# Design and Implementation of Business Logic Layer Object-Oriented Design versus Relational Design

Ali Alharthy
Faculty of Engineering and IT
University of Technology, Sydney
Sydney, Australia
Email: Ali.a.alharthy@student.uts.edu.au

*Abstract*—**Object-oriented programming has become one of the mainstream programming paradigms in software engineering, whereas relational models are predominant in commercial data processing applications. There is strong competition between these models for dominance in the building of modern applications, especially after the emergence and spread of object-relational mapping technology. This paper addresses the question of whether the object-oriented approach is better than the traditional approach in terms of flexibility with respect to changing requirements.**

***Keywords-object-oriented design; relational design; requirement changes; maintenance***

## I. INTRODUCTION

Currently, most business logic layers of modern applications are constructed using either an object-oriented model or a relational model. The object-oriented model is based on software engineering principles such as coupling, inheritance, cohesion, and encapsulation, whereas the relational model is based on predicate logic and set theory principles [1]. The object-oriented model chains the building of applications within objects that have both data and behavior. The relational model supports the storage of data in tables and the treatment of that data with data manipulation language within the database through stored procedures and externally through structured query language. The relational model is currently used in many database systems [1]. Object-oriented technology is also commonly used in database application development. The difference between the two technologies is called the object-relational impedance mismatch [2][3]. In particular, when objects need to be stored in a relational database, object-relational mapping (ORM) appears to play an important role in overcoming the problem of impedance mismatch. ORM is a new technology that allows applications to access relational data in an object-oriented manner [4][5]. With the widespread use of ORM technology, domain objects are built as objects, and the application logic manipulates these objects in a pure object-oriented manner. The critical issue that arises is whether such an object-oriented model for business logic layers is a good choice in general. Proponents of the object-oriented approach have tended to assume that an object-oriented business model will make the system easier to maintain, easier to extend, and easier to reuse.

The object-oriented approach has been advocated as a tool for improving developer productivity and software quality [6][7]. Moreover, it has been suggested that development using object-oriented programming enhances productivity by simplifying understandability, program design, and maintenance in comparison to traditional approaches [8].These studies have maintained that using the object-oriented approach would help reduce the maintenance cost of software. However, there are few complete experimental results that support the claim that there is an advantage in the maintainability of programs developed with the object-oriented approach over those developed with traditional approaches [7][9].

The objective of this paper is to extend this body of knowledge by critically examining this assumption and to carefully compare the applicability and flexibility of the object-oriented system to those of the relational system. The findings from this project will be significant for practical applications in which the business logic layer is implemented in an object-oriented fashion, which is a growing trend in enterprise computing.

The rest of the paper is organised as follows. Section II presents the motivation for the study. Section III outlines the investigation method. Sections IV, V, VI, and VII present the case studies, and Section VIII reports the experimental results. Section IX concludes the paper.

## II. MOTIVATION

Today, changing requirements have become a fact of life for software developers. Many studies have shown that changes in software were one of the reasons why various projects failed. For example, a study by the Standish Group found that only 37% of information technology projects are considered successes and that 21% of projects are considered failures [10]. The remaining 42% are considered ‘challenged’—defined as late, over budget, or having failed to meet expectations. Requirement changes are the major cause of this phenomenon. Such changes can occur during the development and maintenance phase in order to accommodate user and business requirements. Therefore, there is a need to identify a flexible approach that can deal with requirement changes.

However, ORM is very popular and widely used. According to Russell [3], in order to access data stored in relational databases, most modern applications are built using ORM technology rather than the traditional approach.

 ISBN: 978-1-61208-438-1 

It has also been argued that using ORM tools can help reduce project costs. Moreover, proponents of the object-oriented approach have tended to assume that an object-oriented business model will make the system easier to maintain, easier to extend, and easier to reuse. On the other hand, proponents of the traditional approach have argued that not all the world must be handled in objects. In addition, they have maintained that there is some native incompatibility between ORM code and databases. They also maintain that although object-oriented development promises to reduce maintenance effort, these promises are not based on reliable experimentation [11]. Indeed, there is a significant lack of research on whether the object-oriented approach is better than the traditional approach in terms of flexibility in the face of requirement changes.

