Testdokument

Allgemeines Vorgehen

Im Rahmen der Implementierung eines Softwareproduktes in größerem Ausmaß sind Tests unabdingbar. Besonders bei der Vielzahl an Funktionen des ComorbidityViewer müssen die Methoden ohne Fehler funktionieren, um ein reibungsloses Zusammenspiel der einzelnen Komponenten zu garantieren. Aufgrund des IVRM- Modells ist es schwer Klassen unterschiedlicher Komponenten getrennt zu testen, da jede Schicht auf die vorhergehende Schicht aufbaut. So benötigt z.B die data-Schicht die Schnittstelle zur Datenakquirierung, die Klassen des dbConnector-Pakets. Dennoch wurde versucht jedes Paket mit seinen Klassen gekapselt von anderen Paketen zu testen.

Komponententest

Die Test- Klassen zum jeweiligen Paket sind im Paket mit den selben Namen des zu testenden Pakets aufzufinden.
Alle Klassen des jeweiligen Pakets werden mit dem höchsten Grad der möglichen Kapselung von anderen Paketen auf die Korrektheit ihrer Methoden verifiziert, wobei Integrationstests in kleinen Ansätzen mit inbegriffen sind. Aufgrund der aufbauenden Eigenschaft der Komponenten ist es entscheidend die Reihenfolge der Verifizierung einzuhalten.

Db_Connector-Paket:

DataBaseConnectorTest.java

ICDFileReaderTest.java

DBMetaDataTest.java

DataBaseReaderTest.java

Data-Paket:

GraphManagerTest.java

ClusteringDataTest.java

DiseaseCategoriesTest.java

visual_abstraction-Paket:

CVVisualizationTest.java

db_Connector Paket

DataBaseConnectorTest.java:

Der DataBaseConnector ist verantwortlich für die Verbindung zur Datenbank, die essentiell für komplette Laufzeit des Programms ist, deshalb wird getestet ob nach dem Aufruf der Methode connect() eine Verbindung ersellt wurde. Die Existenz eines Objekts vom Typ Connection wird mittels der Methode existsConnectionToDBMS() überprüft.

Da unser Programm von uns konzipierte Datenbankschemen benötigt muss getestet werden, ob diese im

richtigen Format erstellt werden. Beim Erstellen einer DB, mittels unseres Programms, werden eine ICD-Datentabelle angelegt, die jeden Code einen Namen zuordnet sowie eine Metadaten-Tabelle, die Informationen aller importierten Datensätze und die dazugehörige Anzahl an Patienten beinhaltet.

Da die Applikation auf der Basis einer Datenbank funktioniert, muss die Funktion zum Einlesen und importieren in die DB, getestet werden. Des Weiteren ist darauf zu achten, dass die Text-datei unserem Format entspricht. Diese Konventionen werden beim Einlesen der Datei durch das Fangen der SQL-Exception bei falschen Schema und dem anschließenden Rollback behandelt. Der Nutzer erhält eine Fehlerausgabe.

Ums sich die Daten visualisieren zu lassen, muss eine Methode existieren die die aktuelle Tabelle bereit stellt. Diese Tabelle wird standardmäßig beim Importieren eines Datensatzes gesetzt oder bei der Auswahl des Benutzers. Es muss getestet werden, das die erstellte Tabelle als Default- Relation gesetzt wurde

Methode	Testergebnis
`testConnect()`	OK
`testExistConnectionDBMS()`	OK
`testCreateSchema()`	OK
`testselectDB()`	OK
`testGetCurrentDataBase()`	OK
`testExistConnectionToDB()`	OK
`testGetConnection()`	OK
`testGetMetaData()`	OK
`testFillEdgesTable()`	OK
`testTableExists()`	OK
`testGetCurrentTable()`	OK
`testRemoveEdgeTable()`	OK

ICDFileReaderTest.java:

Die ICDFileReader- Klasse wird beim Erstellen einer Datenbank verwendet, um die ICD-Datentabelle zu erstellen. Dabei wird eine Textdatei in die Datenbank importiert. Der Test überprüft, anhand der bekannten Anzahl der ICDCodes und der Anzahl der Tupel in der Tabelle, die Richtigkeit des Imports der Daten.

