Islamic University of Gaza

Deanery of Post Graduate Studies

Faculty of Information Technology

Developing Agile Applications Using Iterative Database Design Model

By

Emad Omar Kehail

Supervised by

Prof. Alaa Al Halees

Mar, 2016

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master in Information Technology

Abstract

Software development are becoming more complex day after day, customer requirements becomes more complex and changing a lot, and software development methodologies are trying to respond to those emergent business needs and requirements.

One of the major business needs of nowadays is the ability to quickly respond to business requirements and changes. Therefore, the software development process has been moved to be more agile by using what has been agreed to call Agile Software Development Process. Such agile methods, XP and Scrum for example, have been widely used lately instead of the traditional software development methods such as Waterfall and Spiral.

However, the business still needs to store data, and Database Management Systems (DBMS) are still the de facto for the business software. DBMS relying completely on Database Design process that follow traditional up-front design process which is sequential by nature. The Data Analyst needs to develop a complete Entity Relationship Diagram (ERD) which is a result of a normalization process that generates the tables and their relationships.

This research developed a model that integrates the database design techniques with Scrum Agile practices. The new model did not sacrifice the features of the database design techniques, yet the model helped to make the database design process more agile by distributing the database design process among the Scrum development process.

We evolve our new model by using Focal Point approach and then adding an Abstraction Layer at the database level which contains business logic related to data implemented using stored procedures and functions, and we find that this helps to reduce the impact of the changes implemented at the database level and to achieve the goal with percentage around 64% of the time needed to achieve the same goal using the traditional upfront design. This is in addition to the flexibility of the new system when it comes to adapt new changes since the results showed that the new model is around 80% more flexible than using upfront design approach.

Keywords: ***Agile Software Development, Database, SCRUM***

Arabic Abstract

تطوير تطبيقات أجيل باستخدام نموذج تكراري لتصميم قاعدة البيانات

أصبح تطوير البرمجيات والأنظمة المحوسبة أعقد من ذي قبل، فقد أصبحت متطلبات العميل معقدة ومتسارعة التغيير، ولهذ تحاول طرق الحوسبة الحديثة أن تلبي متطلبات العمل قدر الإمكان.

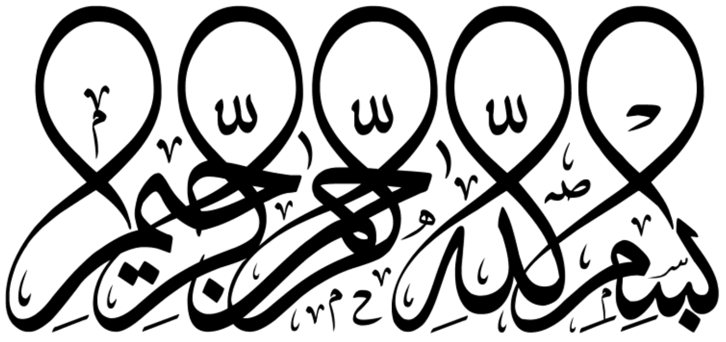
ومن أكثر احتياجات العمل في هذه الأيام هو القدرة على تلبية الطلبات والتعديلات بسرعة. لهذا، أصبحت طرق الحوسبة الجديدة تتجه إلى أن تكون رشيقة ومرنة، وأصبحت تسمى "الطرق الرشيقة في الحوسبة" مثل "سكرم" و "إكس بي" اللتان تستخدمان بشكل واسع في حوسبة الأنظمة بدلاً من الطرق التقليدية مثل "الأسلوب اللولبي" أو "شلال المياه".

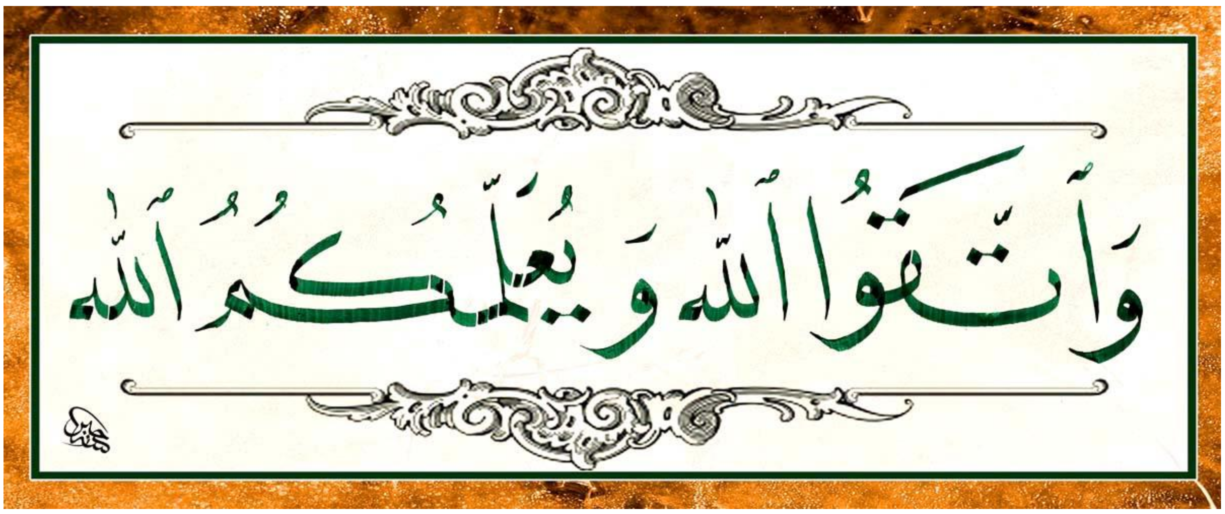
إضافة إلى ذلك، الأعمال التجارية بحاجة ماسة إلى تخزين البيانات، ولهذا بقيت "نظم إدارة قواعد البيانات" شيء أساسي في حوسبة الأنظمة. "نظم إدارة قواعد البيانات" تعتمد أسلوب "التصميم المسبق" في تطويرها، وهي طريقة تسلسلية في طبيعتها، حيث يقوم مصمم قواعد البيانات بتصميم مخطط قواعد البيانات، والذي هو نتيجة لعملية تطبيع الحقول إلى عدة جداول والعلاقات ما بين هذه الجداول.

هذه الدراسة طورت نموذج دمج ما بين أسلوب تصميم قواعد البيانات و الأسلوب الرشيق "سكرم" بطريقة تكاملية. وقد راعى النموذج الجديد أن لا نفقد أي ميزة من ميزات تصميم قواعد البيانات، بل جعل عملية تصميم قواعد البيانات أكثر رشاقة من خلال توزيع هذه العملية على مراحل تطوير الأنظمة المتبعة في أسلوب "سكرم".

قمنا بتطوير هذا النموذج من باتباع أسلوب "النقطة المحورية" وتم أيضاً إضافة "طبقة مجردة" على مستوى قاعدة البيانات. الطبقة المجردة تحتوي على الشيفرة المصدرية" التي تقوم بمنطق العمل الخاص بمعالجة البيانات وذلك باستخدام الإجراءات المخزنة على مستوى قواعد البيانات. وقد وجدنا أن هذا الأسلوب قلل من عدد التعديلات التي كانت تجرى على قواعد البيانات بناء على متطلبات العمل بنسبة 64% مقارنة بتنفيذ نفس التعديل بالأسلوب القديم التسلسي الذي يعتمد على التصميم المسبق.هذا بالإضافة إلى المرونة التي يتميز بها هذا النموذج في قدرته على استيعاب التعديلات حيث أظهرت النتائج أن النموذج الجديد مرن بنسبة تصل إلى 80% مقارنة مع الأسلوب التسلسلي الذي يعتمد على التصميم المسبق لقواعد البيانات.

كلمات مفتاحية: **تطوير البرمجيات الرشيق، قواعد البيانات، سكرم.**


بسم الله الرحمن الرحيم


وَٱتَّقُوا۟ ٱللَّهَ وَيُعَلِّمُكُمُ ٱللَّهُ

Dedication

To the soul of my Parents

To my beloved wife and children

To my family

To all friends

Acknowledgments

All praise is for Allah, the Almighty for providing me with the strength to accomplish this thesis and for guiding me at every stage of my life.

This thesis is the result of years of work whereby I have been accompanied and supported by many people. It is wonderful that I now have the opportunity to express my gratitude to all of them.

I'd like to thank everyone who has helped me in completing this work. I submit my highest appreciation to my thesis advisor **Dr. Alaa M. Alhalees**, who supported me and helped me in each step. I would like to express my deep and sincere gratitude to him. His knowledge and personal guidance have provided a good basis for the present thesis.

I would also like to thank **Eng. Mohammed Elkhoudary** and **Eng. Alaa Alsalehi**, for their valuable technical help and appreciated efforts.

Also, I would like to take this opportunity to express my profound gratitude to my beloved family, especially my wife, and to my family - without whom I would never have been able to achieve so much.

Last, but certainly not least, I offer my thanks and appreciation to all of those who supported me in any respect during the completion of the research.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ASD	Agile Software Development
BCNF	Boyce Code Normal Form
BDUF	Database Upfront
CORBA	Common Object Request Broker Architecture
DAO	Data Access Objects
DBA	Database Administrator
DCOM	Distributed Component Object Model
DM	Data Modeling
ERD	Entity Relationship Diagram
JDBC	Java Database Connectivity
LOB	Large Object
O/R	Object Relational
OLTP	Online Transaction Processing
OO	Object Oriented
OOD	Object Oriented Design
OOP	Object Oriented Programming
RDBMS	Relational Database System
SDLC	Software Development Life Cycle
SQL	Structured Query Language
XP	Extreme Programming

CHAPTER I: Introduction

Nowadays software systems are big if not even huge. They tend to be complex and interconnected with other software and hardware systems. Also, the data related to those systems are getting large in size and this adds more complexity to the database design for those systems [1].

All of that, and when added to the business needs that are changing rapidly, and customer requirements which are continuously changing as well. Then; the software development process is getting more and more complex day after day [1] [2].

Therefore, applying traditional Software Development Lifecycle methodologies such as waterfall to such huge systems would give us the following results [3]:

- **Poor Visibility.** This is because of the sequential nature of the waterfall model.
- **Can't handle changes.** The customer has to wait long time in order to see beta version of the software, and it is very hard to go back to modify the requirements because of the high cost of the changes.
- **Poor quality.** The errors are discovered at late stages and only after the software is released to testing.
- **Higher risks.** The software is delivered late to the market, which means other competitors could deliver before which much more exiting features.

This is because traditional software development, especially those who are database-dependent, are sequential in nature, figure (1.1) represents a typical waterfall software development approach. This adds extra delaying time to the developed software since the software developers have to wait for the data molders to design the Conceptual Level, Entity-Relationship Diagram (ERD), and then convert it into a physical database design. This process usually tends to generate an optimized database structure such as the third normal form 3rd NF or Boyce Code Normal Form BCNF [4].

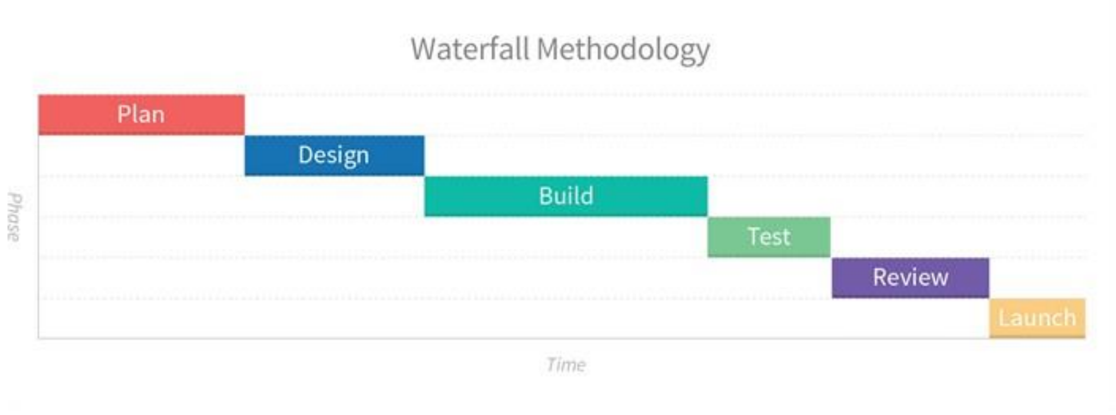


Figure 1.1: Typical Waterfall Software Development Process
[5]

That's why in the last years the adoption of the Agile Software Development models is rising up and adopted rapidly [2]. There is an adoption of the Agile techniques in many software development firms, whether these firms are commercial, governmental or academic [1].

This rapid adoption of agile practices because Agile practices promised of the following [6]:

- ***Individuals and interactions*** over processes and tools
- ***Working software*** over comprehensive documentation
- ***Customer collaboration*** over contract negotiation
- ***Responding to change*** over following a plan

And this yielded many benefits such as [6]:

- Early return on investment
- Short time to market
- Improved quality
- Enhanced client relationships
- Better team morale

Figure (2.1) represents a typical agile approach in software development.

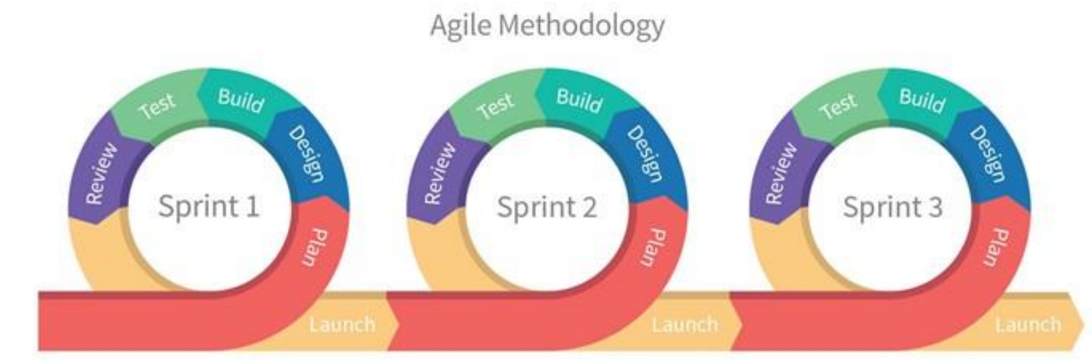


Figure 1.2: Typical Agile Software Development Process
[7]

That's was for the Software Development techniques, but what about the Database layer?

Nowadays, NoSQL databases engines are rising up rapidly, and anyone might think the golden solution is just to start using NoSQL databases since they are dedicated to handle large datasets. However, this is not the right way to go. It is true NoSQL database are meant to handle large data, but RDBMS database still outperform them when it comes to OTLP processing and data consistency [8] [9].

Therefore, there is still a need to use the relational databases for OLTP based systems as stated by Ambler in [10] "Unfortunately, most data-oriented techniques are serial in nature, relying on specialists performing relatively narrow tasks, such as logical data modeling or physical data modeling. ***Therein lies the rub – the two groups need to work together, but both want to do so in different manners***"

Because of that, the ***Evolutionary Database*** techniques start to rise up [11] [10] to help Software Developers and Architects to unify the concept of Agile Software Development along with the Database Design and Modeling [12]. This will greatly help to release working versions of the desired software very quickly without sacrificing the concepts of the database modeling and normalization rules.

1.1 Statement of the Problem

The traditional Database Design and Modeling is an up-front design, and despite the fact that this has been followed through many years, this way of database design and modeling lacks the flexibility to conform to the new Agile Software Development models.

This is supported by Ambler in [10] “The traditional approach to data modeling does not reflect the evolutionary approach of modern methods such as the RUP and XP, nor does it reflect the fact that business customers are demanding new features and changes to existing functionality at an accelerating rate. The old ways simply aren’t sufficient any more, if they ever were”.

Moreover, today software systems are big, if not even huge. The time consumed into designing and modeling the database is very long. This means the developers have to wait for the data modeler to finish the Logical Design, then they have to wait for the Database Administrator to finalize the Physical Design, then, and after all of that, the developers can start working on the business logic and the user interface.

1.2 Objectives

1.2.1 *Main objective*

The main objective of this work is to make the database design and modeling more agile by designing a process to be followed in an Agile Software Development Lifecycle, mainly Scrum, and merge this process with the Scrum framework.

1.2.2 *Specific objectives*

- Design a process to merge the database model along with the agile practices.
- Implement a pilot project to verify the new model.
- Evaluate the effectiveness of the new model.

1.3 Research Scope and Limitations

- This research proposes a model for Agile Database Modeling that will be integrated into Agile Software Development practices. The research will only concentrate on the database layer.
- The research will only focus on how to integrate and adapt Agile Database techniques with an existing Agile Model.
- The new proposed model will be applied to a pilot software system in order to test its applicability.
- We choose questionnaires to collect feedback from developers and to test our model.

1.4 Importance of the Project

Up-front database design and modeling takes long time, and when the developer starts developing the business logic and the interface, then the requirements usually change. This will result in customer dissatisfaction.

Therefore, the database design and modeling has to move one step forward, but, without losing any of its characteristics. The up-front design has to be abandon and a way of an incremental design has to be considered.

