiteren Verlauf nicht näher eingegangen werden, da sich diese Arbeit ausschließlich mit dynamischen Testverfahren beschäftigt.

Im Gegensatz zu statischen wird bei dynamische Testverfahren das Testobjekt mit Eingabedaten beschickt und zur Ausführung gebracht, um zu überprüfen ob dessen Verhalten mit den Qualitätsanforderungen übereinstimmt [26]. Für die Ausführung muss das Testobjekt stets in Form eines ablauffähigen Programms vorliegen, was im Komponententest und im Integrationstest noch nicht der Fall ist, weshalb das Testobjekt in einen Testrahmen (engl. testing framework) eingebettet werden muss. Ein Testrahmen besteht aus einem Platzhalter (engl. stub), der die noch nicht implementierte Ein- und Ausgabeschnittstelle des Testobjektes simuliert, und aus einem Testtreiber, der das Testobjekt von außen mit den Eingabedaten der Testfälle versorgt und dessen Ausgaben entgegen nimmt. Der Testrahmen kann entweder selbst entwickelt oder auf der Basis eines generischen Rahmens (z.B. des für Java verfügbaren JUnit [45]) erstellt werden, indem dieser an das Testobjekt angepasst wird.

Dynamische Software-Tests können im Allgemeinen in vier Kategorien unterteilt werden [26]:

- funktionaler Test oder Black-Box-Test
- nicht-funktionaler Test
- strukturorientierter Test oder White-Box-Test
- änderungsbezogener Test oder Regressionstest

Das Ziel eines **funktionalen Tests** [26] ist die Prüfung der Software auf ihre Übereinstimmung mit den, in der Spezifikation festgelegten, funktionalen Anforderungen[20]. Ob die Software die geforderten Funktionen erfüllt, wird ermittelt anhand des Vergleichs der Ausgabe eines Testob-

[19] System, auf dem eine Software vom Anwender im Rahmen der geschäftlichen Tätigkeit produktiv eingesetzt wird
[20] eine Beschreibung der Qualitätsanforderungen wurde bereits in Abschnitt 3.1 Software-Qualität gegeben

jektes - als Reaktion auf die Eingabedaten eines Testfalls - mit der spezifizierten und theoretisch korrekten Ausgabe, deren Wert ebenfalls im Testfall hinterlegt ist. Bei dem dazu verwendeten Testverfahren, dem Black-Box-Verfahren, wird die Reaktion des Testobjektes (System oder Software-Komponente) ausschließlich von einem außen liegenden Punkt (PoO - Point of Observation) betrachtet und das Testobjekt wird als eine Art schwarzer Kasten angesehen. Auch die Steuerung des Testobjekts anhand der Versorgung mit Eingabedaten erfolgt von einem außen liegenden Punkt (PoC - Point of Control). Da der innere Aufbau, also die Programmstruktur, des Testobjektes bei einem Black-Box-Test nicht bekannt ist, werden die Testfälle dafür aus der Spezifikation abgeleitet. Zur Testfallerstellung existieren verschiedene Verfahren, als ein Beispiel sei die Äquivalenzklassenbildung genannt. Bei dieser wird die mögliche Menge an Eingabedaten in Äquivalenzklassen geteilt und aus jeder Klasse nur ein Repräsentant im Test eingesetzt, da davon ausgegangen wird, dass das Testobjekt für alle Eingabedaten in der Äquivalenzklasse die gleiche Reaktion zeigt.

Der Grad der Erfüllung von nicht-funktionalen Qualitätsanforderungen, wie der Zuverlässigkeit und Effizienz, kann mit Hilfe des **nicht-funktionalen Tests** für ein Testobjekt nachgewiesen werden [26]. Für einen nicht-funktionalen Test kann zum Teil auf diejenigen Testfälle eines funktionalen Tests zurückgegriffen werden, die einen Querschnitt durch die Funktionalität des Gesamtsystems darstellen. Bei deren Ausführung kann dann die nicht-funktionale Größe gemessen werden. Beispiele für Tests dieser Art sind:

| Testart | Testziele |
|---|---|
| Performanztest | Verarbeitungsgeschwindigkeit & Antwortzeiten der Software |
| Lasttest | Software-Verhalten bei steigender Systemlast (viele Transaktionen) |
| Test der Zuverlässigkeit | Verhalten der Software im Dauerbetrieb |
| Test auf Robustheit | Toleranz gegenüber Fehlbedienung & HW-Ausfall, Fehlerbehandlung |
| Test der Benutzbarkeit | Erlernbarkeit, Angemessenheit der Bedienung, Verständlichkeit |

**Tabelle 3-3: Beispiele nicht-funktionaler Tests nach [26]**

Ein **strukturorientierter Test** [26], oder auch White-Box-Test, betrachtet im Gegensatz zum Black-Box-Test das Testobjekt von einem innen liegenden Punkt (PoO) und zielt darauf ab alle Teile des Programmcodes bei der Suche nach Fehlern abzudecken. Die Steuerung des Testobjekts (PoC) erfolgt dazu ebenfalls von innen, das heißt Variablenwerte und Funktionsparameter können direkt modifiziert werden. White-Box-Tests werden aufgrund der notwendigen Kenntnisse über den inneren Aufbau des Testobjektes fast ausschließlich auf der Ebene des Komponenten- und Integrationstests angewendet. Die zugehörigen Testfälle werden nicht wie bei Black-Box-Test ausschließlich auf der Spezifikation basierend erstellt, sondern hauptsächlich anhand der inneren Programmstruktur des Testobjektes. Ein Verfahren zur Testfallerzeugung ist zum Beispiel die Anweisungsüberdeckung, bei der die Testfälle alle oder eine bestimmte Mindestquote der im Programm enthaltenen Anweisungen aufrufen.

Schließlich ist an dieser Stelle der änderungsbezogene Test oder **Regressionstest** (regression testing) anzuführen. Dieser wird nach Korrekturen oder Weiterentwicklungen einer Software angewendet, um zu überprüfen, dass durch die Modifikationen im Quellcode keine Seiteneffekte

eingetreten sind oder unbeabsichtigt zusätzliche Fehler eingebaut wurden. Regressionstests werden daher auf allen Testebenen durchgeführt [26]. Aufgrund der meist beschränkten Ressourcen in Unternehmen können nicht alle der auf einer Testebene definierten Testfälle im Regressionstest verwendet werden. Aus diesem Grund ist eine Entscheidung zu treffen, welche der Testfälle für den Regressionstest wirklich notwendig sind („Testfallselektion“) oder welche hinsichtlich des Testziels den größten Nutzen bringen, um diese an den Anfang des Tests zu legen („Testfallpriorisierung“) und deren Ausführung vor dem Aufbrauchen der Ressourcen zu sichern. Des Weiteren kann eine Entfernung redundanter Testfälle, welche dieselben Fehler aufdecken wie andere Testfälle und damit keinen zusätzlichen Nutzen aufweisen, zur Verringerung der Testsuite-Größe („Testfallreduktion“) von Vorteil sein um einer Ressourcenknappheit beizukommen [46]. Diese Verfahren werden von Malishevsky et al. [47] hinsichtlich ihres Kosten-/Nutzen-Verhältnisses empirisch untersucht.

# 4 Analyse der Software-Qualität und Testfallselektion mittels Data Mining

In diesem Kapitel werden nun die im Rahmen der vorliegenden Arbeit gewonnen Erkenntnisse bezüglich des generellen Ablaufs eines Data-Mining-Projektes zur Analyse der Stärken und Schwächen einer Software präsentiert.

## 4.1 Projektziele

Um die Qualität eines Software-Produktes bereits während der Entwicklungsphase positiv beeinflussen zu können, sind eine kontinuierliche parallele Durchführung von Tests sowie direkt anschließende Maßnahmen zur Fehlerbeseitigung notwendig. Da das ständige und wiederholte Durchführen von Software-Tests, sofern es von Hand erfolgt, einen enormen Arbeits-, Zeit- und Kostenaufwand verursacht, werden vor allem bei großen und komplexen Softwareprojekten heute Lösungen zur Automatisierung verschiedener Aufgaben im Testprozess eingesetzt, zum Beispiel im Bereich der Testfallerstellung, der Testdurchführung und der Testauswertung.

Aufgrund der Durchführung von automatisierten Tests mit Hilfe von Testwerkzeugen - für die Programmiersprache Java sei auf Modultestebene zum Beispiel JUnit genannt - ist es bei der Überprüfung der Qualität einer Software heute möglich, auf diese Software eine Vielzahl von Tests mit unterschiedlichen Kombinationen von Daten und Parametern anzuwenden. Dieser Umstand führt dazu, dass die systematische Auswertung der Testergebnisse zur Analyse der Stärken und Schwächen einer Software über die verschiedenen Versionen hinweg kaum mehr möglich ist. Auch nimmt, bei einer Vielzahl von Testfällen, der Testaufwand trotz Automatisierungslösungen ein enormes Ausmaß an. Abhilfe können hier Ansätze für einen effizienten und wirtschaftlichen Einsatz der verschiedenen Testverfahren schaffen.

An diesen Punkten setzt das in der vorliegenden Arbeit - unter Verwendung von Data-Mining-Methoden - entwickelte Black-Box-Verfahren an, welches, basierend auf den Ergebnissen vergangener Software-Tests, eine Bestimmung der qualitativen Schwachstellen einer Software und einen effizienten Einsatz von Testverfahren ermöglichen soll.

## 4.2 Verwandte Arbeiten

Auf dem Gebiet der Effizienzsteigerung im Bereich der Software Qualitätssicherung kann bereits eine große Anzahl verschiedener Forschungsaktivitäten festgestellt werden. Es existieren zum Beispiel verschiedene Ansätze zur Messung und Analyse von Software-Prozessen und Software-Produkten, welche als Wegweiser für die im Bereich der Software-Technik zu fällenden Entscheidungen dienen können. Ein bekannter Vertreter dieser Analyseverfahren ist die Orthogonal Defect Classification (ODC) [48]. Dieses Verfahren dient der Identifizierung von Phasen des Entwicklungsprozesses, denen aufgrund von qualitativen Problemen eine besondere Aufmerksamkeit zu schenken ist. Diese Phasen werden ausfindig gemacht, indem die in der

Software gefundenen Fehler (z.B. Architekturfehler) der Prozessphase zugeordnet werden, in der sie entstanden sind (z.B. der SW-Design-Phase). Diese Zuordnung wird von Entwicklern der Software von Hand mittels einer Klassifizierung der Fehler durchgeführt. Diese können sich dabei an einem Klassifikationsschema orientieren, in dem Informationen zu früheren Fehlern und deren Prozesszuordnung gesammelt wurden. Trotz dieser Richtlinie ist diese manuelle Klassifizierung von Subjektivität geprägt, was ein großer Nachteil dieses Verfahrens darstellt. Mit dem in dieser Arbeit vorgestellten Verfahren soll dem Problem des subjektiven Einflusses der Entwickler bei der Analyse der SW-Qualität begegnet werden, indem die Identifizierung fehleranfälliger Produktteile (nicht Prozesse) mittels Data Mining automatisiert durchgeführt wird.

Neben der eben beschriebenen, sind im wissenschaftlichen Bereich weitere Arbeiten zu finden, die die Analyse der Software-Qualität zum Inhalt haben, jedoch einem anderen Ansatz nachgehen. Diese Arbeiten beinhalten die Vorhersage der Anzahl von Fehlern (Fehlerrate) in einer Software oder deren Teilen basierend auf Data Mining, um deren Qualitätsentwicklung zu beschreiben oder die Fehleranfälligkeit der Software zum aktuellen Zeitpunkt zu bestimmen oder zu einem späteren Zeitpunkt (z.B. nach der Auslieferung an den Auftraggeber bzw. Kunden) vorherzusagen. Ein Ansatz zur Lösung letzterer Problemstellung wird beispielsweise von Nagappan et al. [49] gegeben, die die in der Vergangenheit nach der Auslieferung einer Software besonders fehlerträchtigen SW-Komponenten untersuchten und einen Zusammenhang mit ihrer Code-Komplexität fanden. Bei der Code-Komplexität handelt es sich um einen Vertreter der Qualitätsindikatoren, die bereits im Abschnitt „Software-Qualität“ erwähnt wurden. Auf Basis der von einem Data-Mining-Modell gelernten Zusammenhänge kann die Fehleranfälligkeit nach der Produktauslieferung für neu entwickelte SW-Komponenten anhand von deren gemessener Code-Komplexität vorhergesagt werden. Hudepohl et al. [50] verwenden ein ähnliche Strategie, jedoch mit dem Unterschied, dass die betrachteten Qualitätsindikatoren von Vornherein festgelegt sind und nicht mittels Data Mining ermittelt wird, welche Indikatoren eine statistische Korrelation zur Fehleranfälligkeit aufweisen. Auch bezieht sich ihre Arbeit auf Ausfälle vor der Auslieferung einer Software und nicht auf die danach festgestellten. Ostrand et al. [51] widmen sich ebenfalls diesem Thema und sagen die Quellcode-Dateien mit der höchsten Fehlerrate im nächsten SW-Release voraus. Ihre Modelle weisen eine recht gute Vorhersagegenauigkeit auf. 20% der Dateien mit der höchsten, für den nächsten Release vorhergesagten Fehlerrate enthielten zwischen 71% und 92% der tatsächlich entdeckten Fehler.

Ramler [52] nutzt die Vorhersage von fehleranfälligen Software-Komponenten, welche auf der Grundlage verschiedener Qualitätsindikatoren für Software sowie auf Informationen aus Software-Versionierungssystemen und Fehlerdatenbanken erfolgt, zur auf dem Risiko basierenden Testfallpriorisierung. Risiko bezeichnet, vergleichbar mit der Risikoprioritätszahl aus dem Bereich der Software-FMEA-Analyse, die Wahrscheinlichkeit des Auftretens von Fehlern in der getesteten Komponente, kombiniert mit den Auswirkungen der Fehler. Bei Ramler's Lösung handelt es sich um einen ähnlichen Ansatz wie in dieser Arbeit, mit dem Unterschied, dass seine Lösung auf strukturellen Eigenschaften einer Software basiert und damit zu den White-Box-orientierten Ansätzen zählen kann, wohingegen in dieser Arbeit ein Black-Box-orientierter Ansatz entwickelt wird, der auf einer Auswertung der Eingaben und Ausgaben von dynamischen SW-Tests basiert. Des Weiteren nimmt er eine Testfallpriorisierung auf Basis des Risikos vor, während in dieser Arbeit die Bedeutung von aufgedeckten Fehler nicht in das Verfahren zur Priorisierung von Testfällen eingeht.

Auch andere Forschungsgruppen bemühen sich um Lösungen zum Thema Effizienzsteigerung bei SW-Tests. Hier sind, neben der Testfallpriorisierung, vor allem die Bereiche Testfallselektion, Testfallreduktion anzuführen. Die Lösungsansätze sind dabei entweder White-Box- oder Black-Box-orientiert. White-Box-Ansätze basieren, analog zur Lösung von Ramler, auf Qualitätsindikatoren zur Messungen von strukturellen Software-Eigenschaften in Kombination mit anderen Informationen, wie denen aus Fehlerdatenbanken extrahierten Daten. Diesen White-Box-orientierten Lösungen stehen die Black-Box-basierten gegenüber, wie zum Beispiel die von Last et al. [53]. Deren Ansatz besteht darin, mittels eines so genannten Info-Fuzzy Network (IFN) automatisiert Äquivalenzklassen von Testfällen anhand von deren Eigenschaften zu bestimmen und auf diese Weise die Anzahl der im Test benötigten Testfälle zu reduzieren. Zur Ermittlung der Eigenschaften werden die Zusammenhänge zwischen Testeingaben und den Testausgaben einer Software untersucht. Ein ähnlicher Ansatz ist bei Raamesh et al. [54] zu finden. Der Unterschied dieser Arbeiten zur vorliegenden besteht erstens darin, dass ein Baumklassifikationsverfahren angewendet wird, um die Schwächen der Software zusammen mit den Testfalleigenschaften aufzudecken. Zweitens wird die Auswertung von Software-Tests in dieser Arbeit nicht dazu genutzt um Testfälle zu ihrer Reduktion in Äquivalenzklassen einzuteilen, sondern um die Testfälle von einem Klassifikationsalgorithmus nach ihrer Fehlerentdeckungswahrscheinlichkeit absteigend sortieren (priorisieren) zu lassen und anschließend zu selektieren.

Ein weiterer Themenbereich umfasst die Analyse von Abhängigkeiten im Auftreten von Fehlern in verschiedenen SW-Komponenten und die Vorhersage des Aufwandes für deren Lokalisierung und Korrektur im weiteren Entwicklungsprozess. Song et al. [55] führen dazu Assoziationsanalysen und Klassifikationsanalysen durch, um für Fehler Zusammenhänge der Form $a \wedge b \Rightarrow c$ zu finden (wenn Fehler *a* und *b* zusammen auftreten, dann wird Fehler *c* höchstwahrscheinlich auch auftreten), anhand derer Entwickler in der Fehlerfindung und Projektmanager beim effizienten Einsatz von Testressourcen unterstützt werden sollen. Dieses Themengebiet ist zwar ebenfalls der Analyse der Qualität von Software gewidmet, damit wird sich diese Arbeit jedoch nicht beschäftigen.

Die meisten der zum Thema gefundenen Arbeiten versuchen die Problemstellungen mit Hilfe eines White-Box-Ansatzes zu lösen, zum Beispiel um, basierend auf den Zusammenhängen zwischen verschiedenen gemessenen Eigenschaften einer Software und der Anzahl der in der Vergangenheit in dieser Software gefundenen Fehler, Aussagen über die aktuell in einer Software enthaltenen beziehungsweise in Zukunft zu erwarteten Fehler zu treffen. Wissenschaftliche Arbeiten mit Black-Box-Ansätzen zur Lösung der gesetzten Data-Mining-Ziele konnten zum aktuellen Zeitpunkt nur wenige (zum Beispiel Last et al. [53]) gefunden werden, was bedeutet, dass in diesem Bereich noch genügend Forschungsbedarf vorhanden ist.

## 4.3 Ziele der Auswertung mittels Data Mining

Das Ziel der Anwendung von Data-Mining-Verfahren in dieser Arbeit ist in erster Linie die Offenbarung von Schwachstellen in einer Software, also derjenigen Bestandteile der Software, die mit hoher Wahrscheinlichkeit in der Zukunft ein Fehlverhalten aufweisen werden. Des Weiteren soll anhand der Data-Mining-Analyse eine systematische Identifizierung der Software- und eventuell auch der Hardware-bedingten Umstände, die für diesen Zustand verantwortlich sind, möglich

sein. Der Zweck einer intensiven Analyse des Zustandes oder Verhaltens einer Software kann die Ableitung von entsprechenden Maßnahmen aus den Ergebnissen zur Verbesserung der Software sein. Außerdem könnten die gewonnen Erkenntnisse genutzt werden, um bei der weiteren Entwicklung von als fehleranfällig eingestuften Software-Teilen besondere Vorsicht walten zu lassen.

Das zweite große Ziel dieser Analyse ist die Erhöhung der Effizienz von Software-Tests. Um diese zu erreichen, soll der Großteil der Ausfälle von Software-Komponenten so zeitig wie möglich in der Testphase aufgezeigt werden, um die verbundenen Fehler dementsprechend frühzeitig beseitigen zu können. Aus diesem Grund sollen die Testfälle nach ihrer Wahrscheinlichkeit der Fehlerentdeckung absteigend sortiert und diejenigen mit der höchsten Wahrscheinlichkeit an den Anfang zukünftiger Tests gelegt werden. Aus der Menge derjenigen mit einer geringen Wahrscheinlichkeit ist eine gewisse Anzahl zu selektieren. Da anzunehmen ist, dass die Komponenten, bei denen in der Vergangenheit bereits vermehrt Fehler aufgetreten sind, wahrscheinlich auch in Zukunft eher Fehler enthalten als andere Komponenten, wird die Fehlerentdeckungswahrscheinlichkeit eines Testfalls davon abhängen, wie fehleranfällig die von ihm getestete Komponente (generell oder in Bezug auf seine Parameter) ist.

Die aus den genannten Einzelzielen formulierbaren konkreten Fragen, die mittels geeigneter Data-Mining-Verfahren beantwortet werden sollen, sind in Tabelle 4-1 dargestellt. Zur Erreichung der gestellten Data-Mining-Ziele werden ein oder mehrere Data-Mining-Verfahren angewendet. Die Auswahl der oder des geeigneten Data-Mining-Verfahrens ist Inhalt des nächsten Abschnitts.

| Nr. | Frage |
|---|---|
| 1 | Welche Teile einer Software sind besonders anfällig für Fehler? |
| 2 | Welche Einflussgrößen stehen damit in Zusammenhang? |
| 3 | Welche Testfälle besitzen die höchste Fehlerentdeckungswahrscheinlichkeit? |

**Tabelle 4-1: Einzelziele der Data-Mining-Analyse**

## 4.4 Wahl des Data-Mining-Verfahrens

Zur Bestimmung der einzusetzenden Data-Mining-Verfahren wird deren Eignung bezüglich jedes einzelnen Data Mining-Ziels evaluiert.

#### 1 - Welche Teile einer Software sind besonders anfällig für Fehler?

Das zu wählende Data-Mining-Verfahren soll im Stande sein Schwachstellen in der Software zu identifizieren, also diejenigen Einheiten, die über mehrere Versionen der Software hinweg ausgefallen sind. Man könnte an dieser Stelle, sofern es die Daten zulassen, noch eine Unterteilung anhand der Art des Fehlverhaltens vornehmen, um die besonders fehleranfälligen

Software-Einheiten (intermittierendes Fehlverhalten) von den weniger kritischen Fällen (transientes Fehlverhalten) zu trennen. Dies wird in der späteren Fallstudie nicht der Fall sein, da die Testresultate in den aufeinander folgenden Software-Versionen kein intermittierendes Fehlverhalten zeigen, die Daten also nicht dafür geeignet sind.

Die Clustering-Methode ist an dieser Stelle nicht einsetzbar, da dies lediglich Gruppen von Software-Einheiten bildet, deren Eigenschaften und Testergebnisse ähnlich sind, jedoch keine Aussagen darüber ermöglicht, welche der Einheiten in den Gruppen als Schachstellen gelten. Auch die Regressionsmethode ist an dieser Stelle ungeeignet, da mit dieser nur numerische Größen vorhergesagt werden können und nicht ob eine Software als fehleranfällig gilt oder nicht. Des Weiteren macht die Anwendung der Assoziationsmethode an dieser Stelle keinen Sinn, da mit dieser ebenfalls keine Klassifikation der Software-Einheiten vorgenommen werden kann, sondern eher die Abhängigkeiten zwischen den einzelnen Einheiten aufgedeckt werden könnten.

Die Klassifikationsmethode scheint hier auf den ersten Blick geeignet zu sein. Denn mit dieser kann die Zugehörigkeit von Testfällen, beziehungsweise der mit ihnen geprüften Software-Einheiten, zu Klassen (zum Beispiel *„fehlgeschlagen“* und *„nicht fehlgeschlagen“*) vorhergesagt werden. Die Klasse „fehlgeschlagen“ würde dann anzeigen, dass es sich bei der Software-Einheit um eine Schwachstelle bzw. bei dem Testfall um einen unbedingt erforderlichen „Teilnehmer“ eines künftigen Tests handelt. Die Vorhersage der Klassenzugehörigkeit basiert dabei auf einer Kombination von Testergebnissen, die von vorhergehenden Versionen der Software vorliegen, und den verschiedenen in den Testfällen enthaltenen Parametern, indem das verwendete Klassifikationsverfahren diese Kombination als Anzeichen für einen erneuten Fehlschlag in einer zukünftigen Version der Software sieht oder nicht. Der Algorithmus hat diese Kombinationen mit ihrer entsprechenden Klassenzugehörigkeit zuvor anhand von historischen Daten, also Testfällen die bereits den Klassen zugeordnet wurden, gelernt. Ein Testfall wurde in den historischen Daten dann der Klasse „fehlgeschlagen“ zugeordnet, wenn die geprüfte Software-Komponente in einer der bisherigen SW-Versionen ein Fehlverhalten gezeigt hat.

**2 - Welche Einflussgrößen stehen damit in Zusammenhang?**

Zur Lösung dieser Aufgabe kann ebenfalls die Klassifikationsmethode angewendet werden, da die Einflussgrößen bereits im Ergebnis der vorhergehenden Aufgabe enthalten sind, sofern zur Erstellung des Klassifikationsmodells ein Verfahren eingesetzt wird, das für den Anwender verständliche Klassenbeschreibungen zur direkten Ableitung der Einflussgrößen erzeugt. Das Entscheidungsbaumverfahren stellt macht dies möglich, denn bei diesem sind die Parameter, die im Zusammenhang mit der Schwäche bestimmter Software-Einheiten stehen aus dem erzeugten Entscheidungsbaum direkt ablesbar. Aus diesem Grund wird ein Entscheidungsbaumverfahren zur Erstellung des Klassifikationsmodells und damit zur Lösung der ersten und zweiten Data-Mining-Aufgabe eingesetzt.

**3 - Welche Testfälle besitzen die höchste Fehlerentdeckungswahrscheinlichkeit?**

Die Beantwortung dieser Frage basiert auf der Analyse der ersten und zweiten Frage, denn sie kann ebenfalls mit Hilfe des erstellten Klassifikationsmodells beantwortet werden, welches automatisch eine absteigende Sortierung (Priorisierung) der Testfälle nach ihrer Fehlerentdeckungswahrscheinlichkeit durchführt.

Die theoretischen Grundlagen des zur Klassifikation der SW-Einheiten und der darauf basierenden Priorisierung und Selektion von Testfällen eingesetzten Entscheidungsbaumverfahrens werden im Folgenden erläutert.

## 4.5 Theoretischer Hintergrund des gewählten Verfahrens

Zuvor soll jedoch die Bedeutung der im weiteren Verlauf verwendeten Begriffe zur Schaffung eines einheitlichen Verständnisses geklärt werden.

**Terminologie**

Eine *Klasse* ist ein virtuelles Konstrukt zur Zusammenfassung von Objekten, die gleiche oder ähnliche Ausprägungen (Zustände, Werte) bestimmter Merkmale aufweisen. Ein Merkmal wäre zum Beispiel die „Länge“ eines Objektes und dessen Ausprägung „30 cm“. Die Gesamtheit der systematisch gebildeten Klassen wird als *Klassifikation* oder als Klassensystem bezeichnet. Die Klassifikation ist das Endprodukt der *Klassifizierung*, dem Prozess zu Bildung von Klassen ähnlicher Objekte. Dieser Prozess kann auf zwei verschiedene Arten ablaufen, nach dem *top-down*-Ansatz[21] oder dem *bottom-up*-Ansatz[22]. Bei einem bottom-up-Vorgehen werden Objekte hinsichtlich ihrer Merkmale untersucht, um sie bei Vorliegen ähnlicher Merkmalszustände zu Klassen zusammenzufassen. Bei einem top-down-Ansatz werden die Klassen vorgegeben und die, eine jeweilige Klasse kennzeichnende Kombination von Merkmalszuständen bestimmt, zum Beispiel mit Hilfe von Erfahrungswissen oder mathematisch-statistischen Berechnungen, und in Form so genannter *Entscheidungsregeln* festgehalten. Anhand des Abgleichs der konkreten Merkmalszustände von zu klassifizierenden Objekten mit diesen Entscheidungsregeln, können die Objekte den vordefinierten Klassen zugeordnet werden. Dabei werden nur die aussagekräftigsten Merkmale in den Entscheidungsregeln berücksichtigt, um die Komplexität beim Abgleichen der einzelnen Merkmalszustände der Objekte mit den Regeln so gering wie möglich zu halten.

Die Instanz, welche eine Klassifizierung vornimmt, wird bezeichnet als *Klassifikator*. Dies kann eine Person sein, die eine Klassifizierung von Hand vornimmt, oder ein Software-Algorithmus, wie er zum Beispiel im Intelligent Miner implementiert ist, der die Klassifizierung automatisiert durchführt. Im weiteren Verlauf des Kapitels wird ausschließlich auf die Arbeitsweise von automatischen Klassifikatoren eingegangen, da nur diese in der Arbeit eingesetzt werden. Diese computergestützte Klassifizierung erfolgt nach dem top-down-Prinzip, das heißt, basierend auf der statistischen Analyse der Merkmalswerte von bereits in der Vergangenheit zu vordefinierten Klassen zugeordneten (bekannten) Objekten einer Menge, der sogenannten *Trainingsmenge*,

[21] schrittweises Vorgehen von „oben nach unten“, das heißt vom Abstrakten zum Konkreten
[22] Vorgehen zum top-down-Ansatz entgegengesetzt gerichtet

werden vom automatischen Klassifikator Entscheidungsregeln zur späteren Klassifizierung neuer (unbekannter) Objekte gelernt.

