Comparison between C# and Java

in implementation of a university desktop app
Abstract
This thesis is about comparison between C# and Java in implementation of a university desktop application. In .Net world the prominent IDE is MS Visual studio and DB is MS SQL server. In Java domain we have several IDEs and techniques to approach the problem. The knowledge to choose between these techniques to use is as much as important if a programmer can gain it, he can apply it in many other software engineering domains like a personal management system, library system, etc.

Therefore the importance of a descriptive and applied experience is showing itself up. This thesis has started with a discussion to choose NetBeans IDE and application definition, and use case modeling then it goes through comparison and implementation part. The comparison will go in depth for each use case in this project in its implementation in C# and Java and see which techniques are used. This includes comparison between these two apps in Software quality factors.

Keywords: C#, Java, .Net, Desktop application, MS SQL server, DB, Software quality, NetBeans, MS Visual studio.
Acknowledgments

Special thanks to my supervisor Tobias Anderson Gidlund for helping me through this degree project and giving me this applied topic and my family who are really my good friends and always supporting and inspiring me entire my life and anybody who courage me to finish my master degree.

Thank you Mathias Hedenborg, thesis examiner for his hints specifically about Software quality chapter and anybody who helps me with this thesis proof reading.
Contents
1. Introduction .................................................................................................................................................. 1
  1.1 What is the problem and goal that this thesis covers ................................................................. 1
  1.2 Intended audience ......................................................................................................................... 1
  1.3 What you need to use this thesis ............................................................................................... 2
  1.4 How this thesis is structured ..................................................................................................... 3
2. A discussion and motivation on tools and techniques in this application ................................. 4
  2.1 What is the application .................................................................................................................. 4
  2.1.1 UDA System Requirements Specification ........................................................................ 4
  2.2 A background on C# and Java in this app ................................................................................ 5
  2.3 Java development environments ............................................................................................... 5
  2.3.1 Oracle JDeveloper ............................................................................................................. 6
  2.3.2 Eclipse ............................................................................................................................... 7
  2.4 Summary ........................................................................................................................................ 8
3. Requirement engineering and Design ................................................................................................. 9
  3.1 UDA Project Glossary .................................................................................................................. 9
  3.2 UDA Use case diagram ............................................................................................................... 9
  3.3 Actor semantics ........................................................................................................................ 10
  3.4 Use case semantics .................................................................................................................... 10
  3.5 C# Class diagram ..................................................................................................................... 11
  3.6 Java Class diagram .................................................................................................................... 12
  3.7 Database diagram ....................................................................................................................... 13
  3.8 Summary ..................................................................................................................................... 13
4. Java vs. C# in an applied implementation view for developers .................................................. 14
  4.1 C# implementation roadmap ...................................................................................................... 14
  4.2 Java implementation roadmap .................................................................................................. 15
  4.3 Use case: CRUDBranch ............................................................................................................. 16
  4.4 Use case: ProfessorsRegistration ........................................................................................... 16
  4.5 Use case: StudentRegistration .................................................................................................. 17
  4.6 Use case: CRUDCourse ............................................................................................................ 17
  4.7 Use case: CourseRegistration .................................................................................................. 17
  4.8 Use case: GradeRegistration ................................................................................................... 17
  4.9 Use case: GradeDeclaration ..................................................................................................... 18
  4.10 Use case: UnitSelection ...................................................................................................... 18
  4.11 Use case: LogonStudent ......................................................................................................... 18
  4.12 Use case: LogonAdmin ........................................................................................................... 18
  4.13 Summary ................................................................................................................................... 19
5. Java vs. C# in Software quality factors ............................................................................................ 20
5.1 A background on Software quality ................................................................. 20
5.2 Visualizing metrics ...................................................................................... 21
  5.2.1 Classes which are large and complex .................................................... 21
  5.2.2 Classes which are complex and tightly coupled with other parts of UDA ................................................. 23
  5.2.3 Classes which have low cohesion and high coupling .......................... 24
  5.2.4 Classes which will often change since they depend on a lot of other classes .................................................................................. 26
5.3 Reusability and Maintainability of UDA ...................................................... 28
5.4 Summary ...................................................................................................... 30
6. Conclusion ..................................................................................................... 31
7. Future work ................................................................................................... 33
References ............................................................................................................. 34
Appendices ............................................................................................................ 36
  Appendix A VizzMaintenance raw data .......................................................... 36
  Appendix B Understand raw data .................................................................. 37
  Appendix C Use case semantics ..................................................................... 38
List of Abbreviations

ADF: Application Development Framework(for Oracle)
CBO: Coupling Between Objects(in SQ)
CMM: Capabilities Maturity Model(in SQ)
CRUD: Creation, Read, Update, Delete(in DB)
DAC: Data Abstract Coupling(in SQ)
DB: Database
DIN: German Institute of Standards
DIT: Depth In Inheritance Tree(in SQ)
DOS: Denial of service(in Hacking)
DS: Data Set
EF: Entity Framework(in .Net)
GB: Giga Byte
GUI: Graphical User Interface
ID: Identification
IDE: Interactive Development Environment
ISO: International Standards Organization
JPA: Java Persistence API
JSR: Java Specification Request
LCOM: Lack of Cohesion of Methods(in SQ)
LND: Loop Nesting Depth(in SQ)
LOC: Lines Of Code(in SQ)
MCC: McCabe Cyclomatic Complexity(in SQ)
MDI: Multiple Document Interface
MS: Microsoft
NOC: Number of Children(in SQ)
RFC: Response set For a Class(in SQ)
SQ: Software quality
SWT: Standard Widget Toolkit
TCC: Tight Class Cohesion(in SQ)
UDA: University Desktop App
UML: Unified Modeling Language
VS: MS Visual Studio IDE
WMC: Weighted Method Count(in SQ)
WPF: Windows Presentation Foundation(in .Net)
List of figures

Figure 3.1: UDA Use case diagram ......................................................................................... 10
Figure 3.2: C# Class diagram .............................................................................................. 12
Figure 3.3: Java Class diagram ............................................................................................ 12
Figure 3.4: UDA Database diagram ...................................................................................... 13
Figure 4.1: C# implementation roadmap ................................................................................ 14
Figure 4.2: Java implementation roadmap ............................................................................. 15
Figure 5.1: Size vs. Complexity in Java .................................................................................. 21
Figure 5.2: Size vs. Complexity in C# .................................................................................... 22
Figure 5.3: Getting a basic understanding of UDA in Java ..................................................... 23
Figure 5.4: Getting a basic understanding of UDA in C# ....................................................... 24
Figure 5.5: Cohesion (TCC) VS. Coupling (DAC) in Java ...................................................... 25
Figure 5.6: Cohesion(LCOM) VS. Coupling(RFC) in Java ...................................................... 25
Figure 5.7: Cohesion VS. Coupling in C# .............................................................................. 26
Figure 5.8: Change sensitive classes in Java ....................................................................... 27
Figure 5.9: Change sensitive classes in C# ....................................................................... 27
List of tables

Table 2.1: UDA System Requirements Specification ................................................................. 5
Table 2.2: Supported features for UDA in Oracle JDeveloper .................................................. 7
Table 2.3: Supported features for UDA in Eclipse ................................................................. 7
Table 2.4: Supported features in IDEs .................................................................................... 8
Table 3.1: UDA Project glossary ............................................................................................ 9
Table 3.2: Actor semantics .................................................................................................... 10
Table 5.1: Figure 5.1 outlier classes legend ........................................................................... 21
Table 5.2: Figure 5.2 outlier classes legend ........................................................................... 22
Table 5.3: Software quality matrix [21][19] ........................................................................... 28
Table 5.4: Table 5.3 legend [19] ........................................................................................... 28
Table 5.5: UDA factor measurement in Java ........................................................................ 29
Table 5.6: UDA factor measurement in C# ........................................................................... 30
Table 0.1: Use case: ProfessorsRegistration ....................................................................... 38
Table 0.2: Use case: StudentRegistration ............................................................................ 38
Table 0.3: Use case: CRUDCourse ....................................................................................... 39
Table 0.4: Use case: CourseRegistration ............................................................................. 39
Table 0.5: Use case: GradeRegistration ............................................................................... 39
Table 0.6: Use case: GradeDeclaration ................................................................................. 40
Table 0.7: Use case: UnitSelection ....................................................................................... 40
Table 0.8: Use case: LogonStudent ....................................................................................... 40
Table 0.9: Use case: LogonAdmin ......................................................................................... 41
Table 0.10: Alternative flows: SQLException ....................................................................... 41
Table 0.11: Alternative flows: LogonFailed .......................................................................... 41
1. Introduction

This thesis is going to compare Java and C# in implementation of a university desktop app. Implementation and comparison in this implementation, both are concerned. One app is written in C# and another is in Java. This app will automate the working flow in a university. Teachers and students and admission officers can interact with this system and do their everyday jobs in a university.

