HOCHSCHULE OSNABRÜCK

UNIVERSITY OF APPLIED SCIENCES

Fakultät Management, Kultur und Technik

Campus Lingen

Institut für Management und Technik

Studiengang Wirtschaftsinformatik

Sommersemester 2018

LV: Softwareentwicklungsprojekt

Prof. Dr. Ralf Buschermöhle

Prof. Dr. Reinhard Rauscher

Systemdokumentation

Travelling Salesman Problem

Datum: 29.09.2018

Vorname, Name	Matrikelnummer
Andrej Drobin:	654726
Julian Geerdes:	609659
Deniz Kücüktas:	653858

Inhaltsverzeichnis

1. Einleitung 2

2. Projektgrundlage 5

3. Entwurfsmuster 6

3.1 Use-Case-Diagramm 6

3.2 Anwendungsfälle 7

3.3 Datenspeicherung 9

3.4 Klassendiagramm 9

3.5 Zustandsdiagramm 11

4 Bibliotheken 12

5. Testprotokolle 12

6. Entwicklungsumgebungen 22

6.1 IntelliJ 22

6.2 Visual Paradigm 22

7 Test-Coverage 22

Eidesstattliche Erklärung 23

Abbildungsverzeichnis

Abbildung 1 Use-Case Diagramm ,,Travelling Salesman Problem"..... 7
Abbildung 2 Klassendiagramm ,,Travelling Salesman Problem“ 10
Abbildung 3 Zustandsdiagramm ,,Travelling Salesman Problem" 11
Abbildung 4 Test-Coverage …….22

Tabellenverzeichnis

Tabelle 1	Übersicht Anwendungsfälle	S.6
Tabelle 2	Anwendungsfall – ID0001	S.7
Tabelle 3	Anwendungsfall – ID0002	S.7
Tabelle 4	Anwendungsfall – ID0003	S.7,8
Tabelle 5	Anwendungsfall – ID0004	S.8
Tabelle 6	Verwendete Bibliotheken	S.11
Tabelle 7	Testfall – TN0001	S.11
Tabelle 8	Testfall – TN0002	S.11
Tabelle 9	Testfall – TN0003	S.12
Tabelle 10	Testfall – TN0004	S.12
Tabelle 11	Testfall – TN0005	S.12
Tabelle 12	Testfall – TN0006	S.12
Tabelle 13	Testfall – TE0001	S.13
Tabelle 14	Testfall – TE0002	S.13
Tabelle 15	Testfall – TE0003	S.13
Tabelle 16	Testfall – TT0001	S.14
Tabelle 17	Testfall – TT0002	S.14
Tabelle 18	Testfall – TT0003	S.14
Tabelle 19	Testfall – TT0004	S.14
Tabelle 20	Testfall – TT0005	S.15
Tabelle 21	Testfall – TT0006	S.15
Tabelle 22	Testfall – TT0007	S.15
Tabelle 23	Testfall – TT0008	S.15,16
Tabelle 24	Testfall – TW0001	S.16
Tabelle 25	Testfall – TR0001	S.16

Tabelle 26	Testfall – TG0001	S.17
Tabelle 27	Testfall – TG0002	S.17
Tabelle 28	Testfall – TG0003	S.17
Tabelle 29	Testfall – TG0004	S.18
Tabelle 30	Testfall – TG0005	S.18
Tabelle 31	Testfall – TG0006	S.18
Tabelle 32	Testfall – TG0007	S.18
Tabelle 33	Testfall – TG0008	S.19
Tabelle 34	Testfall – TG0009	S.19
Tabelle 35	Testfall – TB0001	S.20
Tabelle 36	Testfall – TB0002	S.20
Tabelle 37	Testfall – TB0003	S.20

1. Einleitung

Die Systemdokumentation ist vor allem an den Administrator gerichtet, um den Aufbau der ,,Travelling Salesman Problem“ Software nachzuvollziehen und gegeben falls weiterzuentwickeln. Die Dokumentation legt das Hauptaugenmerk auf die erstellten Entwurfsmuster. Unter anderem stellen wir Diagramme wie das Use-Case-Diagramm oder das Klassendiagramm vor. Des Weiteren wird der Inhalt aller Anwendungsfälle veranschaulicht. Hier werden die Schritte aufgezeigt, welche notwendig sind, um das ,,Travelling Salesman“ Problem zu initialisieren, grafisch darzustellen und mit einer geeigneten Heuristik zu lösen. Im Anschluss sind die einzelnen Testprotokolle und die verwendeten Programmierwerkzeuge festgehalten.

1.2 Projektgrundlage

Im Rahmen des 4 Fachsemesters des Studiengangs Wirtschaftsinformatik bekommen Studenten die Möglichkeit, angeeignetes Wissen und Fähigkeiten in Form eines Softwareentwicklungsprojektes in die Praxis umzusetzen. Unsere Aufgabe ist es dabei, das ,,Travelling Salesman Problem“

oder auf Deutsch, Das Problem des Handelsreisenden zu Lösen. Die Aufgabe der Problemstellung umfasst folgendes: Ein Handlungsreisender betrachtet eine gewisse Anzahl Städte (n-Stück), die er alle von einer Anfangsstadt so schnell wie möglich, aber nur einmal besuchen möchte und am Ende zur Ausgangsstadt zurückkehrt. Das Projekt wird in der Programmiersprache Java umgesetzt und umfasst eine Grafische Benutzeroberfläche, welche das Visualisieren ermöglicht.

2 Analyseergebnisse

In den Analyseergebnissen wird der Projektablauf bestimmt, wobei die Anforderungen als Grundlage dienen. Resultate, die aus dem Pflichten- und Lastenheft entstanden sind, werden hier zum Teil erneut aufgezeigt.

