Implementation of Test-Driven Development in Data Access Layer within a Business System Development

Yunarso Anang[1,4], Masakazu Takahashi[2] and Yoshimichi Watanabe[3]

[1] University of Yamanashi, Kofu, JSQC, g14dma01@yamanashi.ac.jp
[2] University of Yamanashi, Kofu, mtakahashi@yamanashi.ac.jp
[3] University of Yamanashi, Kofu, JSQC, nabe@yamanashi.ac.jp
[4] Institute of Statistics, Jakarta, anang@stis.ac.id

Abstract: Test-driven development (TDD) is a software development practice which enables developers to design the software so the codes are loosely coupled and testable thus have high quality in the term of maintainability. However, in a system involving expensive and complex resource such as database access in a typical business system, the implementation of TDD is not necessarily easy and straightforward. In this paper, we describe our approach of implementing TDD in the data access layer within a business system development. Our approach differs from those have already been proposed and practiced where the mock objects are used to fake the database access. We designed a data access adapter which works as an adapter for the real database connectivity while still being fast enough for a regular unit test. In order to evaluate our approach, we applied it in a real business system development project. We are utilizing software versioning system in order to measure the effort made in response to the bug report as well as the customer requirement for modification after the delivery of the software product. While the time for development is increased, we successfully confirm the effectiveness of the implementation of TDD compared to non-TDD.

Keywords: code maintainability, code quality, low coupled code.

1 Introduction

Test Driven Development (TDD) is a software development practice, where instead of creating the test after the development, the test (preferably automated) is created before starting the development of the production code (Beck, 2008). TDD is not only a testing technique, but rather a design approach and has thus not only influence on the external quality of a product, but also on the internal code quality (Wels, 2012). By practicing TDD, the developer is encouraged to write only the necessary functional codes to make the test pass and performs any necessary code refactoring. However, as mentioned in (Beck, 2008), to test an object which relies on an expensive or complicated resource, such as a database, a Mock Object (Mackinnon, et al., 2001), a fake version of the resource, is used. In order to implement TDD in the development of a business system, which typically relies on the database, the appropriate design of the system should be considered accordingly.

In this paper, we describe how we implement TDD in the data access layer within a business system development. In our implementation, we applied the concept of layered architecture (Fowler, et al., 2002) and Ports and Adapters pattern also known as Hexagonal architecture (Cockburn, 2005). Our approach enables developers to practice TDD and write testable codes in all layers of the software. Also, because we designed an adapter which provides access to the underlying database, the developer can be sure the code he/she wrote will work not only in the "mock" world but in the "real" world, where the code needs to connect to the database.

In order to evaluate our approach, we apply our approach to the development of a real business system development project. We evaluate the effectiveness of our approach by monitoring the feedback from the customer after the software has been delivered to the customer. By utilizing software versioning system, we also measure the developer's effort to perform any modification to the codes in order to answer to the customer bug report as well as the requirement for modification.

The rest of this paper is organized as follows. Section 2 describes related studies. Section 3 describes the proposed framework used to apply TDD combined with Ports and Adapters architecture in the business system development. Section 4 describes the steps to perform the development using the proposed framework. Section 5 describes the case study we conducted to evaluate the proposed method. Section 6 discusses the result of this study. Finally, we conclude this paper in Section 7.

2 Related Studies

TDD is a software development practice which applies the following two simple rules:

- Write a failing automated test before you write any code.
- Remove duplication.

The developer only writes any necessary code to make the test code pass. And because the TDD technique is done in a small increment change, it leads to the excessive tasks of code rewriting and refactoring. However, by practicing TDD, the developer is encouraged to write only the necessary functional codes and performs any necessary code refactoring.

The effectiveness of TDD has been studied by Dogša et al. (Dogša & Batič, 2011) targeted on an industrial case study. The paper stated that the TDD developers produced higher quality code that is easier to maintain, although the decrease in productivity has been observed. The maintainability is considerably better because unit-test provides a safety net when the source is changed, and writing code as is really needed leads to better design of code with lower complexity. As for the lower productivity, the mindset to acquiring the TDD strictly seems to be the reason. George et al. reported that 16% more time is taken for development using TDD (George & Williams, 2003). The test-first approach requires additional and careful preparation of the test unit.

Bhat et al. also studied evaluating the efficacy of the TDD with industrial case studies (Bhat & Nagappan, 2006). The study observed a significant increase in quality of the code developed using TDD compared to non-TDD fashion. While the study observed an extra upfront time for writing the tests, they stated that the productivity of the team was not impacted by the additional focus on producing automated test cases.

