Studentgegevens

Cursuscode: IM0202
Opdracht 2 visualisatie

Marco Huijben (838316640) en Ivo Willemsen (851926289)

Werkverdeling

Het opstellen van de criteria waaraan de software zou moeten voldoen, is door ons beiden uitgevoerd. De cache is verdeeld in twee lagen en beide hebben gewerkt aan de implementatie hiervan. Marco heeft implementatie van de hiërarchie voor zijn rekening genomen en Ivo heeft gewerkt aan de codering van het scatterdiagram. Ivo heeft vervolgens de integratie van beide componenten uitgevoerd terwijl Marco in de tussentijd gewerkt heeft aan het losstaande treemap component. Het verslag is door ons beiden gemaakt.

Visualisatie criteria

Het systeem zou aan de volgende criteria moeten voldoen:

- De software moet inzicht verschaffen in de decompositie van de applicatie in componenten (architectuur view). De gebruiker zou m.b.v. muisnavigatie een decompositie moeten kunnen uitvoeren door op het hoogste niveau (de applicatie) op de verschillende packages te klikken en vervolgens dieper in de hiërarchie te navigeren (gebruikscriteria).
- Het systeem moet de gebruiker in staat stellen om de complexiteit, duplicatie en grootte van de aanwezige componenten op een visuele en tekstuele manier af te leiden. Het type component moet worden kunnen afgeleid door verschillende kleuren te gebruiken, de complexiteit en/of grootte van de componenten zal worden afgeleid door het gebruik van verschillende horizontale en verticale groottes van de GUI objecten (esthetische criteria).
- De gebruiker moet na het opstarten van het systeem, in één oogopslag de zogenaamde ‘outliers‘ kunnen identificeren (qua complexiteit en grootte). Het is niet de bedoeling dat de gebruiker bij de bepaling van deze outliers, door de structuur van de applicatie moet navigeren (gebruikscriteria).

Beoogde componenten

Aan bovenstaande gebruiks- en esthetische criteria zal worden voldaan middels de implementatie van de volgende beoogde systeemcomponenten:

- Een ‘node-link diagram‘[2]. De hiërarchie van een willekeurige applicatie kan worden weergegeven door een ‘node-link diagram‘ dat de gebruiker in staat stelt om op een interactieve manier (m.b.v. de muis) een decompositie uit te voeren van de verschillende lagen binnen het systeem. Het hoogste niveau betreft de applicatie zelf, het laagste niveau is een ‘unit‘ (bijvoorbeeld een methode in Java). Tussenliggende lagen zijn de verschillende packages. In Java kan een bestand meerdere klassen bevatten. In dit geval wordt het bestand dan ook als een aparte laag beschouwd. Aan de esthetisch criteria zal worden voldaan door de grootte van de verschillende GUI-componenten recht evenredig te maken met de grootte van de weerspiegelde packages, bestanden of methoden. Verder zal de kleur van het GUI-object aangeven om welk type laag het gaat.
- Een interactief ‘scatter plot matrix diagram‘[2]. Dit diagram zal de complexiteit van methoden uitzetten tegen de grootte van de methoden. Indien het systeem opstart, zal het systeem alle methoden in overweging nemen bij de constructie van het scatterdiagram. Naarmate de gebruiker dieper in de hiërarchie duikt, zal ook het scatterdiagram worden ververst, zodat alleen die methodes worden gebruikt die ook tot het betreffende package of bestand behoort. Ook stelt het systeem de gebruiker in staat stellen om ‘in te zoomen‘ op een gedeelte van het diagram: Indien de gebruiker op een bepaald kwadrant van het diagram klikt, zullen de gegevens van het betreffende kwadrant worden uitvergroot middels een ander scatter diagram dat onder het hoofddiagram zal worden geplaatst. Indien de gebruiker over een data punt met de muis beweegt, zal het systeem informatie in een tekstvak verstrekken omtrent de betreffende methode, de plek in de hiërarchie, de complexiteit en de grootte van de methode Het component zal generiek worden opgezet, met dien verstande dat het een ‘library component‘ zal worden dat ook in andere use-cases kan worden ingezet waar men behoeft heeft aan het uitzetten van drie variabelen middels een scatter plot matrix diagram. De variatie van de derde variabele zal worden

aangegeven door het toekennen van verschillende kleuren van de data punten in het diagram. In deze use-case, zullen er twee variabelen tegen elkaar worden uitgezet. Bij de bepaling van de duplicatie is er voor gekozen om op bestandsniveau een vergelijking te doen. De duplicatie op het niveau van methoden kan dus niet worden getoond aangezien de benodigde gegevens niet aanwezig zijn.