Methoden	Testergergebnis
`testReadDefaultFileIntoDB()`	OK

DBMetaDataTest.java:

Die Klasse DBMetaData ist zuständig für die Verwaltung aller Kantentabellen, aus denen der Graph generiert werden kann. Es wird getestet, ob beim Erstellen die richtige Relation gesetzt wurde. Des Weiteren wird überprüft, ob alle Tabellen, die Kantendaten erfassen, zurückgegeben werden. Wenn eine Tabelle in der

DB gelöscht wird, muss der Eintrag für die jeweilige Tabelle ebenfalls in der Metadaten- Tabelle gelöscht werden, um eine Inkonsistenz der angezeigten Tabellen auf der GUI und der tatsächlich in der DB befindenden Tabellen zu eliminieren.

Methoden	Testergebnis
`testSetDefaultRelation()`	OK
`testGetDefaultRelation()`	OK
`testGetAllTables()`	OK
`testDeleteEdgeTable()`	OK

DataBaseReaderTest.java:

Diese Testklasse testet die Funktionen explore() und search(), die auf die Datenbank zugreifen mittels eines DataBaseSource- Objektes von Prefuse. Anhand einer Beispiel-datei wird die Korrektheit der zurückgegebenen Tabellen, der oben genannten Methoden auf Gleichheit überprüft. Zugleich wird beim Erstellen einer Instanz des DataBaseReaders die Korrektheit der Update-Methode überprüft, der somit eine Verbindung zur Datenbank erstellt.

Methoden	Testergebnis
`testExplore()`	OK
`testSearch()`	OK

Data Paket

Für jede zu testende Klasse wird die WhiteMaleNet3- Datei verwendet und Phi als signifikantes Kantengewicht mit einem Mindestwert von 0,1.

GraphManagerTest.java:

Da alle Spalten entscheidend für die Korrektheit des Programms sind, da sie weiterverwendende Daten enthaltenen wird die Anzahl der Spalten überprüft.

Es wird die Funktion explore() getestet, die einerseits für die Initialisierung verantwortlich ist, sowie für das Explorieren eines existierenden Graphen. Zum Test wird die Anzahl der Tupel in der Kantentabelle des Graphen der Anzahl der Ergebnisse einer SQL- Query verglichen. Des Weiteren wird der Inhalt der Kantentabelle mit den Ergebnis der Query inhaltlich verglichen.

Um einen neuen für eine neue Krankheit zu initialisieren muss ein existierende Graph gelöscht werden, dass heißt die Inhalte der Kanten- und Knotentabelle muss nach dem Löschen leer sein.

Das Wechseln des Kantengewichtstyp des Graphen kann nur geändert werden, wenn der Graph leer ist, da es sonst zur Inkonsistenz des Graphen kommt. Deshalb ist bei dem Aufruf zum Ändern des Gewichtstyps eine Abfrage ob der Graph leer ist. Dieses Verhalten wird getestet.

Um häufige Datenbankabfragen zu vermeiden werden schon explorierte Krankheiten mittels einer boolean Variable gekennzeichnet. Krankheiten werden somit nur exploriert, wenn die boolean Variable auf false gesetzt ist.

Methoden	Testergebnis

`testColumnCount()`	OK
`testExploreGraph()`	OK
`testNodeData()`	OK
`testClearGraph()`	OK
`testswitchWeightType()`	OK
`testExpandedValue()`	OK

ClusteringDataTest.java:

Bei eingeschaltetem Clustering und bei Exploration des Graph wird durch die `update()` -Mehtode des Observerinterface automatisch, eine Cluster Tabelle erstellt. Es wird beim Test überprüft, ob dieses Zusammenwirken der GraphManager- Klasse als Observable und der ClusteringData Klasse als Observer funktioniert. Verifiziert wird die Cluster- Tabelle durch die entstehende Cluster- textdatei. Es wird die Anzahl der Cluster mit der Anzahl der Zeilen in der Tabelle verglichen. Semantisch werden die Knoten, die in der Textdatei stehen mit den Knoten, die in der Tabelle in einer LinkedList von Strings für jedes Cluster gespeichert sind, verifiziert. Die Verifikation mit der Textdatei ist möglich, aufgrund der Korrektheit des BorderFlow- Algorithmus

Wenn der Graph geleert wird muss auch ebenfalls die Cluster Tabelle geleert werden.