As part of the evaluation of TDD, Wasmun et.al has conducted a case study including the test of the database (Wasmus & Gross, 2007). However, they reported that their implementation of the test of the database can be a time-consuming procedure. From the paper, it was not clear, how they perform TDD for the database access. Ryu et.al proposed a framework of the mock object for TDD which is claimed to be able to lessen the period and save the cost in making the mock object (Ryu, et al., 2005). The framework can also be applied to a database environment by utilizing mock objects. The concept is applicable but still need more consideration to our need, which is to provide a development environment reliable to the real-world system. Ou wrote a guide for practicing TDD in database development (Ou, 2003). The author used an object-relational mapping tool to connect to the local development database as the target for database connectivity and access test. The author claimed that testing against local development database enables faster coding of database access development. It might be true, but relying on a tool for generic database access may not be reliable due to the variations not only in the vendors of the database but also in their versions.

The Ports and Adapters, or also referred to as Hexagonal, is an architecture pattern which intends to allow an application to equally be driven by users, programs, and automated test to batch scripts, and to be developed and tested in isolation from its eventual run-time devices and databases (Cockburn, 2005). Figure 1 shows an application having two active ports, those are user-side and data-side, with several adapters for each port (added file adapter to the original figure). As events arrive from the outside world at a port, a technology-specific adapter converts it into a usable procedure call or message and passes it to the application. The application is blissfully ignorant of the nature of the input device.

Łaskawiec et al. analyzed several software architectures used in Java, which also compared the Ports and Adapters architecture and the layered architecture (Łaskawiec, 2016). For the layered architecture, because it makes developers focus on the technical side of the application, it often shadows business goals of the application and introduces a lot of technical details into the presentation layer making the application less intuitive for the business partner (e.g. outsourced developers). In the other hand, the Ports and Adapters architecture is focused on solving the business problems and dedicating an adapter for each client. Vernon states that the Ports and Adapters architecture considers the domain model as the heart of the software architecture (Vernon, 2013). Ports and Adapters architecture can be used to host a bounded context application and to facilitate other styles such as service-oriented (Microsoft), REST (representational state transfer) (Fielding, 2000), event-driven, and others.

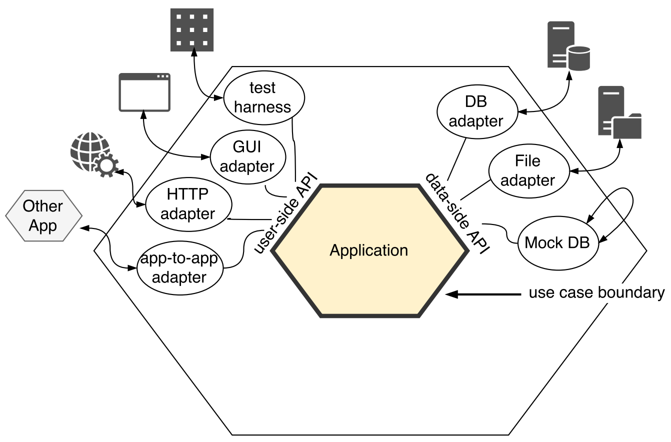


Figure 1 Ports and adapter architecture

In this paper, the Ports and Adapters architecture is used to construct the framework to apply the TDD in the development of a business system which typically involves a database access.

3 The Proposed Framework

TDD is a development practice where the developer starts developing the program by first writing the test program. Then, the developer can develop the program, starting with minimal implementation just to make the test program pass. Finally, the developer can refactor the program he/she to remove duplications. These three steps should be repeated for each software functionality to be implemented. The key which makes the TDD success is the automated test.