In the C# part, we do not have several choices of Interactive Development Environment (IDE) and frameworks and the prominent IDE and framework is MS Visual studio and .Net and Database(DB) is MS SQL server, Of course we have SharpDeveloper IDE too as an example but the most popular one is chosen. In Java side we have several IDEs like NetBeans, Eclipse, and Oracle JDeveloper.

As a developer point of view, we have very strong motivation to understand what should we choose as an IDE in Java or more important what we should choose between Java and C# for developing a desktop app. In this project all tools are free to use and download, and NetBeans has chosen as Java IDE since as you will read it gives a great support to programmer for desktop app and a simple java app is chosen with Swing and not NetBeans platform to make the app runnable in other IDEs and more general and standard. For connecting to DB in Java, Java Persistence API (JPA) and RowSet are used and the DB itself is Oracle DB.

1.1 What is the problem and goal that this thesis covers

Meanwhile one of the goals of this thesis is implementation of these two apps another is a comparison between these two. Comparison will be done in an applied implementation view for developers and with Software quality factors. Therefore we can use this project as an experience how fast and with fewer problems one can develop an enterprise desktop app.

As a constraint, this is not going to write about how to program, but it will be written about main features of chosen techniques and algorithms and why they have been chosen. Java and C# will be compared in these two apps use case by use case and not generally and the features that helps to program these apps will be focused on.

To simplify the problem of comparison and understanding how to program, two roadmaps will be given, One for Java implementation and another for C# implementation. Then the reader with having these two roadmaps and reading the details in the use case implementations can easily understand how to program these two apps and see the differences between these programming languages.

Another constraint here is, DBs will not be compared, but it may be written about the DB features as long as they are chosen by this project and have a difference between C# and Java to use them. It should be mentioned here, this thesis will avoid to writing which programming language is better but it tries to compare their applications in UDA implementation.

1.2 Intended audience

This thesis is very useful for a developer who has experience in one of programming languages Java or C# and wants to start to use another. Developer can see how University Desktop App (UDA) is programmed in his own adapted programming language and how
those data structures and techniques are converted to the other programming language for implementing the real same scenario. Programmer may also does not want to waste his time to choose between different Java IDE’s to develop a desktop app.

You need to know at least a basic understanding of these techniques and products before using this thesis:

- It is supposed that the reader already have knowledge of use case modeling and Unified Modeling Language (UML).
- It is assumed that the reader knows about DB concepts like primary keys and relationships.
- It is supposed that readers have knowledge of C#.Net 4.0 desktop applications and Java 7 Graphical User Interface (GUI) and Swing and DB programming. In .Net, this includes specifically LINQ and in Java JPA.
- It is assumed that the reader knows about MS Visual studio IDE 2010 and NetBeans 7.0.1 and also has passed from Oracle JDeveloper 11g& Application Development Framework (ADF) R2 and Eclipse.
- It is supposed that the reader has passed a course in Software quality and has a basic understanding in its terms like Factors, Criteria, and Metrics and methods.

1.3 What you need to use this thesis
The following products you need in order to run and implement these two apps:

- NetBeans 7.0.1
- Oracle DB 11g R2 express edition (XE): this has some restrictions but they do not have any influence on this thesis project and quiet reasonable for installing it on a developer machine. The restrictions are such as you can not store more than 11 Giga Byte (GB) in XE DB or you can use up to 1GB memory and use one CPU on the host machine [13].
- MS SQL Server 2008 R2 express edition
- MS visual studio 2010 or MS visual C# 2010 express edition
- Microsoft .Net framework 4.0
- Java 7 JDK: you can use the JDK 6 too. The differences between these two JDKs that may influence on the project are on exceptions part and you may just get a compile error in some forms in their exceptions code part when programmer is using the MultiCatch ability. Another part of the code you may get error is Diamond inference. Firstly, this app’s code is written in JDK6 and when it is migrated to JDK 7 in order to gain its benefit these kind of operator are changed by the hint of NetBeans IDE, so you may get error on these lines of code too.
- VizzMaintenance for evaluating Software quality in Java. It is possible to add it in your eclipse as a plugin.
- Understand, which is needed to have a set of metrics for your C# implementation Software quality evaluation.
All these products are free for download except Understand which is available for 15-days trial period.

1.4 How this thesis is structured
Chapter 2 is discussing the background on programing languages and IDEs and the app itself. A supported features table for IDEs will be depicted in its summary. For defining the app its requirements will be gathered traditionally in a table too.

In chapter 3 it will be discussed the use case modeling and system analyzing part. In chapter 4 it will go straight forward to Comparison part. This chapter compares between Java and C# in the form of use cases comparison with focus on implementation view for developers. Chapter 5 will use Software quality factors to compare C# and Java implementation of UDA. In last chapters it will be written a future work in the form of very recent technologies and a final conclusion for the comparison in these two programming languages in this thesis.
2. A discussion and motivation on tools and techniques in this application

In this chapter, firstly the application will be defined by gathering its requirements. Later a background in C# and Java is given, and then NetBeans IDE will be compared with two other famous IDEs, Eclipse and Oracle JDeveloper. Of course we have a number of other IDEs but more famous and free ones are chosen. This comparison will be on their features which have been used in implementation and the supported features in IDEs will be depicted in a table in the summary.

2.1 What is the application

This app is going to automate the process which is being done in a university every day by different actors like teachers and students and admission officers. This app has participated in a programming exhibition in Stockholm and is available online and for easier understanding its functionalities you can see a video online about it:

http://artofcode.se/university-desktop-app[22]

For defining what this application is, we need to define its requirements model.

“Requirements tell us what we should build, not how we should build it.”[2]

In Use case modeling these requirements will be covered by Use cases. Now for simply defining the app we gather all its requirements in table 2.1. This is a more traditional way and simple way than Use case modeling and is just for simply app definition.

2.1.1 UDA System Requirements Specification

UDA System Requirements Specification’s table has a standard structure which includes ID, Details, Type, Priority columns.[2]

<table>
<thead>
<tr>
<th>ID</th>
<th>Details</th>
<th>Type</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The UDA shall authenticate all users in the system.</td>
<td>Registration, Functional</td>
<td>MustHave</td>
</tr>
<tr>
<td>2</td>
<td>The UDA shall register all the user and hold their personal information</td>
<td>Registration, Functional</td>
<td>MustHave</td>
</tr>
<tr>
<td>3</td>
<td>The UDA shall not allow DOS attack in its user interface’s text fields.</td>
<td>Security, NonFunctional</td>
<td>CouldHave</td>
</tr>
<tr>
<td>4</td>
<td>The UDA shall be written in standard Java and C# programming languages</td>
<td>ComplianceTo-Standards, NonFunctional</td>
<td>MustHave</td>
</tr>
<tr>
<td>5</td>
<td>The UDA shall allow the professors to register student’s grades</td>
<td>Courses, Functional</td>
<td>MustHave</td>
</tr>
<tr>
<td>6</td>
<td>The UDA shall allow the admins to CRUD the courses</td>
<td>Courses, Functional</td>
<td>MustHave</td>
</tr>
<tr>
<td>7</td>
<td>The UDA shall allow the student to see their courses</td>
<td>Courses, Functional</td>
<td>MustHave</td>
</tr>
<tr>
<td>8</td>
<td>The UDA shall allow the admins to CRUD the courses</td>
<td>Branches, Functional</td>
<td>MustHave</td>
</tr>
<tr>
<td>9</td>
<td>The UDA shall allow the admins to CRUD professors</td>
<td>Professors, Functional</td>
<td>MustHave</td>
</tr>
<tr>
<td>10</td>
<td>The UDA shall allow the student to</td>
<td>Selection, Functional</td>
<td>MustHave</td>
</tr>
</tbody>
</table>
choose their courses

<table>
<thead>
<tr>
<th></th>
<th>The UDA shall give courses to professors in the terms in different sections</th>
<th>Section, Functional</th>
<th>MustHave</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>The UDA shall use JavaFX for a nicer user interface in Java part.</td>
<td>ComplianceTo-Standards, NonFunctional</td>
<td>WanttoHave</td>
</tr>
<tr>
<td>12</td>
<td>The UDA shall give the possibility to AdminUsers to announce news</td>
<td>News, Functional</td>
<td>WanttoHave</td>
</tr>
<tr>
<td>13</td>
<td>The UDA shall distribute on several servers for its DB</td>
<td>ComplianceTo-Standards, NonFunctional</td>
<td>ShouldHave</td>
</tr>
<tr>
<td>14</td>
<td>The UDA shall use a complete version of DB and not an express version.</td>
<td>ComplianceTo-Standards, NonFunctional</td>
<td>ShouldHave</td>
</tr>
</tbody>
</table>

Table 2.1: UDA System Requirements Specification

2.2 A background on C# and Java in this app

In C# part, Windows form is used to develop user interface of this app and this is perfectly work for this app and already satisfy the requirements. In this project it was not necessary in first version to use newer technique like Windows Presentation Foundation (WPF). Learning new technique need additional fees and time so for the first release windows form is enough rather than WPF.