## III. INVESTIGATION METHOD

The investigation is performed using a number of case studies and by introducing a variety of requirement changes in order to evaluate how the two approaches cope with them. For the implementation, we used Java Database Connectivity (JDBC), a representative relational system, and Hibernate, a representative ORM framework, as well as MYSQL, a relational database. All of these are popular open-source products. In order to measure the overall implementation effort associated with JDBC and Hibernate due to new/changed requirements, we used the code size produced in the completion of a task—the code size was measured in lines of code and takes into account lines added, modified, and deleted—as well as the time required to complete a task. To measure the code size, we used a free tool to compare the source code files after each implementation. The case studies implementation has been done by a developer who has six years experience in Web and Database applications development.

## IV. FIRST CASE STUDY

We chose a simple case study to make an initial comparison of the effort involved in implementing the two technological approaches and changing them in response to requirement changes.

### A. Problem statement

A company requires a Car Park application to maintain information about employees and their parking permits. The car park has a number of parking spots, which are divided into three areas: A, B, and C. Employees who want a permit have to pay a fee on a quarterly basis, which will be automatically deducted from their salary. The purpose of the Car Park application is to help the car park manager process the employees' applications for parking permits. Each employee has an ID, a name, and a phone extension. Each permit has a permit number, the car's registration number, and the section where the car can be parked. An employee can have at most two permits. Employees may change their extension in the course of their employment. When employees get a new car and want to use it instead of the old one, they have to discontinue the current permit and apply for a new one.

### B. Comparison of the findings of the initial construction of the two approaches

TABLE I. FINDINGS OF THE INITIAL CONSTRUCTION

| Program | Files | SLOC | Total/ Lines | ET |
|---|---|---|---|---|
| Hibernate | CPSystem.java | 141 | 251 | 4 h 30 min |
| | Permit.java | 47 | | |
| | Employee.java | 47 | | |
| | HibernateUtil.java | 16 | | |
| | Employee.hbm.xml | 17 | 49 | |
| | Permit.hbm.xml | 14 | | |
| | Hibernate.cfg.xml | 18 | | |
| | Total | 300 | | |
| JDBC | CPSystem.java | 283 | 283 | 3 h |

Table I summarises the findings of the initial construction of the Car Park system using the two approaches. The table shows that even though there are no significant differences between the two approaches with respect to the effort measured by size of source code, the Hibernate approach took more time than the JDBC approach. In fact, with Hibernate we had to deal with six files, whereas with JDBC we had to deal with only one file. Therefore, the Hibernate approach took about 4.3 h, compared to 3 h for JDBC.

### C. Impact of requirement changes on the two approaches

Because requirements change frequently in practice, it is useful to see how different approaches cope with requirement changes. For the initial investigation regarding requirement changes, we made the following change: in the Terminate Permit use case, instead of deleting the permit (as we did before), we labelled the permit as terminated.

### D. Comparison of findings after first requirement change

TABLE II. FIRST REQUIREMENT CHANGE

| Program | Files | V1 | V2 | A | M | D | S | ET |
|---|---|---|---|---|---|---|---|---|
| Hibernate | CPS.java | 141 | 149 | 10 | 4 | 2 | 16 | 40 min |
| | Permit.java | 47 | 65 | 8 | 0 | 0 | 8 | |
| | Emp.java | 47 | 47 | 0 | 0 | 0 | 0 | |
| | Emp.hbm.xml | 17 | 17 | 0 | 0 | 0 | 0 | |
| | Perm.hbm.xml | 14 | 15 | 1 | 0 | 0 | 1 | |
| | Hiber.cfg.xml | 18 | 18 | 0 | 0 | 0 | 0 | |
| | HiberUtil.java | 16 | 16 | 0 | 0 | 0 | 0 | |
| | Total | 300 | 327 | 19 | 4 | 2 | 25 | |
| JDBC | CPSy.java | 283 | 283 | 0 | 3 | 0 | 3 | 10 min |

V1 = before the change; V2 = after the change; A = add; M = modify; D = delete; S = summation of A,M, and D; ET = estimated time

 ISBN: 978-1-61208-438-1 

As Table II shows, there are significant differences between the two approaches with respect to the implementation effort measured by the size of the source code. The implementation of the new requirement changes with Hibernate required a total of 25 lines of code, compared to only 3 lines of code using JDBC. In addition, the implementation of the new requirement changes with Hibernate took about 40 min, whereas it took only 3 min with JDBC. Indeed, it is evident that JDBC offered more flexibility with regard to both time and effort.