2.1 Funktionale Anforderungen

Hauptziele:

FA-1: Das Programm soll dem Benutzer ermöglichen beliebig viele Knoten und Kanten zu erstellen.

FA-2: Nearest-Neighbour dient als Standard Algorithmus.

FA-3: Knoten und Kanten sollen grafisch dargestellt werden.

FA-4: Zufällige Knoten und Kanten sollen generiert werden.

FA-5: Brute-Force Algorithmus soll zum Vergleich dienen.

FA-6: Die Tour wird als Graph ausgegeben.

3. Entwurfsmuster

Um die jeweilige Situation näher zu beschreiben, haben wir erlerntes Wissen genutzt und diverse Diagramme wie: Klassendiagramm, Use-Case-Diagramm und Zustandsdiagramm erstellt.

3.1 Use-Case-Diagramm

Ziel jedes Softwareentwicklungsprozesses ist die Erfüllung bestimmter Anforderungen. Das Use-Case-Diagramm verschafft einen Überblick über

das, was die Software im eigentlichen können muss. Wichtige Funktionen werden zueinander in Beziehungen gesetzt.

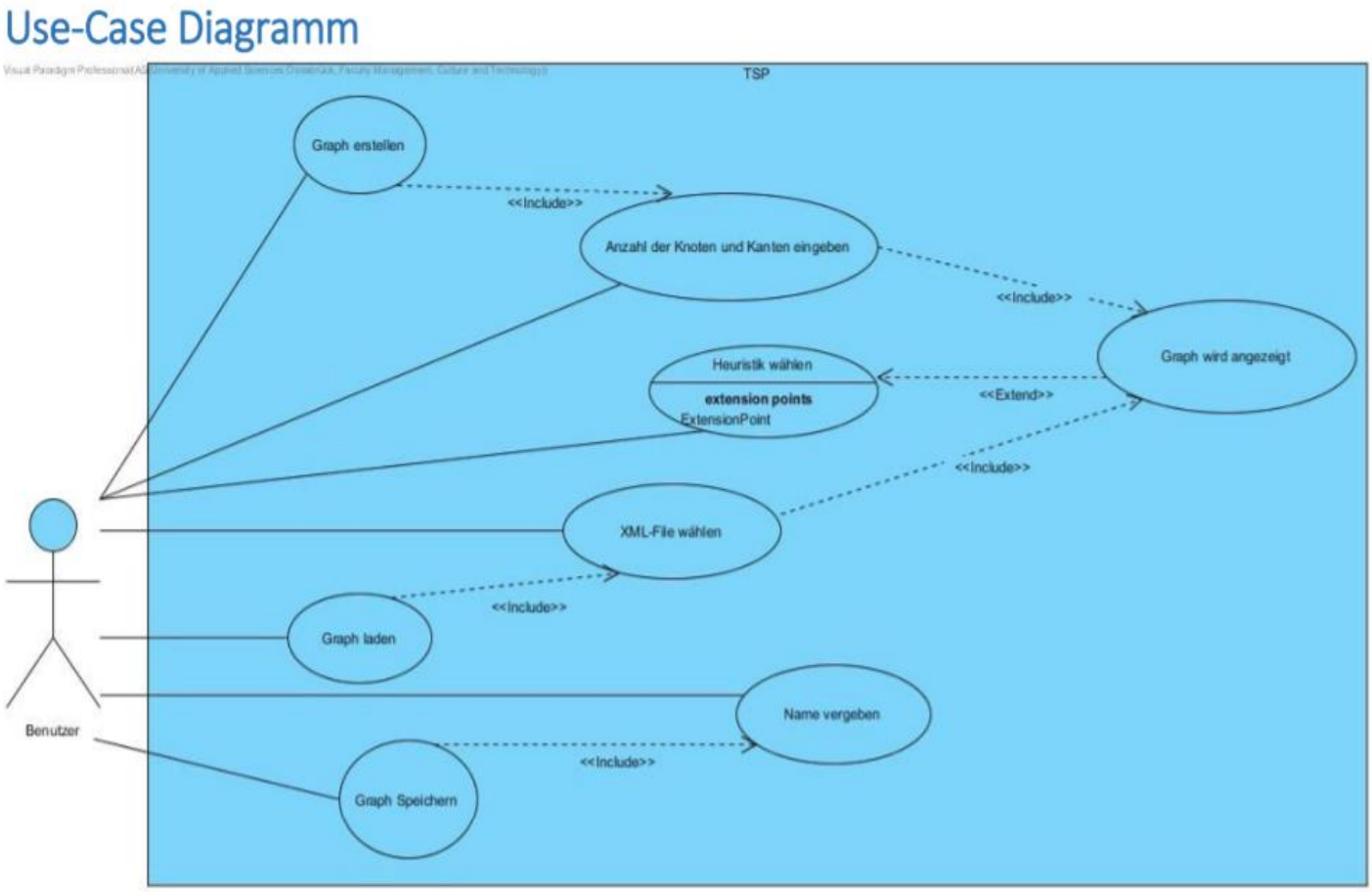


Abbildung 1 Use-Case Diagramm ,,Travelling Salesman Problem"

3.2 Anwendungsfälle

Einzelne Fallbeispiele des Systems werden durch Anwendungsfälle beschrieben. Alle Anwendungsfälle, die ein Akteur in der Software ausführen kann, werden dargestellt. Jeder Anwendungsfall aus dem Use-Case Diagramm wird in einer Tabelle zusammengefasst und spezifiziert.

