ACIT'2017
The International Arab Conference on Information Technology
Yassmine Hammamet, Tunisia
December 22-24, 2017

Integrate Database Design Techniques with Agile Applications

Alaa M. El-Halees
Faculty of Information Technology
Islamic University of Gaza
Gaza, Palestine

Emad O. Kehail
Faculty of Information Technology
Islamic University of Gaza
Gaza, Palestine

***Abstract*— One of the major business needs nowadays is the ability to respond to business requirements and changes quickly. Therefore, the software development process has been moved to be more agile by using what has been agreed to call Agile Software Development Process. However, the business still needs to store data, and Database Management Systems (DBMS) are still the de facto for the business software. DBMS is relying completely on database design process that follows traditional up-front design process which is sequential by nature. 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 help 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. We found that the new model helped to reduce the impact of the changes implemented at the database level and to achieve the goal with a 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

I. Introduction

The business needs are changing rapidly, and customer requirements are continuously changing as well. However, the software development process is getting more and more complex day after day. Therefore, applying traditional software development lifecycle methodologies such as waterfall to such large systems would give us troubles such as: poor visibility, cannot handle changes, poor quality and higher risks [1]**.** This is because traditional software development, especially those who are database-dependent, are sequential in nature. This adds extra delaying time to the developed software since the software developers have to wait for the data modelers to design the Conceptual Level, Entity-Relationship Diagram (ERD), and then convert it into a physical database design, and after all of that, the developers can start working on the business logic and the user interface.

Because of that, the evolutionary database techniques start to rise up [3] [4] to help Software Developers and Architects to unify the concept of Agile software development along with the Database Design and Modeling [5]. This greatly helps to release working versions of the desired software very quickly without sacrificing the concepts of the database modeling and normalization rules.

Using evolutionary database techniques solve the problem of Up-front database design and modeling that takes a long time, and when the developer starts developing the business logic and the interface, then the requirements usually change. This normally results 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.

The main objective of this paper 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. This allows agile software developers to easily embed database design in the agile practices which in turn helps them to present their work to the client quickly. As a results, this helps to quickly release versions of the desired product as well as quickly fixing errors in the early stages of the software development.

The rest of the paper is structured as follows: the second section discusses the related works, the third section addresses our methodology and implementation, the fourth section about evaluation method, section five gives the results , while the sixth section implies the conclusion and future works of the paper.

II. Related Works

There is limited research on developing Agile data techniques that helps both Agile software engineers and database

designers to work together effectively. For example, Morien in [11] introduced the concept of the "Focal Entity". The "Focal Entity" is a starting point that used to start designing the data model in an iterative manner using the conceptual, logical and physical design along with the process of these entities. The author has developed a Tactical Model of Development based on the selection of 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), to link it more with Agile practices. The author has suggested using Scrum Agile practices along with the Focal Entity Approach. The Scrum "sprint" best fit for the "Tactical Model" since it could be used to develop the sprint backlog. The Tactical Model is the basis for allocating the sprint tasks to the Personal Sprint Backlog, and the outcomes of this model are the contracted outcomes of the sprint.

Ambler in [13] introduced the Agile Modeling concept, where the Agile-DBA needs to cooperate with the rest of the development team to the evolutionary design database. The author has proposed database refactoring by using views and direct modifications to the database tables and defined how these modifications can be reduced by means of encapsulation layer. Because of such business values, it needs to keep 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.

Harriman et al. [19] proposed to achieve an iterative model for data modeling. They 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, incrementally, applied Agile discipline to the database development. That, eventually, reduced up-front design work to just-in-time work that matched their 1 to 2 week development iterations.

III. Methodology and Implementation

In our research paper, we proposed a model to develop the Agile-Database model. As explained in figure 1, the model consists of four steps. Each step in our model has a corresponding step in Scrum framework. The steps at the Agile-Database process have to be executed in parallel with the normal Scrum Agile development process. That, it has to be merged and completely integrated into Scrum Agile model. Following are details of each step.

Step One: Conceptual Design

Usually, application development needs preparation to establish a general understanding of the new application goal and objectives. Therefore, series of meetings are needed to identify these objectives. During these meetings, the data analyst need to start to form a general idea about the data needs for this application.