Methoden	Testergebnis
`testClusteringUpdate()`	OK
`testClusterData()`	OK
`testAutomaticallyCluster()`	OK
`testAutomaticallyRemove()`	OK

Visual_abstraction

Dieses Paket enthält diverse Action-Klasse und layout-Klassen. Da alle Klassen auf die Visualisierung des Graphen abzielen und in der CVVisualization enthalten sind bzw. initialisiert werden, wird nur die CVVisulazation Klasse getestet. Folglich werden aufgrund der Aggregation, der Layout- und Action-Klassen mit verifiziert.

Aufgrund der engen Zusammengehörigkeit der CVVisualization-Klasse und des Graphen der GraphManager – Klasse wird das Zusammenspiel dieser Klassen überprüft. Die ständige Aktualisierung der Klasse wird durch das Observer-Pattern erreicht.

Bei leeren Graphen dürfen die Actions nicht laufen, da NullPointerException entstehen , da einige Actions auf Dataen zugreifen wollen, z.B die NodeSizeAction, die verantwortlich ist für die unterschiedliche Größe der Knoten. Diese Action würde kein Minimum bzw Maximum finden, wenn keine Werte existieren an denen sie sich orientieren kann.

Methoden	Testergebnis
`testRunActions()`	OK
`testSwitchLayout()`	OK
`testDecoratorShow()`	OK

`testGetVisualItem()`	OK
`testAnimation()`	OK
`testStopSpringLayout()`	OK

Integrationstest

Die Integrationstests werden durchgeführt , wenn alle Komponenten auf ihre Korrektheit überprüft wurden. Bei den Integrationstest wird das Zusammenspiel der Systemkomponenten verifiziert.
Die Interaktionen zwischen GraphManager und DataBaseReader wurden im Komponententest des GraphManagers durchgeführt.
Das Zusammenspiel zwischen der CVVisulalization und dem GraphManager wird schon bei denKomponententest der CVVisualization- Klasse verifiziert, da diese Klasse eine starke Abhängigkeit zum Graphen der GraphManager- Klasse aufweist. Eine Erstellung eines VisualGraph mit den geforderten Daten würde einen größeren Aufwand erfordern, deshalb wurde die CVVisualization- Testklasse mit dem GraphManager aufgerufen.
Des Weiteren wurde die ClusteringData Klasse mit dem GraphManager getestet, da diese Klasse die relevanten Daten für das Clustering beinhaltet.
Unabhängig voneinander agieren das Clustering und die Visualisierung der Cluster, deshalb wurde eine Testklasse ClusteringVisualizationTest.java angelegt, die verschiedene Testszenarien überprüft.

Systemtest

Der Systemtest ist das Validieren der Funktionen, die vom Auftraggeber gefordert waren, mit denen die das fertige Programm beinhaltet.

Funktionen	enthalten
Datenimport von IcdCode-Daten und HuDiNe-datensätze	ja
Auswahl vorhandener Datensätze	ja
Anzeige eines Teilggraphen mit Informationen	ja
Auswahl verschiedener Ansichten des Graphen	ja
Anzeige des Graphen in Tabellenform	ja
Fokussierung eines Knotens in der Knotenliste und im Graphen(bidirektionales Verhalten)	ja
Zoomen	ja
Suche nach Krankheitsnamen und Anzeige der Suchergebnisse	ja
Expandieren von Knoten	ja
Cluster berechnen und anzeigen	ja
Einstellung von Kantengewichtstyp und minimalen Kantengewicht	ja
Bildexport des Teilgraphen	ja