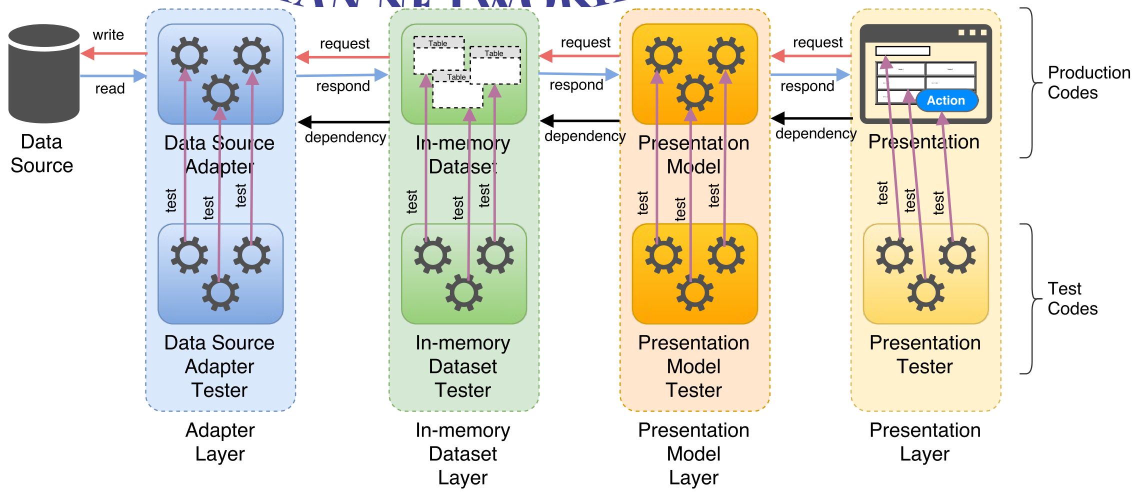


Figure 2 Proposed framework conceptual diagram

In a typical business system, the main functions appear in the user interface. The approach of the proposed method of implementing TDD starts from the user interface. There, the developer starts writing the code by making the test code for each information to be shown to the user interface, or to be received from the user interaction. These test codes should be written independently from the application to be developed, so they can be executed (tested) without first launching the real application. The production code to be implemented, typically, will access a database, sending a request to the database, retrieving data from the database to be logically processed, and give back the result to the user. The concept of layered architecture combined with Ports and Adapters architecture is applied to ensure the TDD is appropriately implemented in each layer. In the data access layer, special attention has been paid in designing the data access adapter, to provide a reliable testing mechanism to real database connectivity. Figure 2 shows the conceptual diagram of the proposed framework.

First, the framework is divided into two big groups of codes: the production codes and the test codes. The production codes are the codes of which the software product uses in production when the end-user uses the product. All of the compiled version of the production codes is delivered to the customer. The test codes are the codes of which used by the developer to test the functionality of the production codes. The compiled version of the test codes is not delivered to the customer. They are completely separated and independent from the production codes. It means that to run the production codes, the test codes are not necessary.

Second, the production codes are divided into 5 layers: Presentation, Presentation Model, In-memory Dataset, Adapter, and Data Source. The followings are the detailed description for each of those layers.

3.1 Presentation

The Presentation is where the end-user interacts with the application. A typical example is a Form with graphical user interface. In a typical .NET Framework[1] desktop application developed using the Visual Studio[2] integrated development environment (IDE), a Form is composed of the designer code and the controller code. Both the designer and the controller codes are parts of the same class.

The designer code: It contains the description of the class constructor and graphical control contained in a Form including the type, name, attributes, and also the declaration of the event handler. The codes written in the designer code are automatically generated by the IDE.

The controller code: It contains the description of the class field and method, event handler, and other codes written by the developer. The developer will write a reference to the attached Presentation Model class instance.

GUI Form is resource expensive and is difficult to conduct an automated test. The developer should avoid writing code for domain specific logic, such as codes for data validation in this layer. The developer should only write codes for data validation or domain specific logic in the Presentation Model. Figure 2 shows a test for the Presentation, but the description is not covered in this paper.

Other than a Form with graphical user interface, the Presentation can be just a console, or other forms of output, such as a report (e.g. for print or pdf), a file (e.g. CSV or spreadsheet), an email, or a sound.

3.2 Presentation Model

The Presentation Model is where the developer writes the codes for the domain logic of the application, particularly for the Presentation to which it is responsible. The domain logic includes the data retrieval, manipulation, and validation, or the logical response to the event in the Presentation. The production code of the class representing the Presentation Model layer should be declared as public, and all members intended to be tested also need to be declared as public.

The code in Presentation Model sends requests to the Adapter to receive data from the data source, and receives the response in the form of In-memory Dataset. The In-memory Dataset then can be processed for Presentation.

The Presentation Model is also responsible for performing domain logic, such as for data validation. For example, in an authentication form in the Presentation layer, when it needs to authenticate user and password, then the actual data validation for the authentication is written in the Presentation Model, not

[1] .NET Framework: A framework for running application, originally developed by Microsoft for Windows platform.
[2] Visual Studio: An integrated development environment (IDE) developed by Microsoft.

in the Presentation. This allows the developer to test the authentication code without the need to launch the application, thus allow the TDD to be conducted.

Typically, one class of Presentation Model is only responsible to one Presentation class. Codes in this Presentation Model can refer to one or more codes from the lower layers.

3.3 In-memory Dataset

In-memory Dataset is a container for the in-memory data tables representing data retrieved from the Data Source, such as tables or query results from databases, or files from disk. It could be a set of data items or just one scalar value.

The in-memory data tables contain disconnected data retrieved from the database, which means, once the data have been retrieved from the data source, any modification to the in-memory data table is not populated to the data source automatically. And in the same manner, once the data have been retrieved from the database, any update in the data source is not populated to the in-memory data tables automatically.

In-memory Dataset can be used to store data retrieved from the Data Source. Or, it can be used as a mock object of the real data. In that case, the data is generated by the developer in the test code.

3.4 Adapter

The Adapter is where the developer writes codes to access the Data Source and to give feedback to the application. Applying the concept from the Ports and Adapters architecture, the Adapter is technology-specific to the Data Source, when sending the request to (or retrieving the response from) the Data Source. However, the feedback to the application should no longer be technology-specific. Any data item, type, and value should be generic to the application.