For DB access, DataSet and LINQ are used. LINQ is an ORM technique[14]. LINQ is invented to cover the shortage to give the programmer the ability to access the DB in an objective way of thinking too. Previously programmer had to develop their programs objectively and at the end they themselves had to convert it to the Database tables or they have to read the database tables and instantiate related objects by hands. Now with LINQ they can work directly with object and the related table or class is created by.Net.

In Java implementation, Swing library and standard Java app are used. Standard Widget Toolkit (SWT) or NetBeans platform exist too, but this project complies with a base standard in order to run without additional libraries.

For DB access, JPA is used. A DataHandler class is written to simulate the DataSet and DataTables in .Net. JPA is like LINQ in Java programing. It gives ORM ability to programmers. Another alternative here is Hibernate, which is an external library. JPA is included in JDK itself so it complies with standard better without additional library.

2.3 Java development environments

This section briefly discusses about NetBeans, Oracle JDeveloper, and Eclipse IDEs and tries to focus on their helping features for UDA project and their deficiencies. When it is referred to NetBeans, it is writing about NetBeans Java app and not NetBeans platform, and when it is discussing about Eclipse, it is written about its last version which is distributed with a windows builder tool. This section tries to discuss about these two IDEs one by one and compare them with NetBeans. Generally when it is written about comparison this report compares for the specific version which is mentioned and at this time that this thesis is written. Some pros and cons to use them will be counted and finally in the summary these IDEs pros and cons will be getting together in a table to simply see and compare them.
2.3.1 Oracle JDeveloper

In 2010 Sun was acquired by Oracle. Sun is the founder of Java and NetBeans IDE [10]. You might consider this IDE to develop this desktop app and firstly this project was implemented with Oracle JDeveloper too. Now two major disadvantages and three Advantages of this IDE in this project will be counted in the next paragraphs.

The first problem with this IDE is, it does not have a support for Multiple Document Interface (MDI) Swing form creation. So you would need to change a form attribute and convert it to a MDI form after its creation or meanwhile that you are creating a Master detail form from your table in your DB. NetBeans IDE can create a MDI app separately from scratch.

I believe that, the main problem of this IDE is, querying and filtering data, in order to customize the dragged and dropped table or the created table with master/detail form wizard, are not as much as easy in comparison with NetBeans. In fact in order to customize the table on your form you need to deal with the View object implementation classes. An example of filtering and querying can be query of students by their student ID which these kinds of task would be typical in this kind of project. Data layer is separated from View layer in the created application by this IDE. So accessing to the Data layer from View to customize the already created table would be programmatically. However Master/detail wizard form creation and as this is mentioned drag and drop tables supporting in this IDE exist in compare with Eclipse that do not exist. In the other hand in NetBeans IDE you use RowSet implementation in some forms that data filtering is needed and because of its simplicity to use, makes the programmer life easier.

Here, it should be written that for changing the appearance of a JTable, Oracle JDeveloper is giving a great support. You can easily change a specific column and fetch as an example student name by its ID from another table like MS visual studio. This feature does not exist in NetBeans and Eclipse as easier as this IDE does. Of course, in NetBeans as an example you can fetch the Branch ID in Student table from Branch table and change the table renderer. However this advantage works when you have already filtered your data.

Another good aspect here would be Navigation bar. In drag and drop table creation or Master/detail wizard form creation this IDE builds a Navigation bar. This Navigation Bar is for ADF library that works well in Swing forms too. Unfortunately we donot have this support in NetBeans like Oracle JDeveloper and MS Visual studio. In NetBeans you need to implement a way to make pagination for your table, and this is possible with RowSet page size and next and previous page methods.

The support for JPA programming is good too. You can easily create with this IDE your JPA entities and controllers like NetBeans.

As the conclusion for this part, the deficiency in giving a simple help to the programmer with filtering data and querying data in Swing forms is as much as important which NetBeans IDE is chosen because of its simplicity. As an example the specific student records by its ID should be acquired from DB table which programming this kind of typical tasks in Oracle JDeveloper is hard. In table 2.2, Supported features in this IDE will summarized as pros and cons.
When you start this project with this IDE you will see that there are two ways to develop a windows app, Swing and SWF. Swing is chosen since it is based on Java itself and it is not needed to add some extra libraries to run. In order to work with Swing library in Eclipse you need the windows builder. The Eclipse windows builder is recently introduced and is possible to install this new software to your Eclipse. When you start your work in this project with Eclipse you will approach three problems.

Firstly, it does not support you with JPA controllers. It has a wizard to just create JPA entities but like Oracle JDeveloper or NetBeans which have wizard to JPA controller too this IDE does not support this feature.

Secondly, when you want to work with DB, when you want to simply drag and drop a table from DB on to your Swing forms it does not support. In NetBeans or Oracle JDeveloper you do this easily too. In NetBeans it creates a list/set/collection of your entities.

Finally, this IDE does not have any master/detail form creation from tables in your DB. NetBeans and Oracle JDeveloper IDE have this feature and it can come very handy. Table 2.3 will summarize supported features for UDA in eclipse as pros and cons.

### Table 2.2: Supported features for UDA in Oracle JDeveloper

<table>
<thead>
<tr>
<th>Supported features in Oracle JDeveloper</th>
<th>Pros+/Cons-</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPA</td>
<td>+</td>
</tr>
<tr>
<td>JPA Controller</td>
<td>+</td>
</tr>
<tr>
<td>Drag and drop table creation</td>
<td>+</td>
</tr>
<tr>
<td>Master/detail form creation wizard</td>
<td>+</td>
</tr>
<tr>
<td>Table customization</td>
<td>+</td>
</tr>
<tr>
<td>MDI forms wizard</td>
<td>+</td>
</tr>
<tr>
<td>Navigation bar in table creation</td>
<td>+</td>
</tr>
<tr>
<td>Simplicity in desktop app</td>
<td></td>
</tr>
</tbody>
</table>

### Table 2.3: Supported features for UDA in Eclipse

<table>
<thead>
<tr>
<th>Supported features in Eclipse</th>
<th>Pros+/Cons-</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPA</td>
<td>+</td>
</tr>
<tr>
<td>JPA Controller</td>
<td></td>
</tr>
<tr>
<td>Drag and drop table creation</td>
<td></td>
</tr>
<tr>
<td>Master/detail form creation wizard</td>
<td></td>
</tr>
<tr>
<td>Table customization</td>
<td>+</td>
</tr>
<tr>
<td>MDI forms wizard</td>
<td>+</td>
</tr>
<tr>
<td>Navigation bar in table creation</td>
<td>-</td>
</tr>
<tr>
<td>Simplicity in desktop app</td>
<td></td>
</tr>
</tbody>
</table>

2.3.2 Eclipse

When you start this project with this IDE you will see that there are two ways to develop a windows app, Swing and SWF. Swing is chosen since it is based on Java itself and it is not needed to add some extra libraries to run. In order to work with Swing library in Eclipse you need the windows builder. The Eclipse windows builder is recently introduced and is possible to install this new software to your Eclipse. When you start your work in this project with Eclipse you will approach three problems.

Firstly, it does not support you with JPA controllers. It has a wizard to just create JPA entities but like Oracle JDeveloper or NetBeans which have wizard to JPA controller too this IDE does not support this feature.

Secondly, when you want to work with DB, when you want to simply drag and drop a table from DB on to your Swing forms it does not support. In NetBeans or Oracle JDeveloper you do this easily too. In NetBeans it creates a list/set/collection of your entities.

Finally, this IDE does not have any master/detail form creation from tables in your DB. NetBeans and Oracle JDeveloper IDE have this feature and it can come very handy. Table 2.3 will summarize supported features for UDA in eclipse as pros and cons.
2.4 Summary
The table 2.4 summarizes the comparison which is done for this project to choose NetBeans IDE (Note: For detail comparisons see section 2.3). Our requirements for UDA have been gathered traditionally and informally and we have defined its boundaries by this section of this report.