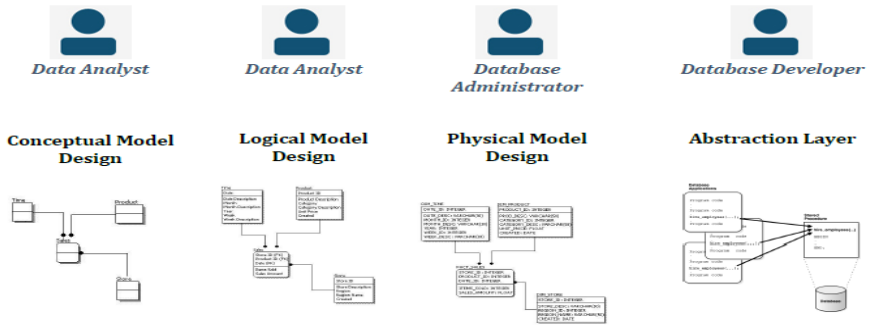

Figure 1: Proposed Agile-Database model

During these meetings, a set of user stories need to be developed, briefly analyzed, and prioritized. These user stories form the Product Backlog of the Scrum framework. The data analyst must attend 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. The data analyst must focus on the business needs and how the model will 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.

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 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.

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 in this step and his opinion is crucial when it comes to choose 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 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.

At this stage, the data analyst designs the Logical Model. The attributes of each entity need to be defined. Also, entities primary and foreign keys are clearly identified as well.

As each user story is being discussed, the data analyst attends 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 just implements what is considered enough for this user story. However, the changes to 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 need to have another discussion with the database administrator to convert the current Logical Model to the most appropriate Physical Model. There could be some changes to 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.

The Database Administrator converts the Logical Model designed by the data analyst to the Physical Model. Only the design of the tables needs to 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 starts working on creating the Abstraction Layer. The database developer codes the necessary Stored Procedures, Functions, Packages, and Database Triggers. 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.

The Database Developer has to follow some guidelines 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 needs to 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.

IV. Evaluation Method

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

A. Proposed Software System

A pilot system, which is a proposed restaurant model, which has been chosen for development in order to evaluate the new model. Our proposed system called " 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 [6] for database models, and it has been chosen because of its simplicity but yet contains an adequate number of database tables that can serve the purpose of the research. The system has 11 user stories.

B. Development Teams

Two teams of developers have been prepared. Team (**A**) develops the proposed system using up-front database design, while team (**B**) develops the proposed system using the new Agile-Database model. Both teams are good in database and programming and has almost the same experiences in software development.

C. Evaluation Criteria

Two evaluation criteria have been developed to help measure the performance of the two teams. Criteria for team (A) are:

1) *ERD Time*: Overall working hours consumed by the all the stakeholders to develop the ERD.
2) *Productivity Rate*: Number of user stories accomplished.
3) *Customer Engagement*: How much customer engaged in the project. Value from 1 to 10. 1 is rare, 10 is very engaged. Use the number of meetings held by the customer.
4) *Customer Satisfaction*: How much customer satisfied. Value from 1 to 10. 1 is unsatisfied, 10 is very satisfied.
5) *Flexibility:* How much the system flexible to adapt changes . Value from 1 to 10. 1 is hard, 10 is very easy.
6) *User stories Divergence:* What actually required compared to what actually developed, Number of user stories cancelled, changed, added to the system.
7) *The Cost of Change:* How much cost of change at the database level. The formula is: 1x for any change done in the Conceptual Model, 2x for any change done to the Logical Model,4x for any change done at the Physical Model,1x for any change done at the code.
8) *Time Needed*: Overall working hours consumed by the all the stakeholders to develop the system.

Criteria for team B are:

1) *Understanding the New Model:* How much hard to understand the new model. Value from 1 to 10. 1 is hard, 10 is extremely easy.
2) *Easiness of the New Model Usage:* How much easy to use the model. Value from 1 to 10. 1 is hard, 10 is extremely easy.