This will allow agile software developers to easily embed database design in the agile practices which in turn will help them to present their work to the client quickly. This will help to quickly release versions of the desired product as well as quickly fixing errors in the early stages of the software development.

1.5 Thesis Organization

This thesis is divided into five major chapters which are structured about the objectives of the research. The thesis is organized as follows:

Chapter I: Gives a brief introduction about the thesis. It presents the statement of the problem and the general and specific objectives.

Chapter II: Presents the background of the Agile concept and the concept of the database design. This chapter discusses the features of Agile and how it is an incremental model while the database in an up-front model by nature.

Chapter III: Presents the related work and current used agile methodologies. Moreover, it presents the concept of the database design, and how the up-front database design mismatch with the current agile methodologies. Also, this chapter discusses how database design could be integrated and used with agile practices

Chapter IV: Includes the methodology used to develop the new proposed model. It presents how to the model will be integrated with Scrum agile practices, and furthermore, it explained and in detailed steps, how to use the model effectively.

Chapter V: Explains the experiment steps and the requirements of the proposed software system that has been used by the working teams. In addition, it presents the results from the working teams who developed the system, compare them, and discuss the findings of the results.

Chapter VI: Draws the conclusion and summarize the research achievements of the new model and the obtained results, and suggests future work.

CHAPTER II: Background

This chapter will review the background of this research, the chapter also discusses the traditional up-front database design and the usage of the Agile models in the software industry. Also, this chapter presents how and why traditional database design contradicts with the nature of the iterative and incremental agile practices.

2.1 Agile Methodology

2.1.1 Agile and traditional Software Development Lifecycle Models

Many software development life cycle (SDLC) exist in the market, and this made it hard for the enterprises to choose the most suitable one of them [4]. For a very long time, Waterfall SDLC has been used to develop software systems, followed by other SDLC such as V-Model, then finally Agile has arrived [4].

Balaji et. al. in [4] has mentioned that there are "*A number of software development life cycle (SDLC) models have been created: waterfall, spiral, V-Model, rapid prototyping, incremental, and synchronize and stabilize. Waterfall model is the Sequential development model*", and therefore they have implemented a comparative study about the advantages and drawbacks of each of the following three SDLCs:

- Waterfall Model
- V-Model
- Agile Model

To see which one of them will be the most suitable SDLC model for an enterprise. The result of the study has reached the following:

- If requirement changes frequently and smaller projects, deliver product in short period time with skilled resources then we can choose "Agile model".
- If requirement is clear, larger project then we choose "Waterfall model"
- If requirement changes, larger project, proper validation to take place in each phase, tester to be involved in early stages of development, then we can choose "V-Model"."

It is clear that the study has stated that when there are requirements changes, then it is better to go for Agile Model, and when the requirements are clear it is good to go for traditional SDLC. However, the study did not take into considerations that the requirements needed to be frozen, not only clear. Freezing requirements is not an option in the rapid and evolving business.

Therefore, Agile Model is still the most suitable Model for SDLC when the requirements are changing frequently.

Moreover, traditional SDLC like waterfall might fit well before, but now it is inapplicable to modern software development. This is what Ambler in [13] has stated "*Data-oriented BDUF is a viable way to build software. But it's certainly not agile, and it certainly doesn't reflect the realities of most modern application-development efforts. It might have worked for you 20 years ago, although I doubt it was your best option back then either (I was naively working like this in the 1980s, by the way), but it isn't appropriate now. It is time to rethink your approach to data-oriented development and adopt evolutionary techniques.*". This is because of, but not limited to, the following:

1. **One size does not fit all**

   Data oriented modeling can't fir well for all projects, some projects needs other type of modeling [13].

2. **It isn't just about data**

   It is not only data, there are processes, forms, and reports. All of this must be considered when designing an application [13].

3. **You can't think everything through at the start**

   It is hard, if not impossible, to know everything needed from start, Ambler has described this in [13] as "Room Decoration" where you need to move some parts of the furniture around till you feel it is perfect [13].

4. **It doesn't easily support change**

   When everything is designed up-front, and work has been split off to more than one team; then it is hard to do changes later on [13].

2.1.2 Agile Software Development Models Adoption and Usage

On the other hand of the equation, the increase demand of software products has impact on the rapid growth of software industry. Today, there are many new software businesses has started and yet more to come. In spite of this, there are a lot of software products failures and business bankruptcy stories [1]. The failure of these software projects are because of the inappropriate selection of a SDLC that is able to respond to business requirements and business needs [1].

That’s why there is a need to use Agile Models, which is an iterative incremental approach to software development and capable of handling business requirements changing quickly [1].

Also, Begel at. el. in [2] has conducted a survey on using Agile Software Development (ASD) using a web-based survey at Microsoft. The survey targeted Microsoft employees who are working in “development, testing and management roles” [2] and also “directly involved in the production of software.” [2].

The importance of this study is that it is the first one focuses on large scale industry in software development company such as Microsoft. Moreover, the study has identified the most commonly used Agile practices inside Microsoft [2]. In this study, the employees were asked about the ASD methodology they are using, it found that 125 out of 192 of this question’s responses indicated they used the Scrum ASD methodology [2].

Figure (2.1) from Begel at. el. study [2] represents the ASD methodologies that have been used by the employees in the conducted study.

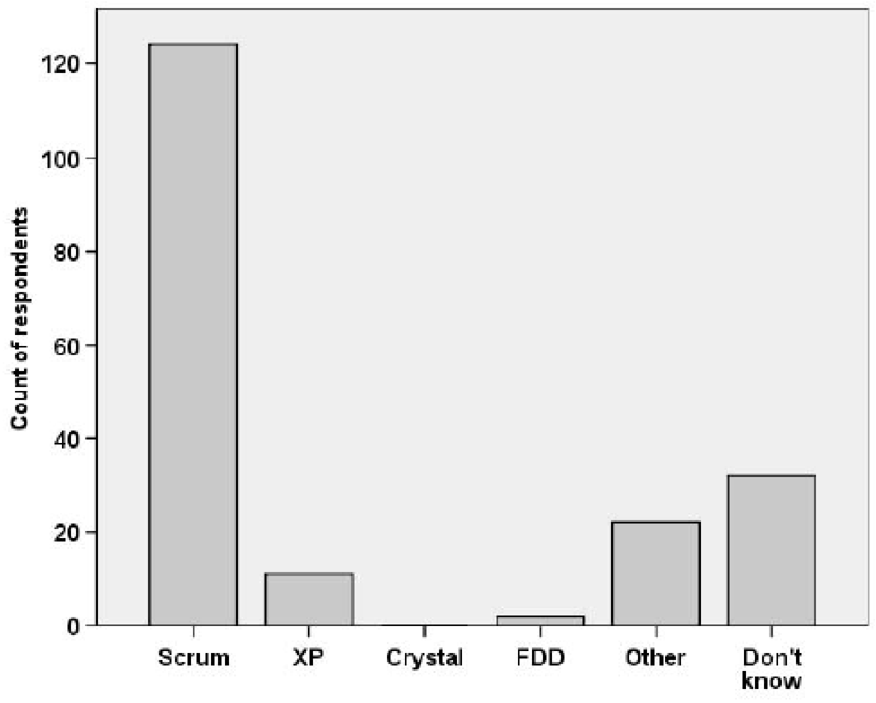


Figure 2.1: Different ASD Methodologies
[2]

Another important result of the study [2] is that it has measured what practices of Agile processes have been adopted by the employees who were targeted. Figure (2.2) represents these practices.

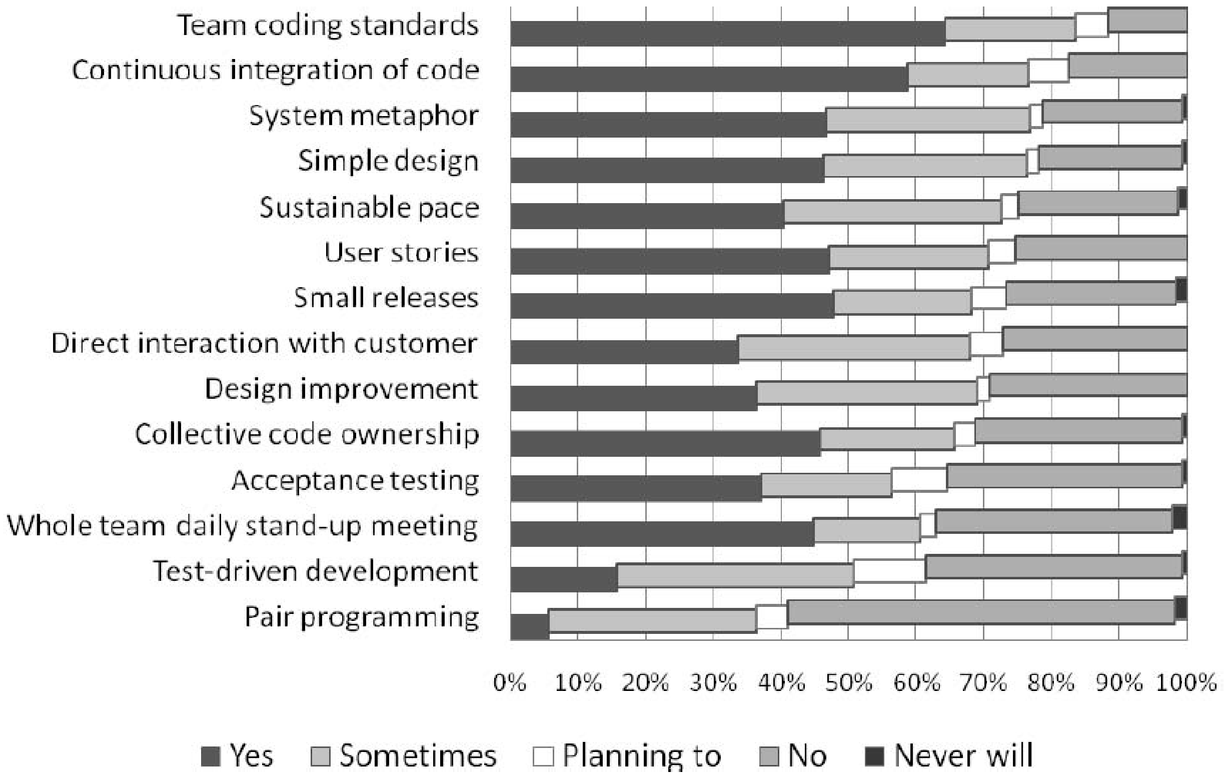


Figure 2.2: Percentage of Usage of Agile Practices [2]

From figure (2.2) we can observe that teams who are using Agile are concerned about things more than others. For example, we can see that there are concerned about Coding Standards and Continuous Integration of Code at the first place. Moreover, we can see that Small Releases and User Stories are important to them as well. This could be because User Stories are the customer point of view of the system and it is one of the requirements gathering tools that help them to figure out the system. Also, the Small Releases could be of importance to them because it helps them to engage the customer more frequently with the development team and have them do any corrective actions to get the system developed as how the customer expected.

Another important result of the study [2] is about the team attitudes and moral factors, the results are depicted in figure (2.3). The result *"My team's developers and testers collaborate more with Agile than before when adopted*

Agile methods." scores more than 60% of Strongly agree and Agree, and scores more than 85% if we add the Neutral results to it.

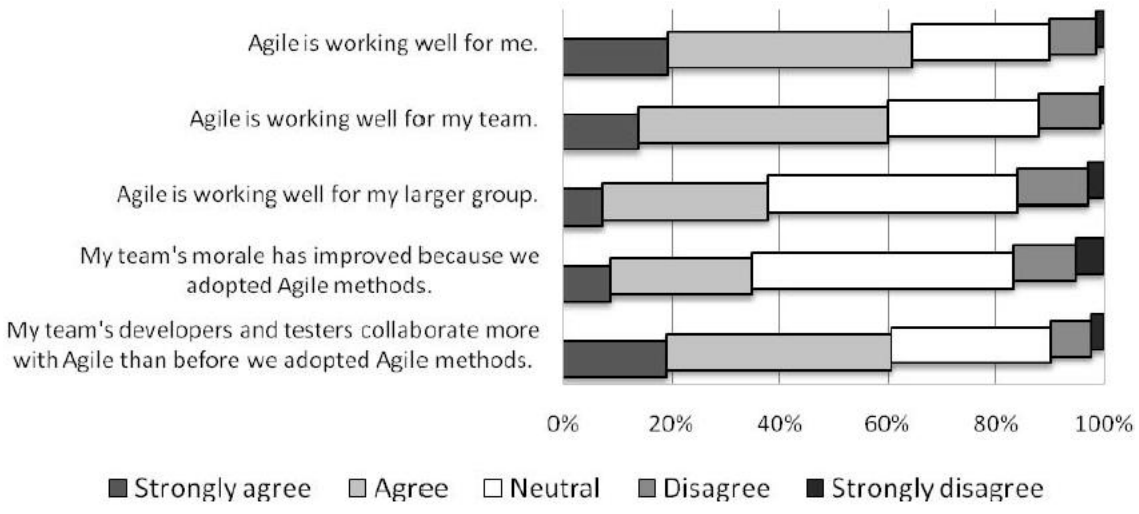


Figure 2.3: Team Attitudes and Morale Factors Concerning ASD

[2]

This result indicates that Agile, and especially Scrum since it was the most used ASD methodology, is able to help team, with different skills (developers and testers) to collaborate more. Therefore, it should not be a problem or impedance to add more team members with other skills, such as Data Modeling, to the Agile team.

Begel at. el. in [2] has expressed the above result as "*Among people who currently use Agile (left side of the graph), 89.7% like or are neutral to ASD. A more important point is that among groups that do not use ASD, 92.8% said they liked or were neutral to ASD, indicating that a vast majority of developers are open to trying ASD in the future*" [2].

Furthermore, and also when it comes to the ASD benefits, the most benefit was the improved communication among the team members. Daily Scrum meetings helps to bring team members with different skills along together and communicate effectively [2]. It is best expressed in Begel at. el. [2] "Team members are aware of what each of the others is working on".

In addition, "Software functionality progress can be checked and monitored much more frequently rather than at the end of the long milestones." [2].

Table (2.1) summarizes the most 10 benefits of the ASD methodologies conducted by the study [2].

Table 2.1: Benefits of Agile Development Methodologies

1.	Improved Communication and Coordination	121
2.	Quick Releases	101
3.	Flexibility of Design – Quicker Response to Changes	86
4.	More Reasonable Process	65
5.	Increased Quality	62
6.	Better Customer Focus	50
7.	Improved Focus -- Better Prioritization	28
8.	Increased Productivity	26
9.	Better Morale	23
10.	Testing First	22

[2]

On the other hand of the Begel et. al. study [2], there were some ASD problems that could be summarized as follows:

- ASD do not work well for large teams, especially teams that are more than 30 members.
- Others say that Scrum daily meetings were not efficient for large teams especially when it is led by weak Scrum Master.
- Agile and non-Agile members' interaction problems; some Agile team members say *"Interaction with non-Agile teams is hard because they do not understand that you can guarantee that all the sprint items will be completed because the prioritization meeting involves very loose time estimates."* [2]
- Others complains about losing the big picture because they were focusing on the daily work "*you 're so focused on the day to day deliverables.*" [2]. Therefore, the "*focus is on the today's work*" [2] more "*than what the feature team is trying to achieve.*" [2].

Despite the problems mentioned above, ASD methodologies benefits still outweigh these problems, and some of the problems could because some developers were rot trained well about how to use Agile processes and "*Some developers wished they had formal training to do Agile, noting that there were few training options available to them. Many who commented on training appeared to have the idea that if they did not do Agile perfectly then the product or process would suffer*" [2].

Much more could be because losing discipline, Agile development "*is simple, but requires a lot of discipline from the team.*" [2]. Usually, and because ASD methodologies differ in nature than traditional SDLC, "*Agile development often requires change in mindset that developers may not be eager to undertake. Several developers also note that unless there is full adoption by the team, Agile methodologies do not work very well*." [2].

2.1.3 Scrum Framework

Scrum is an Agile framework has been used to develop software systems at many scales of enterprises [1] [2], and as has been stated by Schwaber et. al. in [14] "*Scrum is a process framework that has been used to manage complex product development since the early 1990s.*".

Moreover, Scrum is a framework that can incorporate processes and techniques in which they can improve the overall Scrum framework [14]. This is because Scrum is composed of teams, their roles, events, artifacts, and rules. Each of these components of the Scrum framework has a special purpose [14] and this makes the Scrum framework customizable.