<table>
<thead>
<tr>
<th>Supported features in IDEs</th>
<th>NetBeans</th>
<th>Oracle JDeveloper</th>
<th>Eclipse</th>
</tr>
</thead>
<tbody>
<tr>
<td>JPA</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>JPA Controller</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Drag and drop table creation</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Master/detail form creation wizard</td>
<td>+</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Table customization</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>MDI forms wizard</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>Navigation bar in table creation</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Simplicity in desktop app</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 2.4: Supported features in IDEs
3. Requirement engineering and Design
UDA would have several functionalities. For knowing that how this app should work and gathering the needs of stakeholders we need a methodology which our documentation are based on. An iterative waterfall development process with UML in this project has been used [9]. A standard and professional way for gathering of functional requirements of a system is Use case modeling [2]. This chapter will be implementation fundamental. Following this section, project glossary will be given. Later UDA use case diagram and use case and actor specifications will be introduced. In the end UDA DB diagram will be drawn.

3.1 UDA Project Glossary
The project glossary will help us to gather all the specific technical words which are used in this project in order to communicate with stockholders and help them to read our use case specifications.

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRUD</td>
<td>Creation, Read, Update, And Delete data in Database.</td>
</tr>
<tr>
<td>Database/DB</td>
<td>Data storage system.</td>
</tr>
<tr>
<td>SQL Exception</td>
<td>An error which a DB might produce.</td>
</tr>
<tr>
<td>ID</td>
<td>Identification number as a unique key to select records in DB.</td>
</tr>
<tr>
<td>Record</td>
<td>Data is stored in DB as a bunch’s of records or row.</td>
</tr>
<tr>
<td>Field</td>
<td>Each record is constructed by several fields.</td>
</tr>
<tr>
<td>Boolean field</td>
<td>A field which holds false or true value.</td>
</tr>
<tr>
<td>Table</td>
<td>DB is constructed by several tables and they hold records.</td>
</tr>
</tbody>
</table>

Table 3.1: UDA Project glossary

3.2 UDA Use case diagram
In a use case diagram we will try to capture and depict the functional requirements of the system in the form of use cases and actors which are interacting with each other. As this development process is iterative, so in the first iteration this is not needed to have a very complete use case diagram, we can add more use cases as we recognize their need. In figure 3.1, we have three actors, Student, Admin, and Professor. Between Admin and Professor we have generalization relationship and AdminUser is an abstract actor. We have ten Use cases in UDA, for each of these use cases we will have a use case semantic later.
3.3 Actor semantics

In table 3.2 we will define all the actors who will interact with UDA.

<table>
<thead>
<tr>
<th>Actor</th>
<th>Semantic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Admin</td>
<td>A professor who is also a personnel as a manager.</td>
</tr>
<tr>
<td>Student</td>
<td>Student at university.</td>
</tr>
<tr>
<td>Professors</td>
<td>Professor at university.</td>
</tr>
<tr>
<td>AdminUser</td>
<td>Someone who uses the system but who is not a student.</td>
</tr>
</tbody>
</table>

Table 3.2: Actor semantics

3.4 Use case semantics

In use case semantic we will use a template which include Use case Name, ID, Brief description, Actors, Preconditions, Main Flow, Post Conditions and Alternative flows. We separate Alternative flows in the end of use case specifications. For writing use case specification we should keep in mind that they should be readable and easy to understand [2]. Therefore details about the main and alternative flow parts will not be given.
### Use case: CRUDBranch

<table>
<thead>
<tr>
<th>ID</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brief description</td>
<td>CRUD operation on branches.</td>
</tr>
<tr>
<td>Actors</td>
<td>Admin</td>
</tr>
<tr>
<td>Preconditions</td>
<td>The admin is logged on to the system.</td>
</tr>
</tbody>
</table>
| Main Flow | 1. The use case starts when an admin want to CRUD a branch.  
2. The system shows a table with branch name and branch id with the ability to delete and add and update its records.  
3. The admin enters Branch ID and Branch Name to add or select one from the table to edit or delete it.  
4. The system CRUD the admin’s selected branch and refresh the branch table. |
| Postconditions | The CRUD has been completed. |
| Alternative flows | SQLException |

Table 3.3: Use case: CRUDBranch

The use case in table 3.3 is an example of the use case semantic tables. The rest of use case semantic tables are available in appendix C. ID is the unique key of each use case, Main flow is describing its work flow and Preconditions and Postconditions are system’s initial state and last state for the use case.

### 3.5 C# Class diagram

Figure 3.2 gives an overview on C# class diagram. This implementation is more database driven which means that more CRUD data with DB with automatic IDE created classes are being done rather than operation between programmer written classes. Therefore this class diagram has been completed with more details after implementation of the program.

Since in drag and drop tables and visually setting properties as an example in DataGridView, C# is building behind the scene some other classes especially in dbDataSet relation, for simplicity to read the class diagram they have been dropped. One can see them with expanding Data source explorer in visual studio. They are branch, crs, fullstudview, fullprof, Grade, prof, sec, secByLastTerm, select, selectByLastTerm, selectByStudID, selectByTerm, and stud.

Again for simplicity to read this class diagram and Database driven implementation of this app, simple class UML notation has been used. Detail class attributes have been added in implementation and after drag and drop tables and interacting with Database for programing the app. MainContainer is main entry to the program. In figure 3.2, it can be seen that LINQ Data context classes, profs, studs, have just been used in login classes.
3.6 Java Class diagram

Figure 3.3, is an overview on Java Class diagram. For simplicity to read simple UML notation class has been used and Package notations have been dropped in overview class diagram. MDIApp is the main entry of the program. It can be seen that JPA entities have been used more than LINQ entities class in C# class diagram. JPA entities classes are Prof, Stud, Crs, and Branch.
3.7 Database diagram

The output of UDA DB diagram from MS SQL Server is like figure 3.4. We have 6 tables branch, prof, stud, sec, select, crs. Generally for each entity in the system which interacts with it and has influence on its functionality and its states we need to have a table in our DB. Association relations like sec between prof and crs and select between crs and stud would have a table also. Name and family can be seen in two different separated fields. In this way we can sort by name or family easily later. With help of a view on the table we can have these two fields merged.

3.8 Summary
UDA Database in MS SQL Server for C# programming and the same database structure in Oracle DB for Java part are at our hands. There are 12 functionalities which the UDA needs in order to work.
4. Java vs. C# in an applied implementation view for developers

In this chapter the techniques and algorithms which have been used to implement different use cases in this project in Java and C# will be compared use case by use case. Two main roadmaps for implementing this project in these two programming languages will be given. Both GUIs in this project have implemented almost the same. The differences with the main roadmap will be discussed in each use cases after these two main roadmaps, and several comparison points whenever we reach to them will be mentioned explicitly. These two road maps themselves can be seen as a generic use case. In Java road map comparison points will be shown themselves up. A summary comparison between these two roadmaps and these comparison points will be at last.

4.1 C# implementation roadmap

![C# implementation roadmap diagram]

Figure 4.1: C# implementation roadmap

In the C# part the main road path is short and simple since the MS Visual studio and .Net is giving great support. You just need to choose a new windows forms application project and add windows form for each use cases one by one and drag and drop the related table from Data source to the forms and a Navigation bar will be created too. The DataError event handling in DataGridView would be the same too. You can give the customized and nice messages instead of .Net default error messages with implementing this method. The detail implementation may be different and it will be discussed.
4.2 Java implementation roadmap

Figure 4.2: Java implementation roadmap

In Java side and NetBeans, you need to create new Java application, notice that you have Java desktop application or NetBeans platform too, but the problem with NetBeans platform is that it is specific with NetBeans IDE and the problem with Desktop applications in NetBeans is that it is based on the previous version of Java Specification Request (JSR 296) which is not supported anymore by official Java Development Kit.

After that you will create your JPA entities, better in a separate package within the project. Then you will create your JPA Controller for those entities with NetBeans wizard. These JPA entities are used in Logon form implementation directly when you drag the student or professor table from DB on to your form. These are also are used in master/detail forms. Of course you can forget to create yourself these entities and just drag and drop tables on to your forms visually and NetBeans IDE creates them for you too.

Next step, you can deal with writing the JDBC package and the DataHandler class which is using a RowSet implementation by the Sun. Notice here you have different implementations of the RowSet interface, Oracle and Sun. The Sun CachedRowSet implementation has been chosen, Since Oracle Implementation in Pagination closes the RowSet when it reaches at last page and when previous page is called in order to come back and traverse the RowSet, it throws a Closed ResultSet Exeption.