Die genaue Vorgehensweise des Klassifikators bei der Klassifizierung wird durch ein *Klassifikationsverfahren*, in diesem Fall das *Entscheidungsbaumverfahren*, bestimmt. Ein Entscheidungsbaumverfahren präsentiert die Entscheidungsregeln in einer graphischen und daher leicht verständlichen Form, dem *Entscheidungsbaum*. In Abbildung 4-1 ist ein Beispiel für einen solchen Entscheidungsbaum dargestellt. Dieser Baum ordnet das Ereignis der Fruchtbildung bei Chili-Pflanzen der mexikanischen Sorte Jalapeño anhand ihrer Standortbedingungen in zwei Klassen ein, in *ja*, die Chili-Pflanze entwickelt Früchte, und *nein*, die Chili-Pflanze wird keine Früchte tragen. Er dient demzufolge der Ermittlung von günstigen Standortbedingungen für Jalapeño-Pflanzen, um die Basis für eine ertragreiche Ernte von Chili-Früchten zu legen.

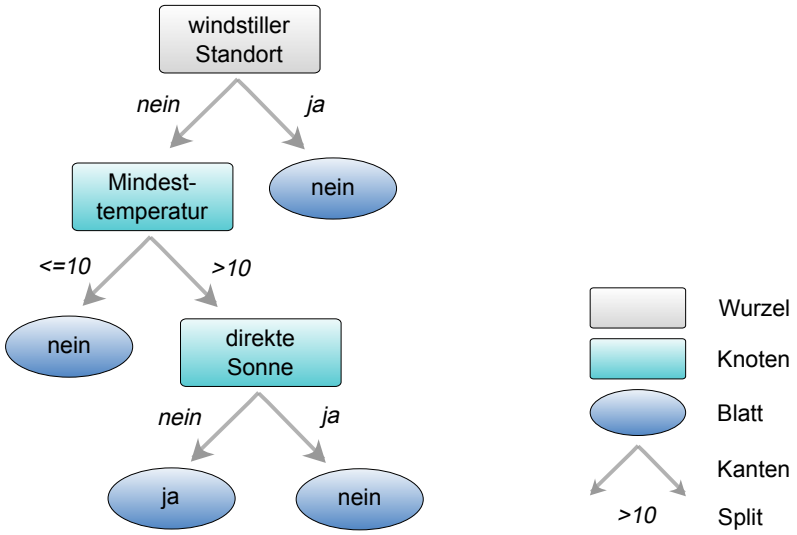



**Abbildung 4-1: Aufbau eines Entscheidungsbaumes**

Ein Entscheidungsbaum besteht aus den in der 4-1 gezeigten Elementen: Wurzel, Knoten, Blätter, Split und Kanten. Die *Wurzel* stellt die Basis des umgedrehten Baumes dar, die alle Objekte der zu analysierenden Trainingsmenge enthält. Ihr entspringen zwei oder mehr *Kanten*, die sich an den *Knoten* des Baumes verzweigen bis sie schließlich in den *Blättern* münden. Da die Wurzel und Blätter genau genommen ebenfalls Knoten darstellen, werden die im Inneren des Baumes liegenden Knoten zu deren Unterscheidung auch als *interne* Knoten bezeichnet. Diese Unterscheidung wird im Folgenden vernachlässigt, da bei Verwendung der Bezeichnung „Knoten“ stets ein innerer Knoten gemeint ist. Die Verzweigung an den Knoten im Baum erfolgt durch eine Teilung, auch bezeichnet als Partitionierung oder *Split*, der Trainingsmenge in mehrere Teilmengen. Der Knoten steht dabei für dasjenige Merkmal der in der Trainingsmenge enthaltenen Objekte, anhand dessen Werten der Split erfolgt. Der genaue Ablauf zur Erstellung eines solchen Entscheidungsbaumes wird nachfolgend auf der Grundlage des Beispiels mit den Jalapeño-Pflanzen näher erläutert.

**Vorgehensweise zur Erstellung eines Entscheidungsbaumes**

Um die Vorgehensweise des automatischen Klassifikators mathematisch beschreiben zu können, ist zuvor eine formale Definition der Menge von Objekten und der zugehörigen Klassen notwendig. Die Menge $X$ mit $n$ Objekten kann, vorerst ohne Klassenzuordnung, formal beschrieben werden als

$$X = \{\overrightarrow{x}_1, ..., \overrightarrow{x}_n\} \subset \mathbb{R}^p \tag{4.1}$$

wobei jedes Objekt durch einen der möglichen Merkmalsvektoren $\overrightarrow{x}$ in einem $p$-dimensionalen reellen Raum repräsentiert wird und jede Dimension $p$ für eines der bei den Objekten auftretenden Merkmale steht. Der Wert, den ein Objekt für jeweils eines seiner $p$ Merkmale annimmt, wird durch die Größe der Verschiebung des Merkmalsvektors in der entsprechenden $p$-ten Dimension ausgedrückt. Zur Einbeziehung der Zuordnung eines jeweiligen Objektes zu einer Klasse $j$ aus der Menge der Klassen $C = \{1, ..., j, ..., J\}$ wird ein eindimensionaler Klassenvektor $\overrightarrow{y}$ verwendet, der auf ein Element der Menge mit den $J$ Klassen zeigt und dessen Menge $y$ für alle $n$ Objekte in der Form

$$y = \{\overrightarrow{y}_1, ..., \overrightarrow{y}_n\} \subset C \tag{4.2}$$

definiert ist. Im Beispiel der Jalapeño-Pflanzen existieren $J = 2$ Klassen, sodass für den Klassenvektor gilt: $y \subset \{1, 2\}$. Der Index 1 repräsentiert dabei die Klasse „ja“ im Entscheidungsbaum (Jalapeño-Pflanze trägt Früchte) und der Index 2 die Klasse „nein“ (Jalapeño-Pflanze trägt keine Früchte). Die Trainingsmenge $T$, als Teilmenge der bereits klassifizierten Objekte mit $n$ Elementen, lässt sich allgemein als Konkatenation (Verkettung) der Merkmalsvektoren im $p$-dimensionalen Raum und der eindimensionalen Klassenvektoren spezifizieren:

$$T = (X, y) = \{(\overrightarrow{x}_1, \overrightarrow{y}_1), ..., (\overrightarrow{x}_n, \overrightarrow{y}_n)\} \subset \mathbb{R}^{p+1} \tag{4.3}$$

Aufgabe des Klassifikators ist es nun, eine Funktion $f$ zu finden mit:

$$f : \mathbb{R}^p \to C, \overrightarrow{x} \mapsto \overrightarrow{y} \tag{4.4}$$

Diese ordnet dem Merkmalsvektor $\overrightarrow{x}$ eines beliebigen Objektes den Klassenvektor $\overrightarrow{x}$ und damit jedem Objekt eine Klasse zu. Zur Generierung einer solchen Funktion wird vom Klassifikator ein Entscheidungsbaum erstellt und aus diesem Entscheidungsregeln extrahiert, die die Funktion repräsentieren. Die Entscheidungsregeln werden in einem Klassifikationsmodell gespeichert und können zur nachfolgenden Klassifizierung von neu hinzugekommenen Objekten oder zur Ableitung der eine bestimmte Klasse kennzeichnenden Merkmalswerte, und damit einem Erkenntnisgewinn, dienen.

Zur Erstellung eines Entscheidungsbaumes wird dieser Knoten für Knoten - von der Wurzel beginnend - aufgebaut, indem an jedem Knoten ein Merkmal bestimmt wird, anhand dessen sich die Menge der Objekte am „besten“ nach ihrer Klassenzugehörigkeit aufteilen („splitten“) lässt, um dann die aus der ursprünglichen Menge erzeugten Teilmengen an die nächsten Knoten weiterzureichen und diese dort anhand des nächsten, „besten“ Merkmals erneut aufzuteilen, bis in den einzelnen Blättern des Baumes nur noch Objekte mit einer gleichen Klassenzugehörigkeit vorliegen. Die Splits werden dabei anhand der verschiedenen Werte, die bei den Objekten für das Split-Merkmal auftreten, durchgeführt. Der Klassifikator zerlegt dazu die einzelnen Merkmalsvektoren der Objekte in ihre Komponenten (eine Komponente je $p$-ter Dimension bzw. je Merkmal) und betrachtet für alle Objekte einer bestimmten Menge diejenigen ihrer Merkmalsvektorkomponenten gemeinsam, die der Dimension $p$ des aktuell betrachteten Merkmals angehören. An der Wurzel werden die Merkmalsvektorkomponenten (Merkmalswerte) der Objekte der gesamten Trainingsmenge herangezogen und in den tiefer liegenden Ebenen des Baumes die Komponenten der Objekte in den einzelnen, bereits an den Knoten gebildeten Teilmengen.

Zur Bestimmung des - nach informationstheoretischen Gesichtspunkten - besten Merkmals für einen Split stehen verschiedene Kriterien zur Auswahl, zum Beispiel die *Entropie*, welche ein Maß für den Informationsgewinn ist, der mit einem Merkmal realisiert werden kann, wobei das Merkmal mit dem größten Informationsgewinn für den Split zu wählen ist. Ein anderes Kriterium ist der so genannte *Gini-Index*, der ein Maß für die „Unreinheit“ (Heterogenität) der Klassenwerte in einer bestimmten Objektmenge darstellt. Bei einer hohen Unreinheit gehören wenige Objekte in einer Menge derselben Klasse an und umgekehrt. Da nach einem „guten“ Split möglichst viele der Objekte in den Teilmengen derselben Klasse angehören sollten, ist dasjenige Merkmal für die Bildung der Teilmengen zu wählen, welches am stärksten zur Verringerung der Unreinheit der ursprünglichen, vor der Partitionierung vorliegenden, Menge beiträgt. Da der Gini-Index in dem Entscheidungsbaumverfahren, welches im Intelligent Miner implementiert ist, Verwendung findet [56], wird dieser im Folgenden ausführlicher betrachtet.

Für den Split der gesamten Trainingsmenge $T$ an der Wurzel des Baumes kann der Gini-Index basierend auf der relativen Häufigkeit $p_j$ einer Klasse $j$ in $T$ (in Anlehnung an [57]) berechnet werden mit:

$$gini(T) = 1 - \sum_{j=1}^{J} {p_j}^2 \tag{4.5}$$

Der Wert des Gini-Index liegt bei einem binären Baum mit zwei Klassen in einem Intervall von $[0,0,5]$ und bei einem Baum mit mehr als zwei Klassen im Bereich von $[0,1]$. Der Minimalwert $gini(T) = 0$ ergibt sich, wenn in einer Menge ausschließlich Objekte derselben Klasse enthalten sind, also bei einer minimalen Unreinheit (maximalen Homogenität). Den für einen binären Baum geltenden Maximalwert von $0,5$ nimmt der Gini-Index an, wenn die Objekte in einer Menge über zwei Klassen gleich häufig verteilt sind, das heißt bei einer maximalen Unreinheit (maximalen Heterogenität). Der bei mehr als zwei Klassen geltende Maximalwert von 1 kann in der Praxis nicht auftreten, da er dann vorliegen würde, wenn eine unendlich große Anzahl von Klassen existiert, der jeweils eines von unendlich vielen Objekten in einer Menge angehört. Die Verringerung der Unreinheit $\triangle gini$, deren Maximierung das Ziel bei einem Split der Trainingsmenge $T$ in $m$

disjunkte Teilmengen $T_1, T_2, ..., T_m$ darstellt, kann für jedes zur Partitionierung verfügbare Merkmal $M$ bestimmt werden mit (in Anlehnung an [58] und [57]):

$$\triangle gini_M(T) = gini(T) - \sum_{i=1}^{m} \frac{|T_i|}{|T|} \cdot gini(T_i) \tag{4.6}$$

Für jede der Teilmengen, die nach einem Split an den Knoten der nächst tieferen Baumebene vorliegenden, wird, wie bereits erwähnt, erneut ein Split durchgeführt. Dazu wird die Ermittlung des für den Split am besten geeigneten Merkmals (aus der Menge der noch verbliebenen) anhand der Gleichungen 4.5 und 4.6 für jede der Teilmengen bzw. Knoten wiederholt.

Zur Veranschaulichung der mathematischen Grundlagen wird eine Beispielrechnung mit den in Tabelle 4-2 aufgeführten Standortinformationen der Jalapeño-Pflanzen durchgeführt. Das Ergebnis der in in Abbildung 4-2 dargestellten Beispielrechnung ist das bestmögliche Merkmal für einen an der Wurzel des Entscheidungsbaumes durchgeführten Split, welches die Unreinheit der Gesamtmenge bezüglich der auftretenden Klassen $\{ja, nein\}$ in den zwei neu gebildeten Teilmengen maximal verringert (Merkmal rot unterstrichen).

| ID | Min.-Temp. in °C | Windstille | direkte Sonne | trägt Früchte? |
|---|---|---|---|---|
| 1 | -2 | ja | nein | nein |
| 2 | 0 | nein | nein | nein |
| 3 | 13 | ja | ja | nein |
| 4 | 11 | nein | ja | nein |
| 5 | 17 | ja | nein | nein |
| 6 | 2 | ja | ja | nein |
| 7 | 3 | nein | ja | nein |
| 8 | 12 | nein | nein | ja |
| 9 | 9 | nein | ja | ja |
| 10 | 10 | nein | nein | nein |
| 11 | 14 | nein | ja | ja |
| 12 | 12 | nein | ja | nein |
| 13 | 15 | ja | nein | nein |
| 14 | 17 | nein | nein | ja |
| 15 | 16 | nein | nein | ja |
| 16 | 11 | nein | nein | ja |

**Tabelle 4-2: Beispieldaten zur Klassifizierung der Fruchtbildung bei Jalapeño-Pflanzen**

Nachdem die Grundlagen des Entscheidungsbaumverfahrens, welches zur Erreichung der gesetzten Data-Mining-Ziele anwendbar ist, vorgestellt wurden, wird in der nächsten Sektion die genaue Vorgehensweise zur Lösung der Aufgaben beschrieben.

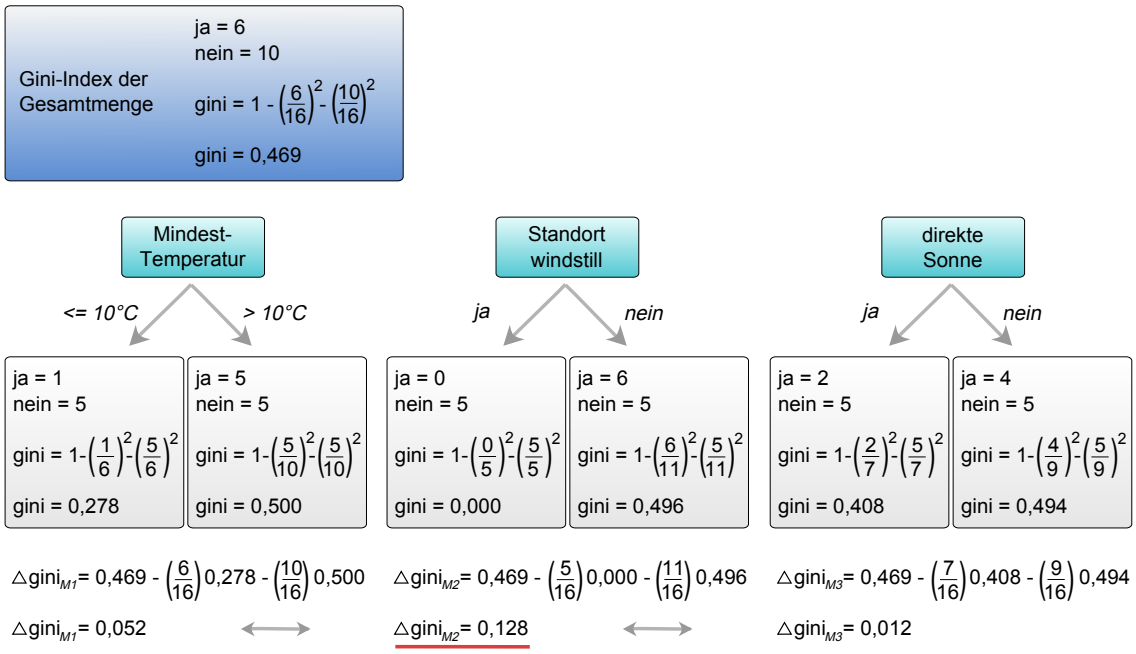



**Abbildung 4-2: Berechnung des an der Wurzel am besten für einen Split geeigneten Merkmals**

## 4.6 Vorgehensweise bei der Analyse mittels Data Mining

Die Grundlage zur Lösung der vorgestellten Data-Mining-Aufgaben, die Analyse der Software-Qualität basierend auf der Ermittlung von Schwachstellen in der Software und die Priorisierung und Selektion von Testfällen zur Erhöhung der Effizienz bei Software-Tests, bildet, wie in Abbildung 4-3 dargestellt ist, die Data-Mining-Analyse.

Diese wird auf die Eigenschaften der Eingaben von dynamischen Black-Box-orientierten Regressionstests und auf die vom Testobjekt als Reaktion auf die Eingaben produzierten Ausgaben angewendet. Eingaben dieser Regressionstests sind Testfälle mit ihren Parametern, wie Eingabedaten und Funktionsparameter, die den Prüflingen im Test übergebenen werden. Die Testausgaben werden repräsentiert von den zu jedem einzelnen Testfall vorliegenden Testergebnissen über die verschiedenen Versionen der Software hinweg. Diese Testergebnisse enthalten Angaben, ob die Reaktionen des Testobjektes auf die Testeingaben in Form von Ausgabewerten mit den erwarteten Reaktionen, den Referenzwerten, übereinstimmen oder nicht.

Diese benötigten Rohdaten sind in einem ersten Schritt zu erfassen und zu untersuchen sowie anschließend auf die DM-Analyse vorzubereiten, da sie in den meisten Fällen nicht in der erforderlichen Form vorliegen. Im nachfolgenden Schritt erfolgt die Erstellung des Entscheidungsbaumes mit den vorbereiteten Daten zu den Testfällen und Testergebnissen unter Angabe verschiedener Parameter für den Erstellungsprozess. Der Entscheidungsbaum soll mit einer berechneten Wahrscheinlichkeit für Testfälle vorhersagen ob diese fehlgeschlagen (und damit Fehler entdecken) oder nicht. Die Güte der Vorhersage und die Anwendbarkeit der Entscheidungsregeln auf neue Testfälle ist anschließend zu beurteilen. Den Abschluss bildet die Interpretation der DM-Ergebnisse. Anhand der als Ergebnis ermittelten Wahrscheinlichkeiten der Fehlerentdeckung können die Testfälle für die weitere Verwendung in Tests priorisiert und selektiert werden.

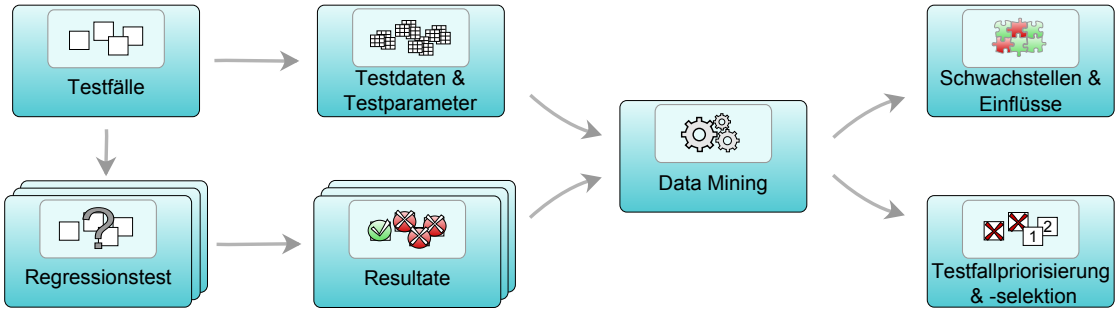



**Abbildung 4-3: Black-Box-Verfahren zur Auswertung von SW-Tests mittels Data Mining**

Auch können die fehleranfälligen SW-Komponenten, zum Teil in Kombination mit weiteren SW-Eigenschaften, aus den Split-Merkmalen des Entscheidungsbaumes abgelesen werden, da diese für das Fehlschlagen der Testfälle verantwortlich sind.

### 4.6.1 Datenerfassung

Für die Data-Mining-Analyse müssen, wie bereits erwähnt, die Testfallparameter aus den Testfällen und die Eigenschaften der Testeingaben (z.B. Verwendung ungewöhnlicher Werte) vorliegen. Diese können aus den zugehörigen Quellen wie Testdefinitionsdateien und Eingabedatendateien gewonnen werden. Die ebenfalls benötigten Testresultate für alle der in den Tests verwendeten Testfälle sind gewöhnlich in Testberichten zu den einzelnen Regressionstests enthalten, welche zur Erfassung der Daten benötigt werden.

Sobald die relevanten Datenquellen, welche die Informationen der benötigten Art enthalten, sondiert wurden, gilt es herauszufinden, ob alle benötigten Daten enthalten sind. Fehlen noch Daten, sollte überlegt werden, auf welchem Wege und unter welchen Bedingungen diese beschafft werden können. Eine Bedingung könnte zum Beispiel sein, dass spezielle Rechte für den Zugriff auf die Daten vorhanden sein müssen und diese eventuell mit einer gewissen Vorlaufzeit zu beantragen sind. Auch ist zu klären, ob die Daten in der vorliegenden Form verwendet werden dürfen. Zum Beispiel müssen bei der Verarbeitung personenbezogener Daten die Regelungen des Bundesdatenschutzgesetzes (BDSG) [59] beachtet werden, sodass vor einer Auswertung von Daten eventuell deren Anonymisierung erforderlich wird. Weitere Beschränkungen bezüglich der Speicherung und Verarbeitung sensibler Daten müssen im Kontext der gesetzlichen und anderweitig geltenden Regelungen individuell geklärt werden.

Liegen alle benötigten Daten schließlich vor, sollten Überlegungen bezüglich der Herangehensweise an das Auslesen der Datenquellen angestellt werden. Fragen die man sich dazu stellen könnte sind zum Beispiel: Sind die Dateien formal strukturiert und dafür geeignet ausgelesen zu werden oder müssen diese noch in eine entsprechende Form gebracht werden? Sind zum Beispiel unstrukturierte Textstücke vorhanden die erst kodiert oder aus denen Einzelinformationen extrahiert werden müssen? Existieren bereits Standard-Tools zum Auslesen der Daten oder ist eine eigene Software-technische Lösung in einer gewählten Programmiersprache zu entwickeln? Nachdem mit Hilfe des gewählten oder selbst entwickelten Tools die Daten erfasst wurden, sind diese in Datenbanktabellen oder einfachen Dateien zu speichern. Die Wahl des Speicherortes

hängt von den Anforderungen des verwendeten Data-Mining-Werkzeugs ab, denn einige arbeiten innerhalb einer Datenbank und erwarten das lokale Vorliegen der Daten in derselben und andere arbeiten Datenbank-unabhängig und geben keine Beschränkungen bezüglich des Speicherortes der Daten. Wird eine Software für die nachfolgende Datenvorverarbeitung verwendet, sollten die Daten dort hinein geladen werden, sofern dies erfordert wird.

Anschließend an die Erfassung der Daten erfolgt deren Begutachtung hinsichtlich sofort sichtbarer Eigenschaften, wie deren Größe und Format, um Probleme bei der Kompatibilität mit dem verwendeten Data-Mining-Werkzeug aufzudecken und entsprechende Anpassungsmaßnahmen durchzuführen. Eine tiefer gehende Untersuchung der Daten kann Probleme hinsichtlich der Qualität der Daten zu Tage fördern, deren Lösung in der nachfolgenden Phase der Datenvorbereitung erfolgen sollte. Daten können unter anderem mit Fehlern behaftet, unvollständig oder für das verwendete Data-Mining-Verfahren unvorteilhaft skaliert sein, was zu fehlerhaften Ergebnissen bei der späteren Analyse führen kann. Unvorteilhaft skaliert bedeutet, dass die Wertebereiche unterschiedlicher Datenattribute stark voneinander abweichen, sodass Attribute mit kleinen Werten als weniger relevant erachtet werden als Attribute mit sehr großen Werten, obwohl erstere vielleicht eher von Interesse in Bezug auf das jeweilige Data-Mining-Ziel sind. Die in dieser Phase gefundenen Probleme sollten festgehalten und entsprechende Maßnahmen zu deren Beseitigung angefügt werden.

### 4.6.2 Datenvorbereitung

Diese Maßnahmen zur *Bereinigung* der Daten zur Beseitigung von Problemen, wie zum Beispiel die Behandlung von Ausreißern (stark von den anderen abweichende Werte bezüglich eines Attributes), machen nur einen Teil aller Aktivitäten zur Vorbereitung der Daten auf die Analyse aus. Weitere Aufgaben umfassen die Selektion benötigter Datensätze oder Datenattribute, die Umformung der Daten sowie die Zusammenführung und die Formatierung der Daten.

Die *Selektion* dient der Reduzierung der Datenmenge um, für die Analyse nicht benötigte, Datensätze und Datenfelder, zum Beispiel um die Berechnungsdauer der Analyseverfahren in einem gewissen Rahmen zu halten oder um potenziell wertvolle Informationen nicht in einer „Flut“ von Modellausgaben untergehen zu lassen. Entscheidungsbäume können zum Beispiel bei Vorliegen einer großen Anzahl von Merkmalen sehr viele Knoten enthalten und daher sehr komplex werden, sodass die Suche nach den für das Data-Mining-Ziel interessanten Merkmalen mit einem unnötig hohen Aufwand verbunden wäre.

Durch *Transformationen* (Umsortieren) der Daten kann der Fokus der Analyse auf diejenigen Objekte gelegt werden, die aus der DM-Zielstellung als die zu untersuchenden hervorgehen. Im Bereich der Klassifikationsverfahren sind daher die Inhalte der Tabellen so anzuordnen, dass in jeder Zeile ein Objekt (z.B. entweder Testfall oder SW-Funktion) zusammen mit seinen Attributwerten, vor allem dem für die Klassifikation verwendeten Zielattributwert (Angabe der Klassenzugehörigkeit) aufgeführt ist. Da sich die drei in dieser Arbeit gestellten DM-Ziele erreichen lassen, indem eine Klassifizierung von Testfällen vorgenommen wird, wird der Fokus der Analyse auf die Testfälle gelegt. Für diese werden alle notwendigen Merkmale bestimmt, einschließlich des Zielattributes, welches das Testresultat eines jeden Testfalls enthält, anhand dessen die Testfälle klassifiziert werden. Ein Mittel zur Umformung von Daten ist die *Aggregation* zur Bildung

neuer Werte mittels der Berechnung der Summe, des Durchschnitts, der Minimalwerte bzw. Maximalwerte oder eines Prozentwertes aus den ursprünglichen Werten, die zusätzliche, für die Analyse interessante Eigenschaften der Daten preisgeben. Die *Skalentransformation* dient der Anpassung der verschiedenen Wertebereiche von Merkmalen aneinander, um eine Verfälschung des Analyseergebnisses zu vermeiden. Dies kann dann der Falls sein, wenn numerisch kleine Werte gegenüber numerisch großen Werten bei der Analyse vernachlässigt werden [60]. Um kontinuierlichen (numerischen) Werten, mit denen sich zum Teil nur sehr spezielle Aussagen generieren lassen, mehr generelle Bedeutung zu verleihen, können diese mittels einer *Diskretisierung* in diskrete Kategorien umgewandelt werden. Die ist auch notwendig, wenn kontinuierliche Werte als Zielattribut für eine Klassifizierung vorgesehen sind, denn eine Klassifikationsanalyse kann ausschließlich kategorische Zielwerte vorhersagen. Die anderen vom Zielattribut verschiedenen Merkmale können dabei jedoch sowohl numerischen als auch kategorischen Typs sein. Liegen inhaltlich zusammenhängende Daten zu Objekten verteilt über verschiedene Datenbanktabellen vor, so können diese durch die *Zusammenführung* mittels eines Verknüpfungsoperators (Join) in einer Tabelle konsolidiert werden. Schließlich kann eine *Formatierung* der Daten zu deren Anpassung an die Erfordernisse des Analyseverfahrens erfolgen. Dies ist zum Beispiel notwendig, wenn die Reihenfolge der Attribute in einer Tabelle im verwendeten DM-Werkzeug von Bedeutung ist und das Schlüsselattribut an erster sowie das Zielattribut an letzter Stelle erwartet wird [12] .