ID:	Beschreibung:
A0001	Graph erstellen
A0002	Heuristik wählen
A0003	Graph laden
A0004	Graph speichern

ID	A0001
Anwendungsfall	Graph erstellen
Ziel	Verbindung zwischen Knoten und Kanten anzeigen
Kategorie	Primär
Externe Akteure	Benutzer
Vorbedingung	Knoten und Kanten wurden erstellt
Auslösendes Ereignis	Akteur beschließt einen Graphen zu erstellen
Nachbedingung Erfolg	Graph wurde erfolgreich erstellt
Nachbedingung Fehlschlag	Keine Werte wurden eingetragen
Beschreibung	Abfrage Wie viele Knoten und Kanten erstellt werden sollen.
Erweiterung	/
Alternativen	/

ID	A0002
Anwendungsfall	Heuristik wählen
Ziel	Ausgewählte Heuristik wird ausgeführt.
Kategorie	Primär
Externe Akteure	Benutzer
Vorbedingung	Heuristik wurde gewählt
Auslösendes Ereignis	Heuristik wird ausgeführt
Nachbedingung Erfolg	Heuristik wurde erfolgreich ausgeführt
Nachbedingung Fehlschlag	Graph wurde nicht oder fehlerhaft erstellt.
Beschreibung	Heuristik in einer ChoiceBox auswählen
Erweiterung	/
Alternativen	Alternativen Heuristiken

ID	A0003
Anwendungsfall	Graph laden
Ziel	Ausgewählter Graph wird angezeigt.

Kategorie	Primär
Externe Akteure	Benutzer
Vorbedingung	Ein bereits erstellter Graph existiert.
Auslösendes Ereignis	Akteur beschließt einen Graphen zu Laden
Nachbedingung Erfolg	Graph wurde erfolgreich geladen
Nachbedingung Fehlschlag	Graph existiert nicht
Beschreibung	Load From File Button klicken und die jeweilige Datei auswählen
Erweiterung	/
Alternativen	Anderes Dateiformat

ID	A0004
Anwendungsfall	Graph speichern
Ziel	Erstellter Graph soll gespeichert.
Kategorie	Primär
Externe Akteure	Benutzer
Vorbedingung	Graph wurde erfolgreich erstellt
Auslösendes Ereignis	Akteur beschließt einen Graphen zu speichern
Nachbedingung Erfolg	Graph wurde erfolgreich gespeichert.
Nachbedingung Fehlschlag	Erstellter Graph ist fehlerhaft
Beschreibung	Save into File
Erweiterung	/
Alternativen	/