Here, as a comparison point and a same characteristic behavior of the chosen data structures in C# and Java, it should be written about the Rowset ability like DataTable in .net Dataset, which both are Serializable and one can send them via a socket through network in a
distributed system. Also, these two data structures are both disconnected, so with setting the pageSize to a needed amount you can efficiently work on a reasonable number of records in your memory in the client side from DB server and then hand the updated records back to DB after data manipulating.

The next comparison point can be seen visually in these two roadmaps in differences between DataHanlder and JPA in Java and DataSet and LINQ in C# for data accessing. DataHandler class has written to simulate DataSet. DataSet is visually manageable but this DataHandler is written and managed by codes. JPA and LINQ both are implementing the same techniques. They are giving the programmer ability to work with the objects instead of the SQL statements and queries and relations. They both are Object-Relational Mapping, ORM techniques.

Another comparison point as a difference is when you drag and drop a table from your DB on to your form. Behind the scene C#.Net is creating a DataTable for you in your DataSet or is using already created one. But in Java, NetBeans creates a JPA entities or uses already created one for CRUD operation on that table as an entity. In contradiction, C# is using its traditional way and Data Set (DS) to manipulate data and is not using its recommended approach and ORM technique yet[12].

In this roadmap the main implementation part path is, you would create a new MDI application sample form and then add your JInternalFrame or Master/Detail Sample forms for each use cases. NetBeans creates Master/Detail forms for the CRUD operations on your table by Wizard. But when it reaches to customization of your data and gathering data from different tables there is nothing more accessible than creating Table models classes and connecting to your data with DataHandler class which you just wrote, and designing the GUI on your JInternalFrame class.

If you see the classes in the project main package namely MainUNIApp, you will see a JInternalFrame convertor class which converts the Master/Detail forms which are JPanel to JInternalFrame in order to add to your desktopPane in your main container MDI form.

There are two closing and finalizing programming parts which you should always do when work with RowSet. One is you should always close the Rowset in the after closing event of the GUI form. Another is you need to close the Rowset in Finalize protected method of your Table model class.

4.3 Use case: CRUDBranch
In this use case which has data from just branch table the main road path is just followed in both C# and Java sides. In Java side, the Master/Detail form with JInternalFrame has been added to desktopPane in main MDI container.

4.4 Use case: ProfessorsRegistration
The difference with previous use case here is, in Java part the isPersonal field is customized later. Its column is changed to a ComboBox with the values of 0/1 so the user can choose just between these two values to update the table. The reason to have 0/1 here is that Oracle DB does not have any Boolean field like MS SQL Sever. So in .Net part we have a CheckBox.
4.5 Use case: StudentRegistration
In this use case in .Net part, the studDataGridView bra_id field has been just changed to fetch from branch table and set the ‘DisplayMember’ attribute value to ‘bra_name’ and customize data on my form.

In Java part, in order to have a more beautiful GUI, a Table model class form has been created so data can be customized better. A paginated Rowset has been created too and in order to show the functionality to the stockholders and present this thesis before actual data entry the pageSize has been set to 3. The value of the Branch ID from Branch table has been fetched too.

4.6 Use case: CRUDCourse
This use case like the first use case does not have any special customization in both sides and is following the default roadmaps.

4.7 Use case: CourseRegistration
This use case is like StudentRegistration use case. In C# and Java part, the values of Course and Professor’s name by their IDs have been fetched from their tables, and we have table pagination in both sides.

4.8 Use case: GradeRegistration
In C# side, the implementation is a bit different from previous use cases. From the roadmap again, this use case has data from different tables, Grade and Select. This Grade table is added to our DS by a SQL query and the Select DataTable is added by the .net automatically as its child. This Grade table is joining the sec and crs tables for obtaining the course name. After that the Grade table has been dragged and dropped to the form and inside of that its Select table as its children too.

Here as a comparison point, it should be mentioned the robust visual ability of .net that the column student name could be visually added and fetched from full student name from its fullstud view table by their IDs. The additional fields, term have been removed, since this use case is implemented for the last term.

Finally, the navigation bar has been customized and the delete and add buttons have been removed and the default Exception messages have been handled by implementing the DataError event.

In Java side, two table models have been created. The first table, Sec, is exactly showing the same query result with the .net part. In the mousePressed event of Sec table, the course ID has been got, and the students who got the course from Select table for the specific logon professor in the last term from the DataHandler has been got too, and the result in the second table, Grade, has been shown on the form.

Here as the java side comparison point, this should written that customization of the Grade table needs another implementation of RowSet. If you want to make a more sophisticated appearance like in .net and show student name by their ID you have to use a SQL join query between Select and Sec tables, and you need to use JoinRowSet. The CachedRowSet implementation which is used already satisfied functional requirement by the stockholders.
4.9 Use case: GradeDeclaration
In the C# part, since this use case has data from just one data table in the DS and more or less the same as the roadmap. You need to add a DataTable for fetching from Select tables with the specific student ID and the term that the student enters.

In the Java side, like C# the implementation is not different with roadmap. You need to implement the functionality in the button ActionPerformed event to get the term from the student, and then you need to implement and set just one TableModel for your Grade table.

4.10 Use case: UnitSelection
In C# side, this use case has data from different tables. The first table Sec is built with a join query with Crs and Prof tables in order to have course professor name in the table. The second table is our select table in the last term for the specific student ID. You need to drag and drop these two DataTables which donot have parent and child relationship as it is written on the roadmap.

Then you need to implement the add functionality of the navigation bar to insert values from the first table Sec into second table Select. In the CellClick event of the first table grid event you need to get these values. For the Delete button in navigation bar you need to delete the row that the student has selected. Finally after these adding and deleting, you need to call the Fillby method on your TableAdapters to refresh the update values on your table grids.

In Java side, you create two table models and in the MouseClicked event of the first table Sec you read the values and insert them in the add button ActionPerformed method to the selRowSet in the selTableModel. For deleting you need to use the getSelectedRow method on Sel table and delete that from selRowSet in the selTableModel.

4.11 Use case: LogonStudent
In C# part, after adding the LINQ to SQL classes as the roadmap shows, you have access to all the tables as they are classes with the same table names in your project. So you can Firstly instantiate your LINQ to SQL Data Context class then you can instantiate your student class and with a Foreach loop or with a firstORDefault method go through the student table and check the student ID and password.

In Java side, you need to just drag and drop you student table on to your form and the JPA entity is created by NetBeans. NetBeans give you a collection or a list or a set of your records in the tables as you have defined in your JPA entities creating before. So with a Foreach loop go through it and check the student ID and password.

4.12 Use case: LogonAdmin
This use case is the same as previous use case for AdminUser.
### 4.13 Summary
The table below summarizes the comparison points which have been mentioned between C# and Java in this project.

<table>
<thead>
<tr>
<th>Comparison points</th>
<th>Java</th>
<th>C#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data access</td>
<td>DataHandler</td>
<td>DataSet</td>
</tr>
<tr>
<td>Serialization</td>
<td>RowSet</td>
<td>DataTable</td>
</tr>
<tr>
<td>ORM</td>
<td>JPA</td>
<td>LINQ</td>
</tr>
<tr>
<td>Drag&amp; Drop Table</td>
<td>Resulted in JPA entity creation</td>
<td>Resulted in DataTable usage</td>
</tr>
</tbody>
</table>
5. Java vs. C# in Software quality factors

In this chapter we compare C# and Java in UDA in Software quality point of view. Firstly a background on factors and metric and standards and tools will be given. Then we visualize the metrics values which have been gained with Software quality measurement tools. Then we estimate these metrics in our Software quality model for Maintainability and Reusability factors. At last a summary of these factors will be given.

5.1 A background on Software quality

We have Standard ISO 9126 which discuss about software quality factors. We use maintainability and reusability of these factors to evaluate and later compare these implementations of UDA.[19] This thesis is not going to write which one is better, but this thesis will use direct metrics like Cyclomatic complexity, Lines Of Code (LOC), Weighted Method Count (WMC), Data Abstract Coupling (DAC), Tight Class Cohesion (TCC), Response set For a Class (RFC) and indirect metrics like maintainability, testability, learnability, design, and understandability in each implementation for evaluation on their own, then they can be used in final conclusion in these two implementations in chapter 6.