For example, when the application is using .NET Framework and wants to retrieve data from a SQL Server database, then all types of data from the SQL Server database should be converted to the generic .NET Framework data types. In other cases, when retrieving data from an Oracle database, then all types of data should also be converted to the generic .NET Framework data types. This allows the developer to handle data with the same generic type equally, also to exchange the data no matter where the data came from.

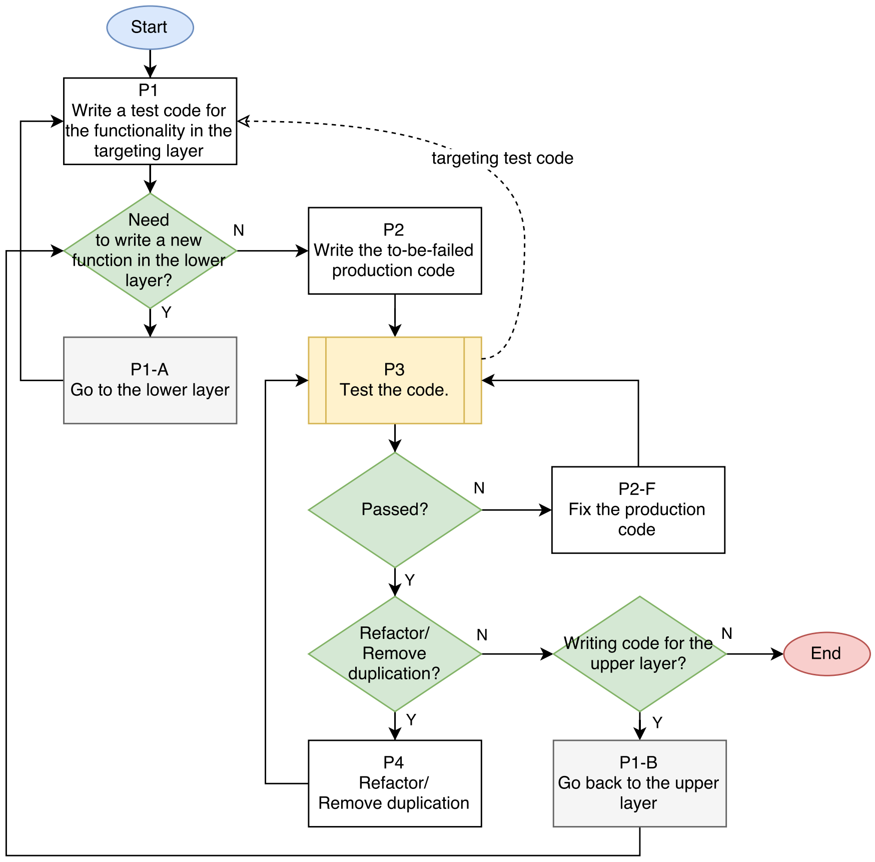
3.5 Data Source

Data Source is the source of the real data. It can be a database, a file, a web service, or a device such as a scanner or digital camera.

From Figure 2, the dependency relationship is consistently from right to left. For example, an instance of a Presentation Model class can be used within a Presentation class, but not in the reverse. Or, in other words, the Presentation Model class instance is responsible for the Presentation class instance, e.g. to give a response to a request from the Presentation class instance, but the Presentation Model class instance is not aware of anything happened in the Presentation class instance. The directions of the reference are also the same for each connection of two layers as shown in the request and respond arrows. Any member of the class which has the role as the interface to other layer has to be declared as public, so it can be tested by the test code which is located outside the production codes.

4 The Proposed Method

Figure 3 shows the flowchart of how to build the application based on the proposed framework described in the previous section. The flowchart shows the flow of tasks to write the code in one layer. Implementation of codes in each layer literally has the same workflow. The workflow is described in the following subsections.



Start
P1
Write a test code for the functionality in the targeting layer
targeting test code
Need to write a new function in the lower layer?
N
Y
P2
Write the to-be-failed production code
P1-A
Go to the lower layer
P3
Test the code.
Passed?
N
Y
P2-F
Fix the production code
Refactor/ Remove duplication?
N
Y
Writing code for the upper layer?
N
Y
End
P4
Refactor/ Remove duplication
P1-B
Go back to the upper layer

Figure 3 Proposed method flowchart

4.1 Write a test code (P1)

The first task to do is to write a test code for the production code we want to implement (P1). In the test code, the developer will write the codes to call the production code (in the same layer) passing any argument it might be required, and receive the feedback from it. If the production codes to be written need to call the not-yet-implemented production code in the lower layer, the developer can first write the not-yet-implemented production code in the lower layer, or instead, writes the production code in the current layer by creating a stub (Pressman, 2010). If the developer chose to write the production code in the lower layer first, then the task restarts from P1 (P1-A).