– Een interactief 'tree map diagram'[2]. Dit component zal de gebruiker in staat stellen om zowel de decompositie van een applicatie uit te voeren, als de complexiteit en grootte van packages en methoden te bekijken. Daarnaast kan inzicht gekregen worden van de duplicatie per bestand. De view zal kunnen worden aangepast door in een combobox het gewenst aggregatie niveau te kiezen (applicatie, package, bestand, klasse of methode).

De beoogde componenten implementeren Shneiderman's mantra[3] op de volgende wijze: Het 'node-link diagram' geeft een **overview** van de structuur van het systeem. Er kan vervolgens worden **ingezoomd** op verschillende packages. Ook kan in het interactieve 'scatter diagram' worden **ingezoomd** op een subset van de data. Men kan detaillistische informatie verkrijgen ('Details-on-demand') door met de muis over een object te navigeren.

Er is aandacht besteed aan Tufte's 'Graphic Design Principles'[4] door op één scherm ('smallest space') in één oogopslag ('shortest time') zicht te krijgen op de complexiteit en unit size 'outliers' binnen een applicatie. Op één scherm wordt zowel de structuur van de applicatie ('node-link diagram') als een meer detaillistische kijk op de methoden binnen de applicatie middels een 'scatter diagram' ('the greatest number of ideas').

Architectuur view keuze

De architectuur van een applicatie wordt weergegeven in een 'Coarse-Grained Polymetric View' en een 'Fine-Grained Polymetric View'[1]. De 'Coarse-Grained Polymetric View' via een hiërarchie weergave wordt bewerkstelligd d.m.v. een 'node-link diagram' (zie vorige paragraaf). De reden dat er gekozen is voor een 'node-link diagram' is het feit dat een boom-achtige structuur zich uitermate goed leent voor het beschrijven van de structuur van bijvoorbeeld een systeem, organisatie of een stamboom. De 'Coarse-Grained Polymetric View' geeft direct toegang tot een 'Fine-Grained Polymetric View' (scatter diagram), waar op een detaillistisch niveau direct te zien zal zijn welke methoden zogenaamde outliers zijn m.b.t. complexiteit en grootte.

Resultaat analyse

Het ontwikkelde visualisatiesysteem voldoet in alle opzichten aan de gestelde criteria. Qua gebruikscriteria stelt het 'node-link diagram' de gebruiker in staat om inzicht te krijgen in de structuur van de applicatie. Door op de applicatie (of ander aggregatieniveau) te klikken wordt in het GUI-object aangegeven wat de geaggregeerde complexiteit en grootte is, waarbij alle onderliggende packages, bestanden en methoden in overweging worden genomen. Een ander gebruikscriterium dat wij gesteld hadden is het in één oogopslag grafisch weergeven welke methoden de boosdoeners zijn van de applicatie m.b.t. complexiteit en/of grootte. Ook het 'tree map diagram' geeft de gebruiker de mogelijkheid om op verschillende niveau inzicht te krijgen in de structuur van een willekeurige applicatie. Ook op het gebied van esthetische criteria kunnen we zeggen dat het systeem hieraan voldoet. In het 'node-link diagram' en in het 'tree map diagram' worden verschillende soorten lagen m.b.v. verschillende kleuren aangegeven. Verder worden in deze diagrammen de grootte van de methoden, bestanden en packages aangegeven door de grootte van de betreffende GUI-objecten.

Onderstaande tabel geeft de meest problematische methoden in welke applicatie zouden moeten worden 'gerefactored' om de complexiteit en/of grootte te reduceren:

Outliers cyclomatische complexiteit en grootte			
Applicatie	Methode	McCabe cyclomatische complexiteit	Unit size
hsqldb	org.hsqldb.FunctionCustom.FunctionCustom.resolveTypes()	275	768
hsqldb	org.hsqldb.FunctionCustom.FunctionCustom.getValue()	241	871
hsqldb	org.hsqldb.cmdline.SqlFile.SqlFile.importDsv()	179	560
hsqldb	org.hsqldb.StatementSchema.StatementSchema.getResult()	181	847
smallsql	smallsql.database.ExpressionArithmetic.ExpressionArithmetic.getBoolean()	134	187
smallsql	smallsql.database.SQLTokenizer.SQLTokenizer.parseSQL()	90	183
smallsql	smallsql.database.SQLParser.SQLParser.function()	85	360

Voorgaande gegevens zijn verkregen door in het 'scatter diagram' op visuele wijze de methodes te selecteren die zich in kwadranten bevinden het dichts bij de rechter bovenhoek. We verwijzen graag naar de Appendix A1 en A2, waar een uitgebreidere lijst is vermeld van methodes die gerefactored zouden moeten worden.