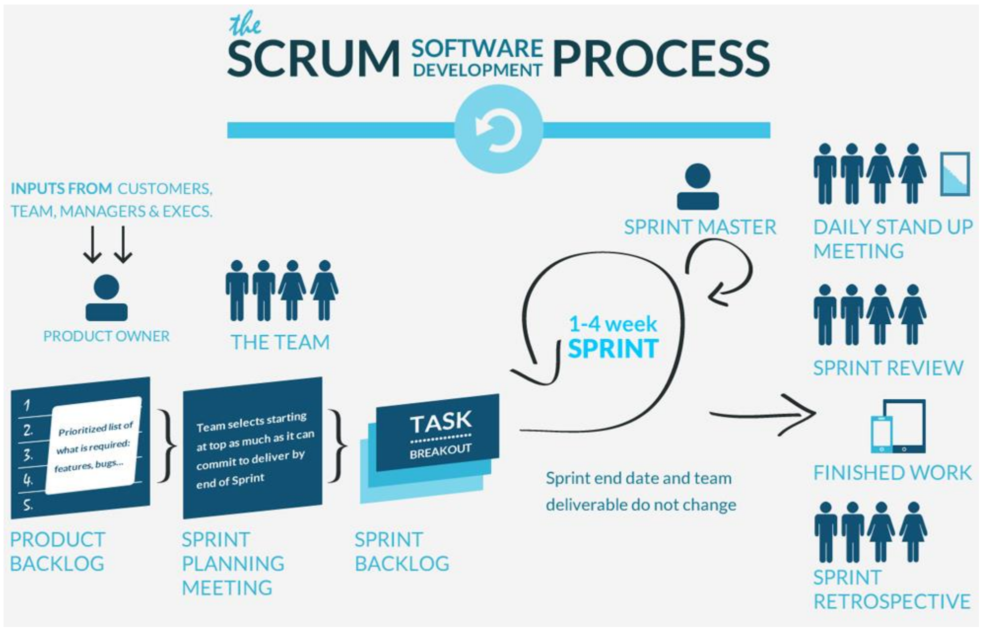
2.1.4 Scrum default roles, events, and artifacts

To better understand Scrum, we have presented the original roles of a typical Scrum Agile model, their activities, ceremonies, and the resulted artifacts in table (2.2).

Table 2.2.2: Original Roles, Activities, Ceremonies, and Artifacts of a typical SCRUM Agile process

Role	Definitions	Activities	Events	Artifacts
Product Owner	- The Product Owner is responsible for maximizing the value of the product and the work of the Development Team. How this is done may vary widely across organizations, Scrum Teams, and individuals.	- Clearly expressing Product Backlog items; - Ordering the items in the Product Backlog to best achieve goals and missions; - Optimizing the value of the work the Development Team performs; - Ensuring that the Product Backlog is visible, transparent, and clear to all, and shows what the Scrum Team will work on next; and, - Ensuring the Development Team understands items in the Product Backlog to the level needed.	- Sprint Planning Meeting - Sprint Review Meeting	- Product Backlog - Sprint Backlog
Scrum Master	The Scrum Master is a facilitator who is accountable for removing the impedances to deliver the sprint goals and deliverables. Scrum Master is not a Team Leader	- Ensure the Sprint is executed the way it is intended. - Acts as buffer in-between the team and as the distracting influences - Rule enforcer - Represents Management to the project team	- Sprint Planning Meeting - Spring Review Meeting - Daily Stand-up Meeting - Sprint Retrospective Meeting	- Burndown charts - Sprint Backlog
Team	The team is responsible for delivering the product. A team is typically made up of 5-9 people with cross functional skills	- Analyze - Develop - Technical communication - Document	- Spring Planning Meeting - Sprint Review Meeting	Based on team's progress the burndown charts are developed by the Scrum Master.

Because of the above, Scrum framework will be the best fit to this research since we need to ***employ new processes and techniques*** in our new model, and we need to ***add new team members*** to the typical Scrum team members.



the SCRUM SOFTWARE DEVELOPMENT PROCESS
INPUTS FROM CUSTOMERS, TEAM, MANAGERS & EXECS.
PRODUCT OWNER
THE TEAM
SPRINT MASTER
DAILY STAND UP MEETING
1-4 week SPRINT
SPRINT REVIEW
Prioritized list of what is required: features, bugs...
Team selects starting at top as much as it can commit to deliver by end of Sprint
TASK BREAKOUT
Sprint end date and team deliverable do not change
FINISHED WORK
PRODUCT BACKLOG
SPRINT PLANNING MEETING
SPRINT BACKLOG
SPRINT RETROSPECTIVE

Figure 2.4: Typical Scrum Process

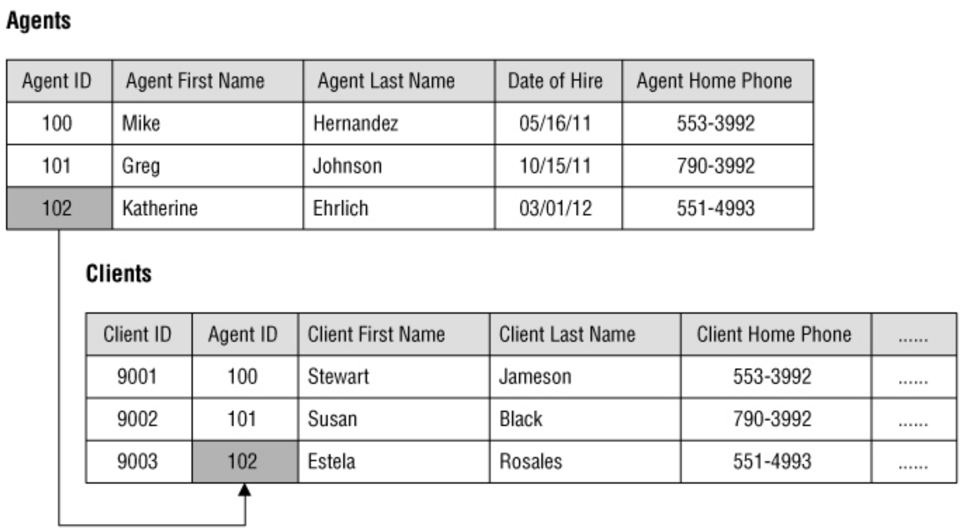
[15]

2.2 Relational Database

Relational database has existed long time ago, almost four decades from now. Through these years, relational database has gained acceptance from many organizations and nowadays software system that worth multibillion are dependent on relational database systems, for example, but not limited to, think of banking systems, airports and travelling booking systems [16].

Dr. Edgar F. Codd is considered the father of the relational model, he created the relational model because of his dissatisfaction with the database models and database products of the time led him to begin thinking of ways to apply the disciplines and structures of mathematics to solve the myriad problems he had been encountering [16].

Relational database stored data in relations, that's it, data are stored in tables, which are composed of fields and records. Each field is of one type of information such as Family Name in which will text data, while a record represents multiple fields such as ID, First Name, Family Name, Birthdate, and Address. Each record is identified by field that contains a unique value, and it is called the Primary Key [16].


Agents

Agent ID	Agent First Name	Agent Last Name	Date of Hire	Agent Home Phone
100	Mike	Hernandez	05/16/11	553-3992
101	Greg	Johnson	10/15/11	790-3992
102	Katherine	Ehrlich	03/01/12	551-4993

Clients

Client ID	Agent ID	Client First Name	Client Last Name	Client Home Phone	......
9001	100	Stewart	Jameson	553-3992	......
9002	101	Susan	Black	790-3992	......
9003	102	Estela	Rosales	551-4993	......

Figure 2.5: Example of How Tables Are Related in Relational Database

[16]

Relational database isolates the logical design from the physical design, the user does not need to know where are the tables physically stored in order to manipulate them. Structured Query Language (SQL) is the standard language that is used to retrieve and manipulate the data in the tables [16].

2.2.1 *Relational Database Design*

Relational database design considered serial in nature [10], the design process usually involves three phases: requirements analysis, modeling, and normalization [16]. Requirements involves meetings with stakeholders and understanding the business being targeted, while modeling is all about modeling the database structure using data modeling methods such as Entity Relationship Diagrams (ERD) to visually represents the database structure.

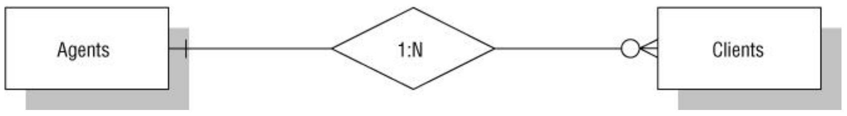


Figure 2.6: How Entities Are Related in ERD

[16]

Finally, normalization is the mathematical process to decompose large tables into smaller ones to avoid problems such as data redundancy and improve the process of data manipulation [16] [17].

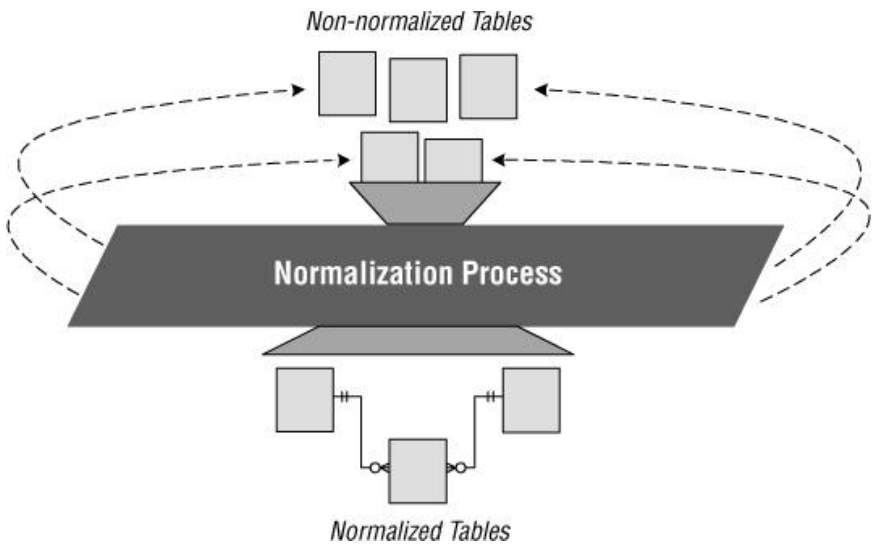


Figure 2.7: Table Normalization in Relational Database

[16]

Furthermore, when designing a database, it is important to consider the following [18]:

- implementation-independent; the design should not specify the technology that will be used.
- application-neutral; the design must and for sure serve the software system that is being analyzed, however, the design must not be limited for this software only.

2.2.2 *The Impedance Mismatch Problem*

Object Oriented Design (OOD) and Object Oriented Programming (OOP) are based on object design that yields classes with relationships such as inheritance and composition. Whereas Data Modeling (DM) and database design (Normalization) based on mathematical approach to normalize the data and reduce redundancy and ensure that the data can be used effectively.

Such differences have been figured out by people involved in the software development industry and has been called "*Object-Relational Impedance Mismatch*" [13]. In other words, "*Applications see data as properties of object classes; relational databases see data as attributes of entities*" [18], and here lies the problem.

Object oriented (OO) applications see things in different manner than how database engines, especially relational database, see it. A very good example is what L. Burns has stated in [18]: "*an application might contain an object class called Invoice, which contains customer data, shipping and billing data, order and item data, tax data, pricing data, and so on. In the database, or at least in* *the data model, these are all represented as separate entities. The reason for this, of course, is that most of this data is used for business purposes other than simply creating invoices. Having all of this data thrown together into a single Invoice table in the database would make it a lot easier to write the invoicing application, but a lot harder to use that data for anything else!*".

The above example clearly represents the problem, Object Oriented developers encapsulate data in a class as attributes, while these attributes usually stored in more than one database table.

However, the mismatch problem is much more than just object-relational problem. Ambler in [13] has categorized these problems as follows in Agile Software Development:

2.2.2.1 *Process impedance mismatch*

Agile Software Development (ASD) use iterative and evolutional approach to develop software system, while many within the data community still use serial or sequential approaches to develop software system. Therefore, there is a need for people involved in the data community to rethink about their approach in order to eliminate, or at least reduce, the mismatch problem [13].

2.2.2.2 *Technology impedance mismatch.*

There are two different paradigms, the object-oriented paradigm is based on software engineering concepts, while the second paradigm, data modeling is based on mathematical principles. Since both of the paradigms and concepts are different, then they will not work smoothly together [10] [13].

2.2.2.3 *The Cultural Impedance Mismatch*

The cultural impedance mismatch problem is something resulted from a political problem between the data community and the object community. Both sides claim their approach is better and the other approach as weakness points. For example, the object community claim that relational databases unable to store objects and deal with them, while the data community claim that object oriented design should be derived by data models [13].

As mentioned above, the greater the mismatch the deeper the problem. This is because there will be a need to write more code and more testing for this code to overcome this mismatch [13].

Therefore, the solution lies in the understanding of the of both communities' needs and requirements, and then tackle this problem in intelligent manner with acceptable trade-offs [13].

2.3 Summary

Developing software systems that are database dependent using Agile methodologies are sort of complex. Many trials and experiments have been done to reach a model that can help to make this easy.

Some of them, like Harriman et. al. in [19] did a nice experiment that clearly proved that traditional software development lifecycles (SDLC) like waterfall are not the best choice for software development nowadays. Moreover, the experiment showed that a means of iterative model could be used when designing and developing a data model.

In addition, Morien in [11] introduced the concept of the "Focal Entity". The "Focal Entity" is starting point that Morien has used to start designing the data model in iterative manner using the conceptual, logical, and physical design along with the process of these entities.

Ambler in [13] introduced the Agile Modeling concept, where the Agile-DBA need to exist and cooperate with the rest of the development team to evolutionary design database. He also shared L. Burns in [18] about the impedance mismatch problem and the need of an abstraction or encapsulation layer to solve the coupling and the impedance mismatch problem.

Database refactoring was a major concern for Ambler and Burns, they have mentioned different ways to deal with database refactoring. Ambler in [10] [13] has thought of database refactoring by using views and direct modifications to the database tables and how these modifications can be reduced by a means of encapsulation layer. On the other hand, Burns in [18] sees database refactoring as something belong to the encapsulation layer, it is must be solved there and the application objects should not be aware of this.

The theories mentioned above have introduced many ways to advance the database design from the being serial and sequential in nature to be more agile and iterative. However, none of them explained how to completely integrate the new iterative database design with an Agile software development technique such Scrum or XP. Therefore, in this thesis, we will design a complete model with detailed steps that clearly explain how to integrate the database design with

Scrum Agile technique and how to move the database design from being sequential and serialized to be iterative and evolutional.

CHAPTER III: Related Work

This chapter will discuss the related work that have been done in the field of Agile methodology, and how it is used to build iterative database design models. Furthermore, the chapter also discusses the database refactoring and other database technique that have been used to make database design more agile and iterative.

3.1 Agile usage with Database Design and Development

At first glance, Agile and Database seems not to be working well together, this has been mentioned by Ambler in [10]: “Unfortunately, most data-oriented techniques are serial in nature, relying on specialists performing relatively narrow tasks, such as logical data modeling or physical data modeling. ***Therein lies the rub – the two groups need to work together, but both want to do so in different manners***”. That’s why there are some, despite few, research about developing Agile Data techniques and models to solve this dilemma by trying to reach a model that helps both parties to work together effectively [11] [12] [19].

Moreover, developers usually focused on specific needs of the data, while Database Administrators (DBAs) and Data Architects focus on the overall data needs of the enterprise [13]. Such differences are normal since everyone is specialized in a certain field and this, as stated by Ambler because “*Specialists have a tendency to become too narrowly focused; they can work so hard to know everything there is to know about a small slice of software development that they can become oblivious of everything else*” [13]. That’s why Ambler has said “*We need to find the sweet spot between these two extremes*” [13].

However, Burns in [18] has another point of view when it comes to data modeling, he believes that “*designing our data around the real-world entities and attributes of the business, we help ensure that our data always has a valid business meaning and value, regardless of how it is used.*”, and therefore developers must be aware that data needs is beyond the scope of one application. Data must be modeled and designed in organization to serve more than one application and in many ways [18].

Furthermore, Burns in [18] believes that “*database should also encapsulate functionality that allows data to be safely updated (in accordance with the appropriate business rules) and quickly accessed in a business-relevant form*”. This means that database engines are not only for storing data, but also for data constraints, business rules as well as data management and processing.

That's why Amber in [13] has described the role of the Agile DBA as "*Agile DBAs apply the values, principles, and practices of AM to evolve their understanding of both the problem domain and of the solution space. They work closely with their teammates, creating models with them and learning new modeling techniques from them. They will create agile models to work through complicated issues or to communicate their work to others.*" [13]. That's it, the Agile DBA work closely and in iterative manner to develop the system with teammates. This means there traditional serial modeling mechanism has to be abandon.

On the way to achieve such iterative model for data modeling, Harriman et. al. in [19] discussed how to liberate the database development with Agile practices by implementing a test on how to develop a software system using Agile techniques and practices while the system is completely based on Database. They *wanted to establish the database as the early foundation for our application. To this end, senior data modeling and database design expertise was brought in*. [19]. They have completely modeled the system and the entity relationship diagram was produced so that *the breadth of the domain that was explored and captured in these artifacts was staggering: in some areas, the data model proceeded the actual application programming by over a year* [19].

Despite this, up-front modeling- should be good and promising, however, the actual results during the project was not at the expected level, since *with our database in place, we were free to focus on the application, making sure to hook it up to the database as we went. As you may guess, this didn't exactly work out as planned. In fact, it created some major difficulties, but along the way we learned some very valuable lessons and came to embrace ongoing database development as a fun part of learning the domain and building the application* [19].