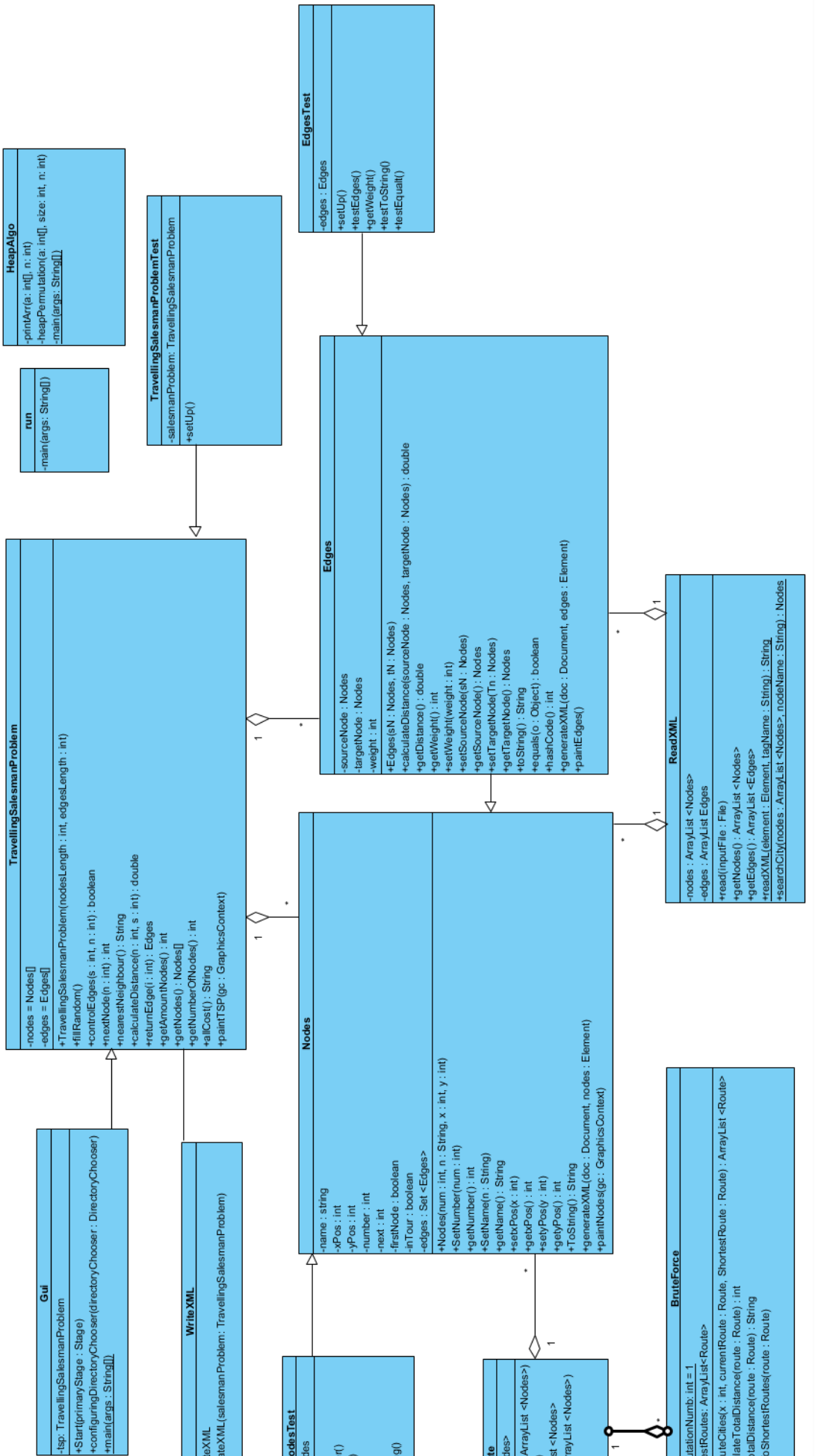
3.3 Datenspeicherung

Unsere Datenspeicherung erfolgt nicht durch eine Datenbank, sondern durch sogenannte XML-Dateien. Dieses Text-basiertes Format ermöglicht uns den Austausch von strukturierter Information. Alle benötigten Werte für die Erstellung, Speichern oder Laden eines Graphen wird mithilfe dieses Formates gespeichert.

3.4 Klassendiagramm

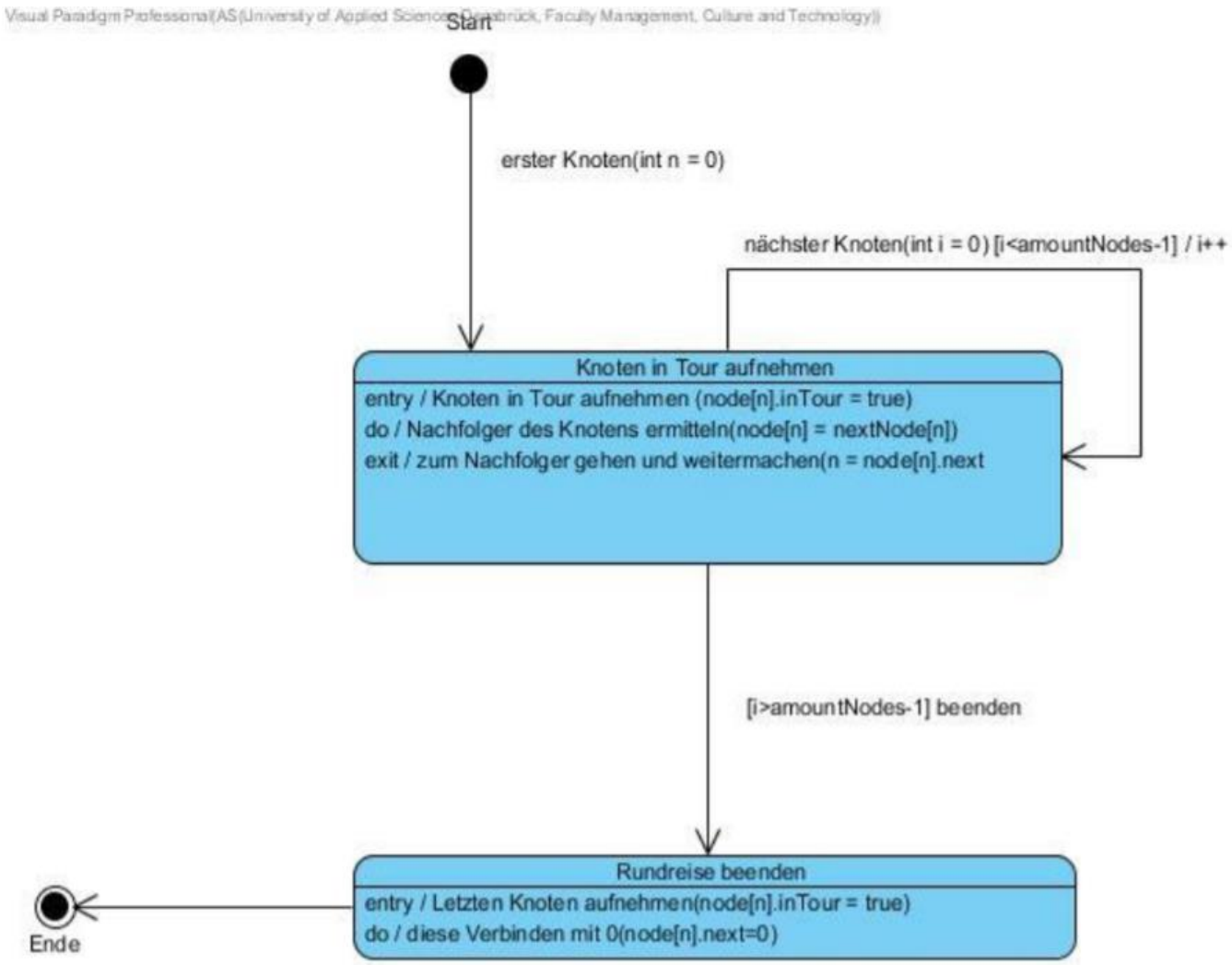
Das Klassendiagramm gilt auch als zentraler Grundbaustein in der UML. Alle notwendigen Klassen und deren Beziehung zueinander werden hier veranschaulicht. Klassen werden als Rechteck dargestellt und enthalten Informationen über Operationen und Attribute. Der Name der Klasse wird im Rechteck an oberster Stelle angegeben. Attribute schildern bestimmte

Eigenschaften für die jeweilige Klasse. Verknüpfungen zwischen den einzelnen Klassen bezeichnet man als Assoziation.