"Software Quality is the entirety of properties and attributes of a product or a process relating to their fitness to fulfill certain requirements." (German Institute of Standards-DIN 55350/11)

As this standard is referring to fulfill certain requirement, these requirements as metrics to compare these programming languages in this project are those metrics which have been just mentioned. For computing these metrics and factors, two tools are used, VizzMaintenance in Java, and Understand in C#. Understand as a stand alone software can be installed in its trial version and VizzMaintenance is free as a plugin in eclipse. Therefore you need to import your Java project to your eclipse to use VizzMaintenance, you do not need to run it, just import it even if without setting its class path since we do not want to run it in Eclipse, we just want to run the VizzMaintenance plugin on it. In this chapter these attributes which have been defined and discussed in advance together with stakeholders and teachers in this thesis will be discussed.

Capabilities Maturity Model (CMM) for this development process is still in defined level, it means that development purpose after passing initial and repeatable CMM levels is to have a qualitative, reliable cost and schedule, and improving but unpredictable quality performance level. In the next releases and iterations of the implementation CMM would be in Managed level to quantitative basis for continuous improvement, and optimizing level, as final level in CMM to consider quantitative statistics over product quality.[15][19]

In C# part it should be noticed that, .Net framework is giving a good visual support in order to develop UDA. For understanding which support it is giving to the programmer, C# implementation road map and its detail description in each use case in chapter 4 should be read. Therefore, the result of each measured metrics in C# part is coming back to .Net framework’s automated code and class creation itself and not the hand written codes and classes of UDA’s programmer himself, and most of the work is configuration.
5.2 Visualizing metrics
In Java side with help of VizzMaintenance we can have a complete set of measured metrics. Just we need to have them visually to better understand and see them. It should be mentioned that Depth In Inheritance Tree (DIT) = Number of Children (NOC) = 0 means that we do not have any inheritance tree and therefore number of children in an inheritance tree either. Understand will help us with WMC, LOC, NOC, RFC, and DIT. After visualization of metrics, we will discuss and interpret our analysis result with focusing on their outliers’ classes and values. To see the raw data you can see Appendix A and B.

5.2.1 Classes which are large and complex
These outlier classes which are complex can indicate classes which are relatively hard in maintainability, understandability, learnability, and testability criteria too. It has this meaning that they are hard to be modified in order to renew them with new requirements or functionalities or to be maintained or understood or learnt or tested by a software engineer in compare with other classes. [19][20] Of course with support of IDE these problems can be reduced.

![Figure 5.1: Size vs. Complexity in Java](image)

Table 5.1: Figure 5.1 outlier classes legend

<table>
<thead>
<tr>
<th>No.</th>
<th>Class Name</th>
<th>LOC</th>
<th>WMC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MDIApp</td>
<td>531</td>
<td>29</td>
</tr>
<tr>
<td>2</td>
<td>SelectionJpaController</td>
<td>189</td>
<td>29</td>
</tr>
<tr>
<td>3</td>
<td>SecJpaController</td>
<td>187</td>
<td>29</td>
</tr>
<tr>
<td>4</td>
<td>JInternalFrameSec</td>
<td>521</td>
<td>19</td>
</tr>
<tr>
<td>5</td>
<td>JInternalFrameStud</td>
<td>517</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 5.1 is outlier’s class names legend for figure 5.1 with their WMC and LOC values. Outlier class No.1 is an outlier in WMC too since it has the most LOC, because In WMC metric, LOC itself, LND, and McCabe Cyclomatic Complexity (MCC) are counted by tool also. Where WMC=29 we have two class with LOC= 187 SecJpaController, and LOC=189
SelectionJpaController which are created by IDE for accessing Sec and Selection JPA entities in DB respectively. These classes have high number of MCC since its methods has a lot of if statements and they increase MCC value, therefor its WMC is an outlier. JInternalFrameSec, and JInternalFrameStud are the highest LOC values after MDIApp. These classes are for creating GUI and customizing and showing DB tables on windows forms.

![Figure 5.2: Size vs. Complexity in C#](image)

<table>
<thead>
<tr>
<th>No.</th>
<th>Name</th>
<th>LOC</th>
<th>WMC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DataSet.DBDataSet</td>
<td>6106</td>
<td>89</td>
</tr>
<tr>
<td>2</td>
<td>DataSet.DBDataSetTableAdapters.TableAdapterManager</td>
<td>868</td>
<td>123</td>
</tr>
<tr>
<td>3</td>
<td>DataSet.DBDataSetTableAdapters.studTableAdapter</td>
<td>475</td>
<td>47</td>
</tr>
<tr>
<td>4</td>
<td>MSUNI.MainContainer</td>
<td>407</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>DataSet.DBDataSetTableAdapters.secTableAdapter</td>
<td>378</td>
<td>33</td>
</tr>
<tr>
<td>6</td>
<td>DataSet.DBDataSet.secDataTable</td>
<td>297</td>
<td>35</td>
</tr>
</tbody>
</table>

Table 5.2: Figure 5.2 outlier classes legend

Figure 5.2 is gained with raw data from Understand tool. In order to gain a more precise image and discussion in Figure 5.2, two outliers IDE automated built with label No.1, 2 have been dropped. Outlier class No.4 MainContainer.cs which is our main entry to program and is calling all other forms therefor has a high LOC. Sec DataTable and TableAdapters in outliers No.5,6, both are gathering data from Crs and Prof entities, therefor they have high WMC and LOC and are complex. studTableAdapter which is also an IDE automated built class show a high LOC and WMC which refers its complexity. One reason to have a high LOC would be it has a lot of fields to CRUD in contrast with other entities.
5.2.2 Classes which are complex and tightly coupled with other parts of UDA

These key outlier classes help us to get a basic understanding of the system. One of the initial steps to understand a legacy system would be its documentation, however it is not sufficient. Therefore these metrics analysis help us to understand better the system.[20]

In figure 5.3, the classes which are in upper right corner of figure are showing the classes which have the highest complexity, WMC and highest coupling, CBO and understanding them will help to gain a basic understanding of UDA. Where CBO=12&WMC=29, class name is MDIApp and where CBO=6&WMC=29, there are two classes, SecJpaController, SelectionJpaController. It should be noted that these two controller classes are not currently used in UDA, since it is using JDBC package and DataHandler class to create JInternalFramesec and JInternalFrameUnitSelection forms. Where CBO=5&WMC=16, class name is JInternalFrameUnitSelection, which is connecting to DataHandler to get data from Sec and Selection tables from DB and fill it in the form and the reader can get an idea about how data customization and handling have been done in UDA.
Figure 5.4: Getting a basic understanding of UDA in C#

In figure 5.4, where CB=57&WMC=89, class name is DBDataSet. Obviously this is the most complex and coupled class in UDA. As much as the reader can understand this class, he can understand further customization of data on it in different forms in UDA better.

5.2.3 Classes which have low cohesion and high coupling

In object oriented design, a very basic rule is that one class should implement one concept, having high coupling means that the class itself does not implement one concept and using many other classes and having low cohesion can indicate that the class is probably implementing more than one concept and these concept could be implemented separately. Therefore, these outlier classes are violating this principle design rule, one class-one concept.

“These classes tend to have either low TCC-values or high DAC-, RFC-values.”[20]

Classes with very low TCC just gather different kind of methods together and do not implement an object concept in UDA and act as a module.

In figure 5.5 where DAC=12, the class name is MDIApp which is our main container MDI form which we are calling all of other classes from. Therefor DAC is high for this sensible class. As definition mentioned too, this class just is acting as a module to instantiate other objects in UDA. Where DAC=6, we have three classes, SecJpaController, SelectionJpaController, DBM. The first two classes are created by IDE automatically and DBM is just for testing purpose. SecJpaController is using Crs, Prof, and Sec entity classes a lot and SelectionJpaController is using Student, Selection, and Crs entity a lot. Therefore their DAC are high.
In figure 5.6 where Lack of Cohesion of Methods (LCOM)=499, the class name is MDIApp. This shows that meanwhile this class has high coupling, it has the lowest cohesion, since its method is not related to each other and each method is calling a separate UDA form, like description in figure 5.5. Where LCOM=231&RFC=23, the class name is Stud which is an automatic IDE built JPA entity. This class is interacting with UDA a lot so it is reasonable that its method can potentially invoke other classes’ methods as definition of RFC. These methods are not interacting with each other a lot which is obeying the definition of LCOM too. [20]
In figure 5.7, where LCOM=96 & RFC=682, No.1 class name is MainContainer and description would be exactly the same as Java part. Where LCOM=80 & RFC=676, No.2 class name is UnitSelection. Methods in this class like Stud in Java, have a high potential to interact with other classes’ methods, therefore as definition of RFC, it has a high value in RFC.