The up-front design of the database was not the best choice Agile developers can use to develop their software, moreover, *as carefully as the database was initially designed, we inevitably came upon gaps.* [19]. In addition, and by time, the process has improved naturally when the developers started to do some simple database tasks themselves. The *benefits of distributed knowledge of the database design and team empowerment* [19] exceeded the expectations and the developers started to be able to handle complex database tasks and the *database was beginning to evolve and improve* [19].

Moreover, and as a result of the up-front design, an *Overloaded Entities* started to appear in the database design. This was because of *some entities containing a large number of fields* and these fields *were very speculative, being created based* *on attempting to anticipate future needs, rather than being driven by identified features and stories.* [19].

Despite the fact that these speculative fields were not as an immediate result of up-front design, they *were originally thought to provide value by allowing future requirements to be accommodated without necessitating schema changes. In reality, however, the costs associated with carrying the unused fields far outweighed the benefits. These costs manifested in several areas.* This is something related to bad design rather than as a result of using up-front design. Database Normalization uses fields resulted from the real analysis of the system not from speculative information.

The cost of the speculative fields become high, this was clear when a senior developer joined the team in the middle of the project and could not figure out the meaning of such fields. Also, during the retrospectives sessions, the speculative fields took more time by team members to discuss them, and the final result was to clean them up, and *to accommodate only features and requirements that were currently identified. The team refrained from speculative data and schema design.* [19].

Furthermore, the experiment that Harriman et. al. in [19] discussed two important points; Integrity Constraints and Data Concurrency Issues.

At first, the development team deferred the implementation of the integrity constraints into the database believing that integrity constraints will make more difficult to build the test units, and the application code will be structured and built in a way that data integrity will not be violated [19]. However, by time, they *did discover a few holes in our application logic that violated data integrity* [19] and this yields to a result that testing for data integrity with application unit tests would be very expensive and unnecessary since this functionality is well-tested by the RDBMS vendor. [19].

The Data Concurrency Issues has been left to the O/R Mapping framework used at the experiment done by Harriman et. al. in [19]. Anyhow, the results were encouraging and they *thought that this was something we could implement at low cost at any point in our project, so we postponed implementation in favor of other seemingly more important business stories. It seemed the agile thing to do: deliver business value with each iteration as prioritized by the customer* [19], but in reality, this was not the case.

After the project matured, the O/R mapping was not helpful and when they *did turn on optimistic locking, it caused the application to break at each one of these* *places.* The development team finally agrees that *concurrency is difficult to test in unit tests. This is another kind of protection better left to the database* [19].

A step further, Morien in [11] has talked about the concept of the Entity Modeling, and how this could be the basis of moving the database design one step forward towards incremental and iterative model.

Morien, and based on the Entity Modeling concept, has focused on the entity as a starting point for an iterative modeling. However, the word entity is defined by Morien as "*it could be said that an entity is highly cohesive, and very loosely coupled with other entities*" [11]. That's it, the entity should not be a standalone object.

The above concept of Entity Modeling helps to setup a basis on how to incrementally develop systems, such "*understanding of Entity Modeling meant that a system could be developed using entities, and then relationships, as the basis of iterations*" [11]. Morien has called this approach a "*Focal Entity Prototyping*", and which is mainly about to develop new systems. Maintenance of existing systems and database refactoring are beyond the scope of Morien's paper.

To clarify it more, Morien in [11] has developed a "*Tactical Model of Development was developed, based on the selection of an Entity to focus upon (thus "Focal Entity") and to elaborate through all of the various appropriate models – Conceptual (Entity Definition), Logical Data (Table Definition), Physical Data (Table Construction), Process (Forms and Reports)*" [11]. The model completely depends on defining what Morien has called "Focal Entity". The process is illustrated in figure (2.9).

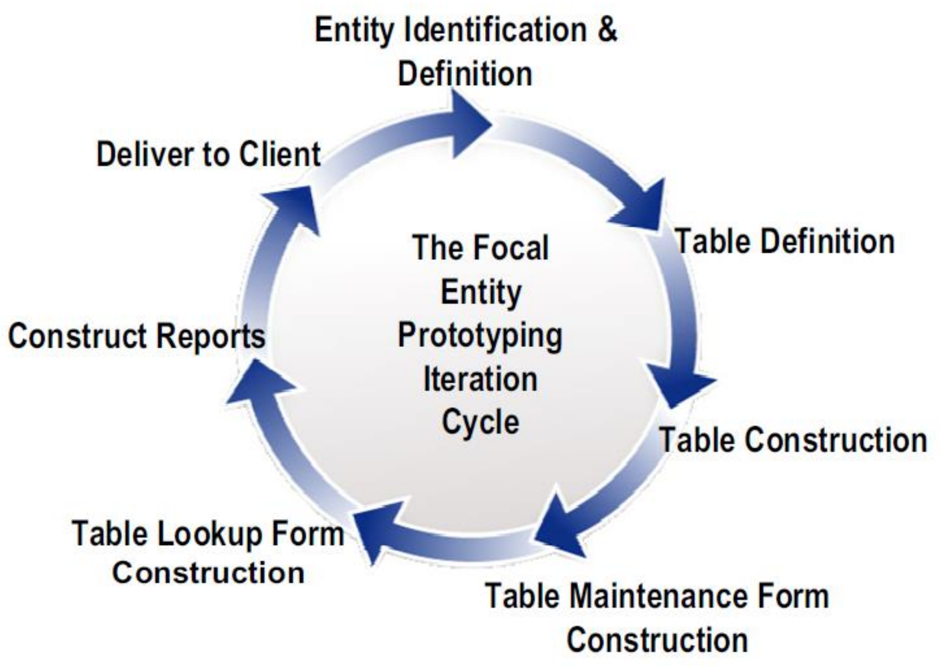

Entity Identification & Definition
Table Definition
Table Construction
Table Maintenance Form Construction
Table Lookup Form Construction
Construct Reports
Deliver to Client
The Focal Entity Prototyping Iteration Cycle

Figure 3.1: Tactical Model of Focal Entity Prototyping

[11]

Also, Morien in [11] has explained how to move from the conceptual model up to the process model. Figure (2.10) represents the concept of the model transformation.

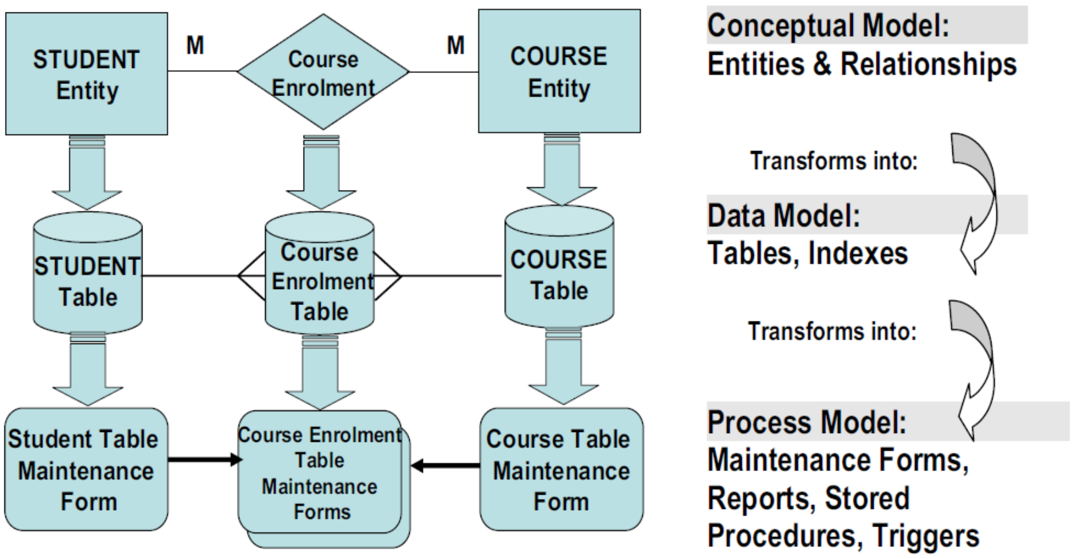

STUDENT Entity
M
Course Enrolment
M
COURSE Entity
STUDENT Table
Course Enrolment Table
COURSE Table
Student Table Maintenance Form
Course Enrolment Table Maintenance Forms
Course Table Maintenance Form
Conceptual Model: Entities & Relationships
Transforms into:
Data Model: Tables, Indexes
Transforms into:
Process Model: Maintenance Forms, Reports, Stored Procedures, Triggers

Figure 3.2: Transformation Between Models
[11]

Morien has mentioned in [11] that the changes could happen to the current design is a sort of adding something to it rather than altering it. If there is a need to alter the existing model, then this is a symptom to *a previously poor design!*

Moreover, Morien sees that his approach could fit well with the O/R models despite that "*there is clearly a problem of transforming Class Models into Relational Database Models; known as the Object-Relational Impedance Mismatch there is nothing in this problem that works against database evolution.*" [11]. However, he still sees that the best is to follow his approach as entity-based rather than class-based approach.

To link it more with Agile practices, Morien in [11] has suggested to use Scrum Agile practices along with the Focal Entity Approach. The Scrum "sprint" will best fit for the "Tactical Model" since it could be used to develop the sprint backlog.

3.2 The need for an Encapsulation Layer

As stated in section 2.4 The Impedance Mismatch Problem, the applications classes and objects need to access the tables stored in the database, but those normalized tables represent a logical view of the data and they have been normalized to add more business values such as eliminating data redundancy and optimize the performance. Because of such business values, there is a real need to keep those tables normalized in the database, and this means that application classes and objects should not access those tables directly; they have to access them through an abstraction or encapsulation layer that exists above those normalized tables [18].

The benefits of encapsulating the database access will be, but not limited to, the following [13]:

- Reduce coupling with a database and thus increase its maintainability and flexibility.
- Implements all data-related code in one place.
- Simplifies the job of application programmers.
- Enables application programmers to focus on the business problem and DBA(s) to focus on the database.
- Gives a common place, in addition to the database itself, to implement data-oriented business rules.
- Takes advantage of specific database and features, increasing application performance.

There is more than one strategy to encapsulate the database access, Amber in [13] has mentioned them as follows:

- **Brute force**

  In brute-force approach the business objects access data sources directly, typically by submitting SQL. For example, in Java applications, this is done via the Java Database Connectivity (JDBC) class library.

- **Data access objects**

  Data Access Objects (DAOs) encapsulate the database access logic required of business objects. The typical approach is for there to be one data access object for each business object, for example the Customer class would have a Customer_Data class. The Customer_Data class implements the SQL code required to access the database.

- **Persistence frameworks**

  A persistence framework, often referred to as a persistence layer, fully encapsulates database access from your business objects. Instead of writing code to implement the logic required to access the database, you instead define meta data that represents the mappings. So, if the Customer class maps to the T_Customer table, then part of the meta data would represent this mapping. Meta data representing the mappings of all business objects, as well as the associations between them, also needs to exist. Based on this meta data, the persistence framework generates the database access code it requires to persist the business objects.

- **Services**

  A service is an operation offered by a computing entity that can be invoked by other computing entities. Services are typically used to encapsulate access to legacy functionality and data, and there is a clear preference within the industry to build new applications following a Web-services-based architecture to facilitate reuse via system integration. Examples of services are Web Services, Stored Procedures, CORBA, and Distributed Component Object Model (DCOM).

3.3 Database Refactoring

A database refactoring is a simple change to a database schema that improves its design while retaining both its behavioral and informational semantics. [13]. However, when it comes to reality, it is not that simple as it looks like. In fact, database refactoring is very hard at the database level and all of this because of the coupling problem. Coupling is the "root of all evil" when it comes to database refactoring; the more things that your database schema is coupled to, the harder it is to refactor [13]. It is hard, as Ambler has stated in [13] because of:

- The application source code
- Other application source code
- Data load source code
- Data extract source code
- Persistence frameworks/layers
- Your database schema
- Data migration scripts
- Test code
- Documentation

And because of this, it is very hard to refactor a database when it is compared to code refactoring. Figure (2.8) represents the worst case scenario of database coupling.

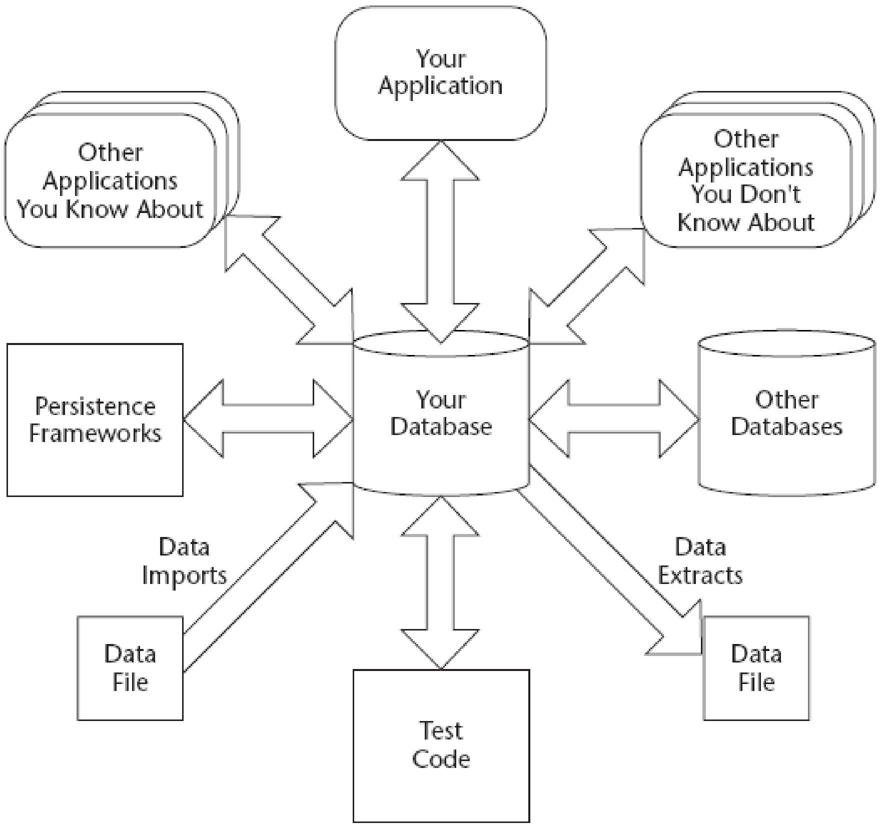


Figure 3.3: Database Coupled with Other Systems and Databases [13]

database refactoring requires a significant cultural change within your organization. Because database refactoring is an enabling technique of the agile data method many of the cultural issues for adopting database refactoring are the same ones that you face adopting the agile data method in general. These cultural issues include a serial mindset within many data professionals, resistance to change, and political inertia.

CHAPTER IV: Methodology and Implementation

This chapter discusses how the new model is developed and gives a detailed description on how to use it effectively. The new developed model is not an alteration to the known Agile models or any of its techniques. In fact, it is an addition to the existing Agile models with objectives to help development team to incrementally develop software systems that are database-dependent. Therefore, the Scrum framework has been used to design and test the new model.

4.1 General Overview of the Model

The new proposed model has been developed to help the development team to develop database-dependent applications using Agile techniques instead of the traditional up-front database design techniques.

However, the model did not sacrifice any of the features of the database design techniques such as normalization and reducing data redundancy. The model preserves all of these features during all of the model development phases.

To understand it better, we will first shortly explain the typical Scrum framework. In any typical Scrum framework, there are ceremonies called Scrum events. These events are linked with the Scrum team members. Each member of the team has a specific role with some activities to do. Also, each team member of a typical Scrum framework will develop, or participate in developing, the Scrum Artifacts.

4.2.1 New Model roles, events, and artifacts

On the other hand, the new developed Agile-Database model team members will work in conjunction with the typical Scrum team members mention in table (2.2). The key players in the new model are: Data Analyst, Database Administrator, Database Developer. Usually, there is a need for one Data Analyst and one Database Administrator. However, there could be a need for more than one Database Developer, and this completely depends on the size of the project that is being developed.

Table (3.1) below explains the Agile-Database team members' roles, activities, ceremonies or events they will participate into, and finally the resulted artifacts.