### E. Further impact of requirement changes on the two approaches

For the second requirement change, suppose a company needs to distinguish between full-time and part-time employees. Part-time employees are paid an hourly rate, whereas full-time employees are assigned a salary.

TABLE III. SECOND REQUIREMENT CHANGE

| Program | Files | V1 | V2 | A | M | D | S | ET |
|---|---|---|---|---|---|---|---|---|
| Hibernate | CPS.java | 149 | 154 | 5 | 2 | 0 | 7 | 1 h |
| | Permit.java | 65 | 65 | 0 | 0 | 0 | 0 | |
| | Emp.java | 47 | 47 | 0 | 1 | 0 | 1 | |
| | PartTime.java | - | 20 | 20 | 0 | 0 | 20 | |
| | FullTime.java | - | 20 | 20 | 0 | 0 | 20 | |
| | Emp.hbm.xml | 16 | 16 | 0 | 0 | 0 | 0 | |
| | Perm.hbm.xml | 17 | 24 | 7 | 0 | 0 | 7 | |
| | Hiber.cfg.xml | 15 | 15 | 0 | 0 | 0 | 0 | |
| | HiberUtil.java | 18 | 18 | 0 | 0 | 0 | 0 | |
| | Total | 327 | 379 | 52 | 3 | 0 | 55 | |
| JDBC | CPSy.java | 283 | 296 | 13 | 3 | 0 | 16 | 20 min |

As Table III shows, the new requirements have had a greater impact on the program implemented through Hibernate, in terms of both the time and the effort required to implement these changes. The implementation of the new requirement changes with Hibernate required a total of 55 lines of code, in contrast to JDBC, which required only 16 lines. This difference represents a nearly 3:1 ratio in quantity of code. Although one of the key benefits of inheritance is minimising the amount of duplicate code in an application by sharing common code amongst several subclasses, the majority of new code is due to inheritance code. Moreover, the implementation with Hibernate took about 1 h, compared to only 20 m using JDBC. As a result, increasing the number of classes that need to be persisted automatically can lead to increased levels of effort and time.

## V. SECOND CASE STUDY

We made the second case study more complicated than the first in order to produce more statistics with which to compare the two approaches. We also made changes that reflect the change in business policy, that is, allowing more than one kind of item to be stored at a shelf location. This change in policy required a change in the structure of the classes. It will provide more data with which to compare the two approaches.

### F. Problem statement

A database is needed to maintain information about the items stored in various warehouses of a company. Design a relational database, which can store the information contained the following:

1. Each warehouse has a phone (not shown) to contact the staff at the warehouse.
2. Shelf locations are of two types: single access and double access.
3. The present policies require that each shelf location, at any time, can be used to store only one kind of item.

TABLE IV. FINDINGS OF THE INITIAL CONSTRUCTION

| Program | Files | SLOC | Total/ Lines | ET |
|---|---|---|---|---|
| Hibernate | PartInWareHouse.java | 141 | 300 | 4 h |
| | Part.java | 30 | | |
| | Warehouse.java | 31 | | |
| | ShelfLocation.java | 45 | | |
| | ShelfLocationPK.java | 37 | | |
| | HibernateUtil.java | 16 | | |
| | Warehouse.hbm.xml | 12 | 58 | |
| | Part.hbm.xml | 13 | | |
| | ShelfLocation.hbm.xml | 15 | | |
| | Hibernate.cfg.xml | 18 | | |
| | Total | 358 | | |
| JDBC | PartInWareHouse.java | 202 | | 2.3 h |

Table IV summarises the findings for implementing the Parts in Warehouses system with the two approaches. Hibernate required a total of 358 lines of code, in contrast to JDBC, which required 202 lines. In addition, Hibernate required about 4 h, whereas JDBC required 2.3 h. Hibernate clearly required more effort and time than JDBC.