4.2 Write the to-be-failed production code (P2)

In this task, the developer writes the production code to make the test code pass. The first step is to run the test (P3), but do not implement anything except throwing a not-implemented-yet exception like the code in C# in .NET Framework shown in Figure 4, to make sure that the test runs but failed.

```
public int Method(string arg) {
    throw new NotImplementedException();
}
```

Figure 4 The to-be-failed production code

4.3 Test the production code (P3)

In this task, the developer runs the automated test suite targeting the production code he/she just wrote. The first time should be failed, and proceed to fix the production code (P2-F) and back to the test (P3). If the test passes, then proceed to the next step, which is the branch of one or two questions depends on the answer of the first question, "Do you need to refactor/remove duplication?". If the answer is yes, then proceed to the refactoring process (P4), but if the answer is no, then there is another question, “Are you writing the code for the upper layer?”, which means, did the developer came from process P1-A or not. If the answer is yes, then he/she goes back to the upper layer (P1-B) and answer the question “Do you need to write a new function in the lower layer?” and iterate the previous processes, or if the answer is no, then the work ends here.

4.4 Refactor/remove duplication (P4)

In this task, the developer removes any duplication in the production code, or any other refactoring tasks (Fowler, et al., 1999), and run the test (P3) again. The developer should avoid writing new code having new software functionality. If the refactoring process involves other production codes, the developer needs to run the test for each of those production codes for regression test. The developer should make sure that all tests pass before finishing the task.

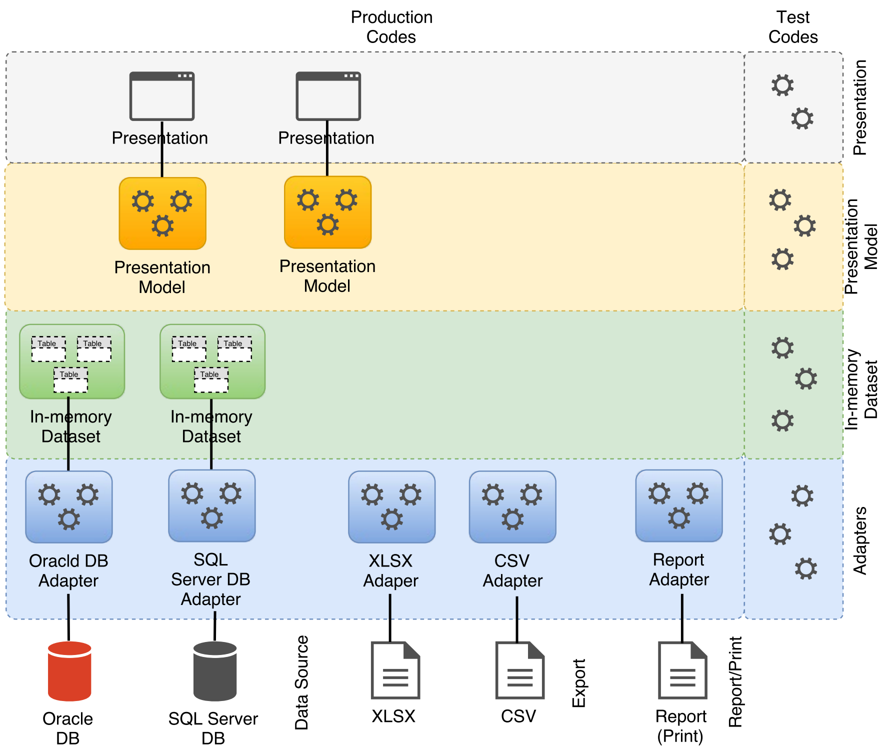


Figure 5 Case study development system architecture

5 Case Study

This section describes a case study conducted in the development of a real-world business system. The development project is held in a software development company S located in Japan. The development of a business system for the return and disposal management is involved in this case study. The system is using an Oracle[3] database which is part of the legacy system and an SQL Server[4] database which is used exclusively by the system. The system retrieves inventory data for return and disposal from the legacy system database and then stores the data along with the delivery information entered by the user to the system database. The system provides outputs to the user on the display screen, spreadsheet files (exported), and reports (printed). The system is developed based on .NET Framework[5]

3 Oracle database website http://www.oracle.com/database/

4 SQL Server is a database developed by Microsoft; website http://www.microsoft.com/sql/

5 .NET Framework is an application framework developed by Microsoft; website http://www.microsoft.com/net/

version 3.5 using Visual Studio[6] 2008. The NUnit[7] version 2.64 is used for the automated test environment. The Subversion[8] is also used in the development for versioning control. The system is developed using the proposed method described in the previous sections. The overall development system architecture is shown in Figure 5.

In the back-end, the system uses two database servers, the Oracle and the Microsoft SQL Server. Because accessing these two different database servers is technically using different codes, we need to separate Adapters for each of these databases. Each Adapter contains some functions to provide different functionalities needed by the application, and most of them provide In-memory Dataset as the response.