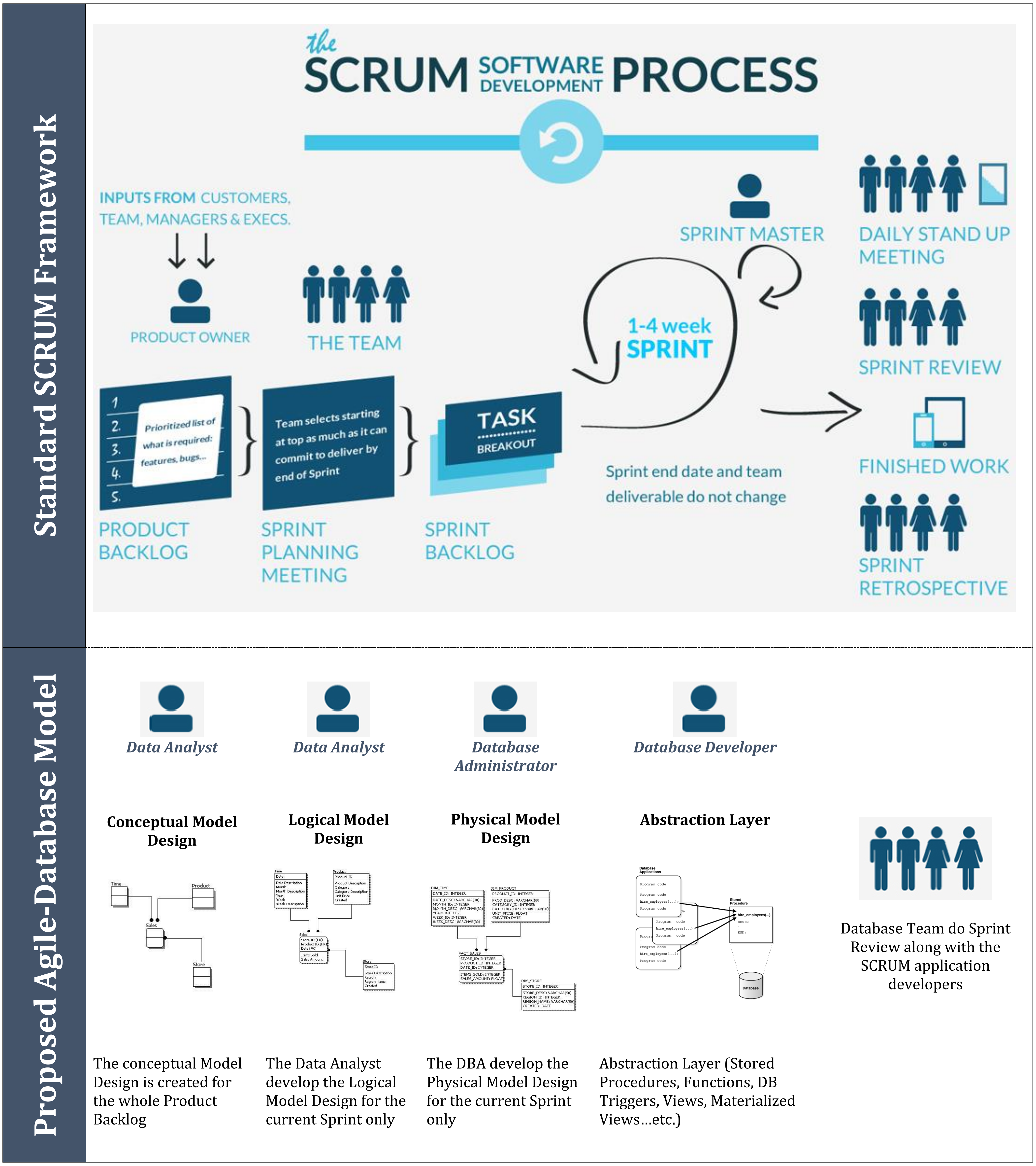
Table 4.1: Roles, Activities, Ceremonies, and Artifacts of the New SCRUM Agile-Database Process

Role	Definitions	Activities	Events	Artifacts
Data Analyst	- The Data Analyst is responsible for understanding the overall picture of the system during the Product Backlog development and then develop the Database Conceptual Logical Design - The Data Analyst will participate in the Sprint Planning Meeting in order to develop the Database Logical Model Design.	- Clearly understand the overall picture of the system; - Participate with the Product Owner to order the items in the Product Backlog to best achieve goals and missions and to ensure items are sorted in database coherent manner. - Ensuring that the Database Conceptual Model is visible, transparent, and clear to all, and shows what the Scrum Team will work on next. - Ensure the Database Logical Model is ready and understood for the current Sprint in action.	- Sprint Planning Meeting - Sprint Review Meeting	- Conceptual Model - Logical Model

Role	Definitions	Activities	Events	Artifacts
Database Administrator	The Database Administrator is responsible for developing the Database Physical Model	- Clearly understand the Logical Model developed by the Data Analyst. - Develop the Database Physical Model.	- Sprint Retrospective Meeting	- Physical Model
Database Developer	The Database Developer(s) is responsible for developing all the database objects in the Abstraction Layer.	- Develop - Document	- Spring Planning Meeting - Sprint Review Meeting	- Stored Procedures - Stored Functions - Packages - Database Triggers - Views Based on team's progress the burndown charts are developed by the Scrum Master.

4.2.2 New Model merged with typical Scrum Framework

Figure (4.1) below explains how the new model has been merged with each of the Scrum phases mentioned above.



Database Team do Sprint Review along with the SCRUM application developers

The conceptual Model Design is created for the whole Product Backlog

The Data Analyst develop the Logical Model Design for the current Sprint only

The DBA develop the Physical Model Design for the current Sprint only

Abstraction Layer (Stored Procedures, Functions, DB Triggers, Views, Materialized Views…etc.)

Figure 4.1: The New Agile-Database Model

4.2 Detailed Overview of the Model

As explained in figure (4.1), table (2.2), and table (4.1); each step in the Scrum framework has a corresponding step in the Agile-Database model. The steps at the Agile-Database process have to be executed in parallel with the normal Scrum Agile development process. That's it, it has to be merged and completely integrated into Scrum. To explain this more, the following is a complete description of each step:

- **Step One: Conceptual Design**
  Usually, application development needs a preparation to establish a general understanding of the new application goal and objectives. Therefore, a series of meetings are needed to clearly identify the project goal, objectives. During these meetings, The ***Data Analyst*** will start to form a general idea about the ***data*** needs for this application.

  During these meetings, a set of user stories will be developed, briefly analyzed, and prioritized. These user stories will form the ***Product Backlog*** of the Scrum framework. The ***Data Analyst*** must attend these meetings in which the ***Product Backlog*** will be developed since it will help to gain additional understanding of the business requirements. It is important that the ***Data Analyst*** should not try to model all the user stories since many of them might not be implemented. At this point, the focus should be on understanding the overall project goal and objectives along with the system boundaries. That's it, he must focus on the business needs and how the model will be help to achieve those business requirements. At this point, the Data Analyst can develop and produce a general ***Conceptual Model Design.*** Only names of the entities and their relationships are needed at this stage of work as explained in figure (4.2).

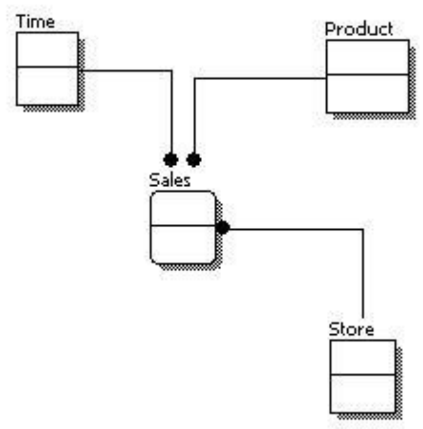


Figure 4.2: Typical Conceptual Model Design
[7]

During the environment setup for the project, the ***Data Analyst*** reviews the ***Conceptual Model Design*** with the development team before actual development starts. The idea here is ***not*** to create a comprehensive model that won't need to change; the idea is to ***agree*** on a model that is "***good enough***" to start development with.

Now the following applies, and repeated, for each Sprint in Scrum process used for application development:

- **Step Two: Logical Design**

  In the Sprint Planning Meeting, the team chooses a ***Sprint*** and its user stories (***Sprint Backlog***) from the ***Product Backlog***. The ***Data Analyst*** is a key player here and his opinion is crucial when it comes to choosing the ***Sprint's*** user stories. The ***Data Analyst*** has to do the best he can to preserve the ***Focal Entity*** concept and to be sure the user stories are **coherent at the data level**. The key point here is to choose user stories that can form a logically related and coherent group of entities from the ***Conceptual Model Design***. These user stories, together, should look like a small independent application as represented in figure (4.3).

  At this stage, the ***Data Analyst*** will design the Logical Model. The attributes of each entity will be defined. Also, entities primary and foreign keys are clearly identified as well.

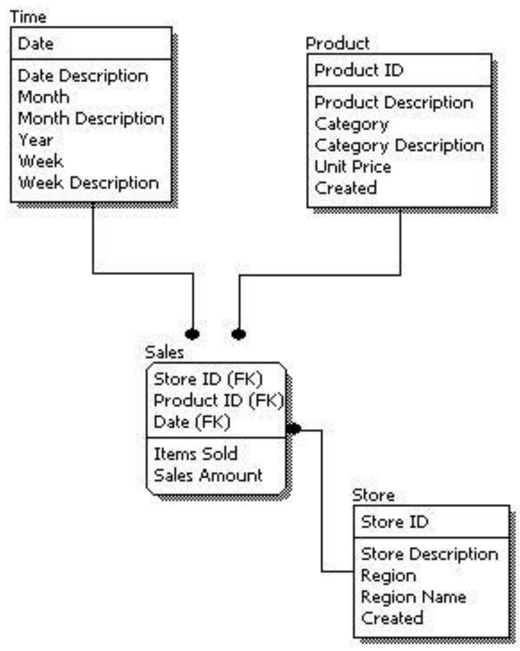


Figure 4.3: Typical Logical Model Design
[7]

As each user story is being discussed, the ***Data Analyst*** will attend this meeting with the Scrum team to investigate any business requirements related to this user story and to reflect it to the ***Logical Model***. The ***Data Analyst*** will just implement what is considered enough for this user story. However, the changes of the ***Logical Model*** could affect previous implemented user stories, therefore, the Scrum developers must work closely with the ***Database Developer*** to ensure they do not bypass the ***Abstraction Layer*** implemented by the ***Database Developer.***

- **Step Three: Physical Design**
  Once the user stories are discussed and agreed to be within the Sprint, the ***Data Analyst*** will have another discussion with the ***Database Administrator*** to convert the current ***Logical Model*** to the most appropriate ***Physical Model***. As said in the previous step, there could be some changes of the ***Physical Model*** as a result of changes in the ***Logical Model***. Therefore, the Scrum developers must not bypass the ***Abstraction Layer*** created by the ***Database Developer*** since the code and the objects in the ***Abstraction Layer*** will do the communication and the manipulation of the data between the application layer and the database physical layer.

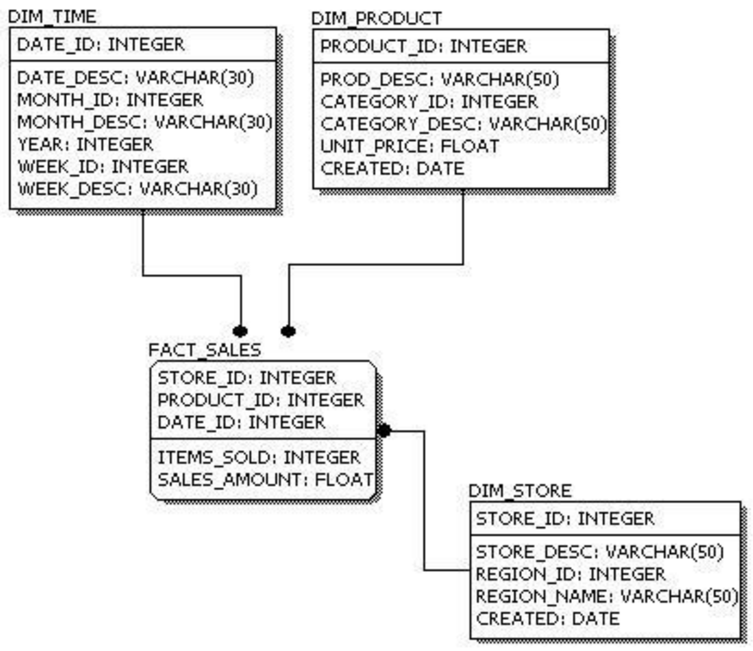


Figure 4.4: Typical Physical Database Model
[7]

- The ***Database Administrator*** will convert the ***Logical Model*** designed by the ***Data Analyst*** to the ***Physical Model*** as represented in figure (3.4). Only the design of the tables will be implemented at this stage. The ***Database Administrator*** is the only authorized person to decide the physical implementation of the database tables (Heap Table, Index Organized Tables, Clustered Tables…etc.).

-
- **Step Four: Abstraction Layer Implementation**
  After creating the ***Physical Model***, the ***Database Developer*** start working on creating the ***Abstraction Layer***. The ***Database Developer*** will code the necessary Stored Procedures, Functions, Packages, and Database Triggers as represented in figure (3.5). These Database Objects will be used by the rest of the Scrum developers to interact with the ***Physical Model*** created earlier by the ***Database Administrator***.

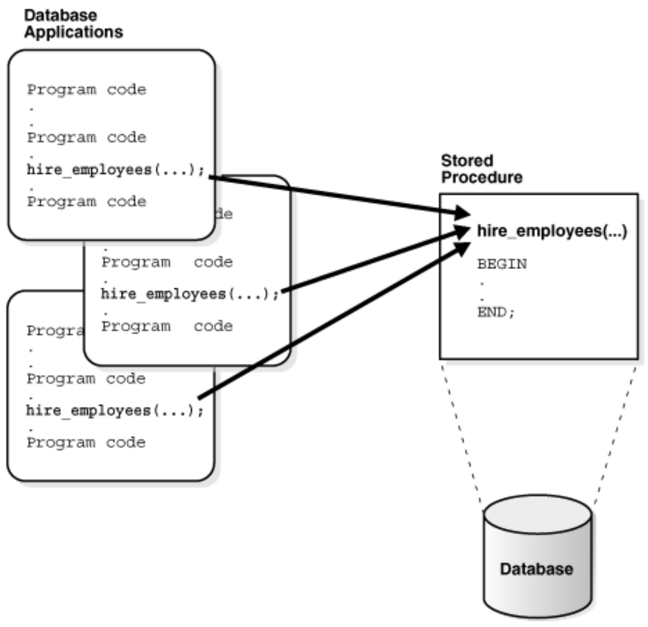


Figure 4.5: Example of Database Stored Procedure
[20]

The ***Database Developer*** has to follow the guidelines in section 4.3.1 when creating these Database Objects. The objectives of these guidelines are to make them as flexible as possible when some changes are requested.

Furthermore, and to respond to the reporting needs of the system, the **Database Developer** will be responsible for creating Database Views and Materialized Views. These views are part of the ***Abstraction Layer***, and their purpose is to hide the ***Physical Model*** from being directly accessed by the Scrum developers. The real benefits of these views will be when there is a need to merge two tables into one table, or even when there is a need to split one table into two physical tables. The views will make these changes hidden beneath the Abstraction Layer and there will be no needs for any changes to be done at the application level by the Scrum developers.

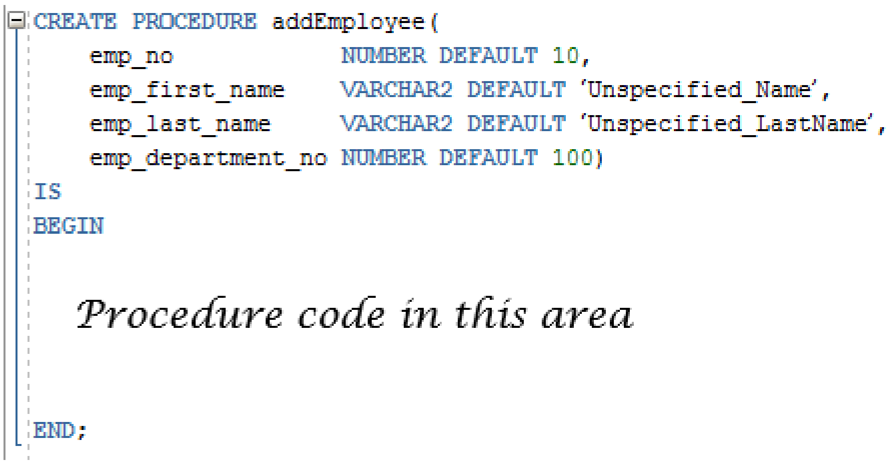
4.3.1 *Guidelines for developing Stored Database Objects*

- The following are the guidelines [21] the ***Database Developer*** has to follow when he starts developing the Database Objects in the ***Abstraction Layer***.

    - **Stored Procedures:** Besides following a naming standards, the ***Database Developer*** has to consider the following when he starts coding Stored Procedures:

        - Return Values: Stored Procedures do not return values, and the ***Database Developer*** must avoid using call by reference parameters unless there is a real need of it. For example, there might be a real need to use call by reference parameters when sending Large Objects data (LOB) to the procedures, in this case, using call by value parameter will result in deep copy of all the data inside the LOB object which size is usually in Gigabytes.