3.5 Zustandsdiagramm

Im Zustandsdiagramm wird das Verhalten eines Systems spezifiziert. Die zur Laufzeit erlaubten Zustände werden aufgezeigt und grafisch dargestellt. Zustände werden durch Rechtecke mit abgerundeten Ecken dargestellt. Die Pfeile zwischen den Zuständen symbolisieren Übergänge von einem Zustand zum anderen. Des Weiteren beginnt das Zustandsdiagramm mit einem Quell- und einem Zielknoten.



7

Abbildung 3 Zustandsdiagramm ,,Travelling Salesman Problem"

4 Bibliotheken

Bibliothek	Erklärung
JavaFX	JavaFX bietet alle APIs für moderne Oberflächen an, z.B. Für GUI Komponenten.
Junit	Junit zum Testen von Java-Programmen.
Random	Erzeugt Zufallszahlen
Jdom	Schnittstelle für den Umgang mit XML-Dokumenten.

5. Testprotokolle

Das Testprotokoll beinhaltet Aufzeichnung über Ereignisse, die beim Anwenden diversen Funktionen auftreten. Je nach Wichtigkeit der einzelnen Anforderungen stehen bestimmte Testfälle im Vordergrund und sollten auf Fehlerfreiheit überprüft werden, sodass am Ende des Projektes die wichtigsten Funktionen laufen. In den folgenden Tabellen sind alle notwendigen Testfälle der jeweiligen Klasse dokumentiert.

Testprotokolle der Klasse ,,Node‘‘:

Testfall	TN0001: Knoten erstellen
Eingabe	Erstellung eines Knotens mit Inhalt
Ablauf	Normale Erstellung eines Objekts
Erwartetes Resultat	Objekt vom Typ ,,Nodes‘‘ wird erstellt.
Tatsächliches Resultat	Objekt vom Typ ,,Nodes‘‘ wurde erfolgreich erstellt.
Fehlerfrei	Ja [x] Nein []
Bemerkung	/

Testfall	TN0002: Zugreifen und ändern der Variable ,,Number‘‘
Eingabe	Auf Nummer zugreifen und eingeben.
Ablauf	Getter-Setter Methoden
Erwartetes Resultat	Auf Nummer wurde zugegriffen und geändert.
Tatsächliches Resultat	Nummer wurde erfolgreich zugegriffen und geändert.
Fehlerfrei	Ja [x] Nein[]
Bemerkung	/

Testfall	TN0003: Zugriff und ändern der Variable ,,Name“
Eingabe	Auf Name zugreifen und eingeben
Ablauf	Getter-Setter Methoden
Erwartetes Resultat	Auf ,,Name“ wurde zugegriffen und geändert.
Tatsächliches Resultat	Auf ,,Name“ wurde erfolgreich zugegriffen und geändert.
Fehlerfrei	Ja [x] Nein[]
Bemerkung	/

Testfall	TN0004: Zugriff auf ,,X und Y“ Variable
Eingabe	Auf X und Y Position zugreifen und ändern.
Ablauf	Getter-Setter Methoden
Erwartetes Resultat	Auf ,,X und Y“ zugreifen und ändern
Tatsächliches Resultat	Auf ,,X und Y“ wurde erfolgreich zugegriffen und geändert.
Fehlerfrei	Ja [x] Nein[]
Bemerkung	/

Testfall	TN0005: Testen der Variable ,,Into“
Eingabe	Die Variable ,,Into“ wird getestet
Ablauf	Getter-Setter Methode
Erwartetes Resultat	Variable ,,Into“ wird getestet.
Tatsächliches Resultat	Variable ,,Into“ wurde erfolgreich getestet
Fehlerfrei	Ja[x] Nein[]
Bemerkung	/

Testfall	TN 0006: Testen der Variable ,,Firstnode“
Eingabe	Ersten Knoten bestimmen.
Ablauf	Getter-Setter-Methode
Erwartetes Resultat	Ein Knoten wird als ,,FirstNode“ bestimmt.
Tatsächliches Resultat	Ein Knoten wurde erfolgreich als ,,FirstNode“ bestimmt
Fehlerfrei	Ja[x] Nein[]
Bemerkung	/

Testprotokolle der Klasse ,,Edges‘‘

Testfall	TE0001: Kante erstellen
Eingabe	Erstellung einer Kante
Ablauf	1. Ein Knoten als Quelle erstellen/bestimmen 2. Ein Knoten als Ziel erstellen/bestimmen
Erwartetes Resultat	Kante wird erstellt
Tatsächliches Resultat	Kante wurde erfolgreich erstellt
Fehlerfrei	Ja [x] Nein []
Bemerkung	/

Testfall	TE0002: Gewichtung der Kante
Eingabe	Ausrechnen, zugreifen und ändern der Gewichtung
Ablauf	1. Knoten als Quelle erstellen/bestimmen 2. Ein Knoten als Ziel erstellen/bestimmen 3. Distanz zwischen Quelle und Ziel berechnen 4. Auf Ergebnis zugreifen 5. Gewichtung Kategorisieren 6. Gewichtung in Kante eintragen
Erwartetes Resultat	Kante erhält Gewichtung
Tatsächliches Resultat	Kante erhält erfolgreich Gewichtung
Fehlerfrei	Ja [x] Nein []
Bemerkung	/

Testfall	TE0003: Duplikat Vermeidung von Kanten
Eingabe	Löschen von doppelten Kanten
Ablauf	1. Eine Kante erstellen 2. Dieselbe Kante spiegelverkehrt erstellen. 3. Mithilfe der Methode ,,equals‘‘ auf Inhalt prüfen. 4. Werte In einem Hashcode speichern
Erwartetes Resultat	Alle doppelten Kanten mit gleichen werten wurden entfernt
Tatsächliches Resultat	Alle doppelten Kanten mit gleichen werten wurden erfolgreich entfernt
Fehlerfrei	Ja [x] Nein []