### 4.6.3 Erstellung des Test-Designs

Um die vom Klassifikator gelernten Entscheidungsregeln hinsichtlich ihrer Eignung zur korrekten Klassifizierung von Objekten mit bisher unbekannter Klassenzugehörigkeit beurteilen zu können, wird die ursprüngliche Datensatzmenge in Teilmengen geteilt. Die Erstellung eines *Klassifikationsmodells*, welches die aus einem Entscheidungsbaum abgeleiteten Entscheidungsregeln enthält, erfolgt durch den Klassifikator auf Basis der ersten Teilmenge der ursprünglichen Datensätze, der so genannten *Trainingsmenge*. Anschließend wird das Modell mit den Entscheidungsregeln auf die restliche Menge der Datensätze, die *Testmenge*, angewendet, um anhand eines Abgleichs der vom Modell vorhergesagten und der realen Klassenzugehörigkeit der Objekte bzw. Datensätze in der Testmenge die Güte des Klassifikationsmodells (der Entscheidungsregeln) zu messen und zu beurteilen. Die Klassenzugehörigkeit eines jeden Objektes wird dabei anhand eines entsprechenden Wertes in der sogenannten *Zielspalte* bzw. dem *Zielattribut* angegeben. Die Teilung der Gesamtmenge an Daten kann dabei auf verschiedenen Arten erfolgen. Zwei der bekanntesten Teilungsverfahren sind der *Random Split*, auch als Simple Split oder Holdout bezeichnet, und die *k-fache Kreuzvalidierung* (engl. *k*-fold cross validation).

#### Random Split

Beim Random Split wird aus der Gesamtmenge zufällig eine bestimmte Anzahl von Datensätzen (gewöhnlich $\frac{2}{3}$ der Gesamtmenge) für die Testmenge ausgewählt und anschließend die restlichen Datensätze (die verbliebenen $\frac{1}{3}$ der Gesamtmenge) der Trainingsmenge zugeordnet (siehe Abbildung 4-4). Das Modell wird dann einmal mit den Trainingsdaten erstellt und anschließend zu seiner Beurteilung auf die Testdaten angewendet.

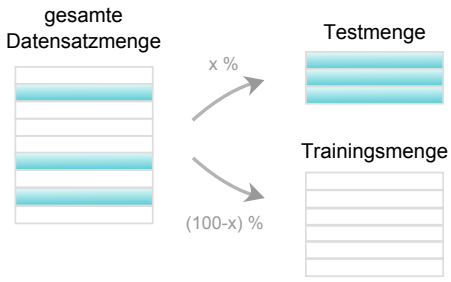



**Abbildung 4-4: Zufällige Teilung der Datensätze mittels Random Split**

### *k*-fache Kreuzvalidierung

Bei der *k*-fachen Kreuzvalidierung wird die Gesamtmenge durch zufällige Auswahl der Datensätze in *k* disjunkte Teilmengen mit annähernd gleicher Größe partitioniert und anschließend das Modell *k* mal mit diesen Mengen trainiert sowie getestet [61]. Dazu wird jeweils eine der Teilmengen als Testmenge verwendet und die anderen $k-1$ Teilmengen werden gemeinsam als Trainingsmenge eingesetzt. Abbildung 4-5 verdeutlicht den den Teilungsmechanismus bei der *k*-fachen Kreuzvalidierung. Die gesamte Genauigkeit *A* (für engl. accuracy) des Klassifika-

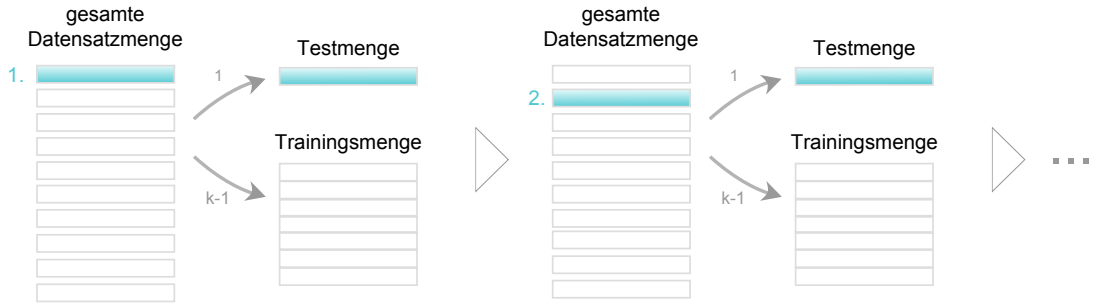



**Abbildung 4-5: Zufällige Teilung der Datensätze mittels k-facher Kreuzvalidierung**

tionsmodells wird danach berechnet, indem aus den *k* einzelnen Genauigkeitsmessungen der Durchschnitt gebildet wird [61]:

$$A = \frac{1}{k}\sum_{j=1}^{k} A_j \tag{4.7}$$

Da für gewöhnlich $k = 10$ gewählt wird, wird diese Form der Kreuzvalidierung (es existieren noch andere) auch 10-fache Kreuzvalidierung genannt [61]. Das Kreuzvalidierungsverfahren ist, da die Modellbildung und -anwendung *k* mal durchgeführt werden muss, mit einem erhöhten Rechenaufwand verbunden. Da Rechenzeit eine knappe Ressource in einem Test darstellt (für den das Verfahren später automatisch angewendet werden soll), wäre die Durchführung dieses Verfahrens im Test nicht möglich. Daher wird es auch in der Fallstudie nicht verwendet und lediglich das Random-Split-Verfahren eingesetzt.

### Stratified Sampling

Sind die in der zu prognostizierende Zielspalte vorhandenen Werte der Datensätze unausgewogen verteilt und treten einige der Werte häufiger auf als andere, so kann es zu einer Bevorzugung von Werten bei der Modellbildung kommen. Das heißt, dass einige Klassifikatoren die Eigenschaften von den Datensätzen, die in der Zielspalte einen häufig aufgetretenen Wert enthalten, höher gewichten als die Eigenschaften der Datensätze mit einem in der Zielspalte selten aufgetretenen Wert, obwohl dieser vielleicht wichtiger oder interessanter für die konkrete Fragestellung ist. Dadurch wird für neue Datensätze überwiegend ein Wert prognostiziert, der auch in der Trainingsmenge häufig auftrat. Um dies zu vermeiden, ist eine Auswahl von Datensätzen für die Trainingsmenge notwendig, bei denen die Werte in der Zielspalte alle mit der gleichen Häufigkeit auftreten. Diesen Verteilungsmechanismus bezeichnet man als *Stratified Sampling* (geschichtete Stichprobenentnahme). Die Vorgehensweise ist in Abbildung 4-6 dargestellt. Dabei wird zuerst

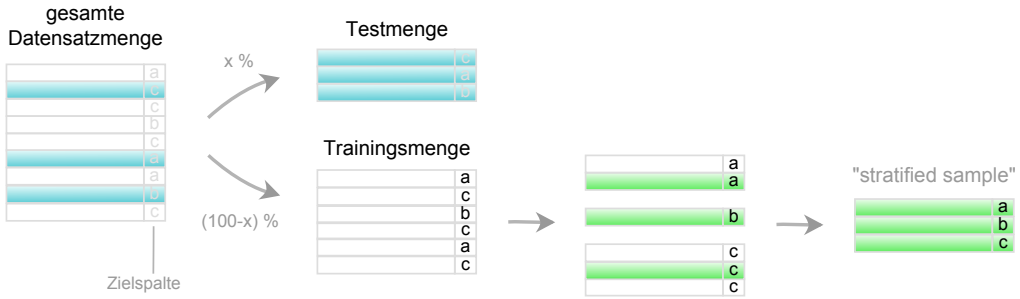



**Abbildung 4-6: Zufällige Teilung der Datensätze mittels Stratified Sampling**

wieder ein Prozentsatz von Datensätzen zufällig aus der ursprünglichen Menge, das heißt mit der ursprünglichen Häufigkeitsverteilung der Zielspaltenwerte, als Testmenge entnommen, da anderenfalls die unabhängige Bewertung der Prognosegüte des Modells nicht mehr gegeben wäre. Der für die Trainingsmenge verbleibende Teil der Datensätze wird anhand von den, in der Zielspalte enthaltenen Werten in Gruppen unterteilt und aus jeder Gruppe zufällig diejenige Anzahl von Datensätzen ausgewählt, die mit der Anzahl von Werten in der kleinsten Datensatzgruppe übereinstimmt. Auf diese Weise wird bei der Modellbildung auf alle Datensätze in der Trainingsmenge, beziehungsweise deren Eigenschaften, das gleiche Gewicht gelegt.

## 4.6.4 Modellbildung

Nachdem das Test-Design definiert wurde, kann die eigentliche Datenanalyse beginnen. Diese verläuft nach dem in Abschnitt Theoretischer Hintergrund des gewählten Verfahrens vorgestellten Prinzip zur Erstellung eines Entscheidungsbaumes.

Dem Klassifikator können für die Baumerstellung Parameter übergeben werden: allgemeine Parameter, die für alle Typen von Klassifikationsalgorithmen gelten, und spezielle Parameter für Baumklassifikationsalgorithmen (z.B. ID3, CART, CHAID, C5.0 [60]). Die einstellbaren Parameter sind abhängig vom verwendeten Data Mining-Werkzeug. So sollen an dieser Stelle einige Beispiele für Modellparameter gegeben werden.

Zu den allgemeine Parametern zählt die Selektion der für die Erstellung des Klassifikations-

baumes relevanten Merkmale der Objekte, indem diese aktiviert oder inaktiviert werden. Die Begrenzung der Merkmale dient der Verringerung der Gefahr, dass das Modell übertrainiert wird. Dies macht sich dann bemerkbar, wenn das Modell in Bezug auf die Trainingsdaten sehr präzise Vorhersagen der Klassenzugehörigkeit trifft, jedoch die Vorhersagegenauigkeit bei den Testdaten rapide abnimmt. Die Ursache dieses Phänomens der Übertrainierung (*Overfitting*) liegt in der Verwendung einer sehr großen Anzahl von Merkmalen bzw. der Verwendung von Merkmalen, die eher unwichtige Eigenschaften der Objekte repräsentieren, das heißt deren Werte einen geringen Informationsgehalt bezüglich der Data-Mining-Ziele besitzen. Diese vielen oder unwichtigen Merkmalswerte führen zu einer Überanpassung des Entscheidungsbaumes und der Entscheidungsregeln an die Eigenschaften der Objekte in der Trainingsmenge, das Modell „spezialisiert“ sich gewissermaßen auf die Trainingsdatensätze. Sobald dann ein neues Objekt (zum Beispiel aus der Testmenge) mit nicht exakt den vom Modell „auswendig“ gelernten Eigenschaften einer Klasse zugeordnet werden soll, kommt es an den Knoten des Baumes zu Entscheidungsfehlern und damit zu einer falschen Klassenzuordnung des neuen Objektes. Ein gewisses Maß an Überanpassung wird es bei realen Daten immer geben, da die Trainingsdatensätze nicht alle existierenden Objekte mit ihren möglichen Eigenschaften repräsentieren können. Ein weiterer allgemeiner Parameter stellt die Festlegung des Zielattributes dar, dessen Werte (Klassenbezeichnungen) vom Modell für die einzelnen Objekte vorherzusagen sind.

Zur Kategorie der speziellen Parameter zählt die Wahl der maximal benötigten Verringerung der Unreinheit, welche bei der Bildung des Entscheidungsbaumes als Abbruchkriterium dient. Sobald an einem Knoten des Baumes der festgelegte Wert erreicht ist, wird an dieser Stelle das weitere Splitten gestoppt und anstatt neuer innerer Knoten werden für die erzeugten Teilmengen direkt Blätter mit den Klassenbezeichnungen angehängt. Die Festlegung dieses Wertes dient daher der Begrenzung des Baumes, bezeichnet als *Pruning*, zur Vermeidung der eben beschriebenen Überanpassung des Modells an die Trainingsdaten. Auch die Festlegung der maximalen Tiefe des Klassifikationsbaumes dient dem Pruning zur Vorbeugung von Overfitting, da bei Erreichen der Knoten in der angegebenen Baumebene die weitere Partitionierung der Datensatzmenge abgebrochen wird und ebenfalls anstatt innerer Knoten die Blätter angehängt werden. Ein zweiter Grund für die Angabe dieses Wertes ist die Verringerung der Komplexität eines Baumes, um eine Interpretation der Ergebnisse nicht unnötig aufwendig zu gestalten. Möglich ist außerdem die Definition einer *Kostenmatrix* zur Gewichtung von Klassifizierungsfehlern. Die Klassen (Zielwerte), deren korrekte Klassifizierung von größerer Bedeutung ist im Vergleich zu anderen Klassen, werden als *positiv* bezeichnet und die weniger wichtigen Klassen als *negativ*. Das Ziel der Angabe der Gewichtung ist also, dass der Klassifikationsalgorithmus die positiven Zielwerte trotz ihrer eventuell geringen Auftrittshäufigkeit bei der Modellerstellung gleichberechtigt behandelt und deren Eigenschaften bei der Baumerstellung nicht vernachlässigt.

### 4.6.5 Modellbeurteilung

Nachdem einige Modelle unter Variation der verschiedenen, dem Klassifikationsalgorithmus zu übergebenden, Parametern erstellt wurden, können diese hinsichtlich der Qualität ihrer Vorhersage beurteilt und verglichen werden. Ein Ziel dessen ist die Abschätzung der sinnvollen Einsetzbarkeit des oder der gebildeten Modelle in der realen Testumgebung. Das zweite Ziel ist die Bildung der Grundlage zur Auswahl des für den späteren produktiven Einsatz im Rahmen der Testauswertung am besten geeigneten Modells aus den verfügbaren, also desjenigen mit

der höchsten Prognosequalität. Zur Beurteilung der Modelle steht in der Literatur eine große Auswahl an verschiedenen technischen Bewertungskriterien zur Verfügung. Im Folgenden erfolgt eine Beschränkung auf die Bestimmung der Klassifikationsfehler mittels einer so genannten *Konfusionsmatrix* (oder auch Fehlermatrix), auf die statistischen Gütekriterien der *Genauigkeit* (auch Richtig-Positiv-Rate oder Trefferquote) und *Zuverlässigkeit* des Modells sowie auf die Interpretation des *Gains-Chart*, eines Diagramms zur graphischen Darstellung des Verhältnisses zwischen der Anzahl der Testfälle und ihrer kumulierten Fehlerentdeckungswahrscheinlichkeit. Die Grundlage zur Beurteilung des Modells bildet das gewählte Test-Design. Einige der möglichen Test-Design-Formen wurden im Abschnitt Erstellung des Test-Designs vorgestellt. Wie bereits erwähnt, werden mit Hilfe der Objekte in der Trainingsmenge ein Entscheidungsbaum und ein Klassifikationsmodell, zur Präsentation des Entscheidungsbaumes und zur Vorhersage der Klassenzugehörigkeit von Testfällen basierend auf dem Baum bzw. auf den aus ihm abgeleiteten Entscheidungsregeln, erstellt. Anschließend wird das so erstellte Modell auf die Testfälle der Testmenge angewendet, um deren Klassenzugehörigkeit zu bestimmen. Anhand des Vergleichs der prognostizierten mit den realen Zielwerten der Testfälle (fehlgeschlagen oder nicht), können die Beurteilungskriterien berechnet und somit das Modell hinsichtlich seiner Leistungsfähigkeit bewertet werden.

**Fehlermatrix**

Eine Fehlermatrix ist ein Mittel zur Analyse der Leistung des Klassifikators hinsichtlich der korrekten Klassifizierung von Objekten (Testfällen). In ihr kann übersichtlich dargestellt werden, wie häufig die Angehörigen einer Klasse $X$ dieser korrekt zugeordnet wurden oder wie häufig sie einer falschen Klasse zugeteilt wurden. Der Aufbau einer Fehlermatrix zur Aufschlüsselung der Korrekt- oder Fehlklassifikationen bei der Einordnung der Testfälle in „fehlgeschlagene“ und „erfolgreiche“ ist in Tabelle 4-3 zu sehen (Bezeichnungen gelten nur für die zweiwertige Klassifikation, also bei binären Entscheidungsbäumen). Die Erstellung einer Fehlermatrix kann dabei sowohl nach der Trainings- als auch nach der Testphase erfolgen, wobei erstere nur in eingeschränktem Maße aussagekräftig ist, da es im Falle einer Übertrainierung des Modells zu einer Verzerrung der Korrektklassifikationen hin zum Positiven kommt.

| | | Vorhergesagte Klasse | |
|---|---|---|---|
| | | fehlgeschlagen | erfolgreich |
| Tatsächliche Klasse | fehlgeschlagen | richtig-positive ($T_p$) | falsch-negative ($F_n$) |
| | erfolgreich | falsch-positive ($F_p$) | richtig-negative ($T_n$) |

**Tabelle 4-3: Aufbau einer Fehlermatrix**

Die Zeilen der Fehlermatrix entsprechen der ursprünglichen (tatsächlichen) Einordnung der Testfälle in die vorhandenen Klassen und die Spalten der Einordnung der Testfälle in die beiden Klassen durch den automatischen Klassifikator. Die Klasse der „fehlgeschlagenen“ Testfälle, also die wichtigere von beiden Klassen (die Testfallpriorisierung basiert auf den Fehlschlägen), wird, wie bereits im vorherigen Abschnitt erwähnt, als *positive* Klasse bezeichnet und die Klasse mit den „erfolgreichen“ Testfällen als *negative* Klasse. Handelt es um ein Klassifikationsproblem mit mehr als zwei Klassen, kann diese Bezeichnung nur dann beibehalten werden, wenn eine

der Klassen als die wichtigste identifiziert werden kann. Diese wird dann als positive und die restlichen Klassen gemeinsam als negative bezeichnet [62]. Bei einer Übereinstimmung der tatsächlichen und der vorhergesagten Klasse eines Testfalls erfolgt die Kennzeichnung mit „richtig“ und andernfalls mit „falsch“. Wurden die in Wirklichkeit fehlgeschlagenen Testfälle durch den Klassifikator der Klasse der erfolgreichen zugeordnet, handelt es sich um eine falsch-positive Klassifizierung und bei Zuordnung der tatsächlich erfolgreichen Testfälle zu den fehlgeschlagenen als falsch-negative Klassifizierung bezeichnet werden. Erstere sind besonders kritisch, da das Auslassen von als erfolgreich vorhergesagten Testfällen, die in Wirklichkeit fehlgeschlagen, mit höheren Folgekosten (z.B. der geminderten Kundenzufriedenheit aufgrund von Ausfällen der Software) verbunden ist, als die unnötige Ausführung von Testfällen die im Ergebnis erfolgreich sind (Testkosten in den meisten Fällen geringer als Fehlerfolgekosten).

Diese Fehlermatrix kann genutzt werden, um die Genauigkeit des erstellten Vorhersagemodells zu bestimmen.

#### Genauigkeit der Vorhersage der Klassenzugehörigkeit

Die Genauigkeit der Modellanwendung auf neue Daten wird auf folgende Weise (basierend auf der Fehlermatrix) berechnet [61]:

$$Genauigkeit = \frac{T_p}{(T_p + F_n)} \tag{4.8}$$

Sie beschreibt allgemein das Verhältnis der Anzahl korrekt in die positive Klasse eingeordneter Objekte zur Summe aller ursprünglich der positiven Klasse zugeordneten Objekte und entspricht daher der Wahrscheinlichkeit einer korrekten Zuordnung der Objekte zur positiven Klasse.

#### Zuverlässigkeit der Vorhersage

Ein zweites Kriterium zur Modellbewertung ist die Zuverlässigkeit, deren Wert gebildet wird anhand des Vergleichs der nach dem Training berechneten Genauigkeit mit der nach dem Test des Modells berechneten Genauigkeit. Sie stellt also ein Maß für die Anwendbarkeit des Modells auf neue Datensätze dar.

#### Gains-Chart

Der Klassifikator gibt, zusammen mit der für einen Testfall der Testmenge vorhergesagten Klasse, zusätzlich noch die Wahrscheinlichkeit (Konfidenz) der korrekten Einordnung des Testfalls in die Klasse an. Diese Wahrscheinlichkeit entspricht bei Testfällen ihrer Fehlerentdeckungswahrscheinlichkeit, da mit jedem Fehlschlagen eines Testfalls die Existenz von Fehlern in der Software angezeigt wird. Anhand dieser Fehlerentdeckungswahrscheinlichkeit werden die Testfälle vom Klassifikator absteigend sortiert und das Verhältnis zwischen einem bestimmten Prozentsatz der Testfälle und der kumulierten Fehlerentdeckungswahrscheinlichkeit dieser in eine Rangfolge gebrachten Testfälle im so genannten Gains-Chart dargestellt (siehe Abbildung 4-7). Ohne das

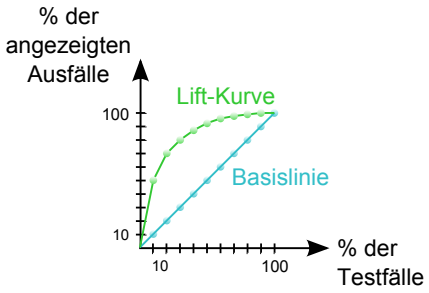



**Abbildung 4-7: Beispiel eines Gains-Chart**

Klassifikationsmodell besteht ein linearer statistischer Zusammenhang (entspricht der Basislinie im Gains-Chart) zwischen dem Prozentsatz der durchgeführten Testfälle und dem Prozentsatz der entdeckten Fehler, welcher der kumulierten Fehlerentdeckungswahrscheinlichkeit der durchgeführten Testfälle entspricht. Das bedeutet, dass bei einer zufälligen Reihenfolge der Testfallausführung, zum Beispiel, nach 10% der durchgeführten Testfälle theoretisch auch 10% der gesamten Fehler entdeckt werden.

Bei der Anwendung des Modells verändert sich dieser Zusammenhang (repräsentiert durch die Lift-Kurve im Gains-Chart). Das bedeutet, wenn die Testfälle in der vom Klassifikator gebildeten Reihenfolge ausgeführt werden, können nach 10% der durchgeführten Testfälle vielleicht bereits 50% der Fehler entdeckt werden, was eine Effizienzsteigerung bei der Testausführung aufgrund des frühzeitigen Auffindens eines Großteils der Fehler zur Folge hätte. Dies ist vor allem dann wichtig, wenn Testressourcen stark begrenzt verfügbar sind, sodass ein Test möglicherweise vorzeitig abgebrochen werden muss und ein gewisser Teil der Fehler in diesem Testlauf nicht mehr gefunden und daher auch nicht beseitigt werden kann. Durch die Verwendung eines präzisen Modells kann diese Anzahl nicht gefundener Fehler reduziert werden.

Die Lift-Kurve im Gains-Chart stellt eine Verbindung von Punkten mit den Koordinaten $(x, y)$ dar, welche folgendermaßen definiert werden können:

$$x = P(fehlgeschlagen), y = Genauigkeit \tag{4.9}$$

Der Anteil $P$ der fehlgeschlagenen Testfälle an der Gesamtanzahl der Testfälle ist, an einem beliebigen Punkt im Koordinatensystem, bestimmbar mit:

$$P(fehlgeschlagen) = \frac{(TP+FN)}{(TP+FN+FP+TN)} \tag{4.10}$$

Die Genauigkeit entspricht der Fehlerentdeckungswahrscheinlichkeit der am selben Punkt im Koordinatensystem vorliegenden Prozentsatz von Testfällen. Sie kann berechnet werden anhand der Gleichung 4.8. Der Lift stellt damit ein Maß für die Effizienz des eingesetzten Modells dar. Das bedeutet, je größer die Fläche zwischen der Basislinie und der Lift-Kurve im Gains-Chart ist, desto präziser klassifiziert das Modell neue Datensätze.

### 4.6.6 Modellverwendung

Wie bereits erwähnt, wird bei der Klassifizierung der Testfälle deren Fehlerentdeckungswahrscheinlichkeit bestimmt, um die Testfälle anhand dieser Fehlerentdeckungswahrscheinlichkeit absteigend zu sortieren. Aufgrund des im vorherigen Abschnitt gezeigten Zusammenhangs zwischen dieser Sortierung und der Erhöhung der Testeffizienz, kann die Sortierung zur Priorisierung der Testfälle im nächsten SW-Test eingesetzt werden.

In einem Gains-Chart ist eine Grenze erkennbar, ab der die Fehlerentdeckungswahrscheinlichkeit bei Zunahme der Testfallanzahl nur noch mäßig ansteigt. Diese Grenze wird dazu verwendet, um die hinsichtlich der Aufdeckung von Fehlern wichtigen Testfälle von den unwichtigen zu trennen. Die wichtigen Testfälle sollten geschlossen für den Test verwendet werden, da mit ihnen eine große Anzahl von Fehlern aufgedeckt werden kann, wohingegen die Anzahl der unwichtigen Testfälle reduziert werden kann. Anhand der Betrachtung der Wirtschaftlichkeit des Einsatzes der weniger relevanten Testfälle, sollte ein bestimmter Prozentsatz aus der Menge dieser Testfälle zufällig ausgewählt und im Test eingesetzt werden, um auf lange Sicht allen Testfällen die Chance zur Ausführung und damit zur Aufdeckung von Fehlern zu geben. Auf die Ausführung dieser unwichtigen Testfälle könnte, unter Hinnahme eines gewissen Restrisikos in Form des Verbleibens unentdeckter Fehler in der Software, aus wirtschaftlicher Sicht gänzlich verzichtet werden, wenn die Testkosten für den Einsatz dieser zusätzlichen Testfälle höher ausfallen als der Nutzen, den ihre Ausführung einbringt.

Der Prozentsatz der noch zusätzlich zu wählenden Testfälle kann demzufolge bestimmt werden, indem die zusätzlich durch diese Testfälle verursachten Testkosten gegen ihren Nutzen (Verringerung des Restrisikos) abgewogen werden. Die Bestimmung der in den Vergleich einzubeziehenden Testkosten und des ebenfalls zu betrachtenden Restrisikos, welches in Form der durch die Testfallausführung ausgebliebenen Fehlerfolgekosten quantifizierbar ist, kann mittels der Adaption einer von Rothermel et al. [47] entwickelten Formel zur Berechnung der Kosten für eine Testfallselektion erfolgen.

Sei $T$ die ursprüngliche (gesamte) Menge der $k$ Testfälle, $F(T)$ die Menge der von den Testfällen aufgedeckten Fehler mit der Anzahl $l$, $T'$ die Menge der jeweils für den Kostenvergleich herangezogenen Testfälle mit der Anzahl $m$ und $F(T')$ die Menge der von den zusätzlich selektierten Testfällen aufgedeckten Fehler mit $n$ Elementen. Des Weiteren werden folgende Größen definiert:

- $C_a(T)$ : Kosten (engl. cost) der Analyse zur Priorisierung der Testfälle
- $C_p(T)$ : Kosten für die Vorbereitung (engl. preparation) der Testdurchführung (inklusive Einrichtung der Testumgebung auf Testsystemen und Lizenzkosten für Software-Testrahmen bzw. Entwicklungskosten für eigene Lösung)
- $C_e(T)$ : Kosten der Ausführung (engl. execution) der Testfälle (enthält z.B. Betriebs- und Instandhaltungskosten des Testsystems)
- $C_c(T)$ : Kosten für die Auswertung (engl. check) der Testresultate
- $C_s(T)$ : Kosten für die Selektion der Testfälle aus der ursprünglichen Testfallmenge
- $C_f(F(T))$ : Kosten die mit Fehlern verbunden sind (Fehlerfolgekosten, z.B. erhöhte Korrekturkosten für spät im Entwicklungsprozess gefundene Fehler)

- $C_m(T)$ : Kosten für die Wartung der gesamten Testfallmenge (für spätere Ausführung der im Moment nicht selektierten Testfälle in künftigen Regressionstests

In einem ersten Schritt ist der Anteil eines Testfalls an den Kosten für Tests mit der ursprünglichen Testfallmenge zu berechnen:

$$C_i = \frac{C_a(T) + C_p(T) + C_e(T) + C_c(T) + C_s(T) + C_m(T)}{k} \tag{4.11}$$

Der Anteil eines Fehlers an den gesamten, von der Menge aller Fehler verursachten Folgekosten ist berechenbar mit:

$$C_j = \frac{C_f(F(T))}{l} \tag{4.12}$$

Die beiden Formeln können dann in einer integriert werden, um das Verhältnis zwischen den Testkosten für die Menge der selektierten Testfälle und den durch die selektierten Testfälle eingesparten Fehlerfolgekosten zu errechnen:

$$C_g = \frac{C_i \cdot m}{C_j \cdot n} \tag{4.13}$$

Diese Formel ist für verschiedene Häufigkeiten der sortierten, für die Selektion vorgesehenen Testfälle anzuwenden, um diejenige Testfallmenge $T'$ für den Test zu wählen, für die der Nutzen gerade noch die Testkosten übersteigt und für die demzufolge gilt: $C_g \leq 1$.