In the front-end, the system provides two main programs which have multiple Forms (Presentation classes in the proposed framework). There are Models (Presentation Models) for each of those Forms. Those Forms retrieve or send data from/to the database via those Models.

There are also some other forms of output to the end-user, those are XLSX (Microsoft Excel files), CSV (comma delimited) files, and reports for printed matter. Because each of those outputs uses technically different codes to generate, Adapters for each of them are provided. These output Adapters also retrieve data from the databases via their corresponding Adapters.

Those parts explained above are implemented in the production codes. In a separate project, there are test codes for testing all the software functionalities in each of those parts.

Table 1 Record of change and other requests (excerpts)

No	対象機能	課題内容	課題区分	対応内容	対応日	View		ViewModel		Adapter								Common	
						Logic	Design	Model	>Test	Data	>Test	Export	> Model	>Test	Report	> Model	>Test	Common	>Test
1	照会画面	画面添付課題1の赤カッコ部分のデータはMTで指定をMTとSTにすると表示されないので、MT～HNはOR条件で抽出してほしい。	修正	修正しました。	2017/02/24					Rev.4397	Rev.4397								
2	照会画面	画面添付課題2の赤いカッコ部分の品目検索ですが、現在は完全一致ですが、前方一致で表示するようにしてほしい。	修正	修正しました。	2017/02/24					Rev.4397	Rev.4397								
3	入力画面	画面添付課題3の赤いカッコ部分の返却左記’区分’の入力は廃棄の時入力しないので必須条件から外して頂きますようお願いします。→課題30で再記載	変更	伝票区分が返却・廃棄の場合、返却先区分と納期を必須条件から外しました。	2017/02/24			Rev.4397											
4	入力画面	画面添付課題4の赤いカッコ部分のAC支給検索で親コード指定すると表示までに、3分かかる。 下記SQL分の品目取得だけだと1秒ほどで取得しているので、 何に時間がかかっているのかご確認よろしくお願いします。 select ESUBJTOD(B.PADRQJ),A.PDY57LN,B.PAY57SRTP,A.PDY57LITM,A.PDDSC1,B.PAY57PTLE from F574211 A,F574201 B WHERE　A.PDDOCO = B.PADOCO AND　B.PAY57SRTP = 'UPCH-29249' AND　A.PDPRP4 = 'AC1' AND B.PAY57SRST < '99'	改善	以下の LIKE クエリを改善しました。 ・客先住所録照会 ・ディスク在庫系の検索 ・AC検索 ・資材検索	2017/02/24					Rev.4397									
5	入力画面	画面添付課題5の赤いカッコで、変更未確認と伝票未発行の物は、受注先名から返却左記名まで赤字で表示するようにしてほしい。→課題31で再記載	修正	修正しました。	2017/02/24					Rev.4397									
6	照会画面	画面添付課題6の赤いカッコで、表示条件を変更未確認と伝票未発行を足してほしいのとガイダンスタイトルの変更をおお願いします。 作業未完了＋変更未確認＋伝票未発行→　タイトル 　作業未・変更未・伝票未 配送未完了＋変更未確認＋伝票未発行→　タイトル 　配送未・変更未・伝票未	追加	伝票未発行の条件を追加しました。	2017/03/02					Rev.4397	Rev.4397							Rev.4397	
			変更	選択項目の内容を変更しました。	2017/03/07					Rev.4403	Rev.4403							Rev.4403	
7	入力画面	画面添付課題7の赤いカッコで、資材送り状とMTLF送り状は、伝票が自動的に出るのではなくて、エクセルデータを出力してほしい。（詳細仕様記載せずすみません。） あと、明細の出力対象は、返却数と転送数の欄に数字もしくわ文字が入っているものを出力対象でお願いします。→課題29へ再記載	変更	変更しました。	2017/03/07							Rev.4403		Rev.4403				Rev.4403	Rev.4403
8	入力画面	画面添付課題8の赤いカッコで、廃棄証明書は、必要と不要の選択とし、文字入力も可能とする。AC関係とDISC関係の両方	変更	変更しました。	2017/02/28	Rev.4397	Rev.4397												
9	完了入力	画面添付課題9の赤いカッコで、完了後に、レ点をいれて完了担当と日時をクリアするボタンを付けてほしい。（完了キャンセル）	変更	変更しました。	2017/02/28	Rev.4397	Rev.4397	Rev.4397	Rev.4397										
10	作業指示画面	画面添付課題10.10-1の赤いカッコで、一部でも完了していると作業指示の追加や変更が出来ないようになっているが、完了していても変更可能としたい。	変更	変更しました。	2017/02/28	Rev.4397													
11	返却指示欄	画面添付課題11の赤いカッコで、客先の担当を登録するので、フリー入力で担当部署と同じ桁の登録としてほしい。	変更	変更しました。	2017/02/28	Rev.4397	Rev.4397			Rev.4397									
12	照会画面	画面添付課題12の赤いカッコで、工場区分データがつくばと御殿場の場合に、 御殿場の指定の時やつくばの指定の時でも、つくばと御殿場の区分のデータを表示してほしい。	修正	修正しました。	2017/02/28					Rev.4397									

To evaluate the effectiveness of the proposed method, a record of requests regarding software change including addition, modification or deletion of requirement, and also bug report or inquiry has been taken. The excerpts from the record are shown in Table 1.