Bemerkung	/

Testprotokolle der Klasse ,,Travelling Salesman Problem"

Testfall	TT0001: Eine beliebige Anzahl von Knoten und Kanten erstellen
Eingabe	nodesLength (int), edgesLength (int)
Ablauf	Werte in den Parameter eingeben
Erwartetes Resultat	Beliebige Anzahl von Knoten wird erstellt
Tatsächliches Resultat	Beliebige Anzahl von Knoten wurden erfolgreich erstellt.
Fehlerfrei	Ja[x] Nein []
Bemerkung	/

Testfall	TT0002: Zugriff auf Anzahl der Kanten
Eingabe	/
Ablauf	Methodenaufruf getAmountEdges()
Erwartetes Resultat	Zugriff auf Anzahl der Kanten
Tatsächliches Resultat	Erfolgreicher Zugriff auf Anzahl der Kanten
Fehlerfrei	Ja [x] Nein []
Bemerkung	/

Testfall	TT0003: Zugriff auf Anzahl der Knoten
Eingabe	/
Ablauf	Methodenaufruf getAmountNodes()
Erwartetes Resultat	Zugriff auf Anzahl der Knoten
Tatsächliches Resultat	Erfolgreicher Zugriff auf Anzahl der Kanten
Fehlerfrei	Ja [x] Nein []
Bemerkung	/

Testfall	TT0004: Zugriff auf Kante vom Array
Eingabe	returnEdge(int)
Ablauf	Wert in Parameter eingeben
Erwartetes Resultat	Zugriff auf die gesuchte Kante
Tatsächliches Resultat	Erfolgreicher Zugriff auf die gesuchte Kante

Fehlerfrei	Ja [x] Nein []
Bemerkung	/

Testfall	TT0005: Zugriff auf Knoten vom Array
Eingabe	returnNode(int)
Ablauf	Wert in Parameter eingeben
Erwartetes Resultat	Zugriff auf den gesuchten Knoten
Tatsächliches Resultat	Erfolgreicher Zugriff auf den gesuchten Knoten
Fehlerfrei	Ja [x] Nein []
Bemerkung	

Testfall	TT0006: Ermittlung des nächsten Knotens
Eingabe	nextNode(int)
Ablauf	1. Knoten wird gesetzt 2. Nächster Nachbarsknoten wird errechnet, der noch nicht besucht wurde
Erwartetes Resultat	Nächster Knoten wird ermittelt
Tatsächliches Resultat	Nächster Knoten wurde erfolgreich bestimmt.
Fehlerfrei	Ja [x] Nein []
Bemerkung	/

Testfall	TT0007: Existenzprüfung der Kante
Eingabe	controlEdges(int, int)
Ablauf	1. Pfad zum nächsten Knoten aus der Methode next-Node() wird überprüft. 2. Pfad wird bestätigt oder abgelehnt.
Erwartetes Resultat	Existenz der nächsten Kante wird entweder bestätigt oder abgelehnt.
Tatsächliches Resultat	Existenz der nächsten Kante wird erfolgreich bestätigt oder abgelehnt.
Fehlerfrei	Ja [x] Nein []
Bemerkung	/

Testfall	TT0008: Nearest-Neighbour Algorithmus
Eingabe	/

Ablauf	1. Beginnt bei Node null 2. Ruft Methode ,,nextNode“ auf 3. Speichert nächsten Knoten 4. Setzt Knoten auf ,,besucht“ 5. Verbindet letzten Knoten mit dem ersten.
Erwartetes Resultat	Setzt alle Knoten auf ,,besucht“
Tatsächliches Resultat	Alle Knoten wurden erfolgreich besucht
Fehlerfrei	Ja [] Nein[x]
Bemerkung	Genügend Kanten müssen vorhanden sein (Sackgasse)

Testprotokolle der Klasse ,,WriteXML“

Testfall	TW0001: Den gewünschten Graphen abspeichern
Eingabe	nodesLength (int), edgesLength (int)
Ablauf	1. Knotenlänge und Kantenlänge erhalten 2. Dokument wird erstellt 3. Vererbung der Eltern- und Kinderknoten 4. Setzt werte durch ,,TagNames“ ein 5. Bestimmt den Speicherort
Erwartetes Resultat	Erstellte XML-Datei für Graphen
Tatsächliches Resultat	XML-Datei für Graph wurde erfolgreich erstellt
Fehlerfrei	Ja [x] Nein[]
Bemerkung	/

Testprotokolle der Klasse ,,ReadXML“

Testfall	TR0001: Den gewünschten Graphen laden
Eingabe	/
Ablauf	1. ReadXML aufrufen 2. Filename eingeben

	3. Speichert die Kanten und Knoten in einer ArrayList anhand der tagNames
Erwartetes Resultat	Lässt Graphen einlesen
Tatsächliches Resultat	Graph wurde erfolgreich eingelesen
Fehlerfrei	Ja [x] Nein []
Bemerkung	/

Testprotokolle der Klasse ,,GUI“

Testfall	TG0001: Erstelle Zufälligen Graphen
Eingabe	Werte für Node und Edge
Ablauf	1. Aufrufen des Menüpunktes ,,Create Random Graph“ 2. Wert in Node eintragen 3. Wert in Edge eintragen 4. Auf Button ,,Submit“ klicken
Erwartetes Resultat	Graph mit Knoten und Kanten wird visualisiert
Tatsächliches Resultat	Graph mit Knoten und Kanten wurde erfolgreich visualisiert
Fehlerfrei	Ja [] Nein[x]
Bemerkung	Wenn mehr Knoten als Kanten existieren,

Testfall	TG0002: Save into File
Eingabe	/
Ablauf	1. Aufrufen des Menüpunktes ,,Save into File“ 2. Gewünschten Speicherort festlegen
Erwartetes Resultat	Änderungen werden erfolgreich gespeichert.
Tatsächliches Resultat	Änderungen wurden erfolgreich gespeichert.
Fehlerfrei	Ja[x] Nein[]
Bemerkung	Funktioniert, vorausgesetzt Graph wurde erstellt.