The rest as in criteria for team (A).

D. Evaluation Process

The evaluation process measures the two teams' performance using items in criteria for teams (A) and (B) criteria as mentioned in section 4.3. The values of the two criteria will be compared to each other to measure the overall performance and efficiency of the new model.

Since it costs more when we do modifications at the database level when the database is growing up, the formula in item 7 "Cost of Change" has been built to comply with this fact. The formula considers that more cost is needed when modifications are 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:

1) Describe the new Agile-Database model: Team (B) only will implement the software using the new proposed model. Team (B) should answer *"Understanding the new model"*.
2) Describe the proposed system to be developed: 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 and the Sprint Logical Model: For Team (B) only since Team (A) is using up-front design. The team should develop a complete Physical Design for the whole system.
4) Review the Sprint Physical Model: Team (A) should develop a complete Physical Design for the whole system. On the other hand, team (B) should develop a Physical Model for the user stories for the Sprint that is currently being developed.
5) Review each Sprint: This is a Scrum sprint review event.
6) Review the Final Product: Final product review.

V. Evaluation Method

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:

A. Specific Results for Team (A)

The team starts developing the system using the traditional up-front database design techniques along with Scrum agile methodology. Table (1) represents team (A) results.

TABLE I. Evaluation Results for Team (A)

	Evaluation Item	Value
1	ERD Time	2
2	Productivity rate	11
3	Review the Conceptual Model	8
4	Review the Sprint Logical Model	7
5	Customer satisfaction	4

6	Flexibility to adapt changes	7
7	Cost of change at the database level	44
8	Time Needed	25 hrs

We can notice 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 like to see things they are familiar with such as forms and reports rather than technical artifacts that do not attract them despite its importance.

B. Specific Results for Team (B)

On the other hand, the team starts developing the system using the new Agile-Database model design techniques along with Scrum agile methodology. Table represents team (B) results.

TABLE II. EVALUATION RESULTS FOR TEAM (B)

	Evaluation Item	**Value**
1	Understanding the new model	9
2	Easiness of the new model usage	8
3	Productivity rate	11
4	Customer engagement	8
5	Customer satisfaction	8
6	Flexibility to adapt changes	8
7	User stories Divergence	7
8	Cost of change at the database level	27
9	Time Needed	16

Also, we can observe that team (B) 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.

C. Shared Results

The shared points from the teams' results are presented together in Table 3 and the scores from the two teams results are presented there as well.

TABLE III. COMPARISON RESULTS OBTAINED FROM TEAM (A) AND TEAM (B)

	Evaluation Item	Team A	Team B	Precedence
1	Productivity rate	11	11	Equal
2	Customer engagement	8	8	Equal
3	Customer satisfaction	7	8	Team B
4	Flexibility to adapt changes	4	8	Team B
5	User stories Divergence	7	7	Equal
6	Cost of change at the database level	44	27	Team B
7	Time Needed	25	16	Team B

The results in Table (3) clearly show that the results for team (B) are better than the results obtained from team (A). They scored the same value for the items "Productivity Rate" and "Customer Engagement". This normal since both of the teams are of adequate skills to accomplish all the user stories, also, 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 a 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 engage the customer in the project early.

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.

User stories Divergence: 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.

The cost of change: 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.

Time Needed: 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.

VI. Conclusion and Future Works

Agile methodology gained respect in software development field; it has been used by many sizes of organizations. On the other hand, relational database engines are still 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 obtained from the team who applied the new model to 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.

In the future, there is a real need for database refactoring practices such as: Reduce the impact of changing the physical model of the database, 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, and get benefit, from some database vendors features that are related to database refactoring such as Oracle.

References

[1] J. Rasmusson, "Agile vs Waterfall," Agile In a Nutshell, 2015. [Online]. Available: http://www.agilenutshell.com/agile_vs_waterfall. [Accessed 13 February 2017].

[2] 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.

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

[4] 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 2017].

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

[6] B. Williams, "Database Answers," 14 October 2001. [Online]. Available: http://www.databaseanswers.org/data_models/restaurant_bookings/index.htm. [Accessed 11 March 2017].