### 4.6.7 Wartung & Kontrolle der Modellverwendung

Die Priorisierung der Testfälle und die Grenze zur Teilung der Testfallmenge verändert sich bei jeder Neuerstellung des Klassifikationsmodells, welche zum Beispiel beim nächsten Software-Release notwendig wird. Die Anzahl der zufällig gewählten, weniger relevanten Testfälle ändert sich zusammen mit dem berechneten Kosten-Nutzen-Verhältnis. Dies ist zum Beispiel dann der Fall, wenn die Kosten für die Ausführung eines jeden Testfalls steigen (bei gestiegenen Betriebskosten des Testsystems oder ähnlichem).

Ferner zu bedenken ist, dass nach einer erfolgten Testfallselektion die Testergebnisse der nicht selektierten Testfälle im Testbericht fehlen. Damit es nicht zur Beeinflussung des Analyseergebnisses zukünftiger Testauswertungen kommt, ist in den Spalten der Tabelle mit den gesammelten Testergebnissen für die fehlenden der Wert *NULL* zu setzen, damit diese vom Klassifikationsalgorithmus bei der erneuten Erstellung eines Modells, welches zur Auswertung der Testergebnisse dient, ignoriert werden.

# 5 Fallstudie ”Entwicklungsprojekt für Data Mining-Software”

## 5.1 Business Understanding

### 5.1.1 Projektziele

Diese Arbeit beschäftigt sich zum einen mit der systematischen Analyse der Zusammenhänge zwischen den Parametern und den Resultaten von Software-Tests zur Generierung von Aussagen über die Qualität einer Software und zum anderen mit der Selektion von Testfällen anhand deren Wahrscheinlichkeit zur Fehleraufdeckung. Diese systematische Analyse wird mittels eines Testobjektes aus der Praxis exemplarisch verdeutlicht, wobei die Durchführung des Data-Mining-Analyseprojektes entsprechend dem CRISP-DM-Prozessmodell erfolgt, welches in Abschnitt 2.4 vorgestellt wurde.

Bei dem im Rahmen dieser Fallstudie verwendeten Testobjekt handelt es sich um den im Abschnitt 2.5 vorgestellten *Intelligent Miner*, die Data-Mining-Komponente innerhalb der von IBM entwickelten Data-Warehouse-Lösung InfoSphere Warehouse. Bei den Tests des Intelligent Miner werden die einzelnen Funktionen und Methoden des SQL/MM-API, sowie deren Zusammenwirken im gesamten Data-Mining-Prozess, und die Easy-Mining-Prozeduren auf ihre korrekte Funktionsweise überprüft. Auch die im Hintergrund ausgeführten Data-Mining-Algorithmen zur Erstellung der verschiedenen Arten von Data-Mining-Modellen werden in diesem Zuge auf das Vorhandensein von Fehlern geprüft.

An dieser Stelle sei der Hinweis gegeben, dass das Testobjekt zufällig selbst ein Data-Mining-Werkzeug darstellt, aber auch jedes andere Software-System als Testobjekt dienen könnte.

### 5.1.2 Beurteilung der Situation und Wahl des Data Mining-Werkzeugs

Die vorliegende Master Thesis wird, wie bereits zu Anfang erwähnt, in Zusammenarbeit mit den Abteilungen Data Mining und Quality Assurance im Bereich Information Management des Unternehmens IBM erarbeitet. Die Hauptaufgabe der Mitarbeiter der Abteilung Data Mining umfasst die Entwicklung des *Intelligent Miner*. Da aus diesem Grund von Seiten der Mitarbeiter eine umfassende und fundierte Hilfestellung bei der Arbeit mit dem Intelligent Miner möglich ist, soll dieser bei der Erprobung des vorgestellten systematischen Analyseverfahrens im Rahmen der in diesem Kapitel beschriebenen Fallstudie eingesetzt werden.

Die Besonderheit am Einsatz des Intelligent Miner als Werkzeug für die Data-Mining-Analyse liegt darin, dass er wie bereits erwähnt auch als Testobjekt dient, auf dessen Basis die Analyse stattfindet. Dies ist jedoch nicht als kritisch zu bewerten, da er keinen Einfluss auf die Ergebnisse der mit ihm durchgeführten Tests hat, denn diese werden von einem speziellen Testwerkzeug vorgenommen, sondern lediglich bei der Analyse seiner Qualität, basierend auf den zuvor unabhängig erzeugten Testergebnissen, mitwirkt.

## 5.2 Data Understanding

Nachdem die Schaffung der Grundvoraussetzung für die praktische Erprobung der Analyseverfahren, die Wahl des Data-Mining-Werkzeugs, abgeschlossen ist, folgt nun die zweite Phase des Data-Mining-Prozesses, welche auch als *Data Understanding* bezeichnet wird.

Innerhalb dieser Phase erfolgt eine Zusammenstellung der für die Analyse benötigten Daten. Dazu werden die Daten erfasst und auf ihre Vollständigkeit hin überprüft. Bei Unvollständigkeit sind die fehlenden Daten zu generieren oder aus zusätzlichen Quellen zu beschaffen. Liegen alle Daten vor, kann man sich in einem nächsten Schritt mit ihren Eigenschaften, wie der Verteilung von Attributwerten, vertraut machen und sie auf Probleme bezüglich ihrer Qualität hin untersuchen, wie Unstimmigkeiten oder fehlende Werte. Auf die Beschreibung gefundener Probleme wird in diesem Abschnitt verzichtet, da diese zusammen mit den Maßnahmen zur Behandlung der Probleme im nächsten Abschnitt zur Datenvorbereitung beschrieben werden.

### 5.2.1 Erfassung der Rohdaten

Da die Untersuchung der Software-Qualität und die Priorisierung und Selektion von Testfällen auf einer Analyse der Zusammenhänge zwischen den Testfalleingaben, also den einzelnen Parametern der Testfälle, und dem für jeden Testfall vorliegenden Ergebnis eines funktionalen Regressionstests basiert, sind diese Informationen für den Intelligent Miner in einem ersten Schritt aus den verschiedenen Quellen zu beschaffen. Tabelle 5-1 zeigt die für diese Untersuchung benötigten Daten und die zugehörige Datenquelle.

| Daten | Quelle | Quelle vorhanden? |
|---|---|---|
| Testfallparameter | Testdefinitionsdateien | ja |
| Eigenschaften der Eingabedaten | Eingabedatendateien | ja |
| Eigenschaften der Eingabemodelle | Eingabemodelldateien | ja |
| SW-Funktionsparameter | Funktionsparameterdateien | ja |
| Typ der geprüften SW-Funktion | API-Dokumentation | ja |
| Testfallergebnisse | Log-Dateien d. Tests | nein |

**Tabelle 5-1: Übersicht der benötigten Daten und deren Quelle**

#### Testfallparameter

Daraus ist erkennbar, dass einige Daten vorliegen, andere noch beschafft beziehungsweise generiert werden müssen. Die Testfälle gehören zur Gruppe der bereits vorhandenen Daten. Für funktionale Tests des Intelligent Miner liegen etwa 3000 Testfälle vor, die über seinen Lebenszyklus hinweg erstellt wurden. Diese besitzen den in Abbildung 5-1 dargestellten grundlegenden Aufbau. Die Definition eines oder mehrerer Testfälle erfolgt in dedizierten Testfalldefinitionsdateien, aus denen die Testfallparameter zu extrahieren sind. Aufgrund der großen Menge an Testfällen wird für letzteres eine automatisierte Lösung angestrebt, die den verbundenen zeitlichen Auf-

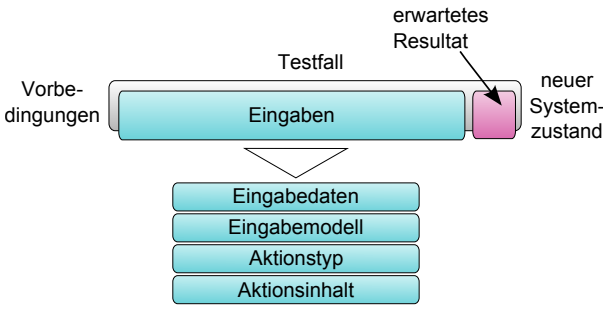



**Abbildung 5-1: Aufbau der Testfälle des Intelligent Miner, Darstellung an [63] angelehnt**

wand enorm reduziert. Eine entsprechendes Software-Werkzeug, das den Inhalt von Textdateien analysieren und die darin enthaltenen Informationen (in diesem Fall die Testfallparameter) für die weitere Verarbeitung zur Verfügung stellen kann, wird allgemein als *Parser* bezeichnet. Es existieren Parser für Textdateien mit einer typischen formalen Struktur des Inhalts, wie Dateien im XML[23]-Format, jedoch konnte zum aktuellen Zeitpunkt kein Parser für die Testfalldefinitionsdateien mit ihrer individuellen formalen Struktur gefunden werden. Aus diesem Grund wurde von der Autorin ein spezieller, auf der Programmiersprache Java basierender Parser zum Extrahieren der Testfallparameter aus den Definitionsdateien entwickelt. Die vom Parser aus oder zu den Definitionsdateien abgeholten Informationen sind:

- Name des Testfalls
- Name der Testfalldefinitionsdatei
- Pfadname der CSV-Datei mit den Eingabedaten
- Pfadname der PMML-Datei mit dem Eingabemodell (Data-Mining-Modell)
- Typ der durchzuführenden Aktion
- Aktionsanweisung (z.B. SQL-Befehl oder Pfadname einer Datei mit Funktionsparametern)
- erwartetes Ergebnis (Referenzwert oder Pfadname zu einer Datei mit Referenzdaten)

Aus dem Testfallname, dem Name der Testfalldefinitionsdatei oder der Aktionsanweisung (je nachdem was die gewünschte Information enthält) wird außerdem der Name und der Typ der geprüften Data-Mining-Funktion sowie der zusammen mit der Funktion eingesetzte Data-Mining-Algorithmus ermittelt. Nun stellt sich die Frage, wie mit den ausgelesenen Daten generell weiter verfahren wird. Die Lösung ergibt sich aus der Tatsache, dass die Intelligent Miner-Funktionen zur Erstellung und Anwendung von Data-Mining-Modellen innerhalb einer DB2-Datenbank ausgeführt werden und daher alle benötigten Daten in der für das Data Mining vorbereiteten Datenbank in Form von Tabellen abzuspeichern sind. Um die Daten strukturiert und mit System abzulegen und dadurch Inkonsistenzen und Redundanzen zu vermeiden, ist die Erstellung eines konzeptionellen Datenbankschemas [64] wichtig. Dieses dient der Festlegung der Entitätstypen (Tabellen) zur Aufnahme von Entitäten (Datensätzen) mit gemeinsamen Attributen (Spalten) und der Definition der Beziehungen (Relationen) zwischen den Entitätstypen. Ein Datenbankschema kann mit Hilfe eines Entity-Relationship-Modells (ERM) [64] modelliert und anhand eines Entity-Relationship-Diagramms (ERD) dargestellt werden, wie zum Beispiel für diese Fallstudie im Anhang auf Seite 83. Zur Erstellung der im Datenbankschema vorgesehenen Tabellen innerhalb der Datenbank

[23] Extensible Markup Language (engl. für erweiterbare Auszeichnungssprache), das XML-Dateiformat dient zur Speicherung hierarchisch strukturierter Daten in Textform

ist neben der Kenntnis der Attributnamen auch das Wissen über die zugehörigen Attributtypen (Datentypen) erforderlich. Diese Datentypen werden automatisch von einem Algorithmus im Parser anhand einer Analyse der ausgelesenen Datenwerte ermittelt, was im Vergleich zu einer manuellen Bestimmung mit einer enormen Aufwands- und Zeitersparnis verbunden ist. Um die Tabellen schließlich in der Datenbank anzulegen und die Daten in diese zu übertragen existieren verschiedene Möglichkeiten. Zum einen könnte der Parser die Daten und die zugehörigen Spaltennamen bzw. -typen in getrennten CSV-Dateien ablegen, damit die Tabellenerstellung und Datenspeicherung mit Hilfe des Datenbank-Administrationswerkzeugs (durch den Import der CSV-Dateien) erfolgen kann. Da letzteres jedoch bei häufigen Änderungen in den Daten (z.B. zur Erhöhung der Datenqualität) zu einem enormen Arbeitsaufwand führen würde, wird der Weg der Automatisierung gewählt. Dazu werden die Tabellen anhand ihrer Spalteninformationen direkt vom Parser mittels SQL-Anweisungen über eine JDBC-Verbindung in der Datenbank angelegt und die Daten übertragen. Für die Testfälle und deren Parameter wird auf diese Weise eine Tabelle mit der Bezeichnung „TESTCASE“ erstellt, deren Attributnamen in Abbildung 5-2 dargestellt sind.

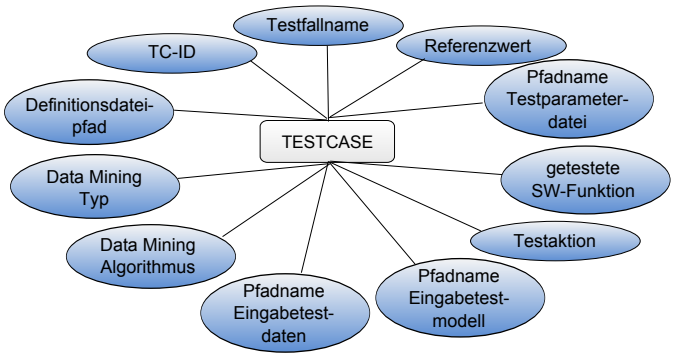



**Abbildung 5-2: Entitätstyp „TESTCASE“ zur Aufnahme der Testfälle und ihrer Parameter**

Weitere Daten, die in der späteren Analyse Verwendung finden sollen, betreffen die Eigenschaften der für die Tests zur Verfügung stehenden Eingabedaten, die in den Testfällen durch Dateipfadnamen referenziert werden.

### Eigenschaften der Eingabedaten

Die Eigenschaften der Eingabedaten der Testfälle werden in die Analyse aufgenommen, um eventuell feststellen zu können, auf welche Merkmale der Eingabedaten die Funktionen des Testobjektes mit einem Fehlverhalten reagieren. Denn sobald für mehrere Eingabedaten mit einer äquivalenten Ausprägung eines bestimmten Merkmals anhand der Data-Mining-Analyse eine Korrelation zum Fehlschlagen einer Software-Funktion festgestellt werden kann, wird davon ausgegangen, dass eine Schwäche in der Software-Funktion offenbart wurde, da die Verarbeitung dieses Merkmals der Eingabedaten, zum Beispiel das Vorhandensein sehr vieler leerer Zeichenketten, in der Funktion beziehungsweise dem von ihr ausgeführten Algorithmus nicht korrekte implementiert wurde. Die nachträgliche Implementierung einer korrekten Datenverarbeitung wäre dann notwendig. Merkmale der Eingabedaten, die dazu geeignet sein könnten, ein solches systematisches Fehlverhalten einer Software-Funktion aufzudecken, sind:

- Größe der Daten (Produkt aus Anzahl der Zeilen und Anzahl der Spalten)
- Anteil fehlender Werte (leere Zeichenketten)
- Anteil kategorischer bzw. numerischer Werte

Diese Liste könnte beliebig erweitert werden, soll jedoch zur Demonstration des Analyseverfahrens genügen. Die Bestimmung geeigneter Merkmale der Eingabedaten sollte ohnehin für jedes Software-Projekt individuell erfolgen. Die Ermittlung der Werte, welche die Eingabedaten hinsichtlich der vorgestellten Merkmale annehmen, könnte wieder manuell, durch Betrachten der Eingabedatendateien, erfolgen oder automatisiert mit Hilfe eines speziellen Algorithmus im Parser. Aus den bereits an früherer Stelle genannten Gründen wird eine Automatisierung bevorzugt. Zur Aufnahme der Werte, die mit Hilfe des Parsers für die Merkmale der einzelnen Eingabedaten ermittelt wurden, wird von ihm die Datenbank-Tabelle „INPUTDATA“ erstellt, welche die in Abbildung 5-3 gezeigten Attribute trägt.

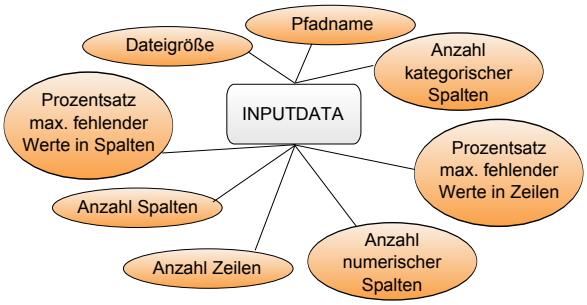



**Abbildung 5-3: Entitätstyp „INPUTDATA“ zur Speicherung der Eigenschaften der Eingabedaten**

### Eigenschaften der Eingabemodelle

Mit dem Fehlschlagen einer Software-Funktion in Zusammenhang stehen könnten auch die Eigenschaften der, von den Testfällen als Eingabe für die Funktionen verwendeten, Data-Mining-Modelle. Testfälle dieser Art dienen zum Beispiel der Prüfung von Scoring-Funktionen des Intelligent Miner, die ein zuvor mit historischen Daten erstelltes Data-Mining-Modell auf unbekannte Daten anwenden. Inkompatibilitäten zwischen der PMML-Version eines Modells und dem verwendeten Scoring-Algorithmus, weil dieser mit neu eingeführten Syntax-Elementen vielleicht noch nicht umgehen kann, könnten zum Beispiel in einem Versagen der Funktion resultieren. Daher wird dieses Merkmal in der Analyse berücksichtigt. Zur Bestimmung der Version eines PMML-Modells wird mit Hilfe des Parsers die entsprechende Zeichenkette aus der Modelldatei ausgelesen. Zusätzlich können der Data-Mining-Typ des Modells und der zur Modellerstellung verwendete Algorithmus aus dem Modell gelesen werden. Die entsprechend benannten Attribute der dafür vorgesehenen Tabelle „INPUTMODEL“ sind, zusammen mit dem Attribut für den Name der jeweiligen Modelldatei, in 5-4 abgebildet.

### SW-Funktionsparameter

Die Anzahl der an eine SW-Funktion übergebenen Parameter könnte ebenfalls eine Schwäche der Funktion offenbaren, und zwar dann, wenn diese nicht mit einer potenziell falschen Anzahl der übergebenen Parameter umgehen kann. Im Zuge der Datenerfassung werden erst einmal die

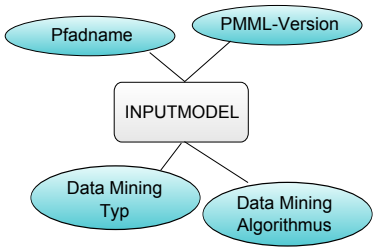



**Abbildung 5-4: Entitätstyp „INPUTMODEL“ mit den Eigenschaften der Eingabemodelle**

Namen der übergebenen Funktionsparameter aus der entsprechenden Parameterdatei ausgelesen und diese erst später in der Datenvorverarbeitungsphase zu einem numerischen Wert (der Anzahl) transformiert. Auf diese Weise könnten die Funktionsparameter noch zur Durchführung anderer Data-Mining-Analysen, zum Beispiel anhand des Einsatzes von Assoziationsverfahren, verwendet werden. Damit soll lediglich aufgezeigt werden, dass die Ergebnisse der Datenerfassung so allgemein wie möglich gehalten werden sollten, genutzt werden diese zusätzlichen Informationen im Rahmen dieser Fallstudie nicht. Daher weist die entsprechende Tabelle „SWFUNCTPAR“ ein Attribut für die Pfadnamen der Parameterdateien und ein Attribut für die in den Dateien enthaltenen Funktionsparameter auf, wie in Abbildung 5-5 zu sehen ist.

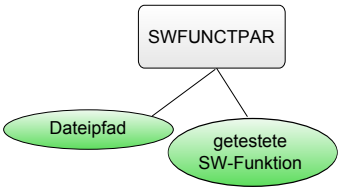



**Abbildung 5-5: Entitätstyp „SWFUNCTPAR“ für die Funktionsparameter**

### Typ der geprüften SW-Funktion

Der Typ der getesteten SW-Funktion wird bestimmt, um nicht nur die Schwächen einzelner SW-Funktionen zu enttarnen, sondern auch die verschiedener Gruppen von SW-Funktionen, zum Beispiel aller Funktionen die den Export von Modellen aus der Datenbank in eine Datei auf der Festplatte betreffen, unabhängig davon ob es sich zum Beispiel um den Export von Clustering- oder Assoziationsmodellen handelt. Die Zugehörigkeit der Funktionen zu Gruppen (Taxonomie) wird auf manuelle Weise der API-Referenz [65] des Intelligent Miner entnommen. Die Attribute der zugehörigen Tabelle „TAXAPI“ werden in Abbildung 5-6 vorgestellt.

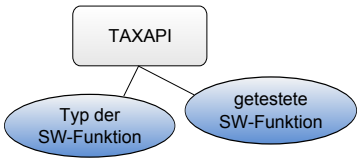



**Abbildung 5-6: Entitätstyp „TAXAPI“ mit der Taxonomie für SW-Funktionen**

### Testfallergebnisse

Wie aus der Tabelle 5-1 ersichtlich wird, liegen die benötigten Log-Dateien mit den Testfallergebnissen von mehreren zurückliegenden Entwicklungsständen (Versionen) des Testobjektes nicht (mehr) vor und sind daher mittels der erneuten Durchführung von Tests zu beschaffen. Der Test erfolgt dazu mit den letzten sechs Intelligent Miner-Versionen, die einen Entwicklungszeitraum von drei Jahren umfassen. Entsprechend dem allgemeinen Ablauf eines Testprozess, wie er auf Seite 24 beschrieben wurde, erfolgt zu Beginn eine Planung der Testaufgaben und des Testziels sowie der benötigten Ressourcen, Zeit und Hilfsmittel. Diese Planung erfolgt in der Praxis, auf die Einbindung eines formalen Testkonzepts wird an dieser Stelle jedoch verzichtet, da es keine Relevanz bezüglich der Ziele des Data-Mining-Projektes aufweist. Die nach der Planung folgende Definition des Testinhalts in Form von Testfällen ist nicht notwendig, da aus früheren Test bereits genügend vorliegen.

Zur Vorbereitung auf die Tests wird die benötigte Testumgebung (siehe Abbildung 5-7), welche einen Testrahmen, die Testfälle, die Eingabe- und Referenzdaten sowie das Testobjekt (den Intelligent Miner) enthält, auf mehreren Testservern mit verschiedenen Betriebssystemen eingerichtet. Dabei werden verschiedene Varianten/Derivate der Betriebssysteme Microsoft Windows, Linux und Unix eingesetzt.

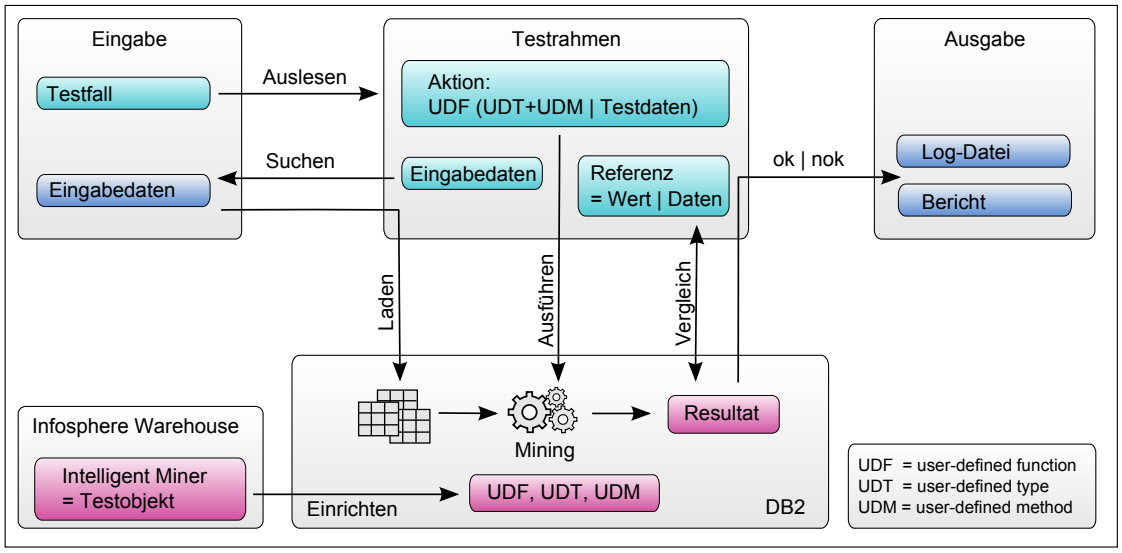



**Abbildung 5-7: Testumgebung zur Generierung der für die Data-Mining-Analyse benötigten Testergebnisse**

Während der Testdurchführung wird jeder einzelne Testfall vom Testrahmen aufgerufen und dessen Parameter ausgelesen. Anhand der im Testfall enthaltenen Pfadangaben zu den Eingabedatendateien werden diese auf dem Testsystem gesucht und, nach Erstellung entsprechender Tabellen, in die DB2-Datenbank geladen. Anschließend wird die im Testfall angegebene Testaktion ausgeführt, das heißt die zu testende API-Funktion des Intelligent Miner wird nach Übergabe der Funktionsparameter in der DB2-Datenbank aufgerufen. Bei der geprüften API-Funktion handelt es sich entweder um eine benutzerdefinierte Funktion der SQL/MM-API oder um eine Gespeicherte Prozedur der Easy-Mining-API. Während der Ausführung der Funktion wird der Testverlauf vom Testrahmen kontinuierlich durch Statusnachrichten in der Log-Datei dokumentiert. Die Ausgabe

der getesteten Funktion, zum Beispiel ein erstelltes Data-Mining-Modell, wird dann in der Datenbank gespeichert und vom Testrahmen mit dem erwarteten Ergebnis, der Referenz, verglichen. Bei Übereinstimmung von Resultat und Referenz wird ein Erfolg in der Log-Datei vermeldet und andernfalls ein Versagen, das auf einen Fehler in der Funktion hindeutet. Nachdem alle Testfälle ausgeführt wurden, wird ein abschließender Testbericht erstellt, der die Endresultate der Tests (Liste der bestandenen und fehlgeschlagenen Testfälle) zusammenfassend darstellt.

Aus den abschließenden Testberichten kann dann das Ergebnis eines jeden Testfalls für alle getesteten Versionen des Intelligent Miner und für alle eingesetzten Betriebssystem zusammen mit seinem Name, dem Name seiner Definitionsdatei sowie der SW-Versions- und Betriebssysteminformation ausgelesen werden. Dazu ist der entsprechende Algorithmus im Parser an die formale Testberichtstruktur anzupassen. Die Fehlermeldungen der Software-Funktionen werden in der späteren Data-Mining-Analyse nicht berücksichtigt, da es sich dabei um Informationen handelt die für neue Testfälle oder neue SW-Versionen noch nicht vorhanden sind und daher das Analyseergebnis durch Übertrainieren (overfitting) des Modells verfälschen würden. Das Gesamtresultat für einen Testfall wird vom Parser ermittelt anhand der Auswertung der Testergebnisse über alle eingesetzten SW- und Betriebssystem-Versionen hinweg. Das bedeutet, sobald ein Testfall für mindestens eine Software-Version auf wenigstens einem Betriebssystem fehlgeschlagen ist, gilt dessen Gesamtergebnis als fehlgeschlagen (engl. failed) und andernfalls als bestanden (engl. passed). Die extrahierten und berechneten Daten werden anschließend wieder direkt vom Parser in eine Datenbank aufgenommen. Die Tabelle „TESTRESULT“ besitzt dazu die in der Abbildung 5-8 dargestellte Attribute.