Testfall	TG0003: Load From File
Eingabe	/
Ablauf	1. Aufrufen des Menüpunktes ,,Load From File“ 2. Zielordner wählen

Erwartetes Resultat	Graph wird geladen und angezeigt.
Tatsächliches Resultat	Graph wurde erfolgreich geladen und angezeigt
Fehlerfrei	Ja [x] Nein[]
Bemerkung	Funktioniert, vorausgesetzt der geladene Graph existiert und enthält Werte

Testfall	TG0004: Starte den Algorithmus
Eingabe	/
Ablauf	1. Aufruf des Menüpunktes ,,Start the Algorithm‘‘
Erwartetes Resultat	Die Kürzeste Route wird farblich gekennzeichnet
Tatsächliches Resultat	Die Kürzeste Route wurde erfolgreich gekennzeichnet.
Fehlerfrei	Ja[x] Nein[]
Bemerkung	/

Testfall	TG0005: Entferne den Graphen von der Benutzeroberfläche
Eingabe	/
Ablauf	1. Aufruf des Menüpunktes ,,Clear the Graph‘‘
Erwartetes Resultat	Erstellter Graph wird entfernt
Tatsächliches Resultat	Erstellter Graph wurde erfolgreich entfernt
Fehlerfrei	Ja[x] Nein[]
Bemerkung	/

Testfall	TG0006: Entferne den Graphen von der Benutzeroberfläche
Eingabe	/
Ablauf	1. Aufruf des Menüpunktes ,,Clear the Graph‘‘
Erwartetes Resultat	Erstellter Graph wird entfernt
Tatsächliches Resultat	Erstellter Graph wurde erfolgreich entfernt
Fehlerfrei	Ja[x] Nein[]
Bemerkung	/

Testfall	TG0007: Wähle Algorithmus
Eingabe	Nearest-Neighbour oder Brute-Force
Ablauf	1. Aufruf des gewünschten Menüpunktes 2. Klick auf ,,Start the Algorithm‘‘
Erwartetes Resultat	Gewünschter Heuristik wird durch den Graphen visualisiert.
Tatsächliches Resultat	Gewünschte Heuristik wurde durch den Graphen visualisiert.
Fehlerfrei	Ja[] Nein[x]
Bemerkung	Der Graph wird zwar korrekterweise angezeigt und gemalt. Dennoch stimmten die Gesamtstrecken nicht überein. Der BruteForce errechnet die letzte Strecke nicht. D.h. die erste Node mit dem letzten Node. Beim Nearest Neighbour sind kleine Abweichungen wegen dem Runden.

Testfall	TG0008: Name und Weights anzeigen
Eingabe	Checkbox
Ablauf	1. Die gewünschte Checkbox anklicken 2. Bestätigen
Erwartetes Resultat	Namen der Knoten und die Weights werden angezeigt
Tatsächliches Resultat	Beide werden angezeigt
Fehlerfrei	Ja[x] Nein[]
Bemerkung	/

Testfall	TG0009: Algorithmus vergleichen
Eingabe	Nearest-Neighbour und Brute-Force
Ablauf	1. Beide Algorithmen aufrufen 2. Bestätigen
Erwartetes Resultat	Beide Heuristiken werden durch den Graphen visualisiert. Und werden mit verschiedenen Farben dargestellt.
Tatsächliches Resultat	Das erwartete Resultat trifft zu.
Fehlerfrei	Ja[x] Nein[]
Bemerkung	/

Testprotokolle der Klasse „BruteForce“

Testfall	TB0001: Die Permutation
Eingabe	Anzahl der Knoten
Ablauf	1. Anzahl der Knoten bestimmen
Erwartetes Resultat	Permutation (Alle Kombinationen werden errechnet und in ArrayListen gespeichert)
Tatsächliches Resultat	Kombinationen wurden gespeichert
Fehlerfrei	Ja[x] Nein[]
Bemerkung	/

Testfall	TB0002: Kürzeste Route
Eingabe	Alle möglichen Touren
Ablauf	1. Anzahl der Nodes bestimmen 2. Zufällig Kanten und Nodes platzieren 3. Gesamtstrecke errechnen und vergleichen 4. Die kürzeste Tour speichern.
Erwartetes Resultat	Kürzeste Tour wird gespeichert und kann abgerufen werden.
Tatsächliches Resultat	Kürzeste Tour wird gespeichert kann aber nur teilweise abgerufen werden.
Fehlerfrei	Ja[x] Nein[]
Bemerkung	/

Testfall	TB0003: Gesamtkosten der Touren errechnen
Eingabe	Tour eingeben
Ablauf	1. Tour einlesen 2. Gesamtkosten werden errechnet 3. Wird als String für die GUI anzeige gespeichert
Erwartetes Resultat	Die Gesamtkosten der Tour wird angezeigt
Tatsächliches Resultat	Nicht die kompletten Kosten werden angezeigt. Der letzte Knoten

	wird mit dem ersten Knoten(Rund-lauf) nicht errechnet.
Fehlerfrei	Ja[] Nein[x]
Bemerkung	Der letzte Knoten wird mit dem ersten nicht verrechnet(Rundgang) fehlt.