        - Parameters Default Values: This is a crucial point about how Stored Procedures can add more flexibility to the ***Abstraction Layer***. The default values of the parameters will help to make the procedures more flexible when there is a need to respond to requirements changing. To clarify this more, let's look at the following example using Oracle Database notation:



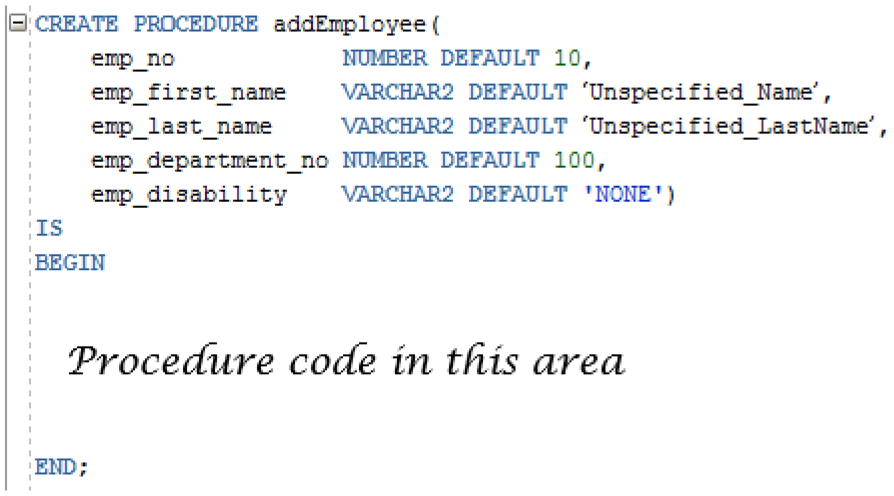
```
CREATE PROCEDURE addEmployee(
    emp_no              NUMBER DEFAULT 10,
    emp_first_name      VARCHAR2 DEFAULT 'Unspecified_Name',
    emp_last_name       VARCHAR2 DEFAULT 'Unspecified_LastName',
    emp_department_no NUMBER DEFAULT 100)
IS
BEGIN

   Procedure code in this area


END;
```

Figure 4.6: Example of Parameters Usage with Stored Procedure

If this procedure is used during application development by the Scrum developers, and after a while new requirements arise; there a new type of employees needed to deal with. These employees have disabilities and they need special treatment since they are either mute or blind, and there is a need to store this piece of information where the value '*B*' stands for BLIND, '*M*' stands for

MUTED, and '*NONE*' stands for normal employees with no disabilities.

Using default values, and after adding the new parameter *emp_disability* with a default value '*NONE*', the new Stored Procedure will be just like the following:



```
CREATE PROCEDURE addEmployee(
    emp_no              NUMBER DEFAULT 10,
    emp_first_name      VARCHAR2 DEFAULT 'Unspecified_Name',
    emp_last_name       VARCHAR2 DEFAULT 'Unspecified_LastName',
    emp_department_no   NUMBER DEFAULT 100,
    emp_disability      VARCHAR2 DEFAULT 'NONE')
IS
BEGIN

  Procedure code in this area

END;
```

Figure 4.7: Stored Procedure with New Parameter with Default Value

The default values will allow the existing code of the Stored Procedures to be called without any modifications. The Scrum developers who invoked the procedure by typing

addEmployee(1234, 'MASA', 'EM', '1024');

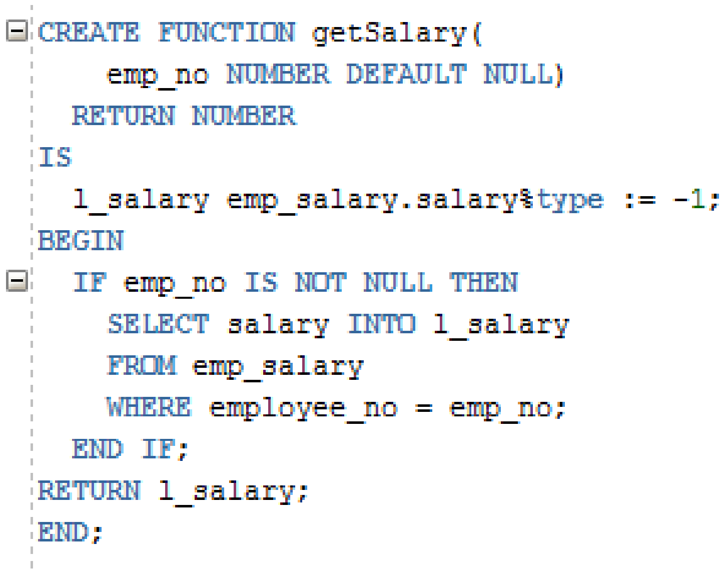
in their code do not have to change anything because of new parameter has a default value, and this default value will be stored using the above notation used to invoke the procedures. This will greatly add more flexibility of the ***Abstraction Layer*** and adhere to the Agile standards in accepting new changes.

- **Stored Functions**: What applies to the *Stored Procedures* applies to function but with the following things to consider:

  - Return Value: *Stored Functions* must return values However; Functions are not allowed to use call by reference parameters by any means. The only return value from the function must be in the *RETURN* statement

  - Parameters Default Values: Same as *Stored Procedures* mentioned above.

The following is an example of a typical *Stored Function* developed using Oracle Database notations.



```
CREATE FUNCTION getSalary(
    emp_no NUMBER DEFAULT NULL)
  RETURN NUMBER
IS
  l_salary emp_salary.salary%type := -1;
BEGIN
  IF emp_no IS NOT NULL THEN
    SELECT salary INTO l_salary
    FROM emp_salary
    WHERE employee_no = emp_no;
  END IF;
RETURN l_salary;
END;
```

Figure 4.8: Example of Stored Function with Parameters

- **Stored Packages:** Some database engines, like Oracle Database, offer the database developers to use Stored Packages which could further add more flexibility, functionality, and performance to the software being developed. The following features of Stored Packages have to considered when developing the ***Abstraction Layer***:

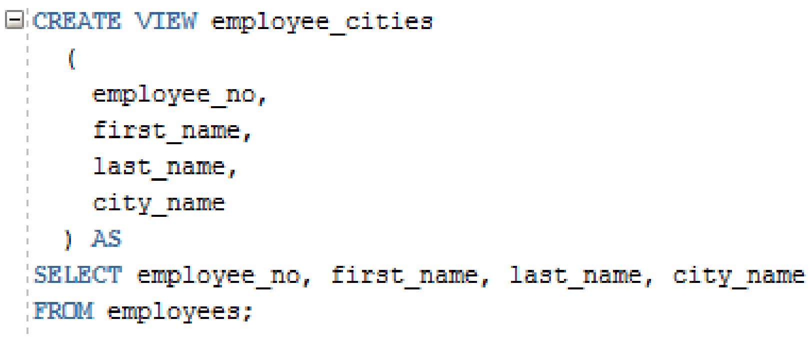
  - Logical Container: *Stored Packages* are logical container for the *Stored Procedures* and Functions. They offer the ***Database Developer*** an option to gather functional-related code together.

  - Encapsulation and Information Hiding: Since *Stored Packages*, especially in Oracle Database, are divided into main parts; *Package Specification* and *Package Body*. Package Specification store the name of the procedures and functions along with their parameters only. Whereas, the *Package Body* stores the code of the procedures and functions defined in the *Package Specification*, and also there could be a private procedures and functions for the package internal use.

  - Overloading: By using *Stored Packages*, the ***Database Developer*** can further extend the flexibility of the ***Abstraction Layer*** by using Overloading. Overloading allows the ***Database Developer*** to invoke the same procedure or function name, but with different code to execute. The *Stored Package* can distinguish this by differentiating them using the procedure or function signature (number and type of parameters).

- **Database Triggers:** Most database engines have the ability to create database triggers. However, they should not be used in any transactional logic. The only thing *Database Triggers* should be used in is for value auditing. When there is a need to audit the values changing, the best place is to plug the code in the ***Abstraction Layer***, and the best place in the ***Abstraction Layer*** is the *Database Triggers*. The code itself could be in a *Stored Procedure*, but its invocation is best to be from a *Database Trigger*.

- **Database Views:** Views, which is built into the database engines, offer a great option of flexibility to the ***Abstraction Layer*** when the developers want to respond to the reporting needs. It is a lot better to build the report using an *SQL SELECT* statement against a *VIEW* rather than *TABLES*. To explain more, consider, as a response to business requirements, there is a need to split as database table called "*A*" into two new tables called "*A*" and "*C*". If the report developed using a *VIEW*, then there is not any need to change the report's *SELECT* statement. All the changes will be done in the ***Abstraction Layer*** by changing the *VIEW*'s *SELECT* statement to retrieve the data by joining the new two tables "*A*" and "*C*".



```
CREATE VIEW employee_cities
   (
     employee_no,
     first_name,
     last_name,
     city_name
   ) AS
SELECT employee_no, first_name, last_name, city_name
FROM employees;
```

Figure 4.9: Database View Retrieve Data from One Table Source

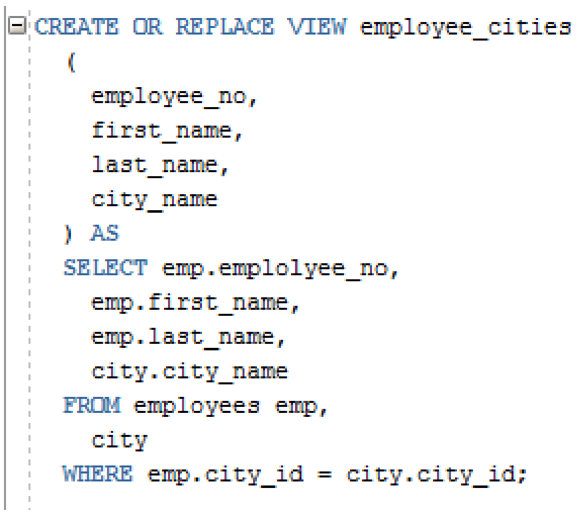

```
CREATE OR REPLACE VIEW employee_cities
  (
    employee_no,
    first_name,
    last_name,
    city_name
  ) AS
  SELECT emp.emplolyee_no,
    emp.first_name,
    emp.last_name,
    city.city_name
  FROM employees emp,
    city
  WHERE emp.city_id = city.city_id;
```

Figure 4.10: Database View Altered to Retrieve Data from Two Tables

- **Materialized Views:** On contrast to normal database views, which do not store data, *Materialized Views* store data and consume storage. Usually, *Materialized Views* are used enhancing reporting performance and speed by de-normalizing the data. However, *Materialized Views* are not database tables, they are still preserve the concept of the database views but with data storage, they retrieve their data from the underlying database tables, and they can be refreshed whenever there are data changes in their underlying tables.

  Because of the features of the *Materialized Views*, they can be an important player in the ***Abstraction Layer***. When there is a complaint from the software users about slow report performance, the only change will happen in the ***Abstraction Layer*** in the database engine. The report that was based on normal database view will be now based on a *Materialized View*. What's make it more attractive is that indexes can be used with *Materialized Views* to better enhance the performance.

  Adapting the concept of the ***Abstraction Layer*** explained above will greatly help Scrum team, in cooperation with the ***Data Analyst***, ***Database Administrator***, and the ***Database Developer***, to respond to new or changed business requirements effectively and efficiently. Moreover, they will keep the data changes to the minimum as possible, and most of the changes related to the data are implemented in the ***Abstraction Layer*** inside the database engine.

4.3 Evaluation

In order to evaluate the proposed Agile-Database Model, we prepared the following:

- **Proposed Software System:** A pilot system, which is a proposed restaurant model, has been chosen for development in order to evaluate the new model.

- **Development Teams:** Two teams of developers have been prepared. Team (**A**) will develop the proposed system using up-front database design, while team (**B**) will develop the proposed system using the new Agile-Database model.

- **Evaluation Criteria:** Two evaluation criteria have been developed to help measure the performance of the two teams. Table (4) will be used by team (**A**), and table (5) will be used by team (**B**).

Table 4.2: Evaluation Criteria for Team (A)

#	Evaluation Item	Value	Description
1.	Time needed to finalize the database ERD		Overall working hours consumed by the all the stakeholders to develop the ERD
2.	Productivity rate		Number of user stories accomplished
3.	Customer engagement during the project		From 1 to 10. 1 is rare, 10 is very engaged Use the number of meetings held with the customer.
4.	Customer satisfaction		From 1 to 10. 1 is unsatisfied, 10 is very satisfied
5.	Flexibility to adapt changes		From 1 to 10. 1 is hard, 10 is very easy
6.	Divergence of what actually required compared to what actually developed		Number of user stories cancelled, changed, added to the system.
7.	Cost of change at the database level		The formula is: 1x for any change done at the Conceptual Model 2x for any change done at the Logical Model 4x for any change done at the Physical Model 1x for any change done at code
8.	Over all time needed for the project		Overall working hours consumed by the all the stakeholders to develop the system

Table 4.3: Evaluation Criteria for Team (B)

#	Evaluation Item	Value	Description
1.	Understanding the new model		From 1 to 10. 1 is hard, 10 is extremely easy
2.	Easiness of the new model usage		From 1 to 10. 1 is hard, 10 is extremely easy
3.	Productivity rate of the new model		Number of user stories accomplished
4.	Customer engagement during the project		From 1 to 10. 1 is rare, 10 is very engaged Use the number of meetings held with the customer.
5.	Customer satisfaction		From 1 to 10. 1 is unsatisfied, 10 is very satisfied
6.	Flexibility to adapt changes		From 1 to 10. 1 is hard, 10 is very easy
7.	Divergence of what actually required compared to what actually developed		Number of user stories cancelled, changed, added to the system.
8.	Cost of change at the database level		The formula is: 1x for any change done at the Conceptual Model 2x for any change done at the Logical Model 4x for any change done at the Physical Model 1x for any change done at code 1x for any change done at the Abstraction Layer
9.	Over all time needed for the project		Overall working hours consumed by the all the stakeholders to develop the system

4.4 Evaluation Process

The evaluation process tries to measure the two teams' performance using items in table (4.2) and table (4.3) mentioned above. The values of the two tables will be compared to each other to measure the overall performance and efficiency of the new model.

Since it cost more when we do modifications at the database level when the database is growing up, the formula in item 8 "Cost of change at the database level" has been built to comply with this fact. The formula considers that more cost is needed when modifications is done at the logical layer or even the physical layer.

To accomplish this, and right after forming the teams' members; a number of sessions will be held with each team individually. The sessions objectives will be as follows:

Table 4.4: Evaluation Sessions

#	Session Description	Team	Comments
1.	Describe the new Agile-Database model	B Only	Team (B) only will implement the software using the new proposed model. Team (B) should answer item No. 1 in table (5).
2.	Describe the proposed system to be developed	A and B	It is essential for each team to understand the new system that will be developed in order to produce the Product Backlog.
3.	Review the Conceptual Model	B Only	Team (A) is using up-front design. The team should develop a complete Physical Design for the whole system.
4.	Review the Sprint Logical Model	B Only	Team (A) is using up-front design. The team should develop a complete Physical Design for the whole system.
5.	Review the Sprint Physical Model	A and B	Team (A) should develop a complete Physical Design for the whole system; item 1 in the table (4) should be answered here. On the other hand, team (B) should develop a Physical Model for the user stories for the Sprint that is currently being developed.
6.	Review each Sprint	A and B	This is a Scrum sprint review event. The idea is to review the answers to the items: 2, 3, 4 ,5, 6, 7 in the evaluation table (4) for team (A). And the items: 3, 4, 5, 6, 7, 8 in the evaluation table (5) for team (B)
7.	Review the Final Product	A and B	Final product review; team (A) should answer item No. 8 in table (4) and team (B) should answer items No. 2 and 9 in table (5)

The above sessions are required and essential by the researcher. However, other sessions could be held upon the request of the teams who will develop the system in order to clarify more issues and to discuss the feedback with the teams.

CHAPTER V: Results

As mentioned in the ***Evaluations*** section in **Chapter IV: Methodology and Implementation**, two teams were composed to develop the proposed system. Each team has been handed the system's user stories to start developing the system and sessions were conducted with each team separately to ensure the understanding of the system. This chapter explains how we test the model, review each team results in each sprint and at the end of the evaluation process, and discuss the findings of the results.

5.1 Experiment Setup

After forming the two teams, a set of rules have been agreed on in order to proceed with the experiment. The following sections will explain how the experiment has been accomplished and also discuss the findings resulted from the work of the two teams.

5.1.1 About the Teams

Team (A) Background

The team is led by Alaa Alsalehi, Alaa loves the smell of code, writing, reading, refactoring and fixing bugs. Alaa loves to help people who want to achieve perfection and create helpful projects, reach success in their business. Alaa is working in IUG as a team leader for student portal, one of the main systems that serves more than 20,000 students. Alaa runs his own start-up -ZAKI- which is a platform for people whom loves cooking to share their knowledge and experience.

Alaa loves mobile development and adding value to people lives providing solutions for their daily problems. Alaa recently focus on Android OS, he is the leader and developer of many heterogeneous - web and mobile - projects. He has a good experience in Google data API specially YouTube API.

Team (B) Background

The team is led by Mohammed Riyad El Khoudary who is a Computer Engineer in Palestine, he got his B.sc and M.sc in computer engineering from the Islamic University of Gaza. Mohammed is fascinated about software engineering and programming, he has written enterprise software for many famous institutes and companies here in Gaza. Mohammed is currently working as a Technical Development Manager at ExaServe company, leading more than 20 developers in challenging software.

5.1.2 OVIPs Restaurant System's User Stories and Scenarios

It is an agile project after all, and therefore, the user stories are crucial for the two teams to start developing the proposed system.
Our proposed system Only for VIPs (OVIPs) is a software system that manages restaurant requests such as customers' orders and table reservation. The system has been derived from a dedicated Internet website [22] for database models, and it has been chosen because it is simple but yet contains adequate number of database tables that can serve the purpose of this research.