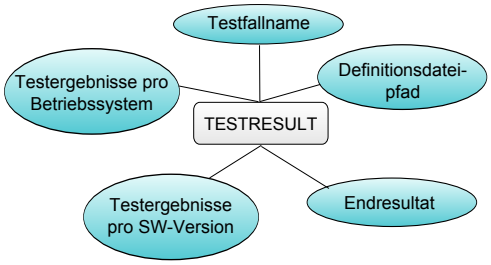



**Abbildung 5-8: Entitätstyp „TESTRESULT“ mit den für jeden Testfall und jede SW-/BS-Version erhaltenen Ergebnissen**

Nachdem alle notwendigen Daten erfasst sind, werden sie zur Sicherung in CSV-Dateien abgelegt. Diese Aufgabe kann mit Hilfe des Datenbank-Administrationswerkzeugs manuell durchgeführt oder automatisch vom Parser (nach der Entwicklung des entsprechenden Codes) übernommen werden, wobei letzterem aufgrund der Zeitersparnis bei häufigen Änderungen der Daten und den damit verbundenen erneuten Sicherungen der Vorzug zu geben ist.

### 5.2.2 Erforschen der Daten

Nachdem die Daten erfasst sind, können diese genauer untersucht werden, zum Beispiel hinsichtlich ihrer univariaten (ein Merkmal betrachtet), bivariaten (gleichzeitige Betrachtung von zwei Merkmalen) oder multivariaten (Betrachtung mehrerer Merkmale zur gleichen Zeit) Verteilung. Zur Betrachtung dieser Werteverteilung sind im Design Studio entsprechende Funktionen verfügbar. Eine mögliche Untersuchung betrifft die Verteilung der Testfälle über die Intelligent

Miner-Funktionen. Dies ist von Bedeutung für die spätere Analyse, denn wenn für einige Funktionen besonders viele Testfälle definiert sind, können diese Funktionen auch verhältnismäßig viele Ausfälle erzeugen und erhalten daher in der Analyse eine besonders hohes Gewicht, was zu einer Verschiebung des Analyseergebnisses führen kann. Eine annähernde Gleichverteilung der Funktionen ist herstellbar durch eine entsprechende Selektion der Datensätze. Da jedoch im Fall des Intelligent Miner einige Funktionen teilweise nur einen Testfall erhalten, würde die Selektion einer gleich großen Anzahl von Datensätzen für jede Funktion zu einem wenig sinnvollen Analyseergebnis führen. Daher werden die Unterschiede in der Verteilung der SW-Funktionen in Kauf genommen. Hinsichtlich der Verteilung der Typen von Mining-Methoden kann ein Ungleichgewicht festgestellt werden, da Testfälle für Sequential Rules (Sequenzanalyse = Spezialfall der Assoziationsanalyse) unterrepräsentiert sind. Hier ist noch reichlich Bedarf an neuen Testfällen für den Integrationstest vorhanden.

## 5.3 Data Preparation

Nachdem in der vorherigen Phase die Eigenschaften und die Qualität der Daten untersucht wurde, werden die Qualitätsprobleme und die zu deren Lösung notwendigen Schritte innerhalb der Vorverarbeitungsphase beschrieben. Die Lösung der Probleme stellt, zusammen mit der Umformung der Daten, die notwendige Voraussetzung dar, um bei der Datenanalyse mittels Data-Mining aussagekräftige Ergebnisse zu erhalten. Das CRISP-DM-Referenzmodell ordnet dieser Phase, wie bereits in Abschnitt Datenvorbereitung beschrieben, fünf Aufgaben zu, die Bereinigung der Daten, die Selektion benötigter Datensätze oder Datenattribute, die Umformung der Daten sowie die Zusammenführung und die Formatierung der Daten.

### 5.3.1 Bereinigen der Daten

Zur Bereinigung der Daten kann es erforderlich sein, Datentypen zu konvertieren, fehlende Werte entweder nachträglich durch generierte oder durch sinngemäße Wert zu ersetzen. Auch sollten widersprüchliche Werte aus den Datensätzen entfernt werden. Diese Bereinigungsoperationen werden direkt in den entwickelten Parser integriert, um die Operationen automatisiert und damit, im Vergleich zu einem manuell zu bedienenden Datenvorbereitungswerkzeug, effizienter ausführen zu können.

Die erste Vorbereitungshandlung betrifft die aus den Testfalldefinitionsdateien ausgelesenen Testfallparameter. Bei einem gewissen Teil der Testfälle werden Angaben bezüglich der Eingabetestdaten, der Eingabemodelle oder der Dateien mit enthaltenen Parametern für die zu prüfenden Software-Funktionen für den Test nicht benötigt. Dennoch sind die Bezeichnungen dieser Testfallparameter in der Definition des Testfalls vorhanden, es ist lediglich kein Wert dahinter angegeben. Für fehlende Werte, oder auch „leere Strings“, ist generell zu überlegen, wie sie in der Datenbank repräsentiert werden sollen, ob zum Beispiel neue Bezeichnungen dafür zu finden sind, die eine eventuell vorhandene Bedeutung des fehlenden Wertes zum Ausdruck bringen. Da der später angewendete Data-Mining-Algorithmus die fehlenden Werte der Testfallparameter ignorieren soll, ihnen also keine Bedeutung beimessen soll, werden die leeren Strings durch *NULL* ersetzt.

Des Weiteren ist die Eindeutigkeit der Trennzeichen in den zur Sicherung der Daten erzeugten

CSV-Dateien zu gewährleisten, falls diese später einmal zu Wiederherstellungszwecken in eine Datenbanktabelle zu laden sind. Mit Hilfe der Trennzeichen werden die einzelnen Datenwerte voneinander abgegrenzt. Das bedeutet, dass sobald in einem Datenwert Trennzeichen auftauchen, würde dieser beim Auslesen aus der Datei fehlerhaft interpretiert. Die in Datenwerten vorhandenen Trennzeichen sind daher durch andere Zeichen zu ersetzen, die den Sinngehalt der Datenwerte nicht verändern. Im konkreten Beispiel enthalten die Namen einiger Testfälle Kommas, die ebenfalls als Trennsymbol in den Datendateien verwendet werden. Ohne eine entsprechende Vorbehandlung würden bei einem späteren Auslesen einer Datei die durch Kommas getrennten Teile eines solchen Testfallnamens ungewollt als einzelne Datenwerte erkannt. Ein Abspeichern eines ausgelesenen Datensatzes in eine Datenbank-Tabelle wäre dann zum Beispiel nicht mehr möglich, da die Anzahl der erkannten Werte in dem Datensatz nicht mit der Anzahl der Spalten der Tabelle übereinstimmen würde, woraufhin die Datenbank einen Fehler zurück gibt. Aus diesem Grund werden die Datenwerte von Anführungszeichen umschlossen. Ein Ersetzen der Kommas in den Datenwerten durch eine anderes Symbol ist aufgrund der potenziellen Verfälschung der Bedeutung des Wertes in keinem Fall zu empfehlen.

Ferner sind Pfadangaben verschiedener Formatierung (absolut oder relativ) aus Gründen der Vergleichbarkeit in ein einheitliches Format zu konvertieren. Daher werden alle Dateipfade relativ, beginnend bei einem definierten Verzeichnisnamen, angegeben und, falls erforderlich, mit Hilfe des entwickelten Java-Programms automatisch gekürzt.

Zusätzlich werden alle Testfälle aus den erfassten Daten entfernt, die in der letzten SW-Version fehlgeschlagen sind, da diese im nächsten Test ohnehin noch einmal ausgeführt werden müssen und daher nicht in die Analyse eingehen. Da die gesamte Menge verfügbarer Testfälle für alle (auch die früheren) SW-Versionen verwendet wurden, einige dieser Testfälle jedoch SW-Funktionen in Kombination mit Parametern testen, die in den früheren Versionen der Software noch nicht implementiert waren, müssen die entsprechenden Testergebnisse von *fehlgeschlagen* in *NULL* geändert werden, damit sie vom Analysealgorithmus ignoriert werden. Andernfalls käme es zu einer negativen Beeinflussung bei der Modellbildung, die dadurch begründet ist, dass alle der betroffenen Testfälle der Klasse fehlgeschlagen angehören und sich das Modell, aufgrund der geringen Unreinheit unter diesen Testfällen, auf deren Eigenschaften spezialisiert und somit übertrainiert wird.

### 5.3.2 Selektieren, Konstruieren, Integrieren, Formatieren

Die in der vorangegangenen Phase dokumentierten notwendigen Schritte zur Anpassung der Daten an die Verarbeitungsmechanismen der Data-Mining-Algorithmen werden nun umgesetzt. Dazu bietet es sich an, ein grafisches Software-Werkzeug zu verwenden, um die notwendigen Transformationen der Daten mit visueller Unterstützung durchzuführen. Da diese Funktionalität bereits im *Design Studio*, einer Komponente des InfoSphere Warehouse, integriert ist, wird dieses zur Umsetzung der Datentransformationen verwendet.

Die Datentransformationen werden durchgeführt mit Hilfe verschiedener grafischer Operatorboxen, die zu einem *Data-Mining-Flussdiagramm („Mining Flow“)* verknüpft werden. Bei dem Flussdiagramm handelt sich um ein Konzept zur visuell gestützten Vorbereitung und Durchführung des Mining-Prozesses. Die Operatoren sind grafische Objekte, die aus der Palette im

Design Studio ausgewählt und in den Arbeitsbereich gezogen werden können. Jeder Operator besitzt bestimmte Eingabe- und Ausgabedatenports zur Verknüpfung mit anderen Operatoren und bestimmte Eigenschaften, die definieren, wie der Operator mit den Daten verfahren soll. Es existieren verschiedene Typen von Operatoren für die Datenvorbereitung: Quellenoperatoren (Table Source), Transformationsoperatoren und Zieloperatoren (Table Target). Quellenoperatoren stellen eine Ansicht (View) - eine Art Kopie - der aus dem Datenbanksystem gewählten Tabelle bereit. Diese Ansicht wird anschließend von einem verbundenen Transformationsoperator verwendet, um die Daten zu verschieben und umzuformen. Zieloperatoren dienen der Speicherung des Transformationsergebnisses in einer neu erzeugten Tabelle innerhalb des Datenbanksystem. Sie sind jedoch nicht unbedingt notwendig, da das Transformationsresultat auch direkt als Eingabe für einen weiteren Typ von Operator, den Mining-Operator zur Modellbildung, dienen kann.

#### Arten der Transformation

Eine detaillierte Beschreibung der im Design Studio zur Verfügung stehenden Transformationsoperatoren gibt das im Internet zu findende Infocenter für DB2-Datenbanken von IBM [66]. Nachfolgend soll, in Anlehnung an das Infocenter, eine kurze Einführung in die Transformatoren gegeben werden.

Ein erster Vertreter dieser Transformationsoperatoren ist der **Auswahloperator**, der dazu dient nur diejenigen Datenmerkmale (Spalten) aus der ursprünglichen Tabelle zu wählen, die eine Relevanz für die Ziele der späteren Data-Mining-Analyse aufweisen und deren Werte den von den Analyseverfahren geforderten qualitativen Bedingungen, wie zum Beispiel keine fehlenden oder inkonsistenten Werte, sowie den technischen Bedingungen, wie Beschränkungen des Datenvolumens oder der Datentypen, entsprechen. Zusätzlich zu beachten ist, dass die Anzahl der Zeilen zur Anzahl der Spalten in der finalen Ergebnistabelle ein Verhältnis von über 5:1 aufweisen sollte, damit die von einem Data-Mining-Algorithmus ausgeführten Berechnungsverfahren zuverlässige statistische Aussagen treffen können.

Ein weiterer Vertreter ist der **WHERE-Bedingungsoperator**, der Zeilen beziehungsweise Datensätze anhand einer Filterbedingung aus einer Tabelle entfernt.

Der **Gruppierungsoperator** dient der Gruppierung von Zeilen einer Tabelle bei gleichzeitiger spaltenweiser Zusammenfassung der in den Zeilengruppen enthaltenen Werte, auch bezeichnet als Aggregation. Unter einer Aggregation oder Verdichtung von Daten versteht man die Zusammenfassung mehrerer Werte zu einem einzigen, stellvertretenden Wert anhand einer bestimmten Aggregationsfunktion, zum Beispiel der Summenfunktion.

Diese Zusammenfassung von Daten können im Zusammenhang mit dem PIVOT-Operator und UNPIVOT-Operator des Design Studio genutzt werden, welche zur Umstrukturierung von Zeilen und Spalten einer Tabelle dienen.

Der **PIVOT-Operator** trägt die Namen von inhaltlich zusammenhängenden Spalten zusammen mit jeweils einem der zugehörigen Werte eines Datensatzes in eine neue Zeile ein. Dies kann zum Beispiel zum Zweck der leichten Erweiterbarkeit von Datensätzen um neue Attribute durchgeführt werden.

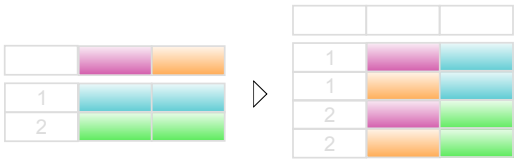


**Abbildung 5-9: PIVOT-Funktion des Design Studio**

Das Gegenstück zu diesem stellt der **UNPIVOT-Operator** dar, der die inhaltlich zusammenhängenden Zeilen eines Datensatzes in Spalten einsortiert. Dies kann zur Schaffung einer besseren Übersicht über die Daten geschehen.

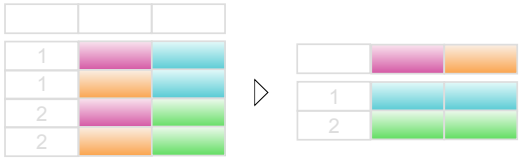


**Abbildung 5-10: UNPIVOT-Funktion des Design Studio**

Eine Sortierung von Zeilen gemäß der Reihenfolge (aufsteigend oder absteigend), die für die zugehörigen Werte in einer oder mehreren Spalten angegeben wurde, übernimmt der **Anordnungsoperator**.

Ebenfalls möglich ist eine Anwendung der SQL-Operationen UNION, INTERSECT und EXCEPT auf virtuelle Tabellen mit Hilfe des **UNION-Verknüpfungsoperators**.

Mittels des hilfreichen **Diskretisierungsoperators**[24] kann eine Umwandlung der kontinuierlichen Werte eines Datenfeldes in diskrete (kategorische) Werte erfolgen. Dazu werden die kontinuierlichen Werte anhand zuvor definierter Grenzen entsprechenden Intervallen (Kategorien) zugeordnet. Anstatt des kontinuierlichen Wertes wird dann der Name der Kategorie in die Ergebnistabelle übernommen. Notwendig ist diese Konvertierung zum Beispiel, wenn Werte eines kontinuierlichen Datenfeldes mit Hilfe eines Klassifikationsalgorithmus (zum Beispiel anhand eines Entscheidungsbaums) vorhergesagt werden sollen, denn dieser akzeptiert ausschließlich kategorische Zielfelder.

Der vorletzte zu betrachtende Operator, der **Eindeutigkeitsoperator**, entfernt redundant vorkommende Zeilen aus der virtuellen Tabelle. Sobald Werte in zuvor angegebenen Spalten bei mindestens zwei Zeilen übereinstimmen, wird nur eine der Zeilen in die Ergebnistabelle übernommen und die anderen werden entfernt.

Schließlich steht noch der **Join-Verknüpfungsoperator** zur Verfügung, der mehrere virtuelle Tabellen mit inhaltlich zusammenhängende Daten in einer einzigen Tabelle integriert. Dabei können im Operator die Spalten der ursprünglichen Tabelle ausgewählt werden, die in der Ergebnistabelle erscheinen sollen. Die Zeilen der ursprünglichen Tabellen werden dann in die Ergebnistabelle übernommen, wenn die in ihnen enthaltenen Datensätze der im Tabellenverbindungsoperator

[24] Die Datenvorbereitung wäre auch in einem speziell vorgesehenen Datenflussdiagramm möglich, da bei diesem jedoch kein Diskretisierungsoperator verfügbar ist, wird sie in einem Data-Mining-Flussdiagramm durchgeführt.

festgelegten Join-Bedingung entsprechen. Für diese zu definierende Join-Bedingung bietet der Tabellenverbindungsoperator im Design Studio vier verschiedene Typen an, den Inner Join, den Left Outer Join, den Right Outer Join und den Full Outer Join.

### Flussdiagramm zur Datenvorbereitung

Eine Übersicht des gesamten Data-Mining-Flussdiagramms zur Datenvorverarbeitung für diese Fallstudie ist im Anhang auf Seite 84 dargestellt. An dieser Stelle soll die Miniaturansicht in Abbildung 5-11 als Grundlage der nachfolgenden Beschreibung der durchgeführten Transformationen dienen.

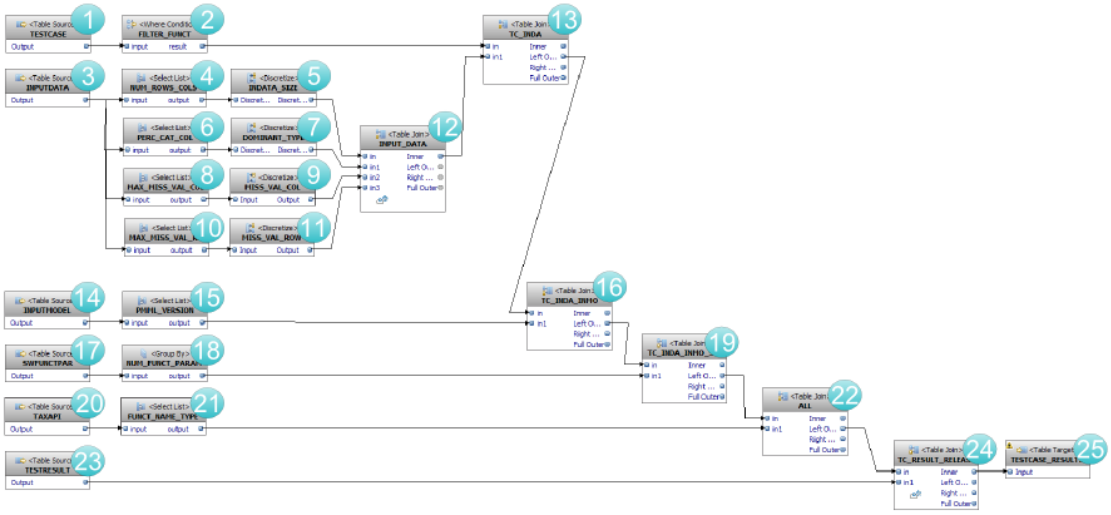


**Abbildung 5-11: Miniaturansicht des Data-Mining-Flussdiagramms zur Datenvorverarbeitung**

Die erste vorzubereitende Tabelle ist die TESTCASE-Tabelle (1) mit den Testfallinformationen, die aus den Testfalldefinitionsdateien ausgelesen wurden. Aus dieser Tabelle werden mittels eines WHERE-Bedingungsoperators (2) diejenigen Testfälle (Datensätze) entfernt, die der Vorbereitung („preparation“ im Testfallname enthalten) oder dem Aufräumen („clean“ im Testfallname) dienen, da diese Testfälle ohnehin im Test benötigt werden und daher nicht für eine Testfallselektion in Frage kommen. Außerdem enthalten diese Testfälle keine Software-Funktionen und stellen somit keinen Informationsgewinn bei der Analyse der Software-Qualität dar. Weitere Anpassungen sind nicht notwendig, da der vom Intelligent Miner verwendete Baumklassifikationsalgorithmus sowohl numerische als auch kategorische Datenattribute als Eingabe akzeptiert und die Daten der TESTCASE-Tabelle ein mittleres Volumen aufweisen (11 Zeilen und 3244 Spalten).

Die zweite Tabelle, bei der Transformationen vorgenommen werden, ist die INDATA-Tabelle (3) mit den Eingabetestdaten zum Test der SW-Funktionen. Die aus den Eingabetestdaten im Datenerfassungsschritt gezogenen Merkmale der Testdaten liegen in proportional skalierter[25] Form vor und sind für eine eher generelle Aussagekraft mit Hilfe einer Diskretisierung in die nominal skalierte Form zu überführen. Dazu werden die einzelnen Merkmalsspalten erst in eine temporäre, virtuelle Tabelle übernommen und anschließend wird die Diskretisierung durchgeführt. Zusätzlich zu den Merkmalsspalten wird zu deren späterer Wiederzusammenführung in jede der virtuellen

[25] zu Maßskalen siehe [60]

Tabellen zusätzlich die Spalte mit den Namen der Eingabedatendateien (Primärschlüssel in der INDATA-Tabelle) übernommen.

Das erste zu bearbeitende Merkmal ist die Größe der Testdaten, welche der Anzahl der gesamten Zellen in den Testdaten entspricht. Zur Bestimmung der Gesamtzahl an Zellen erfolgt in einem Auswahllistenoperator (4) die Selektion der Spalte mit der Anzahl der Zeilen der Testdaten und anschließend deren Multiplikation mit der Spalte die die Spaltenanzahl der Testdaten enthält. Die Spalte mit den resultierenden Werten ($x$ = Anzahl der Zellen) wird danach mittels eines Diskretisierungsoperators (5) einer Diskretisierung in die Intervalle „leer“ ($x = 0$), „klein“ ($0 < x <= 10.000$), „mittel“ ($10.000 >= x < 1.000.000$) und „groß“ ($x >= 1.000.000$) unterzogen.

Das zweite zu bearbeitende Merkmal ist das Verhältnis von numerischen zu kategorischen Werten, für das in einem nächsten Auswahllistenoperator (6) die Anzahl der in den Eingabedaten enthaltenen kategorischen Spalten ausgewählt und durch die Anzahl aller darin enthaltenen Spalten geteilt wird, um bei Vorliegen einer Minderheit von kategorischen Werten ($0\% - 44\%$) im nachfolgenden Diskretisierungsoperator (7) eine Einordnung in das Intervall „numerisch“, bei Vorliegen einer Gleichverteilung von numerischen und kategorischen Werten ($45\% - 54\%$ kategorische Werte) in der jeweiligen Eingangsdatendatei in das Intervall „neutral“ und bei einem Überwiegen der kategorischen Werte ($55\% - 100\%$) eine Einordnung in das Intervall „kategorisch“ vorzunehmen.

Ein drittes Merkmal der Eingabedaten ist der Prozentsatz maximal fehlender Werte in den Zeilen der Eingabedatendatei, der direkt bei der Erfassung der Daten bestimmt wurde, da diese Bestimmung im Java-basierten Parser recht einfach zu realisieren war. Dafür ist im Auswahllistenoperator (8) lediglich noch die Spalte mit den vorbereiteten Werten auszuwählen und anschließend ebenfalls eine Diskretisierung (9) durchzuführen. In diesem Fall wird eine Einteilung in die Intervalle „keine“ (fehlenden Werte) bei einem Prozentsatz von 0%, „gering“ (kleiner maximaler Anteil fehlender Werte in den Spalten) bei einem ermittelten Prozentsatz zwischen $1\% - 30\%$, „mittel“ bei $31\% - 69\%$, „hoch“ bei $70\% - 99\%$ und „vollständig“ bei einem Anteil maximal fehlender Wert in den Spalten von 100% vorgenommen.

Das Pendant dazu bildet der Anteil maximal fehlender Werte in den Spalten der Eingabedatendatei, der ebenfalls im Parser vorberechnet wurde und im Auswahllistenoperator selektiert wird (10). Die im Diskretisierungsoperator (11) gebildeten Intervalle entsprechen denen der für den Anteil fehlendern Werte in Zeilen in (9) gewählten.

Die Zusammenführung der vorbereiteten Eigenschaften der Eingabedaten zu einer einzigen virtuellen Tabelle ist anhand eines Tabellenverbindungsoperators (12) möglich. Die Datensätze der ursprünglichen (virtuellen) Tabellen werden durch die Verknüpfung zusammen in eine Zwischenergebnistabelle übernommen, sofern sie einer, im Tabellenverbindungsoperator festgelegten, Join-Bedingung entsprechen. Die in der Zwischenergebnistabelle erwünschten Spalten können im Tabellenverbindungsoperator aus den ursprünglichen virtuellen Tabellen selektiert werden. In diesem Fall sind dies die vier Spalten mit den diskretisierten Werten, die anhand der Spalte mit den Namen der Eingabedatendateien als Join-Bedingung verbunden werden. Die Verknüpfung der vorbereiteten Tabelle TESTCASE und der eben vorbereiteten Tabelle INDATA erfolgt mittels eines Left Outer Join im Tabellenverbindungsoperators (13), um alle Testfälle zu verwenden,

unabhängig davon, ob die Testfälle Angaben zu den Eingabedaten enthalten oder nicht (nicht in allen Testfällen erforderlich).

Fortgefahren wird mit der Vorverarbeitung der Daten in der INPUTMODEL-Tabelle (14). Dazu wird die benötigte Spalte mit der Angabe der PMML-Version des jeweiligen Eingabemodells ausgewählt (15) und zur Zwischenergebnistabelle von (13) mit Hilfe eines Left Outer Join der Modelleingabedateinamen, die in der Zwischenergebnistabelle und der INPUTMODEL-Tabelle übereinstimmen, hinzugefügt (16).

Für die Tabelle SWFUNCTPAR (17) ist eine Aggregation, das heißt eine Bestimmung der Anzahl der an die SW-Funktionen übergebenen Funktionsparametern im Gruppierungsoperator (18) notwendig. Die Spalte mit den aggregierten Werten wird im Anschluss in die bisherige Zwischenergebnistabelle (16) mittels eines Left Outer Join der Parameterdateinamen eingefügt (19).

Auch für die TAXAPI-Tabelle (20) sind Vorverarbeitungen notwendig, die jedoch lediglich aus der Selektion (21) der relevanten Spalten, also der Kind-Spalte, welche die Software-Funktionen enthält, und der Vater-Spalte, mit den übergeordneten Typen der Funktionen, beinhaltet. Auch diese beiden Spalten werden, durch einen Left Outer Join der Funktionsnamen in der bisherigen Zwischenergebnistabelle und der TAXAPI-Tabelle, mit der Zwischenergebnistabelle integriert (22).

Der letzte notwendige Schritt in der Datenvorbereitung betrifft das Einfügen der Testresultate zu einem jeden Testfall aus der TESTRESULT-Tabelle (23) in die aktuelle Zwischenergebnistabelle mittels eines Inner Join (24), um nur diejenigen Testfälle in die Ergebnistabelle zu übernehmen, für die mindestens ein Testresultat vorliegt. Die Transformation der zu den einzelnen SW-Versionen vorliegenden Testergebnisse zu einem einzigen Resultat, welches als Zielwert für die Klassifikation verwendet werden soll, erfolgte bereits im Java-basierten Parser. Der Grund dafür ist, dass im Design Studio kein Operator und auch kein Standard-SQL-Befehl gefunden werden konnte, der den Abgleich von Zeichenketten (Ergebnisse der einzelne SW-Versionen in dieser Form zusammengehängt) mit regulären Ausdrücken vornehmen könnte, um die Ergebnisfolge bei Vorliegen von mindestens einem Fehlschlag als *fehlgeschlagen* oder andernfalls als *nicht fehlgeschlagen* zu interpretieren. Dazu wäre die Implementierung einer eigenen UDF notwendig gewesen, die aufgrund fehlender Vorkenntnisse einen unnötigen Zeitaufwand bedeutet hätte.

Das Ergebnis der Transformationsoperationen ist die Tabelle TC_RESULT_RELEASE_LEVEL (25), welche nun alle vorbereiteten Daten enthält und die nun in einem neuen Data-Mining-Flussdiagramm als Eingabe für die Data-Mining-Analyse dient.

## 5.4 Modeling

In dieser Phase erfolgt die Anwendung der Data-Mining-Verfahren auf die soeben vorbereiteten Daten. Die Aufgaben in dieser Phase des Data-Mining-Prozesses, zum Beispiel die Einstellung der Parameter für die eingesetzten Data-Mining-Algorithmen, können zum einen durch das Ausführen der einzelnen API-Funktionen des Intelligent Miner in der DB2-Befehlszeile oder automatisiert mit Hilfe eines Java-Programms und zum anderen mit der grafischen Unterstützung

des Design Studios durchgeführt werden. Da die Operatorboxen im Design Studio das Absetzen der Funktionsaufrufe übernehmen und lediglich einige Einstellungen in den Operatorboxen angepasst werden müssen, wird dieses zur Durchführung der Analyse mit den vorbereiteten Daten verwendet.

### 5.4.1 Erstellung des Test-Designs

Ein Test-Design wird für Prognosemodelle - die Ergebnisse der Datenanalyse mittels Klassifikation und Regression - benötigt, um die Qualität der Prognose nach verschieden Aspekten zu beurteilten. Ein Aspekt ist die Vorhersagegenauigkeit des Modells, die bestimmt mit welcher Wahrscheinlichkeit das vorhergesagte Ergebnis für eine bestimmte Klasse mit dem tatsächlichen Wert übereinstimmt. Ein weiterer Aspekt, das Maß der Zuverlässigkeit des Modells, beschreibt inwieweit es geeignet ist, eine Prognose für neue, unbekannte Daten zu erstellen. Der dritte Aspekt betrifft die Rangordnungsqualität, mit der es möglich ist die Eignung des Modells zu beurteilen in Bezug auf die Sortierung von Datensätzen anhand der Wahrscheinlichkeit, mit der diese den vorhergesagten Zielwert annehmen.