Referenties

[1] M. L. en S. Ducasse. Codecrawler: An extensible and language independent 2d and 3d software visualization tool. *In Tools for Software Maintenance and Reengineering*, pages 74–94, June 2005.

[2] V. O. J. Heer, M. Bostock. A tour through the visualization zoo. *Communications of the ACM*, 53(6):795–825, June 2010.

[3] B. Shneidermann. The eyes have it: a task by data type taxonomy for information visualizations. *Proceedings 1996 IEEE Symposium on Visual Languages*, pages 336–343, 1996.

[4] E. Tufte. *The Visual Display of Quantitative Information.* Graphics Press, 2001.

Appendix A1: Hsqldb methodes met hoge complexiteit en/of grootte

Outliers cyclomatische complexiteit en grootte			
Applicatie	Methode	McCabe cyclomatische complexiteit	Unit size
hsqldb	org.hsqldb.test.TestSchemaParse.TestSchemaParse.multiPartKeywords()	8	360
hsqldb	org.hsqldb.dbinfo.DatabaseInformationFull.DatabaseInformationFull.ROUTINES()	26	355
hsqldb	org.hsqldb.test.TestSchemaParse.TestSchemaParse.multiPartKeywords()	8	360
hsqldb	org.hsqldb.dbinfo.DatabaseInformationFull.DatabaseInformationFull.ROUTINES()	26	355
hsqldb	org.hsqldb.util.TransferDb.TransferDb.getTableStructure()	68	371
hsqldb	org.hsqldb.cmdline.SqlFile.SqlFile.processPL()	91	236
hsqldb	org.hsqldb.FunctionCustom.FunctionCustom.getSQL()	88	176
hsqldb	org.hsqldb.cmdline.SqlFile.SqlFile.processPL()	91	236
hsqldb	org.hsqldb.FunctionCustom.FunctionCustom.getSQL()	88	176
hsqldb	org.hsqldb.cmdline.SqlFile.SqlFile.processPL()	91	236
hsqldb	org.hsqldb.FunctionCustom.FunctionCustom.getSQL()	88	176
hsqldb	org.hsqldb.StatementSession.StatementSession.getResult()	98	353
hsqldb	org.hsqldb.cmdline.sqltool.SqlFileScanner.SqlFileScanner.yylex()	130	339
hsqldb	org.hsqldb.cmdline.SqlFile.SqlFile.processBlock()	124	380
hsqldb	org.hsqldb.cmdline.SqlTool.SqlTool.objectMain()	127	413
hsqldb	org.hsqldb.ParserDQL.ParserDQL.readTypeDefinition()	113	361
hsqldb	org.hsqldb.cmdline.SqlFile.SqlFile.displaySqlResults()	138	361
hsqldb	org.hsqldb.cmdline.SqlFile.SqlFile.processBlock()	124	380
hsqldb	org.hsqldb.cmdline.SqlTool.SqlTool.objectMain()	127	413
hsqldb	org.hsqldb.ParserDQL.ParserDQL.readTypeDefinition()	113	361
hsqldb	org.hsqldb.cmdline.SqlFile.SqlFile.processBlock()	124	380
hsqldb	org.hsqldb.cmdline.SqlTool.SqlTool.objectMain()	127	413
hsqldb	org.hsqldb.ParserDQL.ParserDQL.readTypeDefinition()	113	361
hsqldb	org.hsqldb.FunctionSQL.FunctionSQL.getValue()	128	495
hsqldb	org.hsqldb.StatementCommand.StatementCommand.getResult()	137	691
hsqldb	org.hsqldb.server.ServerConnection.ServerConnection.receiveOdbcPacket()	135	762
hsqldb	org.hsqldb.cmdline.SqlFile.SqlFile.processSpecial()	159	494
hsqldb	org.hsqldb.FunctionSQL.FunctionSQL.resolveTypes()	152	474
hsqldb	org.hsqldb.cmdline.SqlFile.SqlFile.processSpecial()	159	494
hsqldb	org.hsqldb.FunctionSQL.FunctionSQL.resolveTypes()	152	474
hsqldb	org.hsqldb.cmdline.SqlFile.SqlFile.importDsv()	179	560
hsqldb	org.hsqldb.StatementSchema.StatementSchema.getResult()	181	847
hsqldb	org.hsqldb.FunctionCustom.FunctionCustom.getValue()	241	871
hsqldb	org.hsqldb.FunctionCustom.FunctionCustom.resolveTypes()	275	768