[6] Visual Studio is a development environment developed by Microsoft; website http://www.microsoft.com/vs/

[7] NUnit is a test suite for .NET framework; website http://www.nunit.org

[8] Subversion is an open source versioning control system; website http://subversion.apache.org

The first 6 columns of the record contain the sequence number, part of programs involved, the content of the request, the category of the request (change, fix or inquiry), the response by the developer, and response date.

To the right, there are columns representing parts of the development system which also represent the layers of the codes. They are View (which referred to as Presentation in the proposed framework), ViewModel (which referred to as Presentation Model), Adapter and Common (mostly contains system-wide static library). Then, by combining with observing the logs taken from the Subversion versioning control system, any parts involved in the software changes are marked with their respective revision number. The record has been summarized as shown in Table 2.

From the summary, we can see that 27 out of 68 number of changes occurred in the graphical user interface. That means that those changes could only be confirmed by launching the application. The rest, 41 or 60.3% of changes, is done outside of the graphical user interface. That means that more than half of the changes could be confirmed by an automated test. From 37 changes occurred in the Adapter, 29 or 78.4% have no impact on the other parts. That means that most of the changes are related to Data Source. Also, there are only 5 changes involving all parts of the graphical user interface, Model, and Adapter, which means that those parts have been developed loosely coupled.

Table 2 Summary of record

Number of requests		64	
Number of total cases of requests		68	
Number of changes in	View (total I only)	27	16
	ViewModel (total I only)	17	9
	Adapter (total I only)	37	29
	ViewModel & Adapter	5	
	View & ViewModel	8	
	View, ViewModel, & Adapter	5	

6 Discussion

Table 3 shows a comparison of time-consuming for software function test between test by launching the application GUI and by using an automated test. The test was conducted on a small desktop application accessing a local database. Three tests with two cases for the first two tests have been conducted. From the result, we can see that the test conducted with the test suite (automated test) for the individual test takes 53% to 59.5% less time than that with application GUI. Also, when conducting multiple tests, the difference becomes more significant, that is 72.6% less time for the automated test.

Table 3 Time-consuming comparison between GUI based and automated test

	Production App (auto)		Test Suite			Difference (% less)	
	Individual	All	1 Case	2 Cases	All	Individual	All
Get User Full Name	2.91		1.21	1.31		54.8%	
Authenticate User	3.40	5.41	1.35	1.38	1.48	59.4%	72.6%
Get User List	2.75		1.29	N/A		53.0%	

**) in seconds*

The time consumed for the automated test is relatively very small which is in the order of magnitude of 1 to 2 seconds, compared to the time consumed for the preparation of the test which is in the order of magnitude of 4 to 5 seconds (several tens of minutes to hours). A related study reported 16% more time for TDD based development (George & Williams, 2003). In the case study, the number of lines of code (LOC) of the test codes is only 6.1% of the total LOC of the production codes. However, considering the benefits, such as code quality, changeability as well as the easiness, and the maintainability, the more effort time is worth to spare.

Furthermore, we calculated the maintainability index of the production codes developed using the proposed method, and compare it with one of the production codes from another similar project which is developed using conventional method. The maintainability index is a value between 0 and 100 that represents the relative ease of maintaining the code (Naboulsi, 2011). The higher value means better maintainability. The maintainability index is calculated using the code analysis feature provided by Visual Studio. Table 4 shows the maintainability index of codes in the View classes which represent the Presentation layer, and the ViewModel and Adapter classes which represent the Presentation Model and the Adapter layers, respectively. In the conventional method, there are only Model classes which responsible for the domain logic and the data access.

Table 4 Comparison of the maintainability index

Class category		Conventional method		Proposed method		Difference	Improvement
		Average	Std deviation	Average	Std Deviation		
View		51.57	5.54	47.76	11.51	-3.81	-7.38%
Model	ViewModel	50.13	6.11	71.13	7.70	17.58	35.08%
	Adapter			64.29	11.67		

From Table 4, we can see that the codes in the ViewModel and Adapter classes which are developed using the proposed method have 17.58 point higher maintainability index or 35.08% improvement than those in the Model classes which are developed using the conventional method. The separation of concern of the Presentation Model and the Adapter seems to be the reason. We observed that the codes within the View codes developed using the proposed method seem to be less maintainable. However, since the standard deviation is higher than those codes developed using the conventional method, it can be considered that there is no significant difference of the maintainability in the View codes.