#### Prüfung der Anwendbarkeit des Modells auf neue Testfälle

Die Abbildung 5-12 stellt ein Flussdiagramm mit einem Ansatz zur Prüfung des erstellten Modells hinsichtlich seiner Anwendbarkeit auf unbekannte Datensätze, in diesem Fall neue Testfälle, dar. Der Zufallsteiler-Operator (2) dient der Teilung aller Datensätze aus der Quelltabelle (1) in

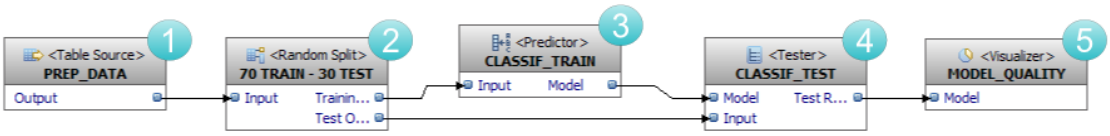



**Abbildung 5-12: Flussdiagramm mit dem ersten Ansatz zur Überprüfung der Modellqualität**

eine Trainingsmenge und eine Testmenge. Die Testmenge wird zufällig aus einem angegebenen Prozentsatz, in diesem Fall 30 Prozent, der Datensatzmenge ausgewählt und der verbleibende Teil, hier 70 Prozent, wird für die Trainingsmenge verwendet. Über den ersten Ausgangsport des Zufallsteiler-Operators wird die Trainingsmenge an den Modellbildungsoperator (3) weitergegeben. Der Modellbildungsoperator erstellt daraufhin mit den Trainingsdaten und den in ihm definierten Parametern ein Klassifikationsmodell und reicht es über seinen Ausgangsport an den ersten Eingangsport eines Testoperators (4) weiter. Der zweite Eingangsport des Testoperators wird mit demjenigen Ausgangsport des Zufallsteiler-Operators verbunden, der die Testmenge der Datensätze bereitstellt. Der Testoperator wendet das erstellte Klassifikationsmodell auf die Testmenge an und vergleicht die erhaltenen Prognosewerte mit den in der Testmenge hinterlegten realen Werten, um anhand dessen die Qualität des Klassifikationsmodells zu errechnen. Diese wird im nachgeschalteten graphischen Darstellungsoperator (5) präsentiert.

### Prüfung der Anwendbarkeit des Modells auf die nachfolgende Software-Version

Abbildung 5-13 enthält ein Flussdiagramm mit einem Ansatz zur Prüfung des erstellten Modells hinsichtlich seiner Anwendbarkeit auf neue Attributwerte, in diesem Fall die Testergebnisse nachfolgender Software-Versionen. Bei diesem Testdesign werden in einem Auswahloperator (2)

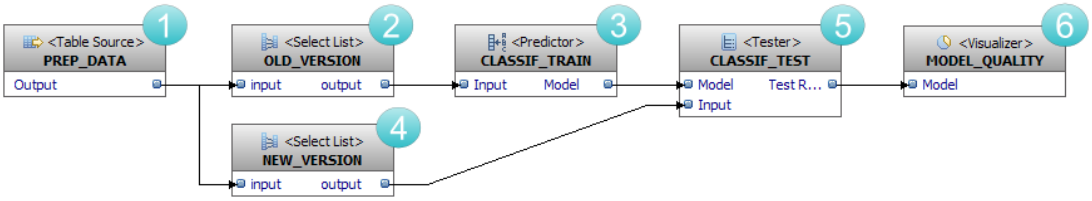



**Abbildung 5-13: Flussdiagramm mit dem zweiten Ansatz zur Überprüfung der Modellqualität**

alle für die Analyse als Eingabe benötigten Spalten sowie die Spalte mit den Testergebnissen der ersten zur Verfügung stehenden Software-Version aus der Quelltabelle (1) ausgewählt und letztere mit einer neuen Bezeichnung versehen (TOTAL_RESULT), um diese als Zielattribut zu kennzeichnen. Anzumerken ist hier, dass es sich bei allen in der Fallstudie betrachteten SW-Versionen des Intelligent Miner um „grobgranulare“ Release-Stände und nicht um seine „feingranularen“ täglichen Builds handelt. Die Datensätze (Testfälle für den Intelligent Miner) mit den gewählten Attributen werden in Form einer virtuellen Tabelle dem Modellbildungsoperator (3) übergeben, welcher mit den Daten und den in ihm definierten Parametern, zum Beispiel dem Name der Spalte mit den Zielwerten für die Vorhersage, ein Klassifikationsmodell erstellt. Dieses gibt er über seinen Ausgangsport an den Testoperator (4) weiter. In einem zweiten Auswahloperator (5) werden die gleichen Spalten wie im ersten Auswahloperator aus der Quelltabelle ausgewählt, jedoch mit dem Unterschied, dass nun die Spalte mit den Testergebnissen der nachfolgenden, zweiten Software-Version selektiert und diese mit der zuvor eingeführten Bezeichnung für das Zielattribut gekennzeichnet wird. Der Auswahloperator stellt die Datensätze mit den gewählten Attributen dem Testoperator an seinem zweitem Eingangsport bereit, damit dieser das mit den Testergebnissen der ersten Version erstellte Klassifikationsmodell auf die neuen Testergebnisse der Nachfolgeversion anwenden kann. Der Testoperator vergleicht dann die mit dem Modell vorhergesagten Testergebnisse für die einzelnen Testfälle mit den realen Ergebnissen um die Qualität der Vorhersage des Modells berechnen zu können. Die ermittelte Prognosequalität kann anschließend im verbundenen graphischen Darstellungsoperator (5) betrachtet werden. Mit Hilfe dieses Test-Designs kann die Modellqualität für alle Paare von aufeinander folgenden Software-Versionen - drei in dieser Fallstudie - geprüft werden, um zu ermitteln, wie zuverlässig das Modell die Testergebnisse für die nächste Software-Version vorhersagen wird. Dieses Wissen ist notwendig, um eine verlässliche Sortierung und Reduktion der Testfälle basierend auf dem erstellten Modell zu garantieren.

Ein Alternative dazu ist die Vorhersage der Testergebnisse der nächsten Software-Version auf Basis der Ergebnisse mehrerer Vorgängerversionen, anstatt einer einzigen, welche ebenfalls in dieser Fallstudie Anwendung findet. Dazu müssen lediglich im ersten Auswahloperator (2) (siehe Abbildung 5-14) Anpassungen durchgeführt werden. Um die Testergebnisse aus einer bestimmten Anzahl mehrerer Vorgängerversionen im Klassifikationsmodell zu berücksichtigen, sind im Auswahloperator die zu den Vorgängerversionen in der Quelltabelle (1) gespeicherten

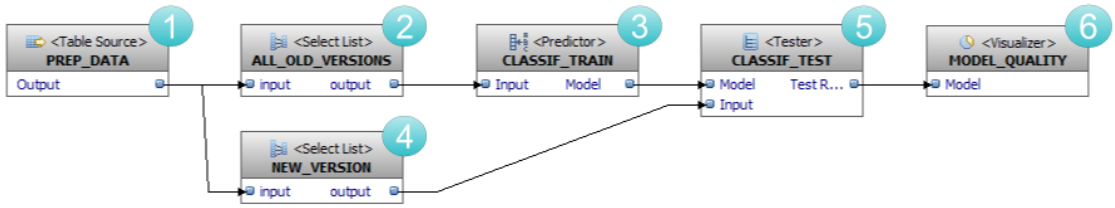



**Abbildung 5-14: Flussdiagramm mit einem alternativen zweiten Ansatz zur Überprüfung der Modellqualität**

Testergebnisse zu aggregieren und in die Spalte mit der Kennzeichnung für das Zielattribut (TOTAL_RESULT) einzutragen. Da die Testergebnisse in den binären Einzelzeichen {0,1} vorliegen, kann die Aggregation realisiert werden anhand des Zusammenhängens (engl. concatenation) der Einzelzeichen von aufeinander folgenden SW-Versionen und einer Überprüfung der entstandenen Zeichenkette auf das Vorliegen von mindestens einer „1“ (Test fehlgeschlagen). Ist dies der Fall, so gilt der Testfall insgesamt als fehlgeschlagen und in der Zielspalte wird ein entsprechender Eintrag („failed“) vorgenommen. Der zugehörige SQL-Befehl für diese Abfrage aus der Quelltabelle namens PREPDATA innerhalb der DB2-Datenbank ist:

```
1 SELECT
2 CASE WHEN ((PREPDATA.VERS1 || PREPDATA.VERS2) || PREPDATA.VERS3) LIKE '%1%'
3       THEN 'failed'
4       ELSE 'notfailed'
5       END AS TOTAL_RESULT
6 FROM PREPDATA;
```

Der Ausdruck ’%1%’ verwendet das Jokerzeichen (engl. wildcard) %, das beim Abgleichen des Ausdrucks mit der zu den Testergebnissen vorliegenden Zeichenkette alle vor und nach der „1“ in der Zeichenkette stehenden Zeichen repräsentiert.

### 5.4.2 Modellbildung

Nachdem die verschiedenen Formen von Test-Designs gewählt und eingerichtet wurde, können in den Klassifikationsoperatoren (z.B. Nummer (3) in Abbildung 5-14) die Parameter für die Bildung des Entscheidungsbaumes angegeben werden. Eine Einführung in die möglichen Parameter erfolgte bereits im Abschnitt „Modellbildung“ auf Seite 44. Bei der Selektion der für die Modellbildung einzusetzenden Merkmale, als Vertreter der allgemeinen Parameter, wurden die in Tabelle 5-2 dargestellten (nach der Datenvorverarbeitung vorliegenden) Merkmale der TC_RESULT_RELEASE_LEVEL-Tabelle als aktiv definiert. Alle anderen Spalten in dieser Tabelle wurden auf inaktiv gesetzt, da bei ihnen keine signifikante Bedeutung hinsichtlich des Mining-Ergebnisses festgestellt werden konnte und auf diese Weise einer Überanpassung des Modells an die Trainingsdaten (bis zu einem gewissen Grad) vorgebeugt wird. Als Zielattribut für die Klassifikation der Testfälle wurde die Spalte TOTAL_RESULT mit den zusammengefassten, aus den einzelnen SW-Versionen vorliegenden Testergebnissen verwendet.

Alle speziellen Parameter, bis auf die Kostenmatrix, wurden bei Standardwerten belassen, da keine der getätigten Werteänderungen zu einer Erhöhung der Modellgenauigkeit beitrugen. Dies

| Merkmal | Bedeutung |
|---|---|
| MINING_TYPE | Typ der getesteten Data-Mining-Funktion, z.B. Klassifikation, Clustering |
| ALGOR | darkgray zusammen mit der Data-Mining-Funktion eingesetzter Data-Mining-Algorithmus |
| TC_TESTED_FUNCT | Name der getesteten Data-Mining-Funktion |
| TC_TESTED_FUNCT_TYPE | Typ der getesteten Data-Mining-Funktion, z.B. Import-Funktion für Data-Mining-Modelle unabhängig vom MINING_TYPE |
| INDATA_SIZE | Größe der Eingabedaten |
| INDATA_DOMINANT_TYPE | überwiegend kategorische oder numerische Werte in Eingabedaten |
| INDATA_MAX_RATIO_MISS_VAL_COL | maximaler Anteil der fehlenden Werte in den Zeilen der Eingabedaten |
| INDATA_MAX_RATIO_MISS_VAL_ROW | maximaler Anteil der fehlenden Werte in den Spalten der Eingabedaten |
| NUM_FUNCT_PARAM | Anzahl der an die Data-Mining-Funktion übergebene Funktionsparameter |

**Tabelle 5-2: Für die Modellbildung aktivierte Merkmale**

kann damit erklärt werden, dass der erstellte Baum aufgrund der Begrenzung der benötigten Merkmale kaum übertrainiert ist und daher keine zusätzlichen Maßnahmen zu seiner Beschneidung (Pruning) vorgenommen werden müssen. Ein Pruning des Baumes kann bei Bedarf ohnehin nachträglich im Modellbetrachtungsoperator (*Visualizer*) durchgeführt werden.

In der Kostenmatrix wurden die *falsch-negativen*, also die für eine Effizienzsteigerung beim Testressourceneinsatz wichtigen Testfälle, die vom Modell als erfolgreich vorhergesagt wurden, jedoch in Wirklichkeit fehlgeschlagen sind, mit einem sechsfachen Gewicht versehen im Vergleich zu den *falsch-positiven*, also denjenigen Testfällen, die als fehlgeschlagen klassifiziert wurden, aber eigentlich erfolgreich waren. Der Faktor mit dem Wert sechs konnte als optimal identifiziert werden, da bis zu diesem Wert die Vorhersagegenauigkeit des Modells stieg, darüber hinaus jedoch keine weitere Verbesserung möglich war.

Die vom Algorithmus ermittelten, wichtigsten Split-Merkmale für die Erstellung des Entscheidungsbaumes sind, für alle Test-Design-Formen im Anhang auf Seite 86 dargestellt. Die Tabelle 5-3 gibt eine Übersicht der jeweils drei wichtigsten Merkmale. Aus dieser wird ersichtlich, dass die Merkmale mit der größten Relevanz sich bei den mit den verschiedenen Test-Designs erstellten Modellen unterscheiden. Unter den am häufigsten auf den ersten Plätzen auftauchenden Merkmalen ist das mit den Namen der getesteten Data-Mining-Funktionen (TC_TESTED_FUNCT) und das mit den im Zusammenhang mit den Data-Mining-Funktionen getesteten Data-Mining-Algorithmen (ALGOR). In der nachfolgende Anwendung der Modelle auf die Testdatensätze wird sich anhand der Vorhersagegenauigkeit zeigen, ob diese relevanten Merkmale wirklich aussagekräftig genug für eine Klassifizierung neuer Testfälle sind.

| Test-Design | 1. Merkmal | 2. Merkmal | 3. Merkmal |
|---|---|---|---|
| Random Split (V1+2+3+4) | TC_TESTED_FUNCT | MINING_TYPE | ALGOR |
| V1 train → V2 test | TC_TESTED_FUNCT | INDATA_SIZE | ALGOR |
| V2 train → V3 test | ALGOR | TC_TESTED_FUNCT | MINING_TYPE |
| V3 train → V4 test | ALGOR | TC_TESTED_FUNCT | MINING_TYPE |
| V1+2+3 train → V4 test | TC_TESTED_FUNCT | MINING_TYPE | NUM_FUNCT_PARAM |

**Tabelle 5-3: Vergleich der drei wichtigsten Merkmale über die Modelle der verschiedenen Test-Designs hinweg**

Ferner ist zu erkennen, dass die gesamten relevanten Merkmale bei den Test-Designs mit den SW-Versionspaaren 2 → 3 und 3 → 4 identisch sind (es existieren bei beiden nur diese drei wichtigen Merkmale), während sie sich von den Merkmalen beim SW-Versionspaar 1 → 2 unterscheiden. Dies kann dadurch begründet sein, dass die Testergebnisse mit den Versionen 2 und 3 sehr ähnlich ausgefallen sind, und demzufolge viele der fehleranfälligen SW-Einheiten bei den beiden Versionen übereinstimmen, während die mit der früheren Version 1 erhaltenen Testergebnisse im Vergleich zu den beiden späteren unterschiedlich ausgefallen sind und daher in Version 1 zum Teil andere SW-Einheiten Schwächen zeigen. Die Konsequenz dieser unterschiedlichen Modellergebnisse ist, dass höchstwahrscheinlich eine erneute Erstellung eines Modells, und damit die erneute Testfallpriorisierung und -selektion, zu Beginn jeder neuen Release-Entwicklungsphase der Software durchzuführen ist, um die Modelle möglichst aktuell bezüglich der fehleranfälligen SW-Bestandteile zu halten und damit die zugehörigen Testläufe möglichst effizient zu gestalten.

Ein Beispiel, wie die nach den Splits vorliegenden Teilmengen hinsichtlich ihrer Werteverteilung aussehen, gibt die Abbildung 5-15 für das Test-Design des Random Split.

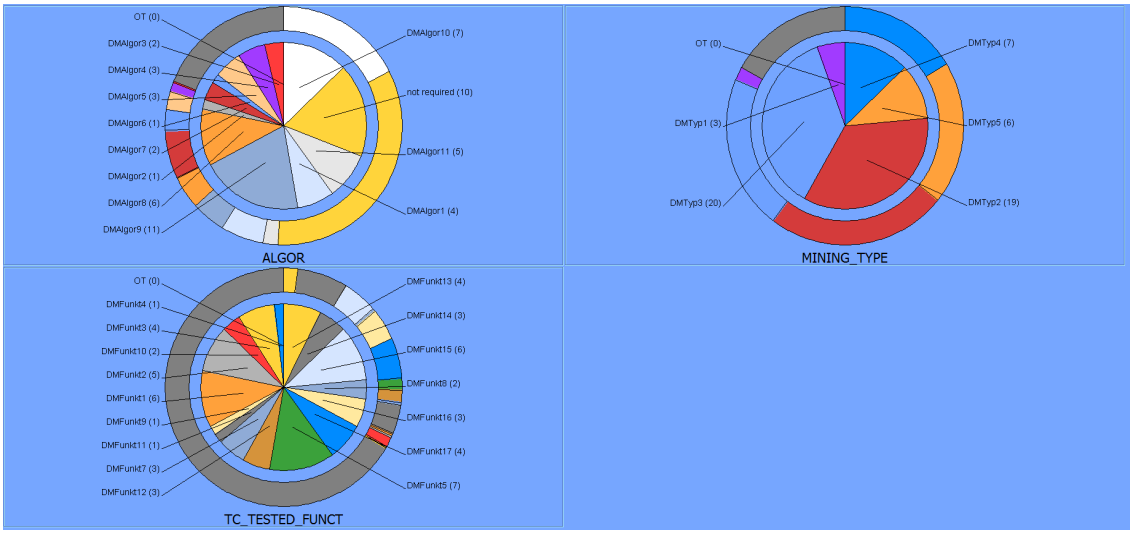



**Abbildung 5-15: Werteverteilung für die wichtigsten Split-Merkmale beim Random Split**

Aus diesen Kuchendiagrammen kann abgelesen werden, welche Eigenschaften der Software am

stärksten zu einem Fehlschlagen der Testfälle beitragen. Die äußeren Ringe um die Diagramme enthalten dabei die ursprüngliche Verteilung der Werte in der gesamten Trainingsmenge und die inneren Kuchendiagramme enthalten die Werteverteilung für die, nach dem Split vorliegende, Teilmenge der fehlgeschlagenen Testfälle. Durch den Vergleich der inneren Diagrammsektoren mit denen des äußeren Rings, kann ermittelt werden, welche Eigenschaften charakteristisch für die fehlgeschlagenen Testfälle sind. Dies können unter anderem die von den Testfällen geprüften Data Mining-Funktionen sein. Genauere Informationen stellt dazu der gebildete Entscheidungsbaum bereit, der mit Hilfe der, in den erstellten Modellen enthaltenen, Entscheidungsregeln präsentiert wird. Ein Beispiel für diese Präsentation gibt die Abbildung 5-16, wieder für das Random-Split-Design. Die mit den anderen Designs erzeugten Entscheidungsbäume werden, aufgrund anderer relevanter Split-Merkmale, einen von diesem Beispiel etwas abweichenden Aufbau zeigen.

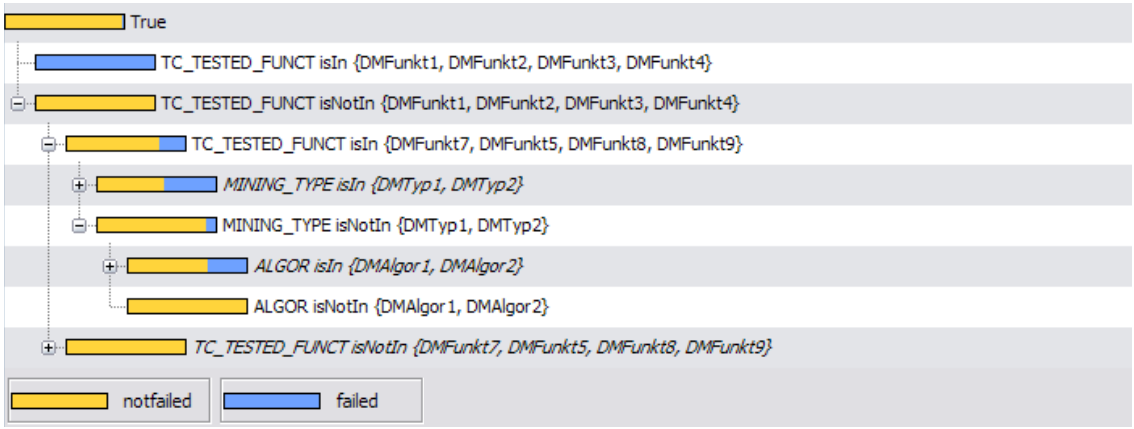



**Abbildung 5-16: Entscheidungsbaum zum Ablesen der fehleranfälligen SW-Funktionen „TC_TESTED_FUNCT“**

Aus den, für die vier SW-Versionen erstellten, Entscheidungsbäume können jeweils die schwächsten SW-Funktionen entnommen werden. Eine entsprechende Übersicht schafft Tabelle 5-4.

| SW-Version | 1. Funktion | 2. Funktion | 3. Funktion | 4. Funktion |
|---|---|---|---|---|
| 1 | DMFunkt1 | DMFunkt2 | DMFunkt3 | DMFunkt4 |
| 2 | DMFunkt5 | DMFunkt3 | DMFunkt7 | DMFunkt8 |
| 3 | DMFunkt5 | DMFunkt3 | DMFunkt7 | DMFunkt8 |

**Tabelle 5-4: Übersicht der vier schwächsten SW-Funktionen in den SW-Versionen**

Diese Informationen bestätigen den Verdacht, dass die festgestellten Unterschiede hinsichtlich der wichtigen Split-Attribute daraus begründet ist, dass in Version 2 und 3 die gleichen SW-Funktionen Schwächen aufweisen, während in der vorhergehenden Version 1 zum Teil andere SW-Funktionen mit qualitativen Problemen behaftet sind.

## 5.5 Evaluation

Nachdem verschiedene Klassifikationsmodelle erstellt wurden, ist zu beurteilen, welches davon das nach technischen Kriterien qualitativ beste darstellt und inwieweit dieses Modell auch inhaltlich zur Erfüllung der gestellten Projektziele beiträgt.

### 5.5.1 Modellbeurteilung nach technischen Kriterien

Wie bereits in Abschnitt „Modellbeurteilung“ auf Seite 45 beschrieben wurde, kann die Beurteilung der erstellten Modelle auf der Basis verschiedener Kriterien erfolgen. Im Folgenden werden daher die Genauigkeit und die Zuverlässigkeit der Modelle, welche unter Verwendung verschiedener Test-Designs auf neue Daten angewendet wurden, sowie die einzelnen, nach den Modelltests vorliegenden Gains-Charts beurteilt. Die Berechnung der Genauigkeit erfolgt anhand der im Abschnitt „Genauigkeit der Vorhersage der Klassenzugehörigkeit“ auf Seite 47 vorgestellten Formel 4.8, wohingegen die Zuverlässigkeit aus dem im Visualizer des Design Studio angezeigten Modelltestresultat abgelesen werden kann.

#### Statistische Gütekriterien

Im Folgenden sind die Ergebnisse der Modellanwendung auf neue Daten für alle verwendeten Test-Designs in Tabellen dargestellt. Diese enthalten eine Fehlermatrix und die nach Gleichung 4.8 (Seite 47) berechneten Werte der Genauigkeit sowie die aus den Modelltestresultaten abgelesenen Zuverlässigkeitswerte.

| | fehlgeschlagen | erfolgreich | Genauigkeit | Zuverlässigkeit |
|---|---|---|---|---|
| fehlgeschlagen | 13 | 12 | 0,520 | 0,866 |
| erfolgreich | 4 | 633 | | |

**Tabelle 5-5: Ergebnisse des Modells mit Random Split**

Tabelle 5-5 zeigt die Ergebnisse für den Test des Modells nach dem Random-Split-Verfahren. Aus den Einträgen ist erkennbar, dass von den 25 tatsächlich fehlgeschlagenen Testfällen aus der Testmenge mit 13 nur etwa die Hälfte richtig klassifiziert wurden. Dementsprechend gering fällt der Wert für die Genauigkeit des Modells bei der Klassifizierung neuer Testfälle aus. Die Zuverlässigkeit nimmt einen akzeptablen Wert an, was bedeutet, dass das Modell nicht aufgrund von Overfitting eine geringe Genauigkeit aufweist, da bei einer solch hohen Zuverlässigkeit das Modell seine Qualität von sich aus gering einschätzt. Das zugehörige Gains-Chart (siehe Anhang Seite 87) zeigt einen stabilen Verlauf, das heißt das Modell ist in der Lage die kritischsten Testfälle als solche zu erkennen und an den Anfang der Rangfolge zu legen. Aufgrund der geringen Genauigkeit ist das Modell trotzdem nur in einem geringen Maß zur Priorisierung und Selektion von neuen Testfällen geeignet.

Etwas anders sieht es bei dem, mit der ersten SW-Version gebildeten Modell zur Klassifizierung neuer SW-Versionen aus (vergleiche Tabelle 5-6. Hier kann bereits eine Genauigkeit von 60%

| | fehlgeschlagen | erfolgreich | Genauigkeit | Zuverlässigkeit |
|---|---|---|---|---|
| fehlgeschlagen | 12 | 7 | 0,632 | 0,845 |
| erfolgreich | 51 | 2743 | | |

**Tabelle 5-6: Ergebnisse des Modells mit V1 train → V2 test**

erreicht werden, bei einer hohen Zuverlässigkeit. Dieses Modell erkennt die kritischsten Testfälle, bis auf ein paar einzelne Falschklassifikationen, relativ gut. Bei einer Vorhersagegenauigkeit von 60% kann das Modell zur Testfallpriorisierung eingesetzt werden, jedoch sollten auf jeden Fall zusätzliche Testfälle aus der Menge der weniger wichtigen hinzugenommen werden, um die Fehlerfindungsrate zu erhöhen.

| | fehlgeschlagen | erfolgreich | Genauigkeit | Zuverlässigkeit |
|---|---|---|---|---|
| fehlgeschlagen | 13 | 5 | 0,722 | 0,952 |
| erfolgreich | 56 | 2779 | | |

**Tabelle 5-7: Ergebnisse des Modells mit V2 train → V3 test**

Das mit der zweiten SW-Version erstellte Modell (siehe Tabelle 5-7 weist eine, im Vergleich zu dem Modell mit der ersten SW-Version, erhöhte Genauigkeit auf. Auch die Zuverlässigkeit ist noch einmal gestiegen. Die Fähigkeit des Modells, die Testfälle geeignet zu sortieren, ist ebenfalls stärker ausgeprägt. Mit einem Einsatz von etwa einem Prozent der Testfälle können schon circa 70% der gesamten Fehler gefunden werden.

| | fehlgeschlagen | erfolgreich | Genauigkeit | Zuverlässigkeit |
|---|---|---|---|---|
| fehlgeschlagen | 13 | 3 | 0,812 | 1,000 |
| erfolgreich | 13 | 2784 | | |

**Tabelle 5-8: Ergebnisse des Modells mit V3 train → V4 test**

Auch das mit SW-Verion drei erstellte und auf Version vier angewendete Modell (siehe Tabelle 5-8) zeigt eine nochmals erhöhte Genauigkeit und Zuverlässigkeit im Vergleich zu den vorherigen Modellen. Die Qualität der Rangbildung ist bei diesem Modell schon sehr gut, denn bei der Ausführung von etwa 1% der Testfälle können bereits 80% der Fehler gefunden werden. Nach dieser Grenze könne mit zusätzlichen Testfällen kaum noch Gewinne bezüglich des Aufdeckens von Fehlern gemacht werden.