### G. Impact of requirement changes on the two approaches

The storage rules change to allow more than one kind of item to be stored at a shelf location. This entails that the cardinality relationship between the two entities Shelf Location and Items must be changed to one-to-many.

### H. Comparison of the findings after first requirement change

As shown in Table V, the new requirements have had a greater impact on the program implemented through Hibernate, in terms of both the time and the effort required to implement these changes. The implementation of the new requirement changes with Hibernate required a total of 139 lines of code, in contrast to JDBC, which required only 38. This difference represents a nearly 4:1 ratio in quantity of code. Indeed, the source of increase in code quantity was due to the addition of an item class with its composite key, which

is not necessary in JDBC. Moreover, the implementation with Hibernate took about 1.30 h, compared to only 30 min with JDBC.

TABLE V. FIRST REQUIREMENT CHANGE

| Program | Files | V1 | V2 | A | M | D | S | ET |
|---|---|---|---|---|---|---|---|---|
| Hibernate | WHouse.java | 141 | 155 | 14 | 5 | 0 | 19 | 1.30 h |
| | Part.java | 30 | 30 | 0 | 0 | 0 | 0 | |
| | Whouse.java | 31 | 31 | 0 | 0 | 0 | 0 | |
| | SLoc.java | 45 | 32 | 2 | 6 | 15 | 23 | |
| | SLocPK.java | 37 | 37 | 0 | 0 | 0 | 0 | |
| | Item.java | - | 29 | 29 | 0 | 0 | 29 | |
| | ItemPK.java | - | 37 | 37 | 0 | 0 | 37 | |
| | HibUtil.java | 16 | 16 | 0 | 0 | 0 | 0 | |
| | Who.hbm.xml | 12 | 12 | 0 | 0 | 0 | 0 | |
| | Part.hbm.xml | 13 | 19 | 6 | 0 | 0 | 6 | |
| | SLo.hbm.xml | 15 | 20 | 5 | 2 | 1 | 8 | |
| | Item.hbm.xml | - | 16 | 16 | 0 | 0 | 16 | |
| | Hiber.cfg.xml | 18 | 19 | 1 | 0 | 0 | 1 | |
| | Total | 358 | 453 | 110 | 13 | 16 | 139 | |
| JDBC | WHouse.java | 202 | 218 | 16 | 20 | 2 | 38 | 40 min |

## VI. THIRD CASE STUDY: ISSUE OF RELATIONAL REPRESENTATION/NAVIGATION

The representation of the relationship is a fundamental issue. In fact, the difference between hierarchy, network, relational, and object-oriented databases is the way in which the relationship is represented. Therefore, if we construct the application with JDBC, we will not experience the navigation problem, whereas the problem arises when the application is constructed with ORM. Thus, we have to decide how to represent the navigation objects.

### *I. Problem statement*

A distribution company supplies various kinds of products to customers on a daily basis according to the standing orders placed by the customers. The company wants to set up a system to maintain information about the products that the company can supply, its customers, and the standing orders.

### *J. Comparison of the findings of the initial construction of the two approaches*

Table VI summarises the findings for implementing the Standing Order system with the two approaches. Hibernate required a total of 268 lines of code, in contrast to JDBC, which required 141. In addition, Hibernate required about 3 h, whereas JDBC required 2 h. Thus, Hibernate required more effort and time than JDBC.

TABLE VI. FINDINGS OF THE INITIAL CONSTRUCTION

| Program | Files | SLOC | Total/ Lines | ET |
|---|---|---|---|---|
| Hibernate | SOSystem.java | 86 | 212 | 3 h |
| | Customer.java | 22 | | |
| | Order.java | 47 | | |
| | Product.java | 41 | | |
| | HibernateUtil.java | 16 | | |
| | Customer.hbm.xml | 10 | 56 | |
| | Order.hbm.xml | 15 | | |
| | Product.hbm.xml | 12 | | |
| | Hibernate.cfg.xml | 19 | | |
| | Total | 268 | | |
| JDBC | SOSystem.java | 141 | 141 | 2 h |

### *K. Impact of requirement changes on the two approaches*

We changed the navigation rule between the objects from unidirectional to bidirectional association.