5.2.4 Classes which will often change since they depend on a lot of other classes
This classes are identifying those change sensitive classes which we should test them last and they help use to estimate change effort. Classes with high DAC or Coupling Between Objects (CBO) values know a lot of other classes and therefore are change sensitive, and changing them can end up with changing a lot of other classes. These classes can reduce the maintainability of the UDA too, and they should be monitored for improving Software quality of UDA in next releases. [19][20]
Figure 5.8: Change sensitive classes in Java

In figure 5.8, where RFC=29 and DAC=12, the class name is MDIApp. The reason is obvious, it is calling other classes and whenever they change, MDIApp will be changed too. Where RFC=23 and DAC=6, the class name is SelectionJpaController, which is an IDE automatic created class for accessing selection DB table. This class also is highly depended on Stud and Crs entity classes and therefore is a highly changing class too.

Figure 5.9: Change sensitive classes in C#

In figure 5.9 Where CBO=57&RFC=193, class name is DBDataSet, which obviously is so change sensitive class since it is under influence of all the data table classes in DB. Where CBO=43&RFC=675, class name is gradeDeclaration. This class also is change sensitive since whenever professor gives grade this class will be changed too.
5.3 Reusability and Maintainability of UDA

We need to have a more precise measurement to have a conclusion for comparison between C# and Java in UDA. We use complexity metrics LOC and WMC to estimate factors, Maintainability and Reusability’s criteria. These criteria are listed under label sub property in table 5.3. This table will be used for our measurement to compute the factors. Table 5.4 is table legend for 5.3. It shows that how we should compute the measurement for each sub property.

<table>
<thead>
<tr>
<th>Category</th>
<th>SubCategory</th>
<th>Metric</th>
<th>Sub Property</th>
<th>Re-Usability</th>
<th>Maintainability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity</td>
<td>Size</td>
<td>LOC</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>StructuralC.</td>
<td>WMC</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Architecture &amp; Structure</td>
<td>Coupling</td>
<td>CBO</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>Cohesion</td>
<td>LCOM</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

Table 5.3: Software quality matrix[21][19]

<table>
<thead>
<tr>
<th>Sign</th>
<th>Description</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>--</td>
<td>highly inversely related</td>
<td>2</td>
</tr>
<tr>
<td>-</td>
<td>inverse coupling between metrics and criteria</td>
<td>1</td>
</tr>
<tr>
<td>++</td>
<td>highly direct coupling between metrics and criteria</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 5.4: Table 5.3 legend[19]

For counting class outliers three methods have been used, 15 outlier classes, 15% outlier classes of UDA, and outliers between value range trend with this formula:

\[
\text{min} + (\text{max} - \text{min}) \ast 85\% = \text{max} \quad \text{(if low is good)}
\]

\[
\text{min} \quad \text{min} + (\text{max} - \text{min}) \ast 15\% = \text{max} \quad \text{(if high is good)}[19]
\]

Which min is the lowest metric value and max is the highest one. We aggregate these metrics in package level, and then we integrate them for each criterion with simply adding them together.
Table 5.5 shows that which packages have the worst Maintainability and Re-Usability when UDA is implemented with Java. Package MainUNIJApp has the worst maintainability in all methods of outlier counting. The better rank means the worst value. The reason would be its worst values in sub properties. This package is the main implementation package and from road map is the main implementation path too. Therefore for changing and adapting it with new functionalities developer should put his most effort on it.

In Reusability factor, MainUNIJApp, TableModel, and JPACTRL are showing the highest ranks. It means that they are hard to use. Because MainUNIJApp and TableModel together are implementing the forms and JTable customizations and they can be hard to understand and learn as sub property shows too. JPACTRL would be hard in Reusability with respect in its sub property, programmer has to deal with it to reuse it and customize data from DB.
Table 5.6: UDA factor measurement in C#

Table 5.6, shows that DataSet namespace is the worst Re-usability factor in all methods of outlier counting in UDA C# implementation. DataSet has been reused almost in all the forms of UDA in C# implementation. Namespace Entities and DataSet.DBDataSetTableAdapters also shows a poor Maintainability index. In regard to its sub property TableAdapters namespace also have a high value and in fact is being used in all the forms of UDA to use data. Entities namespace which contains prof and stud classes, are being used in authentication forms. This namespace and its classes have been created by IDE automatically. Programmer need to be carful in maintenance of these namespaces.

5.4 Summary
In this chapter we have started with the standard and factors definitions which have been used in UDA after that we visualized metrics in order to get a descriptive image of UDA. At the end, we measured two important Software quality factors, Maintainability and Reusability of UDA.
6. Conclusion

This thesis has fulfilled its goals in implementation of a desktop application, UDA and compared C# and Java programming language in UDA. In Chapter 4 we discussed comparison of UDA Java and C# version in their implementations. We have given two road maps in sub chapter 4.1 and 4.2 which helps to develop the same application in this domain and get the developer’s hand who has knowledge in one of these programming languages and want to switch to another. He could compare different techniques descriptively with having these road maps and we have mentioned some comparison points in UDA to use them. Main comparisons points between these roadmaps have been pined points in this project:

- In Java a DataHandler class is needed to access DB, in C#.Net DataSet is used.
- RowSet in Java and DataTable in C# are both Serializable.
- ORM techniques has been used, JPA in Java and LINQ in C#.
- Drag and drop DB tables resulted in JPA entities creation in Java and Datatable usage in .Net.

In chapter 5, we change our view from an applied comparison to a more scientific way with considering Software quality factors Maintainability and Reusability. We measured Software quality criteria and metrics and evaluate these indexes of UDA in chapter 5.3. In chapter 5.2 we visualize different metrics which can help programmers to understand, maintain, test and reuse UDA in its different implementations with Java and C#. The developer in this way can see obviously with measurements if he chooses C# or Java how would be these factors in his final product precisely. The result of this measurement has been pined points in this project:

- In Java MainUNIJApp package has the worst Maintainability index, and MainUNIJApp and JPACTRL and JPACTRL.exceptions have the worst Reusability index.
- In C# side, DataSet.DBDataSetTableAdapters namespace and Entities have the worst Maintainability and Reusability indexes.

To put the Software quality factors measurements in a nut shell, if Maintainability and Reusability indexes in GUI and Data customization on UDA’s forms are the most important aspects to choose between Java and C# as the choice of programming language of UDA, The programmer should go for C#, since in Java side these indexes are showing poor measurements in MainUNIJApp package which is handling GUI creation and data customization on UDA’s forms. One reason according to C# implementation roadmap and its description in each use case would be the great support which .Net framework is giving in order to builds UDA’s windows forms and their data customization.

On the other hand if the developer wants a better Maintainability index for accessing data, he should go for Java since in C# part TableAdapters and Entities namespaces which are handling data access part of UDA, are showing a poor indexes values for this criterion. From implementation point of view, one reason would be usage of ORM technique in Java and NetBeans implementation of UDA. In C# part, .Net Framework is still using its traditional
and not modern approach in accessing data from DB in windows forms of UDA and this is discussed in chapter 4 in one of the comparison points.
7. Future work
JavaFX has the ability to be deployed within a web browser page and it can extend already implemented Java version of UDA. It has already leveraged NetBeans IDE in its version 7.1 release candidate 2 by the date that this report has been written. It can help to make UDA distributed. Therefore would be a good choice to switch on in next release of UDA.[24]

In the .Net world we have Microsoft entity framework (EF) 4.1 and code first technology. Microsoft recently introduced code first and it has recommended EF as the programmer choice for accessing Data [12]. With code first as its name is showing programmer first write his code later with .net compiler the DB will be created.
References


| No. | RuleRefID | name        | Materializability | CIO<|C|Classes | DAC | DT | ILCOM | LCOM | LD | LENV | LOC(C>/classes) | MPC | NAM | NOD | MOD | RFC | TCC | WMC  |
|-----|-----------|-------------|-------------------|------------------|--------|-----|------|-------|------|----|------|-----------------|-----|-----|-----|-----|-----|-----|-----|
| 1.  | 1.         | VizzMaintenance | raw data         |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 2.  | 2.         |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 3.  | 3.         |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 4.  | 4.         |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 5.  | 5.         |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 6.  | 6.         |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 7.  | 7.         |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 8.  | 8.         |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 9.  | 9.         |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 10. | 10.        |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 11. | 11.        |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 12. | 12.        |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 13. | 13.        |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 15. | 15.        |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 16. | 16.        |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 17. | 17.        |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 18. | 18.        |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 19. | 19.        |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 20. | 20.        |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 21. | 21.        |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 22. | 22.        |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 23. | 23.        |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 24. | 24.        |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 25. | 25.        |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 27. | 27.        |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 28. | 28.        |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 29. | 29.        |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 30. | 30.        |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 31. | 31.        |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 32. | 32.        |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 33. | 33.        |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 34. | 34.        |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 35. | 35.        |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
| 36. | 36.        |              |                   |                   |        |     |      |       |      |    |      |                  |     |     |     |     |     |     |     |
## Use case semantics