Das mit drei zurückliegenden SW-Versionen gebildete und mit der aktuellsten Version getestete Modell (in Tabelle 5-9 gezeigt) liegt bezüglich seiner Genauigkeit und Zuverlässigkeit etwa im gleichen Bereich wie das mit der dritten Version trainierte Modell. Einen gravierenden Unterschied gibt es jedoch bei der Rangfolgebildungsqualität. Das Modell hat zwar einige sehr kritische Testfälle richtig erkannt, trifft dann jedoch bei einer ganzen Reihe von Testfällen Fehlentscheidungen

| | fehlgeschlagen | erfolgreich | Genauigkeit | Zuverlässigkeit |
|---|---|---|---|---|
| fehlgeschlagen | 13 | 3 | 0,812 | 0,930 |
| erfolgreich | 54 | 2743 | | |

**Tabelle 5-9: Ergebnisse des Modells mit V1+2+3 train → V4 test**

in der Form, dass diese vom Modell als kritisch eingestuften in Wirklichkeit nicht fehlgeschlagen waren. Dies ist mit einem unnötig erhöhten Kostenaufwand im Rahmen der Tests verbunden. Noch entscheidender für die Einschätzung der Qualität der Rangfolgebildung ist hier der Umstand, dass am unteren Ende der Rangfolge noch einige Testfälle als überhaupt nicht kritisch eingeschätzt werden, die in Wirklichkeit aber sehr wichtig waren, da sie viele Fehler aufdecken konnten. Darüber hinaus konnten bei diesem Modell mit 1% der Testfälle nur 7% der gesamten Fehler entdeckt werden (siehe Gains-Chart im Anhang auf Seite 87). Aufgrund den beiden zuletzt genannten Gründen ist dieses Modell eher ungeeignet für die Priorisierung von Testfällen.

Zusammenfassend kann gesagt werden, dass die zur Priorisierung und Selektion der Testfälle erstellten Modelle zum Teil eine recht hohe Genauigkeit aufweisen. Insbesondere die Modelle, die für paarweise aufeinander folgende SW-Versionen erstellt wurden, zeigen eine Genauigkeit von etwa 60% bis 80% und auch die Rangfolgebildung ist bei diesen Modellen stabil, das heißt es kommt nicht zu einer übermäßig großen Anzahl von Fehlklassifikationen. Diese Modelle sind daher recht gut geeignet für eine Testfallpriorisierung. Auch das Modell, welches mit mehreren vorhergehenden SW-Versionen erstellt und auf die neuste Version angewendet wurde, schneidet hinsichtlich der Genauigkeit recht gut ab, versagt jedoch bei der Rangfolgebildung, der Grundlage zur Priorisierung von Testfällen. Das Modell, mit welchem die Anwendbarkeit auf neue Testfälle geprüft werden kann, weist eine eher mittelmäßige Genauigkeit auf, weshalb eine zuverlässige Priorisierung von neu hinzugekommenen Testfällen nicht in jedem Fall gewährleistet werden kann, da dieser Vorgang von der konkreten Ausprägung der Testfallmerkmale abhängig ist.

### 5.5.2 Inhaltliche Bewertung der Resultate

Es kann im Rahmen dieser Arbeit nicht beurteilt werden, ob das erzeugte Klassifikationsmodell das Projektziel der Effizienzsteigerung beim Einsatz von Testressourcen erfüllt, denn dies ist erst nach dessen Anwendung beim Test der nächsten Produktversion (Release) empirisch evaluierbar. Auch hinsichtlich des zweiten Projektziels, der Generierung zutreffender Aussagen über die Schwächen der Software, kann das Modell im Rahmen dieser Arbeit inhaltlich nicht bewertet werden, da, zur Feststellung des tatsächlichen Vorliegens der ermittelten Schwächen, eine intensive Prüfung der Software von einem Entwickler des Intelligent Miner durchgeführt werden müsste.

## 5.6 Deployment

Diese Phase umfasst, entsprechend des CRISP-DM-Prozesses alle Tätigkeiten zur Integration des Modells in eine Anwendung innerhalb der realen Geschäftsumgebung.

### 5.6.1 Modellanwendung

Die Testfälle, die von einem Testrahmen für den nächsten Regressionstest zu verwenden sind, können basierend auf ihrer Fehlerentdeckungswahrscheinlichkeit durch einen Auswahlmechanismus bestimmt werden, der in einer Anwendung implementiert werden kann. Dieses Auswahlverfahren soll nachfolgend, anhand von beispielhaften SQL-Anweisungen erläutert werden.

Zur Bestimmung der Fehlerentdeckungswahrscheinlichkeit, welche die Grundlage der Auswahl der Testfälle für den jeweils nächsten Test darstellt, wird das erstellte Modell auf die Testfälle einzeln angewendet und die Konfidenz (engl. confidence), also die Wahrscheinlichkeit, dass ein Testfall einen Fehler aufdeckt, berechnet. Anhand eines Grenzwertes dieser Wahrscheinlichkeiten werden später die Testfälle in zwei Gruppen eingeteilt, die kritischen und die weniger kritischen. Die kritischen sind alle im nächsten Test zu berücksichtigen, während aus der Menge der weniger kritischen wiederum eine Teilmenge zufällig zu wählen ist. Der Grenzwert kann mit Hilfe der Ergebnisse aus dem Gains-Chart des erstellten Modells ermittelt werden. Dazu werden zuerst die Gains-Chart-Ergebnisse in Form einer virtuellen Tabelle mittels

```
1  SELECT CAST (ROWCOUNT AS SMALLINT)   AS ROWCOUNT,
2         CAST (SUMACTUAL AS SMALLINT) AS SUMACTUAL,
3         CAST (THRESHOLD AS DEC(5,2)) AS THRESHOLD
4  FROM TABLE (IDMMX.DM_getGainsChart ((SELECT RESULT
5                                       FROM IDMMX."CLASTESTRESULTS"
6                                       WHERE ID='MODELNAME'), 'failed'))
7  ORDER BY NUMROWS_IN_SUBSET;
```

aus dem Modell extrahiert. Die erste Spalte in der resultierenden Tabelle (ROWCOUNT) enthält die Anzahl der Testfälle in den kumuliert aus der gesamten Testfallmenge gebildeten Untergruppen, die im graphischen Gains-Chart an der X-Achse angetragen sind (als Anzahl oder Prozentsatz). Für diese Untergruppen ist in der zweiten Spalte (SUMACTUAL) die Anzahl derjenigen Testfälle enthalten, bei denen der vorhergesagte Zielwert („failed“) korrekt war und die beim graphischen Gains-Chart an der Y-Achse (ebenfalls als reine Anzahl oder als Prozentsatz) festgehalten sind. In der dritten Spalte (THRESHOLD) befindet sich der geringste Konfidenzwert (Fehlerentdeckungswahrscheinlichkeit), der in der jeweiligen Untergruppe bei einem Testfall vorkommt. Der Grenzwert (engl. threshold) zur Teilung der Testfälle in kritische und weniger kritische kann nun durch paarweises Vergleichen der Konfidenzwerte in der dritten Spalte und durch Selektieren des Konfidenzwertes mit der größten Abweichung zum nächsten Wert ermittelt und in einer Variable in der Anwendung gespeichert werden.

Die Namen der als kritisch eingestuften Testfälle, die in vollständiger Zahl im jeweils nächsten Test einzusetzen sind, können anschließend mit Hilfe eines in den Programmcode eingebetteten SQL-Befehls der Form

```
1  SELECT TC_NAME
2  FROM TESTCASE_SCORES
3  WHERE CONFIDENCE >= '' + threshold + ''
4  ORDER BY CONFIDENCE DESC;
```

unter Verwendung der Variable mit dem enthaltenen Grenzwert (hier „threshold“) und einer absteigenden Sortierung der Testfälle nach ihrer Fehlerentdeckungswahrscheinlichkeit abgerufen

und in eine Liste eingetragen werden. Aus der Menge der verbleibenden Testfälle, die mit dem SQL-Befehl

```
1 SELECT TC_NAME
2 FROM TESTCASE_SCORES
3 WHERE CONFIDENCE < '' + threshold + ''
4 ORDER BY CONFIDENCE DESC;
```

abgeholt und in eine eigene Liste gespeichert werden können, ist eine bestimmte Anzahl mit Hilfe eines Zufallsgenerators zu selektieren. Diese Anzahl kann anhand des, im Abschnitt „Modellverwendung“ beschriebenen Kosten-Nutzen-Verhältnisses berechnet werden. Der für einen automatisierten Regressionstest verwendete Testrahmen kann die beiden Listen mit Testfallnamen schließlich zur effizienteren Testdurchführung verwenden.

### 5.6.2 Beispiel für die Einbindung des Verfahrens in den Testprozess

Voraussetzung für die Integration der automatischen Auswertung von Testergebnissen zusammen mit den Testparametern in die reale Umgebung ist die Pflege eins Repositories zur Archivierung der Testberichte vergangener Regressionstests. Außerdem ist das gesamte Verfahren unter Verwendung des Data Mining zur Testfallpriorisierung und -auswahl im jeweils verwendeten Testrahmen für den automatisierten Regressionstest zu implementieren.

Der Testrahmen ruft zuerst einen Parser auf, der alle benötigten Daten sammelt und in eine Datenbanktabelle eines Data-Warehouse einträgt. Der Parser beginnt dazu mit dem Auslesen der Testberichte im Repository zur Extrahierung der Testergebnisse der vergangenen Tests. Anschließend ließt er die Parameter aller verfügbaren Testfälle (sofern diese nicht mehr aus vorherigen Analyseprozessen im Data Warehouse vorliegen) aus den Testfalldefinitionen und ermittelt die Eigenschaften der einzelnen Testfallparameter. Nachdem der Parser das Auslesen beendet hat, ruft der Testrahmen ein SQL-Skript zur Vorverarbeitung der Daten, zur nachfolgenden Erstellung eines Klassifikationsmodells (nur zu Beginn der Entwicklung eines neuen SW-Releases) und zur Anwendung des Modells (Scoring) auf die in der Datenbank gespeicherten Testfälle auf. Mit den priorisierten Testfällen, als Ergebnis des Scorings, ist in der Form weiter zu verfahren, wie es im vorhergehenden Abschnitt Modellanwendung beschrieben ist (Selektion der Testfälle und Abholung der Namen der im nächsten Regressionstest zu verwendenden Testfälle). Nach Abschluss der Testdurchführung ist der Testbericht mit den neuen Testergebnisse wieder im Repository für die nächste Analyse zu speichern. Beim nächsten Regressionstest beginnt der eben beschriebene Prozess von vorn.

# 6 Zusammenfassung und Ausblick

## 6.1 Ergebnisse

Die vorliegende Arbeit präsentiert einen Ansatz zur systematischen Auswertung von Software-Tests mit dem Ziel der Untersuchung einer Software auf Schwachstellen, also die am stärksten fehlerbehafteten Bestandteile einer Software, und der Priorisierung und Selektion von Testfällen für den dynamischen Regressionstest zur Erhöhung der Effizienz des Einsatzes von Ressourcen bei der Durchführung eines Tests. Eine Effizienzsteigerung kann dadurch bewirkt werden, dass diejenigen Testfälle, welche Software-Komponenten mit der größten Anfälligkeit für Fehler testen, an den Anfang eines Tests gelegt werden und auf diese Weise eine frühzeitige Aufdeckung und Behandlung von Fehlern möglich ist. Darüber hinaus kann die bevorzugte Ausführung der wichtigsten Testfälle zur Offenbarung einer größtmöglichen Anzahl von Fehlern dafür sorgen, dass im Falle eines vorzeitigen Testabbruchs, zum Beispiel aufgrund aufgebrauchter Testressourcen, ein Großteil der Fehler bereits gefunden werden konnte.

Der beschrieben Ansatz basiert auf einer Data-Mining-Analyse zur Klassifizierung der, im Sinne einer Entdeckung von Fehlern, besonders wichtigen und weniger wichtigen Testfälle, aufbauend auf einer Untersuchung der Zusammenhänge zwischen den Eingaben, in Form von Testfallparametern, und den Ausgaben, den Resultaten, der Tests. Anhand dieser Klassifizierung, unter Angabe einer Wahrscheinlichkeit für die Zugehörigkeit eines Testfalls zur Klasse der für einen effizienten Test besonders relevanten Testfälle, ist eine Priorisierung und Selektion der Testfälle möglich. Da für diese Klassifizierung der Testfälle ein Entscheidungsbaumverfahren eingesetzt wird, welches die Testeingaben, die für die Klassifizierung der Testfälle von entscheidender Bedeutung sind, an den Knoten des Entscheidungsbaumes in einer leicht verständlichen Form präsentiert, können an diesen Knoten die Bezeichnungen der als besonders fehleranfällig eingestuften Software-Komponenten sowie weiterer Merkmale, die ein Anzeichen für qualitative Probleme in der Software sind, abgelesen werden. Die beschriebene Vorgehensweise wird von der Abbildung 6-1 noch einmal verdeutlicht.

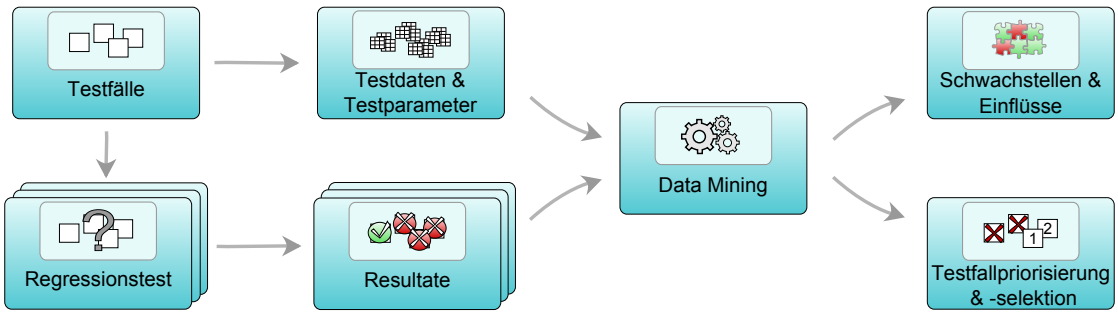



**Abbildung 6-1: Vorgehensweise bei der systematischen Auswertung von SW-Tests mittels Data Mining**

Um diese Vorgehensweise in der Praxis nachzuvollziehen, wurde ein reales Software-Produkt der Analyse mittels Data Mining unterzogen. Dazu wurden alle der in einem Data-Mining-Projekt

notwendigen Schritte durchgeführt, von der Erfassung und Vorbereitung der als Eingabe für die Analyse dienenden Daten, über die Wahl des am besten geeignet erscheinenden Data-Mining-Verfahrens und die Selektion von optimalen Parametern für dieses Verfahren zur Erstellung eines präzisen Modells für die Klassifizierung der Testfälle, sowie die Beurteilung der Eignung des Modells zur Erfüllung der genannten Ziele der Analyse bis hin zur Beschreibung eines möglichen Vorgehens zur Priorisierung und Selektion der Testfälle, welches eine Kosten-Nutzen-Analyse zur Bestimmung der ökonomisch gesehen besten Anzahl zu selektierender Testfälle einschließt.

Die praktische Anwendung des Verfahrens im Rahmen der Fallstudie hat gezeigt, dass die Erstellung eines Klassifikationsmodells mit den Testergebnissen der letzten Software-Version zur Bestimmung ihrer schwächsten Komponenten sowie zur Priorisierung und Auswahl der für den Test der aktuellen Version der Software benötigten Testfälle zuverlässiger ist, als die Erstellung des Modells auf Basis der Testergebnisse mehrerer früherer SW-Versionen. Das erstellte Modell weist eine recht hohe Genauigkeit auf, das heißt es ist in der Lage 80% der in der früheren SW-Version fehlgeschlagenen Testfälle korrekt als fehlgeschlagen zu klassifizieren. Mit dem Einsatz dieses Modells wäre es möglich, durch die Ausführung von 1% der zur Verfügung stehenden Testfälle bereits zwischen 60% und 80% der Fehler in der Software zu offenbaren.

## 6.2 Kritische Würdigung der Arbeit

In der vorliegenden Arbeit wurde ein Verfahren aufgezeigt, mit welchem die Analyse der Qualität einer Software und die Priorisierung, in Kombination mit der Selektion, von Testfällen anhand eines Black-Box-Ansatzes automatisiert durchführbar ist. Ein Vorteil des vorgestellten Ansatzes gegenüber einigen anderen in der wissenschaftlichen Literatur beschriebenen Verfahren zur Analyse der Software-Qualität, wie der Orthogonal Defect Classification, ist die Objektivität bei der Bestimmung der Qualität, da das Verfahren ohne händisch vorgenommene Einschätzungen von Entwicklern oder anderen Projektmitarbeitern auskommt, sondern stattdessen auf einer computergestützten Data-Mining-Analyse unter Verwendung statistischer Methoden basiert.

In der Arbeit wurde außerdem die konkrete Umsetzung des Verfahrens aufgezeigt und ist damit leichter nachzuvollziehen als die Vorgehensweise in vielen anderen wissenschaftlichen Arbeiten, in denen entweder nur die theoretischen Grundlagen erläutern werden oder bei denen zwar eine empirischer Versuchsaufbau dargestellt wird, jedoch in den meisten Fällen ohne die konkreten Aufgaben zur Erfassung und Vorbereitung der Daten. Dabei zählen diese Phasen zu den wichtigsten und am meisten mit Bedacht durchzuführenden im gesamten DM-Prozesses, denn ohne eine geeignete Datenvorbereitung kann in den meisten Fällen kein sinnvolles Analyseergebnis erreicht, also kein wertvolles Wissen aus den Daten gezogen werden.

Verbesserungspotenzial besteht noch bei der Ermittlung der Fehlerentdeckungswahrscheinlichkeit. Die Anzahl der mit einem einzelnen Testfall real aufgedeckten Fehler wäre ein genaueres Maß für die Fehlerfindungsrate (um Fehleranfälligkeit einer Komponente zu bestimmen) als der jetzige Ansatz mit einer Häufigkeit der pro Komponente fehlgeschlagenen Testfälle, denn mit dieser realen Fehleranzahl würden die von einem Testfall doppelt entdeckten Fehler nicht zur Erhöhung der Fehleranfälligkeit beitragen. Dieses Vorgehen würde jedoch eine zusätzliche umfangreiche Datenerfassung aus bestehenden Fehlerdatenbanken voraussetzen.

Da das vorgestellte Verfahren direkt in eine produktive Testumgebung, das heißt in den Testrahmen eingebettet werden kann, ist es mit geringem personellen Aufwand und dementsprechend auch mit geringeren Kosten durchführbar als konventionelle statistische Analysen, die von Hand durchzuführen sind. Da es sich um einen Black-Box-Ansatz handelt, bei dem der strukturelle Aufbau einer Software keine Beachtung findet, könnte das entwickelte Verfahren theoretisch auch auf andere SW-Projekte übertragen werden.

## 6.3 Erfahrungen aus dem Projekt

Aus dieser Arbeit kann für zukünftige Projekte mitgenommen werden, dass eine zu Projektbeginn erstellte, grafische Übersicht der durchzuführenden Aufgaben und benötigten Ressourcen vorteilhaft für den Verlauf des gesamten Projektes sein kann. Denn auf diese Weise können, unter Einholung der entsprechenden Informationen, Aufgaben mit hohem Zeitbedarf, Abhängigkeiten zwischen einzelnen Tätigkeiten sowie die Knappheit von Ressourcen frühzeitig sichtbar gemacht werden. Die bevorstehenden Aufgaben können anhand dieser Erkenntnisse in eine geeignete Reihenfolge gebracht werden, um Ressourcen effektiv zu nutzen und die mit knappen Ressourcen sowie auftauchenden Problemen verbundenen Verzögerungen einfacher ausgleichen beziehungsweise von Vornherein vermeiden zu können.

## 6.4 Ausblick

Die Überprüfung der Übertragbarkeit des entwickelten Verfahrens zur Bestimmung fehleranfälliger Software-Bestandteile und zur Priorisierung von Testfällen auf andere SW-Projekte könnte zum Beispiel Inhalt weiterführender Arbeiten sein. Die Implementierung des Analyseverfahrens im aktuellen Testrahmen, zum einen zur Bereitstellung der Testfallpriorisierung und -selektion im Test und zum anderen zur praktischen Evaluierung der in der Arbeit präsentierten Analyseergebnisse wäre ebenfalls als Thema einer künftigen Arbeit denkbar. Auch die Integration von Daten aus Fehlerdatenbanken und einer Quellcode-Versionsverwaltung in die Data-Mining-Analyse zur Gewinnung weiterer nützlicher Informationen bezüglich der Qualität der betrachteten Software wäre als Weiterführung diese Arbeit vorstellbar.