The system requirements have been expressed in the following user stories and their scenarios:

001	**Customer Registration**
As a customer, I want to register in the OVIPs Restaurant Database so I can use the restaurant system.	

- Scenario #1: The customer is new and never registered before

  Given a customer who has never been registered in the system, ***when*** he enters his information as indicated by the registration form, ***then*** the customer will receive a new username and password to logon to the system.

- Scenario #2: The customer is already registered and has a username

  Given a customer who has been registered in the system, ***when*** he tries to create a new account with the same username, ***then*** the system will display an error message explaining that this username is already taken and offer him either to create a new account or reset the password for this account by sending a new one to the email associated with this account.

002	**Customer Booking**
As a registered customer in the OVIPs Restaurant System, I want to book a table at a specific date and time so I can visit the restaurant the time that suites me.	

- Scenario #1: Booking online

  Given a customer who has been registered in the system, ***when*** he uses the restaurant online system to book a table at a specific date and time, ***then*** the customer will be notified if his booking is successful or not.

- Scenario #2: Booking by Phone

  Given a customer who has been registered in the system, ***when*** he calls the restaurant to book a table at a specific date and time, ***then*** the customer will be notified by phone and an SMS to confirm his booking is successful or not.

003	Customer Book Cancelling
As a registered customer in the OVIPs Restaurant System, I want to able to cancel the booking I had made before so I can come back later.	

- Scenario #1: Online Book Cancelling

  Given a customer who has been registered in the system, ***when*** he uses the restaurant online system to cancel a book that is not due yet, ***then*** the customer will be notified if his book cancellation is successful or not.

- Scenario #2: Cancelling a Book by Phone

  Given a customer who has been registered in the system, ***when*** he calls the restaurant to cancel a book that is not due yet, ***then*** the customer will be notified by phone and SMS to confirm his book cancellation is successful or not.

004	Customer Book Changing
As a registered customer in the OVIPs Restaurant System, I want to able to change the booking I had made before so I pick another suitable time for me.	

- Scenario #1: Online Book Changing

  Given a customer with who has been registered in the system, ***when*** he uses the restaurant online system to change his booking for a table at a specific date and time to a new date and/or time, ***then*** the customer will be notified if his book changing is successful or not.

- Scenario #2: Changing a Book by Phone

  Given a customer with who has been registered in the system, ***when*** he calls the restaurant to change his booking for a table at a specific date and time to a new date and/or time, ***then*** the customer will be notified by phone and SMS to confirm his booking changing is successful or not.

005	Customer Orders
As a registered customer in the OVIPs Restaurant System, I want to review the available menus so I can order the food, drink, or sweets that I like more.	

- Scenario #1: New Orders

  Given a customer with who has been registered in the system, ***when*** he reviews the menu and request a new order for food and/or drink, ***then*** the customer can follow up the status of the order by the restaurant online system and can see how much the order will take to be ready.

- Scenario #2: Changing an Order

  Given a customer with who has been registered in the system, ***when*** he tried to add, drop, or change a quantity of some items in his order, ***then*** the customer will be able to know if his request is accepted or it was too late to change the order.

- Scenario #3: Cancelling an Order

  Given a customer with who has been registered in the system, ***when*** he tried to cancel an order, ***then*** the customer will be able to know if his request is accepted or it was too late to cancel the order.

006	Adding Staff Members
As a restaurant manager for the OVIPs Restaurant System, I want to add staff members and their relative information so I can assign each of them role description.	

- Scenario #1: Adding New Staff

  Given a restaurant manager with the required privileges, ***when*** he adds new staff member to the system, ***then*** the manager will be able to assign this staff an existing role from a list of available roles in the system.

007	Staff Member Management
As a restaurant manager for the OVIPs Restaurant System, I want to be able to change the staff roles so I can better manage them.	

- Scenario #1: Editing Existing Staff Members Roles

  Given a restaurant manager with the required privileges, ***when*** he edits an existing staff member, ***then*** the manager will be able to assign this staff member a new role from an existing role from a list of available roles in the system.

- Scenario #2: Editing Existing Staff Members Information

  Given a restaurant manager with the required privileges, ***when*** he edits an existing staff member, ***then*** the manager will be able to change the member information in the system such contact and address information.

008	Menus Management
As a restaurant manager for the OVIPs Restaurant System, I want to be able to create new menus and manage current menus so I can better serve my customers	

- Scenario #1: Creating New Menus

  Given a restaurant manager or a staff member with the required privileges, ***when*** he creates a new menu in the system, ***then*** he can add new menu items to this menu.

- Scenario #2: Editing Existing Menus

  Given a restaurant manager or a staff member with the required privileges, ***when*** he edits an existing menu, ***then*** the manager will be able to add new menu items, delete menu items, or move menu items from menu to menu.

- Scenario #2: Deleting an Existing Menus

  Given a restaurant manager or a staff member with the required privileges, ***when*** he deletes an existing menu, ***then*** the system will no longer display this menu and its items will be free to be added to other menus.

009	View Orders
As a restaurant manager for the OVIPs Restaurant System, I want to be able to view tables' orders sorted by date or customers so I can track them in case I need to.	

- Scenario #1: Listing Orders

  Given a restaurant manager or a staff member with the required privileges, ***when*** he queries for a specific order by customer name, date, or table, ***then*** he can see the orders' related information such as, but not limited to, who order it, paid or not, date and time of the order.

010	**Editing Orders**
As a restaurant manager, or dedicated staff member in the OVIPs Restaurant System, I want to be able to edit tables' orders so I can respond to the customers' needs.	

- Scenario #1: Editing Orders

  Given a restaurant manager or a staff member with the required privileges, ***when*** he edits a specific order for a specific customer, ***then*** he can change this order by any means such as adding more items to the order, dropping items from the order or even cancelling list of items from this order.
- Scenario #2: Deleting Orders

  Given a restaurant manager or a staff member with the required privileges, ***when*** he deletes a specific order for a specific customer, ***then*** the customer will no longer pay for this order, and the system will indicate that this order was deleted by a privileged user and store that user information with this transaction.

011	**Orders Discount**
As a restaurant manager for the OVIPs Restaurant System, I want to be able to give discounts for any tables' orders so I can better market for my restaurant.	

- Scenario #1: Editing Orders

  Given a restaurant manager or a staff member with the required privileges, ***when*** he gives a specific customer a discount for a specific order, ***then*** this discount will be directly reflected to the order balance and the customer will be able to see this discount using the restaurant system.

5.1.3 *Environment and Tools:*

The development environment for the two teams are as follows:

- **Database Engine**: Oracle Database 11g [23] has been used as the underlying database engine used to develop the system. The two teams have a remarkable experience with Oracle database, and this is important in order to rule out any factor that could affect the development of the system except the complexity of the business rules that needed to be implemented.
- **Modeling Tool**: Oracle SQL Developer [24] has been used to model the database and to develop the Abstraction Layer by Team (B). Oracle SQL Developer is completely integrated with Oracle database and it facilitate the creation of the tables, views, and stored procedures.
- **Development Tool**: NetBeans [25] has been used as the Java [26] development tool. NetBeans is completely integrated with Oracle database engine and it is one of the famous Java development tools.

5.2 Teams Results and Findings

5.2.1 *Team (A) Results*

The team starts developing the system using the traditional up-front database design techniques along with Scrum agile methodology. Based on the ***Evaluation Process*** section in **Chapter III: Methodology and Implementation**, table (6) represents team (A) results:

Table 5.1: Team (A) Results

#	Evaluation Item	Value	Description
1.	Time needed to finalize the database ERD	2	Overall working ***hours*** consumed by the all the stakeholders to develop the ERD
2.	Productivity rate	11	Number of user stories accomplished
3.	Customer engagement during the project	8	From 1 to 10. 1 is rare, 10 is very engaged Use the number of meetings held with the customer.
4.	Customer satisfaction	7	From 1 to 10. 1 is unsatisfied, 10 is very satisfied
5.	Flexibility to adapt changes	4	From 1 to 10. 1 is hard, 10 is very easy

6.	Divergence of what actually required compared to what actually developed	7	Number of user stories cancelled, changed, added to the system.
7.	Cost of change at the database level	0 Conceptual 6 Logical 8 Physical 0 Code 0 Abstraction Layer	The formula is: 1x for any change done at the Conceptual Model 2x for any change done at the Logical Model 4x for any change done at the Physical Model 1x for any change done at code 1x for any change done at the Abstraction Layer
8.	Over all time needed for the project	25 hrs.	Overall working hours consumed by the all the stakeholders to develop the system

5.2.2 *Team (B) Results*

The team starts developing the system using the traditional up-front database design techniques along with Scrum agile methodology. Based on the ***Evaluation Process*** section in **Chapter IV: Methodology and Implementation**, the following results were obtained:

Table 5.2: Team (B) Results

#	Evaluation Item	Value	Description
1.	Understanding the new model	9	From 1 to 10. 1 is hard, 10 is extremely easy
2.	Easiness of the new model usage	8	From 1 to 10. 1 is hard, 10 is extremely easy
3.	Productivity rate of the new model	11	Number of user stories accomplished
4.	Customer engagement during the project	8	From 1 to 10. 1 is rare, 10 is very engaged Use the number of meetings held with the customer.
5.	Customer satisfaction	8	From 1 to 10. 1 is unsatisfied, 10 is very satisfied
6.	Flexibility to adapt changes	8	From 1 to 10. 1 is hard, 10 is very easy

7.	Divergence of what actually required compared to what actually developed	7	Number of user stories cancelled, changed, added to the system.
8.	Cost of change at the database level	3 Conceptual 2 Logical 1 Physical 8 Code 4 Abstraction Layer	The formula is: 1x for any change done at the Conceptual Model 2x for any change done at the Logical Model 4x for any change done at the Physical Model 1x for any change done at code 1x for any change done at the Abstraction Layer
9.	Over all time needed for the project	16 hrs.	Overall working hours consumed by the all the stakeholders to develop the system

Since team (B) followed an incremental database design, the following are the resulted ERD for the pilot system used in this thesis.