TABLE VII. FINDINGS OF THE INITIAL CONSTRUCTION

| Program | File Name | V1 | V2 | A | M | D | S | ET |
|---|---|---|---|---|---|---|---|---|
| Hibernate | SOSys.java | 86 | 86 | 0 | 0 | 0 | 0 | 30 min |
| | Cust.java | 22 | 32 | 10 | 0 | 0 | 10 | |
| | Order.java | 47 | 47 | 0 | 0 | 0 | 0 | |
| | Product.java | 41 | 51 | 10 | 0 | 0 | 10 | |
| | Htil.java | 16 | 16 | 0 | 0 | 0 | 0 | |
| | Cu.hbm.xml | 10 | 14 | 4 | 0 | 0 | 4 | |
| | Or.hbm.xml | 15 | 15 | 0 | 0 | 0 | 0 | |
| | Pr.hbm.xml | 12 | 16 | 4 | 0 | 0 | 4 | |
| | Hib.cfg.xml | 19 | 19 | 0 | 0 | 0 | 0 | |
| | Total | 268 | 296 | 28 | 0 | 0 | 28 | |
| JDBC | SOSys.java | 141 | 141 | 0 | 0 | 0 | 0 | 0 |

As Table VII shows, the new requirements have had a greater impact on the program implemented through Hibernate, in terms of both the time and the effort required to implement these changes. The implementation of the new requirement changes with Hibernate required a total of 28 lines of code and 30 min, in contrast JDBC, which did not require any changes, because navigation is not an issue for it.

## VII. FOURTH CASE STUDY

We made this case study even more complicated and realistic in order to produce much more statistical data with which to compare the two approaches. The case study also highlights the issue of relationship representation and illustrates that the object-oriented approach is more sensitive to the class model than the relational model.

 ISBN: 978-1-61208-438-1 

### L. Problem statement

Eastern Suburb Gymnastics (ESG) is a regional organisation that is responsible for running competitions between the gymnastics clubs in eastern suburbs of Melbourne. The competitions are organised into seasons. ESG needs a system to help organise and maintain the records of the competitions that take place in a single season. The system, in essence, needs to store information on the gymnasts, their clubs, the organisation of the competitions, and the competition results.

### M. Comparison of the findings of the initial construction of the two approaches

TABLE VIII. FINDINGS OF THE INITIAL CONSTRUCTION

| Program | File | SLOC | Total/ Lines | ET |
|---|---|---|---|---|
| Hibernate | GScoringSystem | 237 | 810 | 6 h |
| | Club | 44 | | |
| | Competition | 24 | | |
| | CompetitionPk | 35 | | |
| | Division | 60 | | |
| | EventPk | 46 | | |
| | Event | 37 | | |
| | EventType | 58 | | |
| | Gymnast | 60 | | |
| | Judge | 40 | | |
| | Meet | 52 | | |
| | TeamPk | 46 | | |
| | Team | 33 | | |
| | Score | 22 | | |
| | HibernateUtil | 16 | | |
| | Club.hbm.xml | 13 | 177 | |
| | Competition.hbm.xml | 12 | | |
| | Division.hbm.xml | 15 | | |
| | Event.hbm.xml | 21 | | |
| | EventType.hbm.xml | 14 | | |
| | Gymnast.hbm.xml | 15 | | |
| | Judge.hbm.xml | 17 | | |
| | Meet.hbm.xml | 14 | | |
| | Team.hbm.xml | 14 | | |
| | Score.hbm.xml | 16 | | |
| | Hibernate.cfg.xml | 26 | | |
| | Total | 987 | | |
| JDBC | GScoringSystem | 259 | 259 | 3 h |

Table VIII summarises the findings for implementing the Eastern Suburb Gymnastics system with the two approaches. Hibernate required a total of 987 lines of code, in contrast to JDBC, which required 259. In addition, Hibernate required about 6 h, whereas JDBC required 3 h. It is evident that Hibernate required more effort and time than JDBC. This difference represents a nearly 4:1 ratio in quantity of code. Indeed, the source of the increase in the code quantity was due to a plain old Java objects (POJO) and its mapping files.