### Use case: ProfessorsRegistration

<table>
<thead>
<tr>
<th>ID</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brief description</td>
<td>CRUD operation on professors.</td>
</tr>
<tr>
<td>Actors</td>
<td>Admin</td>
</tr>
<tr>
<td>Preconditions</td>
<td>The admin is logged on to the system.</td>
</tr>
</tbody>
</table>
| Main Flow | 1. The use case starts when an admin want to CRUD a professor.  
2. The system shows a table with professor ID, name, family, password, and isPersonnal Boolean field with the ability to delete and add and update its records.  
3. The admin enters above values to add or select one from the table to edit or delete it.  
4. The system CRUD the admin’s selected professor and refresh the professor table. |
| Postconditions | The CRUD has been completed. |
| Alternative flows | SQLException |

Table 0.1: Use case: ProfessorsRegistration

### Use case: StudentRegistration

<table>
<thead>
<tr>
<th>ID</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brief description</td>
<td>CRUD operation on students.</td>
</tr>
<tr>
<td>Actors</td>
<td>Admin</td>
</tr>
<tr>
<td>Preconditions</td>
<td>The admin is logged on to the system.</td>
</tr>
</tbody>
</table>
| Main Flow | 1. The use case starts when an admin wants to CRUD a student.  
2. The system shows a table with student ID, name, family, address, tell No., registration date, branch ID, password with the ability to delete and add and update its records.  
3. The admin enters above values to add or select one from the table to edit or delete it.  
4. The system CRUD the admin’s selected student and refresh the student table. |
| Postconditions | The CRUD has been completed. |
| Alternative flows | SQLException |

Table 0.2: Use case: StudentRegistration

### Use case: CRUDCourse

<table>
<thead>
<tr>
<th>ID</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brief description</td>
<td>CRUD operation on courses.</td>
</tr>
<tr>
<td>Actors</td>
<td>Admin</td>
</tr>
<tr>
<td>Preconditions</td>
<td>The admin is logged on to the system.</td>
</tr>
</tbody>
</table>
| Main Flow | 1. The use case starts when an admin wants to CRUD a course.  
2. The system shows a table with course ID, name, unit, type with the ability to delete and add and update its records.  
3. The admin enters above values to add or select one from the table to edit or delete it.  
4. The system CRUD the admin’s selected course and refresh the course table. |
| Postconditions | The CRUD has been completed. |
### Use case: CourseRegistration

<table>
<thead>
<tr>
<th>ID</th>
<th>Brief description</th>
<th>Actors</th>
<th>Preconditions</th>
<th>Main Flow</th>
</tr>
</thead>
</table>
| 5  | Giving courses to professors. | Admin | The admin is logged on to the system. | 1. The use case starts when an admin wants to give a course to a professor.  
2. The system shows a table with course ID/name, group ID, term, professor ID/name, capacity, and time with the ability to delete and add and update its records.  
3. The admin enters above values to add or select one from the table to edit or delete it.  
4. The system CRUD the admin’s selected record and refresh the table. |

<table>
<thead>
<tr>
<th>Postconditions</th>
<th>Alternative flows</th>
</tr>
</thead>
<tbody>
<tr>
<td>The course is given to the professor in the system.</td>
<td>SQLException</td>
</tr>
</tbody>
</table>

### Use case: GradeRegistration

<table>
<thead>
<tr>
<th>ID</th>
<th>Brief description</th>
<th>Actors</th>
<th>Preconditions</th>
<th>Main Flow</th>
</tr>
</thead>
</table>
| 6  | Professor wants to give grade to student. | Professor | The professor is logged on to the system. | 1. The use case starts when a professor wants to give grade to a student.  
2. The system shows courses which the professor has got in his last term in a table with course ID/name, group ID, term  
3. The professor click on one of the above table rows and the list of students which took this course will be shown and this second table has student ID/name, group ID, and score with the ability to edit its score field.  
4. The admin enters score values to edit it.  
5. The system registers the admin’s given score and refreshes the table. |

<table>
<thead>
<tr>
<th>Postconditions</th>
<th>Alternative flows</th>
</tr>
</thead>
<tbody>
<tr>
<td>The score is given to the student in the system.</td>
<td>SQLException</td>
</tr>
</tbody>
</table>

### Use case: GradeDeclaration

<table>
<thead>
<tr>
<th>ID</th>
<th>Brief description</th>
<th>Actors</th>
<th>Preconditions</th>
<th>Main Flow</th>
</tr>
</thead>
</table>
| 7  | Student wants to see his grades. | Student | The student is logged on to the system. | 1. The use case starts when a student wants to see his grade.  
2. The system asks the student to enter the term which he wants to see his grades.  
3. The student enters the term.  
4. The system shows his courses which he has given in the term. It can be a table with course ID/name and score. |

---
Use case: GradeDeclaration

**Postconditions**
The score is visible to the student in the system.

**Alternative flows**
SQLException

---

Table 0.6: Use case: GradeDeclaration

---

Use case: UnitSelection

<table>
<thead>
<tr>
<th>ID</th>
<th>Brief description</th>
<th>Actors</th>
<th>Preconditions</th>
<th>Main Flow</th>
</tr>
</thead>
</table>
| 8  | Student wants to select his courses. | Student | The student is logged on to the system. | 1. The use case starts when a student wants to select his courses.  
2. The system shows courses which are given by the system in the last term in a table which would have course ID/name, professor name, group ID, term, and capacity.  
3. The student click on one of the above table rows and it will be added to his unit selection table.  
4. The student can delete its unit selection table rows too.  
5. The system update the student’s chosen courses and refreshes his unit selection table. |

<table>
<thead>
<tr>
<th>Postconditions</th>
<th>Alternative flows</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The course is given to the student in the system.</td>
</tr>
<tr>
<td></td>
<td>SQLException</td>
</tr>
</tbody>
</table>

---

Table 0.7: Use case: UnitSelection

---

Use case: LogonStudent

<table>
<thead>
<tr>
<th>ID</th>
<th>Brief description</th>
<th>Actors</th>
<th>Preconditions</th>
<th>Main Flow</th>
</tr>
</thead>
</table>
| 9  | Student authentication | Student | The student is not logged on to the system. | 1. The use case starts when a student wants to enter to the system.  
2. The system shows a login screen.  
3. The student enters his username and password.  
4. The system checks these two fields from their values in student’s table.  
5. The system logs student on the app. |

<table>
<thead>
<tr>
<th>Postconditions</th>
<th>Alternative flows</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The student is logged on to the system.</td>
</tr>
<tr>
<td></td>
<td>SQLException</td>
</tr>
<tr>
<td></td>
<td>LogonFailed</td>
</tr>
</tbody>
</table>

---

Table 0.8: Use case: LogonStudent

---

Use case: LogonAdmin

<table>
<thead>
<tr>
<th>ID</th>
<th>Brief description</th>
<th>Actors</th>
<th>Preconditions</th>
<th>Main Flow</th>
</tr>
</thead>
</table>
| 10 | AdminUser authentication | AdminUser | The AdminUser is not logged on to the system. | 1. The use case starts when an AdminUser wants to enter to the system.  
2. The system shows a login screen.  
3. The AdminUser enters his username and password.  
4. The system checks these two fields from their values in Prof’s table.  
5. The system logs AdminUser on the app. |

<table>
<thead>
<tr>
<th>Postconditions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The AdminUser is logged on to the system.</td>
</tr>
</tbody>
</table>
### Alternative flows: SQLException

<table>
<thead>
<tr>
<th>Alternative flows</th>
<th>Postconditions</th>
</tr>
</thead>
</table>
| 1. The alternative flow begins when a SQL exception happens. This can be a primary key constraint violation by the actor at any time.  
2. The system display a message with the name of the cause of exception and its message.  
3. The system rollbacks the change to the initial state. | The system is in its initial state. |

Table 0.10: Alternative flows: SQLException

### Alternative flows: LogonFailed

<table>
<thead>
<tr>
<th>Alternative flows</th>
<th>Postconditions</th>
</tr>
</thead>
</table>
| 1. The alternative flow begins when an actor authentication is failed.  
2. The system writes a failed info message.  
3. The system shows the login screen again. | The actor is not authenticated. |

Table 0.11: Alternative flows: LogonFailed
Linnaeus University
School of Computer Science, Physics and Mathematics

SE-391 82 Kalmar / SE-351 95 Växjö
Tel +46 (0)772-28 80 00
dfm@lnu.se
Lnu.se/dfm