From the case study, we can conclude that:

- Automated test can help developer to confirm the codes faster, even for combined tests;
- Separating the domain logic from the Presentation allows developers to conduct automated test;
- Loosely coupled or loosely separated layers of functionality allow the developer to perform changes and tests more independently among layers.
- As the maintainability is one of the software quality characteristic (ISO/IEC JTC 1/SC 7, 2011), the proposed method can help the developer to produce a better quality of software product.

7 Conclusion

Implementing TDD in business system development enables us to develop testable codes which inevitably produce loosely coupled modules, modules which have fewer constraints or dependencies among them, thus makes module's concern more distinct. Our approach of implementing TDD in database access layer enables us to practice TDD in a resource expensive object faster while still maintaining the reliability and confidence the codes will work in the real database connectivity. Furthermore, compared to the development approach without introducing TDD, while there was the additional time necessary for designing and writing the test codes, the developer's effort in answering the bug report and the customer requirement for change is lower. With the proposed method, developers can expect a better quality of software product.

TDD or automated testing, in general, is an effective technique to test the production code with deterministic results. However, in a business system, database content is dynamic. Further work needs to be done to address the nondeterministic behavior in a database application.

Acknowledgement

We would like to thank Mr. Naofumi Takayama and Mr. Akihiro Yoda for allowing us to conduct a case study in the real-world software development and providing the data for evaluation.

References

1. Łaskawiec, S. (2016). The Evolution of Java Based Software Architectures. *Journal of Cloud Computing Research, 2*(1), 1-17.
2. Beck, K. (2008). *Test-Driven Development by Example.* Addison-Wesley.

3. Bhat, T., & Nagappan, N. (2006). Evaluating the Efficacy of Test-Driven Development: Industrial Case Studies. *Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering*, (pp. 356-363).
4. Cockburn, A. (2005). *Hexagonal Architecture*. Retrieved 2017, from http://alistair.cockburn.us/Hexagonal+architecture
5. Dogša, T., & Batič, D. (2011). The effectiveness of test-driven development: an industrial case study. *Software Quality Journal, 19*(4), 643-661.
6. Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures.* Dissertation. Retrieved 2017, from http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
7. Fowler, M., Beck, K., Brant, J., Opdyke, W., & Roberts, D. (1999). *Refactoring: Improving the Design of Existing Code.* Addison-Wesley.
8. Fowler, M., Rice, D., Foemmel, M., Hieatt, E., Mee, R., & Stafford, R. (2002). *Patterns of Enterprise Application Architecture.* Addison-Wesley.
9. George, B., & Williams, L. (2003). An Initial Investigation of Test Driven Development in Industry. *Proceedings of the 2003 ACM Symposium on Applied Computing (SAC)*, (pp. 1135-1139).
10. ISO/IEC JTC 1/SC 7. (2011). *ISO/IEC 25010:2010 Systems and Software Engineerings — Systems and Software Quality Requirements and Evaluation (SQuaRE) — System and Sofware Quality Models.* ISO/IEC.
11. Mackinnon, T., Freeman, S., & Craig, P. (2001). Endo-Testing: Unit Testing with Mock Objects. In *Extreme Programming Examined* (pp. 287-301). Addison-Wesley.
12. Microsoft. (n.d.). Chapter 1: Service Oriented Architecture (SOA). Retrieved 2017, from https://msdn.microsoft.com/en-us/library/bb833022.aspx
13. Naboulsi, Z. (2011). *Code Metrics — Maintainability Index.* Retrieved 2017, from MSDN Blogs: https://blogs.msdn.microsoft.com/zainnab/2011/05/26/code-metrics-maintainability-index/
14. Ou, R. (2003). Test-Driven Database Development: A Practical Guide. *Proceedings of Extreme Programming and Agile Methods — XP/Agile Universe 2003: Third XP Agile Universe Conference*, (pp. 82-90).
15. Pressman, R. S. (2010). *Software Engineering: A Practioner's Approach* (7th ed. ed.). McGraww-Hill.
16. Ryu, H.-Y., Sohn, B.-K., & Park, J.-H. (2005). Mock Objects Framework for TDD in the Network Environment. *Proceedings of the Fourth Annual ACIS International Conference on Computer and Information Science (ICIS'05)* (pp. 430-434). IEEE.
17. Vernon, V. (2013). *Implementing Domain-Driven Design.* Addison-Wesley.
18. Wasmus, H., & Gross, H.-G. (2007). Evaluation on Test-Driven Development: An Industrial Case Study. *Proceedings of the 2nd International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2007)*, (pp. 103-110).
19. Wels, S. (2012, August 22). Test Driven Development. *Proceedings of Agile Seminar 2012.* Retrieved March 19, 2017, from https://sewiki.iai.uni-bonn.de/teaching/labs/xp/2012b/seminar/start