### N. Impact of requirement changes on the two approaches

Here, we investigated how sensitive the two approaches are to the choice of domains modelled.

TABLE IX. FINDINGS OF THE INITIAL CONSTRUCTION

| Program | File Name | V1 | V2 | A | M | D | S | ET |
|---|---|---|---|---|---|---|---|---|
| Hibernate | GSSystem | 237 | 274 | 37 | 0 | 0 | 37 | 1 h |
| | Club | 44 | 44 | 0 | 0 | 0 | 0 | |
| | Competition | 24 | 24 | 0 | 0 | 0 | 0 | |
| | CompetitionPk | 35 | 35 | 0 | 0 | 0 | 0 | |
| | Division | 60 | 60 | 0 | 0 | 0 | 0 | |
| | EventPk | 46 | 46 | 0 | 0 | 0 | 0 | |
| | Event | 37 | 37 | 0 | 0 | 0 | 0 | |
| | EventType | 58 | 58 | 0 | 0 | 0 | 0 | |
| | Gymnast | 60 | 60 | 0 | 0 | 0 | 0 | |
| | Judge | 40 | 40 | 0 | 0 | 0 | 0 | |
| | Meet | 52 | 52 | 0 | 0 | 0 | 0 | |
| | TeamPk | 46 | 46 | 0 | 0 | 0 | 0 | |
| | Team | 33 | 33 | 0 | 0 | 0 | 0 | |
| | TeamMember | - | 55 | 55 | 0 | 0 | 55 | |
| | Score | 22 | 22 | 0 | 0 | 0 | 0 | |
| | HibernateUtil | 16 | 16 | 0 | 0 | 0 | 0 | |
| | Club.hbm.xml | 13 | 13 | 0 | 0 | 0 | 0 | |
| | Comp.hbm.xml | 12 | 12 | 0 | 0 | 0 | 0 | |
| | Divis.hbm.xml | 15 | 15 | 0 | 0 | 0 | 0 | |
| | Event.hbm.xml | 21 | 21 | 0 | 0 | 0 | 0 | |
| | EType.hbm.xml | 14 | 14 | 0 | 0 | 0 | 0 | |
| | Gymt.hbm.xml | 15 | 15 | 0 | 0 | 0 | 0 | |
| | Judge.hbm.xml | 17 | 17 | 0 | 0 | 0 | 0 | |
| | Meet.hbm.xml | 14 | 14 | 0 | 0 | 0 | 0 | |
| | Team.hbm.xml | 14 | 14 | 0 | 0 | 0 | 0 | |
| | TMember..xml | - | 14 | 14 | 0 | 0 | 14 | |
| | Score.hbm.xml | 16 | 16 | 0 | 0 | 0 | 0 | |
| | Hibern.cfg.xml | 26 | 27 | 1 | 0 | 0 | 1 | |
| | Total | 987 | 1093 | 106 | 0 | 0 | 106 | |
| JDBC | GSSystem | 259 | 291 | 32 | 0 | 0 | 32 | 25 min |

Table IX shows that the implementation of the new requirement changes with Hibernate required a total of 106 lines of code, in contrast to JDBC, which required only 32. This difference represents a nearly 3:1 ratio in quantity of code. Moreover, the implementation with Hibernate took about 1 h, compared to only 25 min for JDBC.

## VIII. RESULTS

The results of our critical comparison of the two paradigms in terms of flexibility, which was based on implementation findings, indicate that in the initial construction of the application, using ORM is much costlier than using JDBC. In other words, the level of effort and time required to implement the application is much higher with Hibernate than with JDBC. For instance, the initial construction of the ESG system with ORM required a total of 987 lines of code, in contrast to JDBC, which required 259. This difference represents a nearly 4:1 ratio in quantity

 ISBN: 978-1-61208-438-1 

of code. In addition, ORM required about 6 h, whereas JDBC required only 3 h. Indeed, increasing the number of classes that need to be persisted automatically can lead to increased levels of effort and time.