Appendix A2: Smallsql methodes met hoge complexiteit en/of grootte

Outliers cyclomatische complexiteit en grootte			
Applicatie	Methode	McCabe cyclomatische complexiteit	Unit size
smallsql	smallsql.database.ExpressionArithmetic.ExpressionArithmetic.getBoolean()	134	187
smallsql	smallsql.database.SQLTokenizer.SQLTokenizer.parseSQL()	90	183
smallsql	smallsql.database.SQLParser.SQLParser.function()	85	360
smallsql	smallsql.database.StoreImpl.StoreImpl.writeExpression()	72	151
smallsql	smallsql.database.ExpressionValue.ExpressionValue.accumulate()	69	153
smallsql	smallsql.database.StoreImpl.StoreImpl.writeExpression()	72	151
smallsql	smallsql.database.ExpressionValue.ExpressionValue.accumulate()	69	153
smallsql	smallsql.junit.TestDataTypes.TestDataTypes.runTest()	64	218
smallsql	smallsql.database.DateTime.DateTime.toString()	43	289
smallsql	smallsql.database.SQLParser.SQLParser.datatype()	42	107
smallsql	smallsql.database.SQLParser.SQLParser.expressionSingle()	41	114
smallsql	smallsql.database.Table.Table.requestLockImpl()	35	99
smallsql	smallsql.database.SQLParser.SQLParser.expression()	31	94
smallsql	smallsql.database.DateTime.DateTime.parse()	29	108
smallsql	smallsql.database.JoinScroll.JoinScroll.next()	28	93
smallsql	smallsql.database.ExpressionFunctionConvert.ExpressionFunctionConvert.getObject()	38	87
smallsql	smallsql.junit.TestOperatoren.TestOperatoren.runTest()	1	87
smallsql	smallsql.database.SQLTokenizer.SQLTokenizer.getSQLDataType()	33	61
smallsql	smallsql.database.SSResultSetMetaData.SSResultSetMetaData.getDataTypePrecision()	32	54
smallsql	smallsql.database.Index.Index.findRows()	33	65
smallsql	smallsql.database.Index.Index.addValues()	32	66
smallsql	smallsql.database.ExpressionArithmetic.ExpressionArithmetic.getObject()	33	52
smallsql	smallsql.database.ExpressionFunctionReturnP1.ExpressionFunctionReturnP1.getObject()	30	45
smallsql	smallsql.database.StoreImpl.StoreImpl.getLong()	33	60
smallsql	smallsql.database.StoreImpl.StoreImpl.getInt()	33	61
smallsql	smallsql.database.StoreImpl.StoreImpl.getNumeric()	33	56
smallsql	smallsql.database.StoreImpl.StoreImpl.scanObjectOffsets()	36	61
smallsql	smallsql.database.StoreImpl.StoreImpl.getBytes()	33	59
smallsql	smallsql.database.StoreImpl.StoreImpl.getString()	34	62
smallsql	smallsql.database.StoreImpl.StoreImpl.getDouble()	33	60
smallsql	smallsql.database.StoreImpl.StoreImpl.getBoolean()	33	62
smallsql	smallsql.database.StoreImpl.StoreImpl.getMoney()	33	56
smallsql	smallsql.database.StoreImpl.StoreImpl.getFloat()	33	60
smallsql	smallsql.database.StoreImpl.StoreImpl.getObject()	34	61
smallsql	smallsql.database.Table.Table.requestLockImpl()	35	99

Appendix A3: Overzicht screenshots

De volgende screenshots zijn bijgeleverd:

- complete.png: overview van de applicatie, waarbij de applicatie als een geaggregeerd geheel wordt weergegeven. Het bijbehorende scatter diagram wordt tevens getoond
- package_view: weergave van de decompositie van de structuur van een applicatie. De groene kleur duidt een applicatie aan. Blauw wordt met een package geassocieerd, rood met een bestand en grijs met een methode. Het scatter diagram wordt getoond indien het package wordt geklikt dat overeenkomt met de kleinste blauwe rechthoek
- TreeMap_applicatie.png: overzicht van alle packages, files en methoden, de grootte geeft de LOC weer. De kleur geeft de type (package, file, methode) aan.
- TreeMap_package.png: overzicht van alle packages waarin de LOC wordt weergegeven door de grootte en het aandeel cc van de package t.o.v. van het totaal cc in de vorm van een kleur.
- TreeMap_file.png: overzicht van alle Java files, de grootte geeft de LOC weer en de kleur de duplication.
- TreeMap_class.png: overzicht van alle klassen in de applicatie, de grootte geeft de LOC weer en de kleur de complexity.
- TreeMap_Method.png: overzicht van alle methoden in de applicatie, de grootte geeft de LOC weer en de kleur de complexity.