# Anhang A: Datenerfassung und Datenvorverarbeitung

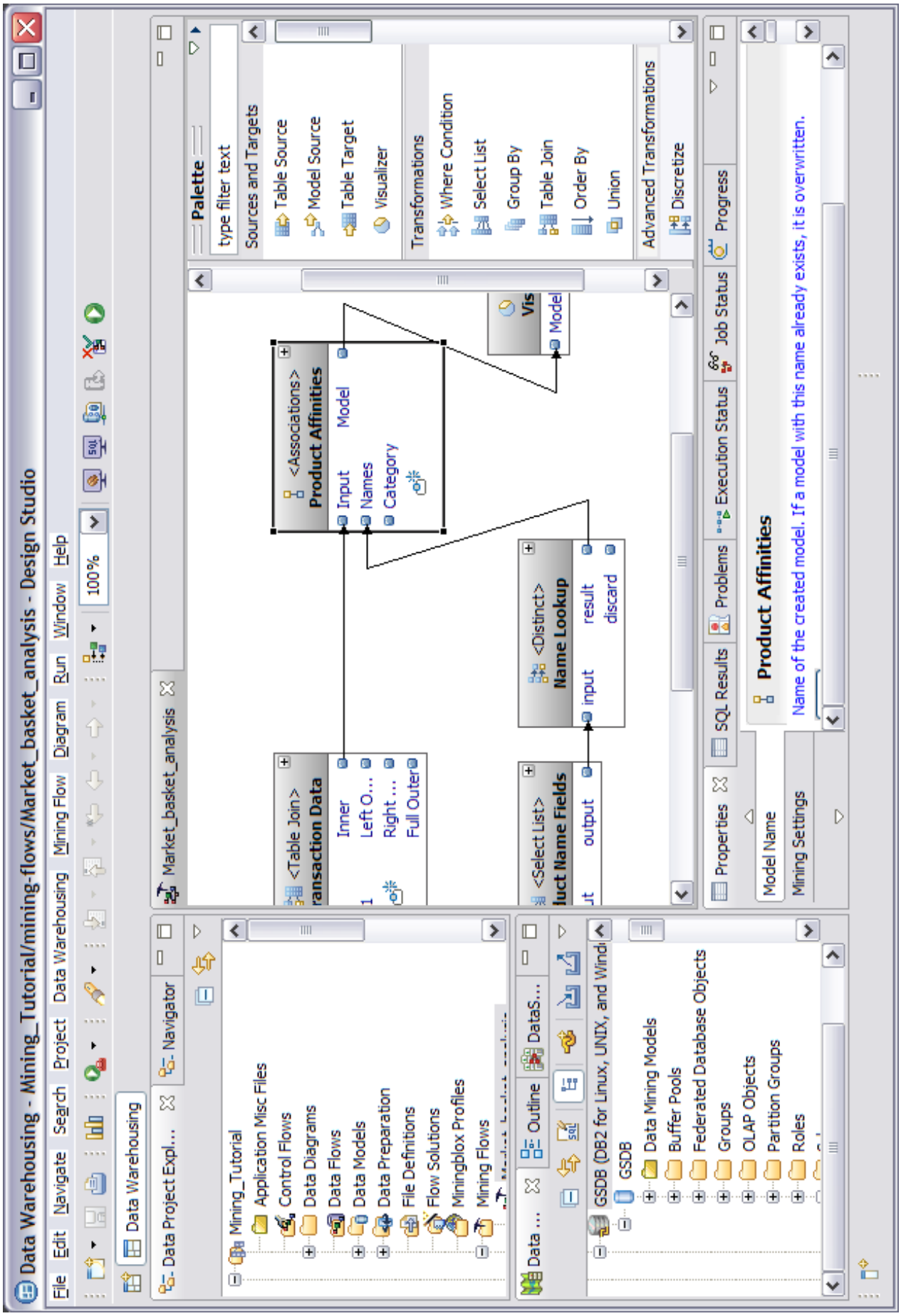



**Abbildung A-1: Grafische Benutzeroberfläche des Design Studio**

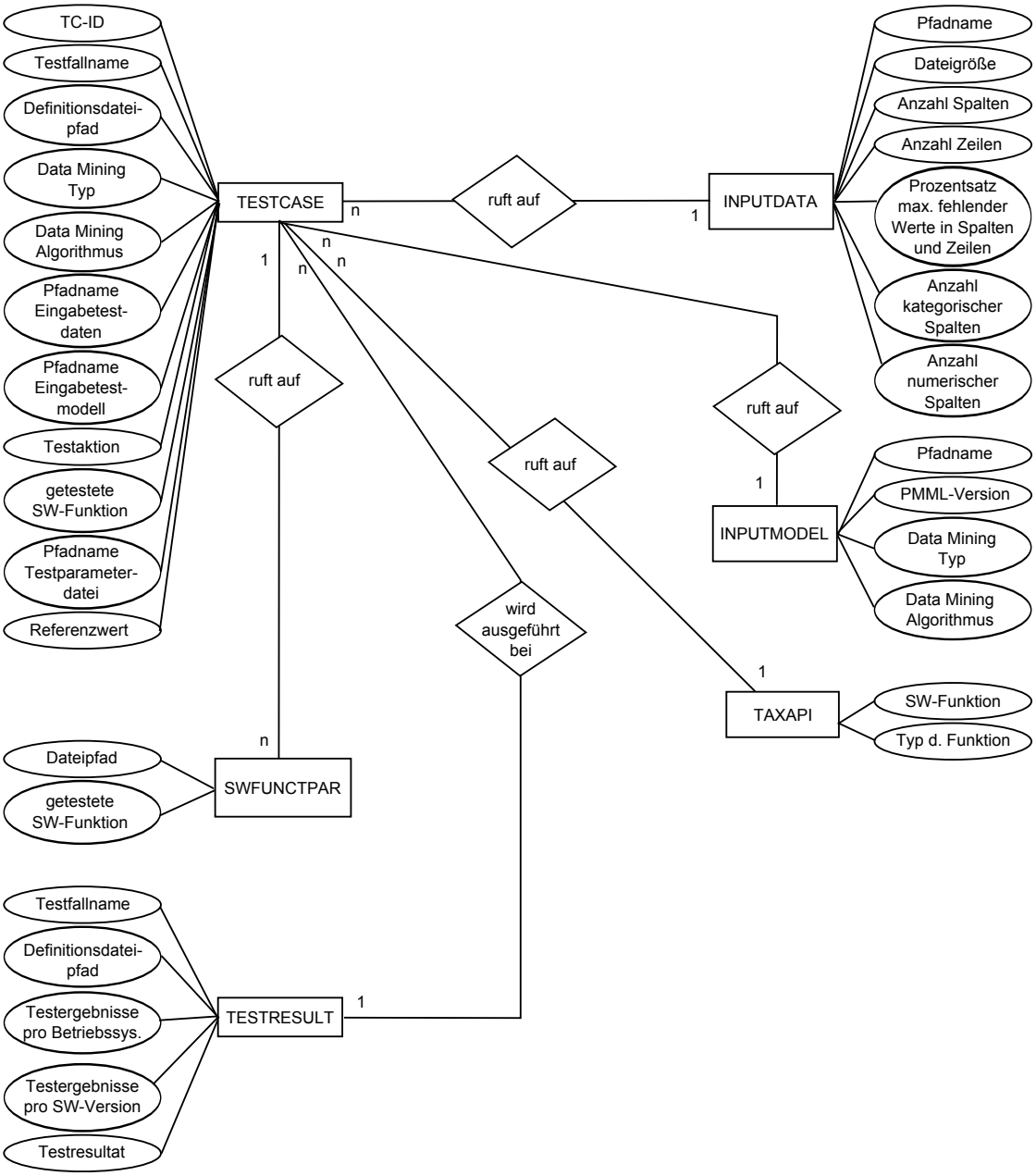



**Abbildung A-2: Datenbankschema**

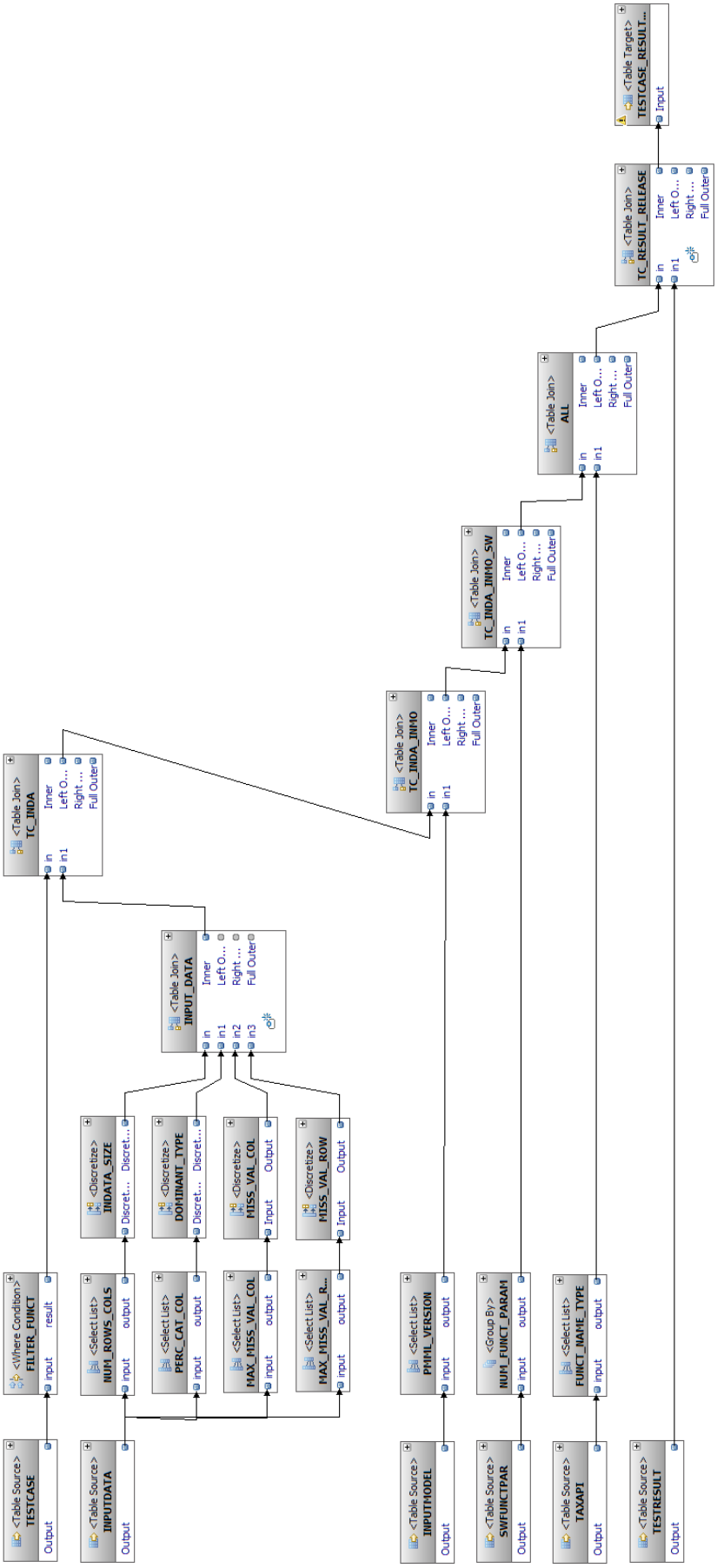
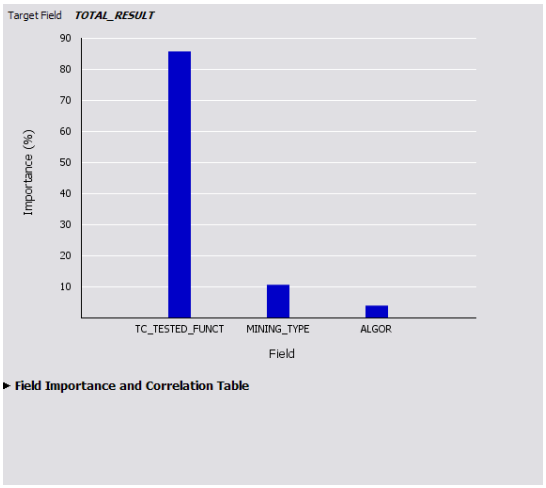
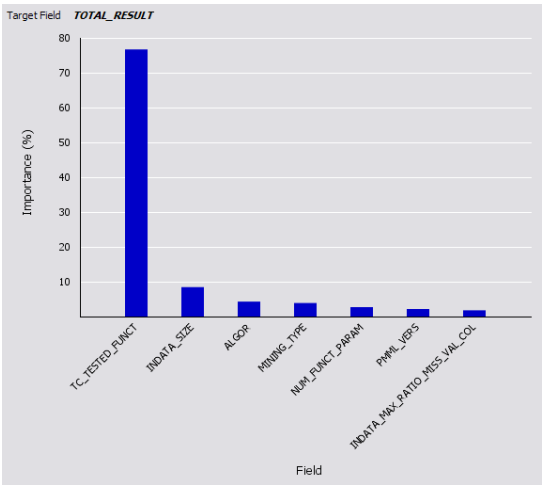



**Abbildung A-3: Data-Mining-Flussdiagramm zur Datenvorverarbeitung**

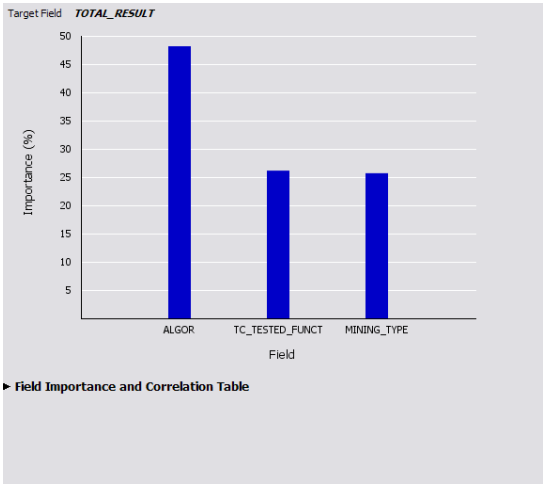
# Anhang B: Modellbildung und Modellbeurteilung



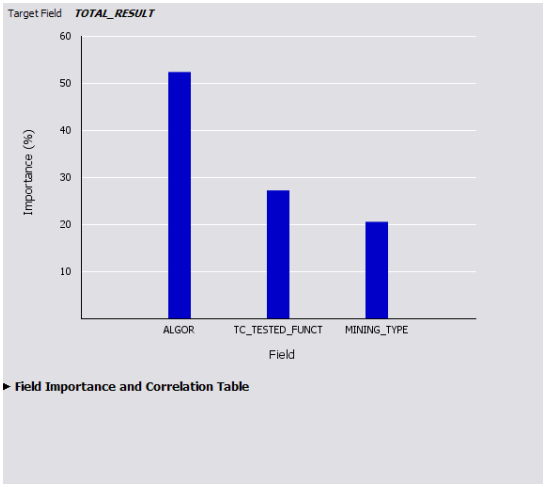

Wichtigkeit der Merkmale bei Random Split



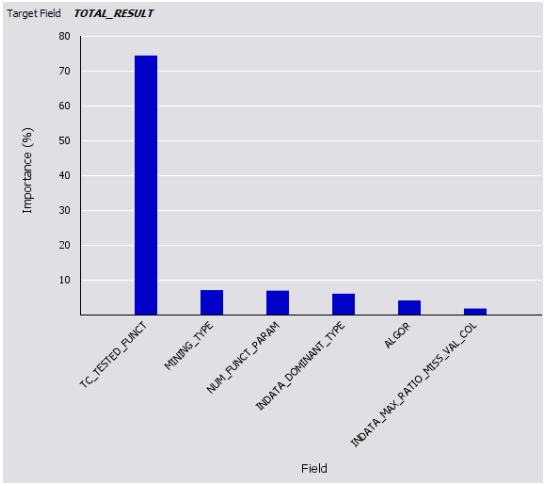

Wichtigkeit der Merkmale bei
Vers 1 = Training & Vers 2 = Test




Wichtigkeit der Merkmale bei
Vers 2 = Training & Vers 3 = Test

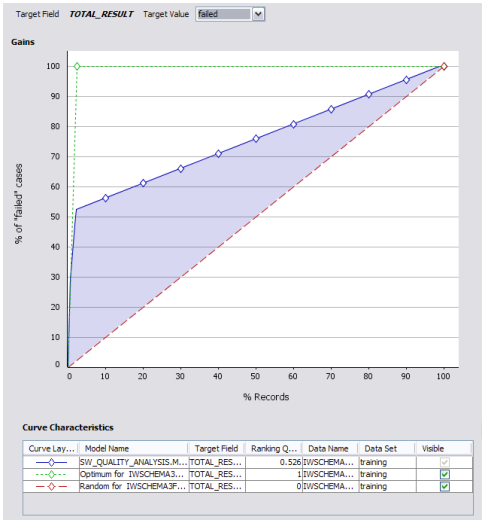



Wichtigkeit der Merkmale bei
Vers 3 = Training & Vers 4 = Test

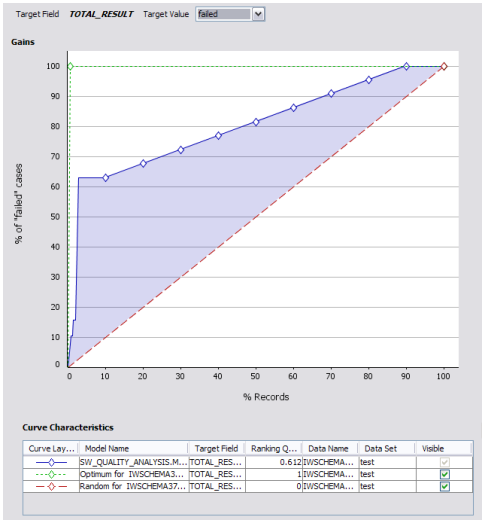



Wichtigkeit der Merkmale bei
Vers 1+2+3 = Training & Vers 4 = Test

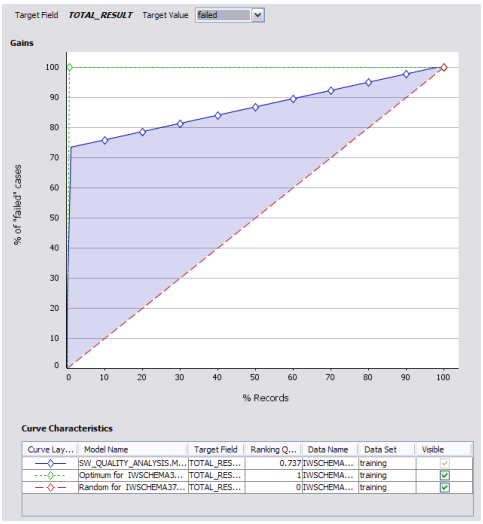
**Abbildung B-1: Wichtigkeit der Merkmale bei den verschiedenen Test-Design-Formen**



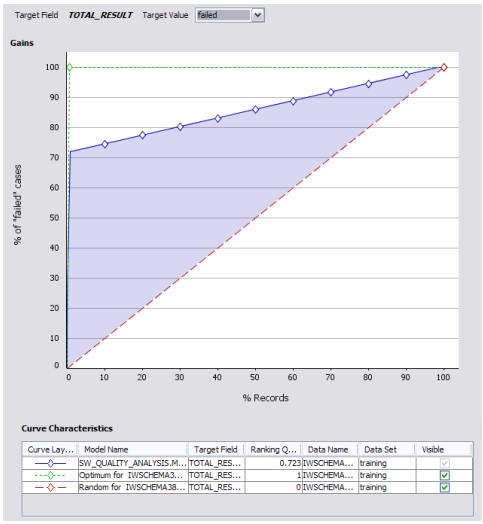

Gains-Chart bei Random Split



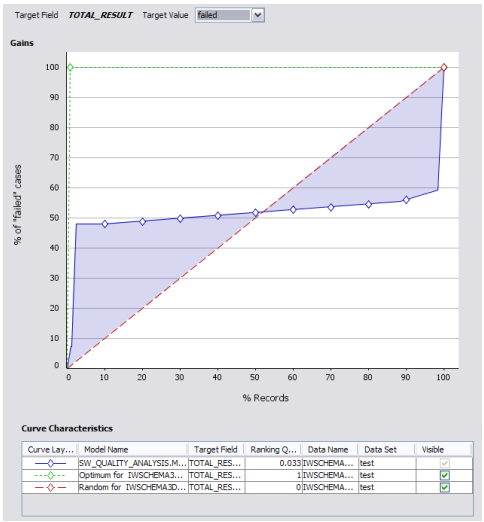

Gains-Chart bei
Vers 1 = Training & Vers 2 = Test




Gains-Chart bei
Vers 2 = Training & Vers 3 = Test




Gains-Chart bei
Vers 3 = Training & Vers 4 = Test




Gains-Chart bei
Vers 1+2+3 = Training & Vers 4 = Test

**Abbildung B-2: Gains-Charts der Modelle aus den verschiedenen Test-Designs**

# Literaturverzeichnis

[1] IBM Corporation: *Das IBM Forschungs- und Entwicklungszentrum in Deutschland*. – Im Internet verfügbar unter http://www-05.ibm.com/de/entwicklung/ueberuns/index.html am 01.05.2011.

[2] IBM Corporation: *IBM Software Group*. – Im Internet verfügbar unter http://www-05.ibm.com/de/ibm/leistungen/software_group.html am 01.05.2011.

[3] Fayyad, Usama ; Piatetsky-Shapiro, Gregory ; Smyth, Padhraic: *From Data Mining to Knowledge Discovery in Databases*. Menlo Park, Kalifornien : AAAI Press, 1996

[4] Schneider, W. ; Toplak, W.: Verkehrsprognosen mit Visuellem Data Mining und Künstlicher Intelligenz. In: *e & I Elektrotechnik und Informationstechnik* 125 (2008), Nr. 6, S. 232–237. – Im Internet verfügbar unter http://www.springerlink.de/content/g47h27k7k8565771/ am 31.05.2011.

[5] Pruess, Paul: *Data Mining in Search of Hidden Oil*. 1997. – Im Internet verfügbar unter http://www.lbl.gov/Science-Articles/Archive/oil-data-mining.html am 19.07.2011

[6] Hippner, Hajo ; Wilde, Klaus: Data Mining im CRM. In: Helmke, Stefan (Hrsg.) ; Uebel, Matthias F. (Hrsg.) ; Dangelmaier, Wilhelm (Hrsg.): *Effektives Customer Relationship Management - Instrumente - Einführungskonzepte - Organisation*, 2008, S. 205–225

[7] Ballard, Chuck ; Rollins, John ; Ramos, Jo ; Perkins, Andy ; Hale, Richard ; Dorneich, Ansgar ; Milner, Edward C. ; Chodagam, Janardhan: *Dynamic Warehousing: Data Mining Made Easy*. IBM Corporation, 2007. – ISBN 0738488860

[8] Beekmann, Frank ; Chamoni, Peter: Verfahren des Data Mining. In: Chamoni, Peter (Hrsg.) ; Gluchowski, Peter (Hrsg.): *Analytische Informationssysteme Business Intelligence-Technologien und -Anwendungen*, 2006, S. 263–282

[9] Chaovalitwongse, W. ; Pardalos, P. M. ; Iasemidis, L. D. ; Suharitdamrong, W. ; Shiau, D. S. ; Dance, L. K. ; Prokopyev, O. A. ; Boginskia, V. L. ; Carney, P. R. ; Sackellares, J. C.: Data Mining in EEG: Application to Epileptic Brain Disorders. In: *Data Mining in Biomedicine* Bd. 7, Springer US, 2007. – ISBN 978–0387693187

[10] Kaushik, Saroj ; Singhal, Naman: Pattern Prediction in Stock Market. In: *AI 2009: Advances in Artificial Intelligence: 22nd Australasian Joint Conference, Melbourne, Australia, December 1-4, 2009. Proceedings* Bd. 5866/2009, Springer Berlin Heidelberg, 2009

[11] The CRISP-DM consortium: *CRoss Industry Standard Process for Data Mining*. – Im Internet verfügbar unter http://www.crisp-dm.org/index.htm am 25.02.2011.

[12] Chapman, Pete ; Clinton, Julian ; Kerber, Randy ; Khabaza, Thomas ; Reinartz, Thomas ;

Shearer, Colin ; Wirth, Rudiger: *CRISP-DM 1.0 Step-by-step data mining guide*. 2000. – Im Internet verfügbar unter http://www.crisp-dm.org/CRISPWP-0800.pdf am 25.02.2011.

[13] Guazzelli, Alex ; Lin, Wen-Ching ; Jena, Tridivesh: *PMML in Action: Unleashing the Power of Open Standards for Data Mining and Predictive Analytics*. CreateSpace, 2010. – ISBN 978–1452858265

[14] The Data Mining Group Consortium: *Data Mining Group*. – Im Internet verfügbar unter http://www.dmg.org/ am 04.03.2011.

[15] Kurz, Andreas: *Data Warehousing Enabling Technology*. Bonn : MITP-Verlag, 1999. – ISBN 978–3826640452

[16] IBM Corporation: *Data-Warehouse*. – Im Internet verfügbar unter http://www-01.ibm.com/software/de/data/infosphere/warehouse/ am 29.01.2011.

[17] IBM Corporation: *Installation architecture for InfoSphere Warehouse on multiple computers*. – Im Internet verfügbar unter http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp?topic=/com.ibm.dwe.install.doc/architecture_luw.html am 29.01.2011.

[18] Burnus, Heinz: *Datenbankentwicklung in IT-Berufen Eine praktisch orientierte Einführung mit MS Access und MySQL*. Vieweg Verlag, 2008

[19] IBM Corporation: *DB2 Enterprise Server Edition*. – Im Internet verfügbar unter http://www-01.ibm.com/software/data/db2/linux-unix-windows/edition-enterprise.html am 01.05.2011.

[20] International Organization for Standardization: *ISO/IEC 13249-6:2006 (Information technology – Database languages – SQL multimedia and application packages – Part 6: Data mining)*. 2006. – Im Internet verfügbar unter http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=38648 am 25.05.2011.

[21] Rahm, Erhard: *Objekt-relationale DBS: SQL:1999 und SQL:2003*. 2008. – Im Internet verfügbar unter http://dbs.uni-leipzig.de/file/dbs2-ss08-kap4.pdf am 25.05.2011.

[22] IBM Corporation: *Introducing database objects and data mining in SQL*. – Im Internet verfügbar unter http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.im.model.doc/c_introducing_database_objects_and_data_mining_in_sql.html am 25.05.2011.

[23] IBM Corporation: *Generating a SQL script for your mining flow*. – Im Internet verfügbar unter http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.datatools.datamining.doc/t_generatingasqlscriptforyourminingflow.html am 21.07.2011.

[24] International Organization for Standardization: *ISO/IEC 9126-1:2001 (Software engineering – Product quality – Part 1: Quality model)*. 2001. – Im Internet verfügbar unter http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=22749 am 13.05.2011.

[25] Partsch, H.: *Qualitätssicherung - Allgemeines*. 2010. – Im Internet verfügbar unter http://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.170/lehre/ss2010-swt2/folien/16.QS_allgemein.pdf am 08.01.2011.

[26] Spillner, Andreas ; Linz, Tilo: *Basiswissen Softwaretest*. dpunkt.verlag, 2007

[27] Armbrust, Ove ; Ochs, Michael ; Snoek, Björn: *Stand der Forschung von Software-Tests und deren Automatisierung*. Fraunhofer IESE, 2004. – Im Internet verfügbar unter http://www.ove-armbrust.de/downloads/Armbrust-RLRLP-SoA-Testing.pdf am 12.07.2011.

[28] Balzert, Helmut: *Lehrbuch der Software-Technik: Software-Entwicklung*. Spektrum Akademischer Verlag Heidelberg Berlin, 2001

[29] International Organization for Standardization: *ISO 9000:2005 (Quality management systems – Fundamentals and vocabulary)*. 2005. – Im Internet verfügbar unter http://www.iso.org/iso/catalogue_detail?csnumber=42180 am 19.07.2011.

[30] dpa, N24: *Defekte Kraftstoffleitung - Erneuter Massenrückruf bei Toyota*. – Im Internet verfügbar unter http://www.n24.de/news/newsitem_6616405.html am 21.05.2011.

[31] Walraven: *Materialeigenschaften von Kunststoffen*. – Im Internet verfügbar unter http://www.walraven.com/library/documents/de/Material-Properties-data-sheet-DE.pdf am 21.05.2011.

[32] Freinatis, Stefan: *Sicherheit und Zuverlässigkeit digitaler Systeme*. 2005. – Im Internet verfügbar unter http://ti.uni-due.de/ti/de/education/teaching/ws0506/szds/Folien-Netz.pdf am 22.05.2011.

[33] Tränkler, Hans-Rolf ; Fischerauer, Gerhard: Grundlagen der Messtechnik. In: Czichos, Horst (Hrsg.) ; Hennecke, Manfred (Hrsg.): *HÜTTE - Das Ingenieurwissen*, Springer Verlag, 2008

[34] Abele, Marcus: *Modellierung und Bewertung hochzuverlässiger Energiebordnetz-Architekturen für sicherheitsrelevante Verbraucher in Kraftfahrzeugen*. Kassel, Universität Kassel, Diss., 2008

[35] Hellebrand, Sybille: *Qualitätssicherung für mikroelektronische Systeme*. 2011. – Im Internet verfügbar unter http://www.date.uni-paderborn.de/fileadmin/lehre/SS2011/Quali/folien/quali11-FT.pdf am 22.05.2011.

[36] Liggesmeyer, Peter: *Software-Qualität: Testen, Analysieren und Verifizieren von Software*. Spektrum Akademischer Verlag, 2009. – ISBN 978–3827420565

[37] Balzert, Helmut: *Lehrbuch der Software-Technik: Software Management, Software-Qualitätssicherung, Unternehmensmodellierung*. Bd. 2. Spektrum Akademischer Verlag Heidelberg Berlin, 1998. – ISBN 978–3827400659

[38] Pentti, Haapanen ; Atte, Helminen: *Failure mode and effects analysis of software-based*

*automation systems*. 2002. – Im Internet verfügbar unter http://www.fmeainfocentre.com/handbooks/softwarefmea.pdf am 18.07.2011.

[39] Schubert, Wilfried: *Vorlesung Softwaretechnik Grundlagen - Die Implementierungsphase*. 2011. – Im Internet verfügbar unter https://www.staff.hs-mittweida.de/~wschub/intranet/ss11/Fach_SWT/Fach_SWT_VL12_gross.pdf am 06.06.2011, Voraussetzung ist ein Login bei der Hochschule Mittweida

[40] Technische Universität Wien: *Testen von Spezifikationen und Programmen VU Softwarequalitätssicherung*. 2004. – Im Internet verfügbar unter http://qse.ifs.tuwien.ac.at/courses/QS/SS04/Studentenleitfaden_Beispielgruppe_1_KF_MF_210304.doc am 19.07.2011.

[41] Boehm, Barry W.: *Software Engineering Economics*. Prentice Hall, 1981

[42] Boehm, Barry W.: Guidelines for Verifying and Validating Software Requirements and Design Specifications. In: Samet, P. A. (Hrsg.): *Euro IFIP 79*, 1979, S. 711–719

[43] Versteegen, Gerhard: *Das V-Modell in der Praxis*. Heidelberg : dpunkt.verlag, 2000

[44] Beck, Kent: *Extreme Programming*. Addison-Wesley, 2000

[45] Meffert, Klaus: *JUnit Profi-Tipps*. Entwickler.Press, 2006

[46] Bauer, Thomas ; Eschbach, Robert ; Guo, Zhensheng ; Kloos, Johannes: *D-5.1: Ansatz für konditionale Testfallreduktion basierend auf vorausgegangenen Testergebnissen*. 2008

[47] Malishevsky, Alexey G. ; Rothermel, Gregg ; Elbaum, Sebastian: Modeling the Cost-Benefits Tradeoffs for Regression Testing Techniques. In: *ICSM '02 Proceedings of the International Conference on Software Maintenance*, IEEE Computer Society Washington, 2002, S. 204 – 213

[48] Chillarege, R. ; Bhandari, I.S. ; Chaar, J.K. ; Halliday, M.J. ; Moebus, D.S. ; B.K. Ray, M.Y. W.: Orthogonal defect classification-a concept for in-process measurements. In: *IEEE Transactions on Software Engineering* 18 (1992), Nr. 11, S. 943 – 956. http://dx.doi.org/10.1109/32.177364. – DOI 10.1109/32.177364

[49] Nagappan, Nachiappan ; Ball, Thomas ; Zeller, Andreas: Mining Metrics to Predict Component Failures. In: *ICSE '06 Proceedings of the 28th international conference on Software engineering*, ACM New York, 2006

[50] Hudepohl, John P. ; Aud, Stephen J. ; Khoshgoftaar, Taghi M. ; Allen, Edward B. ; Mayrand, Jean: Emerald: Software Metrics and Models on the Desktop. In: *IEEE Software* 13 (1996), Nr. 5, S. 56–60. – Im Internet verfügbar unter http://www.computer.org/portal/web/csdl/doi/10.1109/52.536459 am 23.07.2011.

[51] Ostrand, Thomas J. ; Weyuker, Elaine J. ; Bell, Robert M.: Predicting the Location and Number of Faults in Large Software Systems. In: *IEEE Transactions on Software Engineering*

31 (2005), Nr. 4, S. 340–355. – Im Internet verfügbar unter http://www.computer.org/portal/web/csdl/doi/10.1109/TSE.2005.49 am 23.07.2011.

[52] Ramler, Rudolf: *Mining Repositories for Decision-Support in Software Test Management*. 2007. – Im Internet verfügbar unter http://www.soft-net.at/new/wp-content/plugins/download-monitor/download.php?id=33 am 22.06.2011.

[53] Last, Mark ; Friedman, Menahem ; Kandel, Abraham: The Data Mining Approach to Automated Software Testing. In: *KDD '03 Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM New York, 2003. – ISBN 1581137370, S. 388–396

[54] Raamesh, Lilly ; Uma, G.V.: Knowledge Mining of Test Case System. In: *International Journal on Computer Science and Engineering* 2 (2009), Nr. 1, S. 69–73

[55] Song, Qinbao ; Shepperd, Martin ; Cartwright, Michelle ; Mair, Carolyn: Software Defect Association Mining and Defect Correction Effort Prediction. In: *IEEE Transactions On Software Engineering* 32 (2006), Nr. 4, S. 69–82. – Im Internet verfügbar unter http://www.computer.org/portal/web/csdl/doi/10.1109/TSE.2006.19 am 23.07.2011.

[56] IBM Corporation: *Decision tree classification*. – Im Internet verfügbar unter http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.im.model.doc/c_decision_tree_calssification.html am 09.07.2011.

[57] Mola, Francesco ; Siciliano, Roberta: A fast splitting procedure for classifcation trees. In: *Statistics and Computing* 7 (1997), S. 209–216

[58] Kurth, Winfried: *Entscheidungsbäume*. 2004. – Im Internet verfügbar unter http://www-gs.informatik.tu-cottbus.de/~wwwgs/gdm_v03.pdf am 15.07.2011.

[59] Bundesministeriums der Justiz ; juris GmbH: *Bundesdatenschutzgesetz (BDSG)*. 2009. – Im Internet verfügbar unter http://www.gesetze-im-internet.de/bundesrecht/bdsg_1990/gesamt.pdf am 09.05.2011.

[60] Runkler, Thomas A.: *Data Mining – Methoden und Algorithmen intelligenter Datenanalysen*. Vieweg+Teubner, 2010

[61] Olson, David L. ; Delen, Dursun: *Advanced Data Mining Techniques*. Springer Berlin Heidelberg, 2008. – ISBN 978–3540769163

[62] Bramer, Max: *Principles of Data Mining*. Springer London, 2007. – ISBN 978–1846287657

[63] Lutz, Jürgen: *Software Engineering für große Informationssysteme: Softwaretest*. 2004. – Im Internet verfügbar unter http://www.objectarchitects.de/wien2004/%2899%29Softwaretest.pdf am 19.05.2011.

[64] Vornberger, Oliver ; Müller, Olaf: *Datenbanksysteme*. 2003. – Im Internet verfügbar unter http://www-lehre.informatik.uni-osnabrueck.de/~dbs/2003/PDF/skript.pdf am 19.05.2011.

[65] IBM Corporation: *Mining-API - Referenz*. – Im Internet verfügbar unter http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.dwe.navigate.doc/dwe_mining_reference.html am 21.07.2011.

[66] IBM Corporation: *Transform operators*. – Im Internet verfügbar unter http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.dwe.sqw.doc/designing/data_flow/copstransform.html am 16.05.2011.

# Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Böblingen, 29. Juli 2011