Moreover, JDBC is more flexible in the face of requirement changes than is ORM. For example, for an object to be persisted to a database, Hibernate needs a mapping file for all the objects that are to be persisted as well as POJO, which is not required when using the JDBC approach. This means that if we would like to add an attribute to or delete an attribute from a class, we must modify the mapping file of that class to map or delete the attribute, and subsequently we must modify the class itself to add/delete that an attribute with its getter and setter methods. When using JDBC, in contrast, we do not need to undertake these steps. Furthermore, the object-oriented paradigm has an issue related to navigation between objects through association links, whereas navigation is not an issue for JDBC. In addition, determining the direction with UML is not an easy task, which can be considered one of the common mistakes in design decision. In addition, the object-oriented approach is more sensitive to the class model than the relational model. It is worth mentioning that the developer did not use auto-code generation during performing the initial construction implementation, and this could explain the remarkable difference in time between two approaches.

Although the current study has yielded some clear preliminary findings, its design is not without flaws. First, the case studies were small scale as a result of some restrictions, such as the time and effort required for implementation. A further limitation is that the implementation of all the case studies was performed by one developer, which may affect the generalisability of the study's findings to different developers.

## IX. CONCLUSION

This paper addressed the question of whether the object-oriented approach is better than the traditional approach or vice versa in terms of applicability and flexibility to requirement changes. The experimental results show that the object-oriented approach required more time and effort as a result of mapping files. Moreover, the object-oriented approach has an issue of navigation between objects. However, our examination is only the beginning. We believe there is still a need for further research with real projects to yield reliable results. Our future work will focus on conducting more experiments on real projects to validate our results and to investigate flexibility of object-oriented approach to requirement changes.

## REFERENCES

[1] E. F. Codd, "A Relational Model of Data for Large Shared Data Banks," Communications of the ACM, vol. 13, 1970, pp. 377-387.

[2] B. Unger, L. Prechelt, and M. Philippsen, The Impact of Inheritance Depth on Maintenance Tasks: Detailed Description and Evaluation of Two Experiment Replications. Fak. für Informatik Univ., 1998.

[3] C. Russell, "Bridging the Object-relational Divide," *Queue*, vol. 6, 2008, pp. 18-28.

[4] M. I. Aguirre-Urreta and G. M. Marakas, "Comparing Conceptual Modeling Techniques: A Critical Review of the EER vs. OO Empirical Literature," ACM SIGMIS Database, vol. 39, 2008, pp. 9-32.

[5] F. Lodhi and M. A. Ghazali, "Design of a Simple and Effective Object-to-Relational Mapping Technique," in Proceedings of the 2007 ACM Symposium on Applied Computing, 2007, pp. 1445-1449.

[6] S. Sircar, S. P. Nerur, and R. Mahapatra, "Revolution or Evolution? A Comparison of Object-oriented and Structured Systems Development Methods," MIS Quarterly, 2001, pp. 457-471.

[7] G. A. Kiran, S. Haripriya, and P. Jalote, "Effect of object orientation on maintainability of software," in Software Maintenance, 1997. Proc. International Conference on, 1997, pp. 114-121.

[8] M. B. Rosson and S. R. Alpert, "The Cognitive Consequences of Object-oriented Design," Human-Computer Interaction, vol. 5, 1991, pp. 345-379.

[9] M. A. Eierman and M. T. Dishaw, "The Process of Software Maintenance: A Comparison of Object-oriented and Third-generation Development Languages," Journal of Software Maintenance and Evolution: Research and Practice, vol. 19, 2007, pp. 33-47.

[10] S. Group. (2011). The Standish Group International Inc.

[11] E. Arisholm and D. I. Sjoberg, "Evaluating the Effect of a Delegated Versus Centralized Control Style on the Maintainability of Object-oriented Software," IEEE Transactions on Software Engineering, vol. 30, 2004, pp. 521-534.

 ISBN: 978-1-61208-438-1 