6. Entwicklungsumgebungen

Die Entwicklungsumgebung, auch IDE (Integrated Development Environment) genannt ist eine Sammlung der Werkzeuge, die für die Realisierung und Nutzung der Software benutzt wurde.

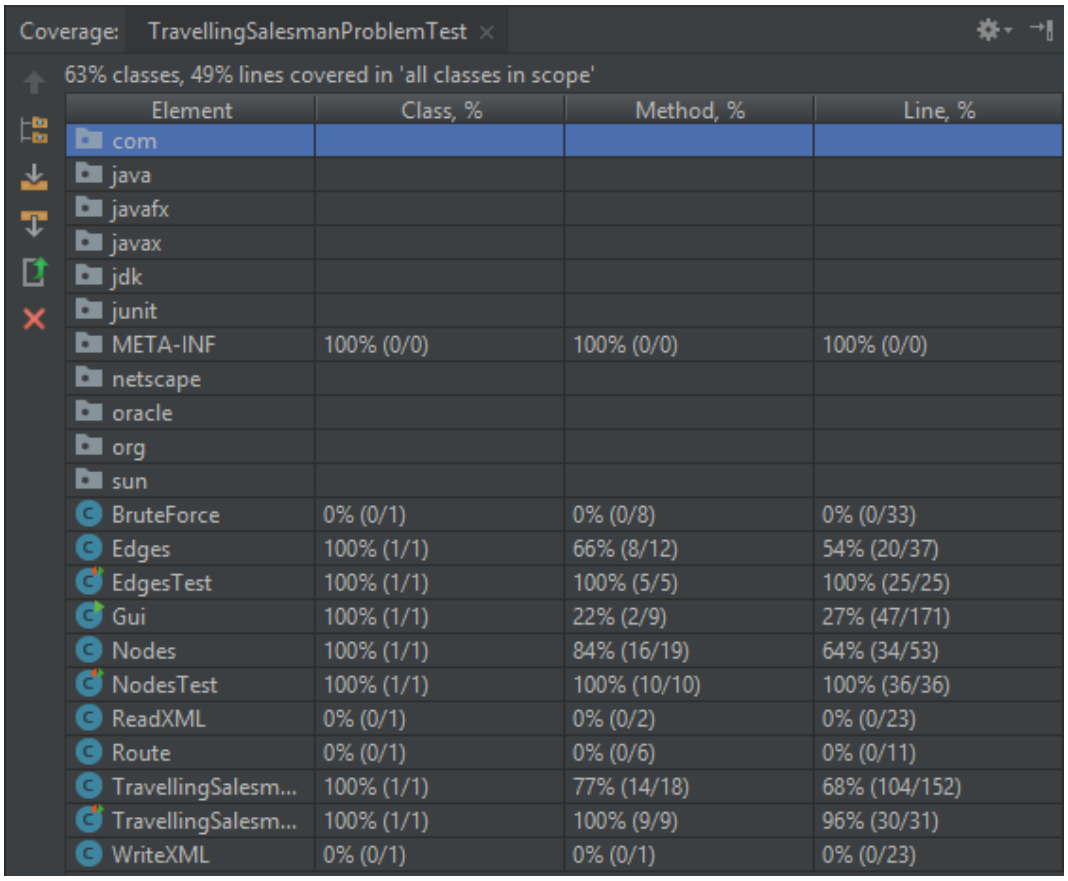
6.1 IntelliJ

IntelliJ IDEA ist eine Entwicklungsumgebung des Softwareunternehmens Jetbrains für die Programmiersprache Java. Besondere Merkmale sind die Unterstützung von z.B. JavaEE, Junit, Ant, ApacheMaven, und GUI Editoren.

6.2 Visual Paradigm

Visual Paradigm ist eine Software, welche die Erstellung von Diagrammen wie z.B. Klassendiagramme nach UML-Standard erleichtert und unterstützt.

7. Test-Coverage



Element	Class, %	Method, %	Line, %
com			
java			
javafx			
javax			
jdk			
junit			
META-INF	100% (0/0)	100% (0/0)	100% (0/0)
netscape			
oracle			
org			
sun			
BruteForce	0% (0/1)	0% (0/8)	0% (0/33)
Edges	100% (1/1)	66% (8/12)	54% (20/37)
EdgesTest	100% (1/1)	100% (5/5)	100% (25/25)
Gui	100% (1/1)	22% (2/9)	27% (47/171)
Nodes	100% (1/1)	84% (16/19)	64% (34/53)
NodesTest	100% (1/1)	100% (10/10)	100% (36/36)
ReadXML	0% (0/1)	0% (0/2)	0% (0/23)
Route	0% (0/1)	0% (0/6)	0% (0/11)
TravellingSalesm...	100% (1/1)	77% (14/18)	68% (104/152)
TravellingSalesm...	100% (1/1)	100% (9/9)	96% (30/31)
WriteXML	0% (0/1)	0% (0/1)	0% (0/23)

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Arbeit selbstständig und nur unter Zuhilfenahme der ausgewiesenen Hilfsmittel angefertigt habe. Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach anderen gedruckten oder im Internet verfügbaren Werken entnommen sind, habe ich durch genaue Quellenangaben kenntlich gemacht.

Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde vorgelegt und auch noch nicht physisch oder elektronisch veröffentlicht.

Lingen (Ems), den 27.09.2018 _____

Andrej Drobin

Lingen (Ems), den 27.09.2018 _____

Julian Geerdes

Lingen (Ems), den 27.09.2018 _____

Deniz Kücüktas