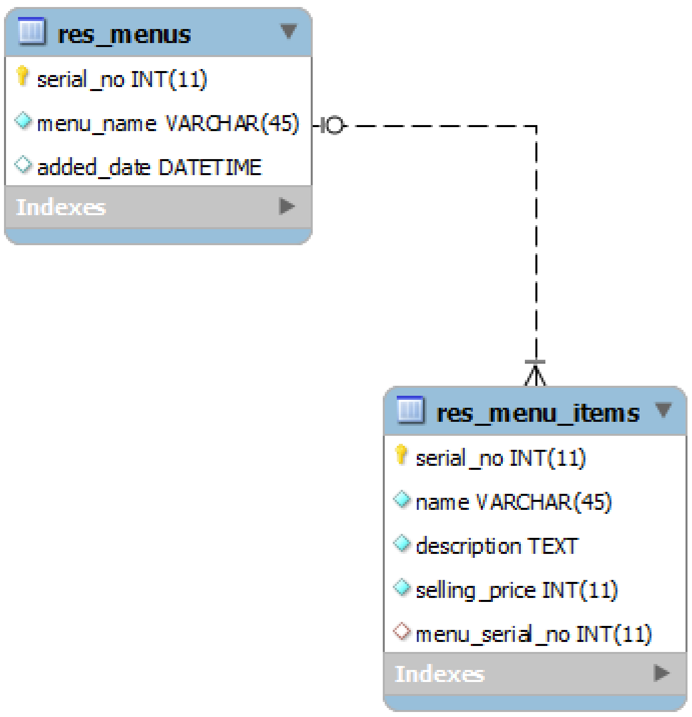


Figure 5.1: Incremental ERD (1) for the pilot system

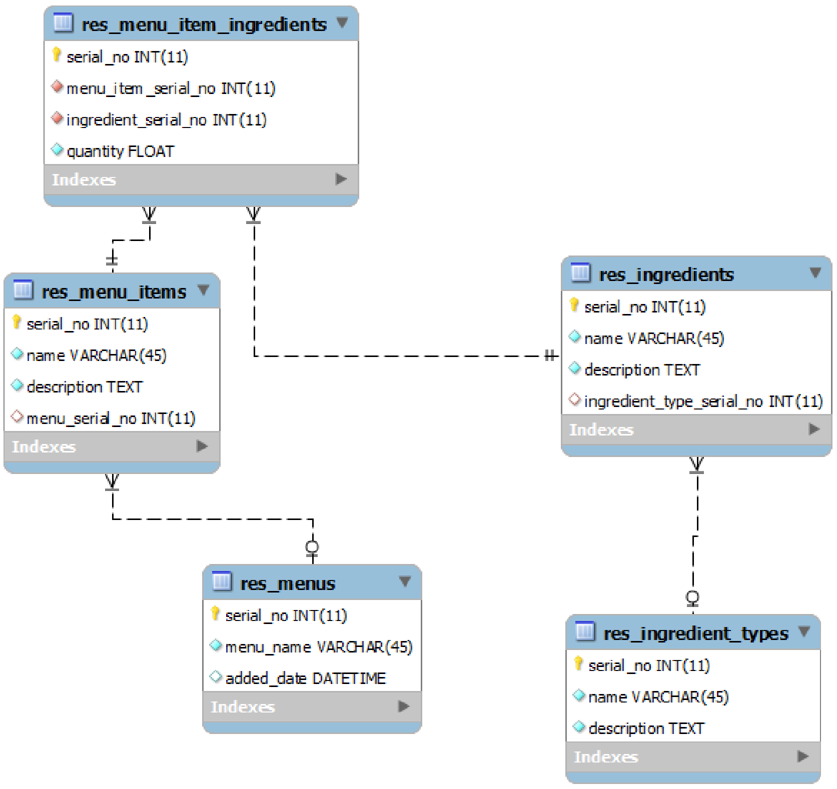


Figure 5.2: Incremental ERD (2) for the pilot system

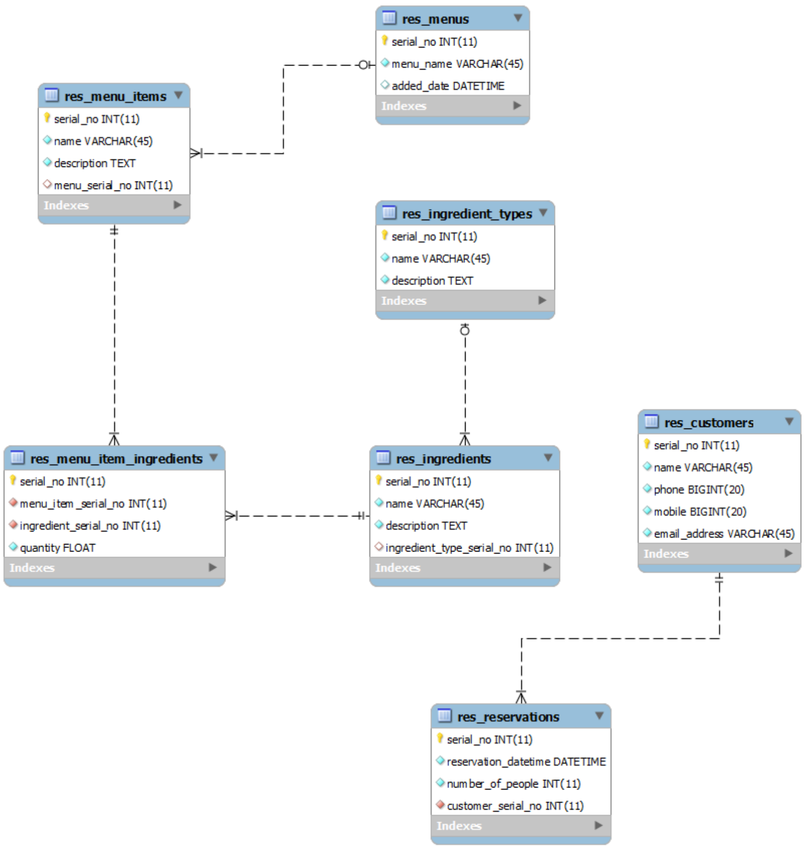


Figure 5.3: Incremental ERD (3) for the pilot system

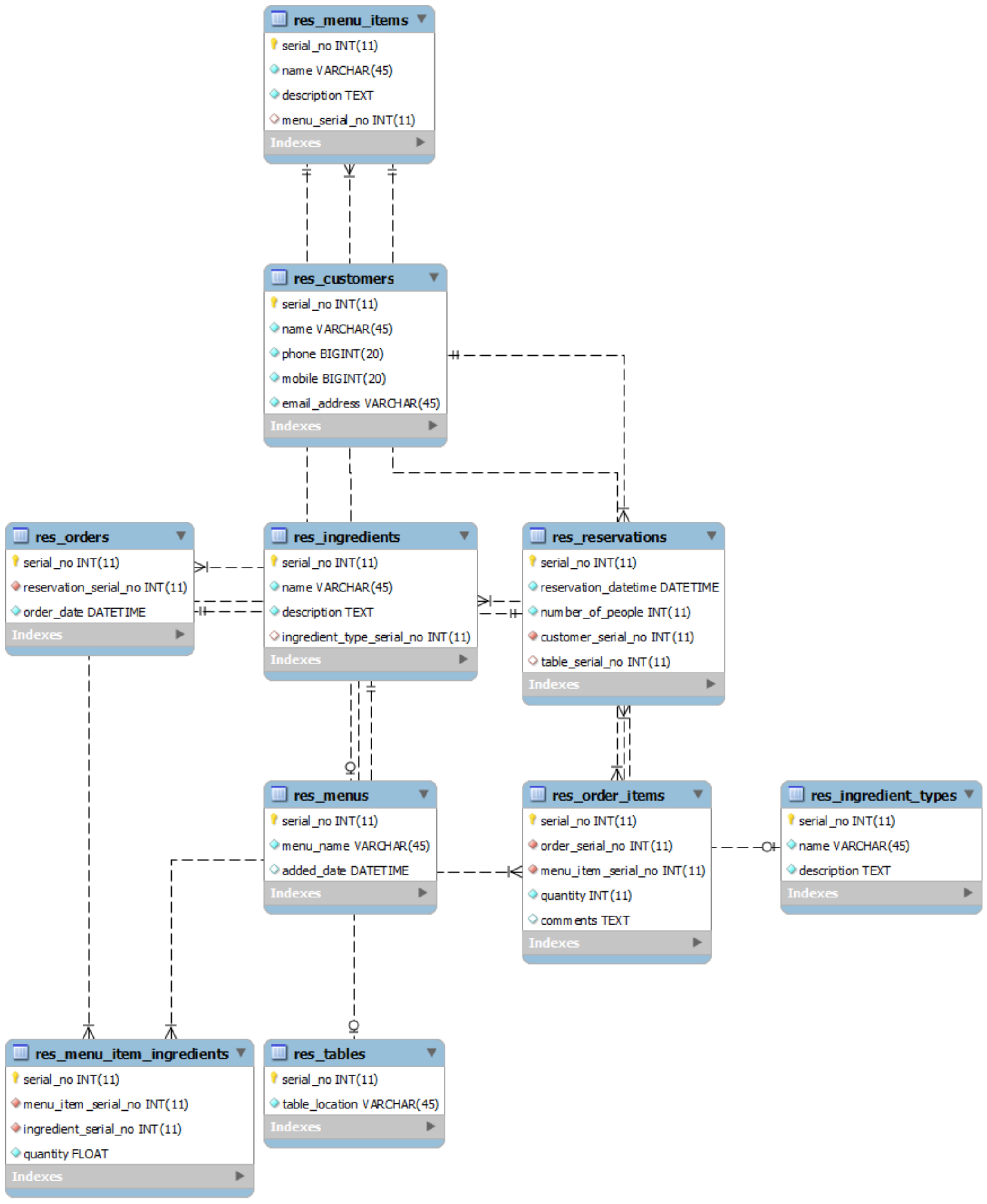


Figure 5.4: Incremental ERD (4) for the pilot system

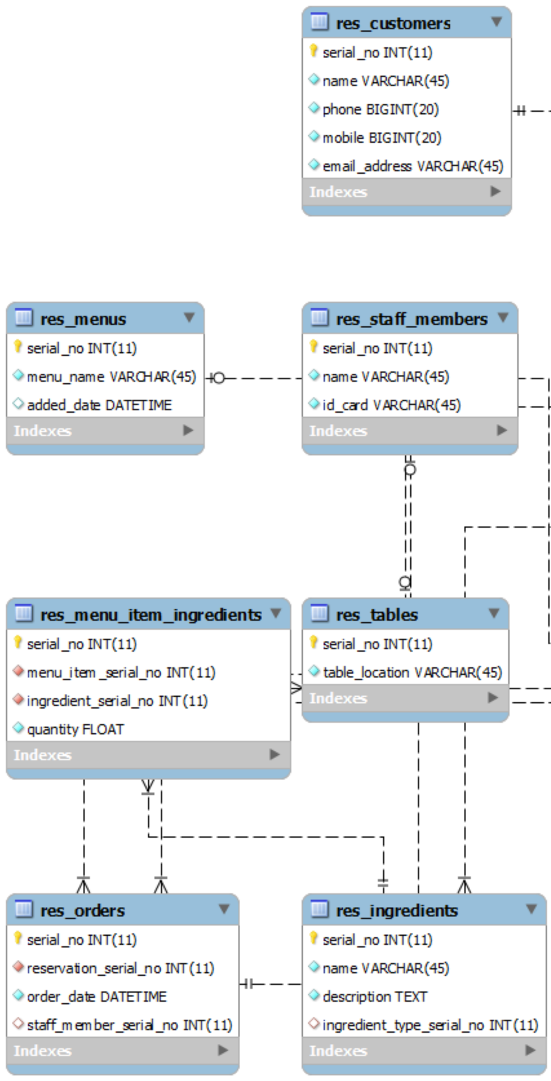


Figure 5.5: Final ERD for team (B)

5.2.3 *Findings*

The results from the two teams' implementation can be categorized into two sections: specific and shared. The specific sections discuss the points that are dedicated to the team, while the shared section discuss the points that are in common between the two teams and are comparable. The following are the details of each section:

5.2.3.1 *Specific Results*

Item (1) "Time needed to finalize the database ERD" in in table (6) represents the specific results for team (A). The item value is the overall time that is needed to accomplish the database ERD when the project starts. The value is 2 working hours excluding any modifications done later to the database ERD.

For team (B), the items (1) “Understanding the new model” and item (2) “Easiness of the new model usage” in table (7) are the team’s specific items. They are bout the understanding the easiness of the model. The values of item “Understanding the new model” is 9 while the value if the item “Easiness of the new model usage” is 8.

It is easy to observe that team (A) has spent 2 hours of designing the final “physical” database ERD, the 2 hours are the first version of this ERD since the user stories have been developed yet. Despite the time spent in modeling the database ERD is considered small, only 2 hours, but its weight is 8% of the overall time needed to accomplish the system. This means the customer will not be engaged in this 8% of the system, and moreover, the customer has to wait more time before he can see actual data entry forms and reports resulted from the sprints. Customers love to see things they are familiar with such as forms and reports rather than technical artifacts that does not attract them despite its importance!

On the other hand, when it comes to team (B) specific results, we can easily observe that team (A) did not face a problem in understanding and adapting the new model to start working on. The item “Understanding the new model” score is 9 out of 10, while the item “Easiness of the new model usage” score is 8 out of 10. Team (B) has completed the system using the new model efficiently and there were no complaints or more technical clarification needed when the work starts.

5.2.3.2 Shared Results

The shared points from the teams’ results are presented together in table (8) and the scores from the two teams results are presented there as well.

Table 5.3: Comparison of The Results for Shared Items

#	Evaluation Item	Team (A) Scores	Team (B) Scores	Precedence
1.	Productivity rate of the new model	11	11	Equal
2.	Customer engagement during the project	8	8	Equal
3.	Customer satisfaction	7	8	Team (B)
4.	Flexibility to adapt changes	4	8	Team (B)

5.	Divergence of what actually required compared to what actually developed	7	7	Equal
6.	Cost of change at the database level	0 Conceptual 6 Logical 8 Physical 0 Code 0 Abstraction Layer	3 Conceptual 2 Logical 1 Physical 8 Code 4 Abstraction Layer	Team (B)
7.	Overall time needed for the project	25 hrs.	16 hrs.	Team (B)

The results in table (8) clearly show that the results for team (B) is better than the results obtained from team (A). They scored the same value for the items "Productivity rate of the new model" and "Customer engagement during the project". This normal since both of the teams are of adequate skills to accomplish all the user stories, also, and since the Agile methodology is used to develop the system, then the customer engagement is expected to exist.

The remarkable results are for the rest of the items. Team (B) scored much better than team (A). The following are discussion of each of the shared items scores for the two teams.

- **Customer satisfaction:** Team (B) has scored 8 out of 10 while team (A) scored 7 out of 10. The difference is not high, and this is logical since the Agile methodology consider customer engagement is crucial in software development. Anyhow, team (B) scored higher score than team (A) because the customer was able to early engage in Scrum sprints because of the new Agile-Database model. Team (B) reduced the startup time needed for the project by delaying the creation of the physical database model to the beginning of each Scrum sprint. Using this technique, team (B) was able to early engage the customer in the project.

- **Flexibility to adapt changes:** Team (B) has scored 8 out of 10 while team (A) has scored 4 out 10. The remarkable difference

was because team (A) needed to do modifications to the physical database design, which was developed up-front, and also do some modifications to the business logic embedded inside the developed forms. At some point of the software development, team (A) leader state "*2nd sprint was not hard 3rd sprint I felt some inability*".
However, for team (B), the physical model was created only once the Scrum sprint is fully discussed and completely agreed.

- **Divergence of what actually required compared to what actually developed:** Both teams scored the same value, this is because the customer is one customer for both of the teams. The feedback from the customer was the same for both teams. This is why this value is the same for both of them.

- **Cost of change at the database level:** Team (B) as done a sort of modifications to the system and the database design during the development. There are three changes done to the database conceptual model; which are actually a result of two modifications done to the database logical model and one modification done to the database physical model.
Also, we can see that most of the modifications that team (B) has accomplished are in the code and the Abstraction Layer.

- **Overall time needed for the project:** Team (B) has scored 16 hours while team (A) has scored 25 hours. This is a remarkable difference. The same outcomes have been accomplished by team (B) with less time. Team (B) needed 64% of the time needed by team (A) to accomplish the same project and to reach the same outcomes.

5.3 Summary

According the results that were obtained from the two teams who were working on the proposed project for this thesis; it is obvious that the results for team (B), who was using our new developed model (Agile-Database), are much better in terms of easiness, flexibility, and time.

The new model has proved to be effective when there is a need to achieve a flexibility and ability to respond to customer changes, and of course, without sacrificing the quality of the developed product. In addition, our new model has reduced the cost of changes since the database physical model creation is divided to the sprints, and thus, the cost of changing something to the database modeling structure is at the conceptual level or at the logical level, and very rare at the physical level. Furthermore, the overall time needed to finalize the work is much less when using the new proposed model, this is a remarkable achievement for the new model, and it resulted because the physical creation of the database model is delayed till the Scrum sprint is discussed and agreed and because of the use of the Abstraction Layer which separates the complexity of the adapting changes to the database from the user interface.

In conclusion, the new model, Agile-Database, has improved the overall experience of the Agile development for the systems that are heavily database-dependent.

CHAPTER VI: Conclusion and Future Works

This chapter concludes the results and the findings of the work, also, it highlights the future work directions.

6.1 Conclusion

Agile gained respect in software development field, it has been used by many size of organizations. Due to the nature of the Agile methodology and its practices; Agile helped developers to more involve the customer in the software development lifecycle, and this resulted in more customer satisfaction.

On the other hand, relational database engines are still the dominant when it comes to the critical software systems that needed transaction consistency and accuracy. However, the traditional up-front design practices for modeling databases are inadequate and inconsistent with today's Agile practices.

Our model, the Agile-Database, has integrated the Agile practices along with the database design practices. The model did not ignore the importance of the database modeling practices, it keeps all the good about database design, but it distributes the modeling phases along with the Scrum phases.

The results from obtained from the team who applied the new model in developing the proposed software showed great improvement when compared to the results obtained from the team who used traditional database development with Scrum. The team who used the model was able to achieve the same results with a percentage around 64% of the time needed by the team who used the traditional up-front design.

Moreover, the new model helped the team to be able to adapt changes with more flexibility with a percentage around 80% when compared to the other team whose ability to adapt the changes was only around 40% for the same software system.

Furthermore, the new model has reduced the cost of the changes done at the database level. This is due to the fact that the physical implementation of the database model is deferred until the Scrum sprint is completely agreed and the developers accept it and start working on it.

Finally, the model did not sacrifice any of the database design concepts such data integrity and reducing data redundancy. We do hope that this model is adapted and used with an international Agile methodology, especially Scrum, which has been used in this research.

6.2 Future Works

The new model has improved the database design practices and integrate it successfully with Scrum practices. Also it helped to reduce the impact of the database refactoring with the use of the Abstraction Layer. However, there is a real need for future work on database refactoring practices such as:

- Reduce the impact of changing the physical model of the database.
- A need to do more testing on more large-scale systems to ensure the results are accurate when it comes to enterprise level applications.
- Develop an algorithm that could be the basis for developing new software that helps software architects to find the “Focal-Entity” in an automated manner, or list candidate “Focal-Entity” to them.
- How to improve, and get benefit, from some database vendors features that are related to database refactoring such as Oracle Edition Based, Oracle Database Replay, and Oracle Online Redefinition.
- Use other agile techniques such as XP.

References

[1] A. M. M. H. a. H. Abushama, "Popular Agile Approaches in Software Development: Review and Analysis," *INTERNATIONAL CONFERENCE ON COMPUTING, ELECTRICAL AND ELECTRONIC ENGINEERING (ICCEEE),* pp. 160-166, 2013.

[2] A. B. a. N. Nagappan, "Usage and Perceptions of Agile Software Development in an Industrial Context: An Exploratory Study," *First International Symposium on Empirical Software Engineering and Measurement,* pp. 255-264, 2007.

[3] J. Rasmusson, "Agile vs Waterfall," Agile In a Nutshell, 2015. [Online]. Available: http://www.agilenutshell.com/agile_vs_waterfall. [Accessed 13 February 2016].

[4] S. a. D. M. Murugaiyan, "WATEERFALLVs V-MODEL Vs AGILE: A COMPARATIVE STUDY ON SDLC," *International Journal of Information Technology and Business Management,* pp. 26-30, 2012.

[5] C. B, "What is Agile Development? An Introduction," ScreenMedia, 4 8 2014. [Online]. Available: http://www.screenmedia.co.uk/blog/2014/08/what-is-agile-development-a-brief-introduction. [Accessed 13 February 2016].

[6] A. Sidky, *A Structured Approach to Adopting Agile Practices: The Agile Adoption Framework,* Virginia: Virginia Polytechnic Institute and State Univ., 2007.

[7] 1keydata, "Data Modeling," 1keydata.com, 2001. [Online]. Available: http://www.1keydata.com/datawarehousing/data-modeling-levels.html. [Accessed 13 February 2016].

[8] J. Han, H. E., G. Le and J. Du, "Survey on NoSQL database," in *Pervasive Computing and Applications (ICPCA), 6th International Conference on*, 2011.

[9] N. Leavitt, "Will NoSQL Databases Live Up to Their Promise?," *Computer,* vol. 43, no. 2, pp. 12-14, 2010.

[10] S. Ambler, "Introduction To Database Refactoring," The Data Administration Newsletter, 1 July 2006. [Online]. Available: http://tdan.com/introduction-to-database-refactoring/5010. [Accessed 13 February 2016].

[11] R. Morien, "Agile Development of the Database: A Focal Entity Prototyping Approach," *Proceedings of the Agile Development Conference,* 2005.

[12] K. S. a. B. Goel, "Impact of Agile and TDD Implementation in Database," *International Journal of Computer Applications,* vol. 22, pp. 26-29, 2011.

[13] S. Ambler, Agile Database Techniques, Wiley Publishing, 2003.

[14] K. Schwaber and J. Sutherland, "The Scrum Guide," Scrum Alliance, 24 September 2014. [Online]. Available: https://www.scrumalliance.org/why-scrum/scrum-guide. [Accessed 13 February 2016].

[15] M. Website, "Maxxor," Maxxor, [Online]. Available: https://www.maxxor.com/software-development-process. [Accessed 4 March 2016].

[16] M. J. Hernandez, Database Design for Mere Mortals: A Hands-on Guide to Relational Database Design, Michigan: Addison-Wesley Professional, 2013.

[17] J. L. Harrington, Relational Database Design and Implementation, 3rd Edition, Morgan Kaufmann, 2009.

[18] L. Burns, Building the Agile Database: How to Build a Successful Application Using Agile Without Sacrificing Data Management, Westfield, NJ: Technics Publications, 2011.

[19] A. Harriman, P. Hodgetts and M. Leo, "Emergent database design: liberating database development with agile practices," in *Agile Development Conference*, 2004.

[20] L. Ashdown and T. Kyte, "Oracle Database Documentation - Oracle Database Concepts," Oracle, 2011 2011. [Online]. Available: http://docs.oracle.com/cd/E25054_01/server.1111/e25789/srvrside.htm. [Accessed 14 February 2016].

[21] S. Feuerstein, Oracle PL/SQL Best Practices, 2nd Edition, O'Reilly Media, Inc., 2007.

[22] B. Williams, "Database Answers," 14 October 2001. [Online]. Available: http://www.databaseanswers.org/data_models/restaurant_bookings/index.htm. [Accessed 11 March 2016].

[23] Oracle, "Oracle Database," Oracle, 2016. [Online]. Available: https://www.oracle.com/database/index.html. [Accessed 9 March 2016].

[24] Oracle, "Oracle SQL Developer," Oracle, 22 December 2015. [Online]. Available: http://www.oracle.com/technetwork/developer-tools/sql-developer/overview/index-097090.html. [Accessed 9 March 2016].

[25] Oracle, "NetBeans.org," Oracle, 2016. [Online]. Available: https://netbeans.org/. [Accessed 9 March 2016].

[26] Oracle, "Java Software," Oracle, 2016. [Online]. Available: https://www.oracle.com/java/index.html. [Accessed 9 March 2016].