Extreme Programming and Database Administration: Problems, Solutions, and Issues

Ahmed M. Hassan
ThoughtWorks, Inc.
651 W Washington Blvd.,
Suite 600, Chicago, IL,
60661 USA
+ 1 312 925 2378
Ahmed.Hassan@thoughtworks.com

Amr Elssamadisy
ThoughtWorks, Inc.
651 W Washington Blvd.,
Suite 600, Chicago, IL,
60661 USA
+ 1 312 961 6504
Amr.Elssamadisy@thoughtworks.com

ABSTRACT
XP focuses on both development and the customer. And because it is recommended that XP be used only with small projects, it is assumed that the developers do the DBA functionality. Recently there has been literature discussing the use of XP with large projects and the various ways that the basic XP principles have been adopted with larger development teams. With any medium to large sized project there is usually a separate Database Administrator (DBA) or DBA team. How does this team fit in with the overall XP team and how or what XP practices are applicable to its responsibilities? In this paper we will tell of our experience doing just that, how the XP process was or was not used by the DBA team, problems we had on the database side, our applied solutions, and solutions that we suggest.

Keywords

XP process, scalability to larger teams, database.

1 INTRODUCTION

This is an experience report describing our findings and suggestions on a large project that has been heavily affected by XP and has grown to include an independent database team. It should be kept in mind that this is a 'lessons learned' format where we only describe what we believe can be changed for the better. That does not imply that we did not succeed in supporting an XP development process with a DBA team; in fact, the opposite is true. We have been very successful in having a DBA team that is agile enough to support a very dynamic and large XP project.

Historically at ThoughtWorks, Inc. we have been using agile methodologies and/or a modified version of XP since January of 2000 [1][2][3]. The particular project that the authors have been working on grew from having a pure development team, to a team with one database administrator (DBA), and now includes a team that has been as large as four dedicated to supporting development and deployment of the application. As the development team adopted XP and molded it to its needs the smaller DBA team was out of the loop during many parts of the development process.

This paper attempts to recognize problems, analyze them, and describe and suggest solutions for database-related issues in a large XP software project. First, we will present the problems we encountered and then we will present the XP practices that we believe address these problems.

2 PROBLEMS ENCOUNTERED WITH AN XP DEVELOPMENT TEAM AND A STANDARD DBA TEAM

Lack of unit tests for views/stored procedures

During testing some views/stored procedures did not return the data that they should have. Developers seemed to take the database code for granted and did not develop unit tests for all of the code parts that relied on the database code. At the same time, the database team was fine with writing code, testing it manually, and assuming that it worked correctly. The truth is that both developers and DBAs were at fault – there was weak informal communication/cooperation in writing these views and stored procedures. In summary, we believe that views and stored procedures should be treated like any other code – they should be written in pairs and should be covered by the unit test suite.

Code and database are not in synch

This is a 2-fold issue:

One: synchronization of constraints (required fields (not null), unique, primary key, and referential integrity (RI)). The application had its own 'understanding' of existing database constraints. The actual database constraints were not exactly the same as those of the application. Most of the 'mismatching' pieces required code changes that took time and effort to fix during a cleanup stage.

Two: synchronization of data items (tables and columns). As the code base grew, many database changes were required. It is not possible to 'forget' about adding a new column to a table since it will be used by the

code. However, it is very easy to 'forget' to remove unused tables/columns in such a low-communication environment. This led to having unused tables and columns remaining in the database (and in a few cases in the code too).

Bad database design decisions
Since most design decisions were driven by developers (including database design), many design choices were not the best choices available. This is due to the fact that not all developers have database design skills. Therefore bad design decisions fell through unnoticed without the DBA team having had the chance to review them.

Missing Database constraint definition
A typical situation for a developer when working on a story card that required new database columns and/or tables was to request a simple change to the database without any constraints. Therefore a column would be added without any not-null constraints or RI constraints even though logically they were needed. At the rush to finish functionality at the end of the iteration, the extra code needed to turn these constraints on was never added and subsequently the change request to add these constraints was never made.

Database changes are a development bottleneck
All database changes were handled only by DBAs. When the team did not have enough DBAs many code changes were waiting for the matching database changes. In other words, database changes became a development bottleneck. This caused developers to either delay delivery of their code or to ask for their database changes prematurely. Some of these premature changes were removed from the code afterwards, but not from the database (code-DB synch problem).

Database changes have a single owner
In a large project like the one that the authors worked on, there were many different environments that developers are not familiar with or even aware of. Database changes had to be applied by DBAs who were able to manage these environments. Therefore, all database related code (views/stored procedures, etc.) was owned by DBAs. Many changes were simple and could have been applied by developers who had to wait for a DBA to apply their changes.

Missing communication
Communication between the development and a 'standard' DBA team most resembled the communication of a client to a development team in a pre-XP process. That is, there is a wall and the client (developers in our case) throws the requirements over the wall to the developers (DBAs in this case). Then DBAs throw the results back over the wall. The problems with this approach are the same problems that existed with the old client-developer relationship. This resulted in two separate teams working on one project.

3 SOLUTIONS TO PROBLEMS ENCOUNTERED

The Planning Game

The planning game is one of the most important activities that were missed. XP is based on quick iterations, and each iteration kicks off with the planning game. This is the feedback that allows the team to 'drive' the application to success and get around potholes. We have caught problems/bad designs that could have been avoided if only a member of the DBA team attended the design meetings (can be replaced by a developer with high database design skills).

The fact is that not all story cards require DBA input. However, it is very difficult at the planning phase to know exactly how things will touch the database. Therefore we recommend that a member of the DBA team (or a developer that is highly skilled in DB design) be present in ALL developer design meetings to be able to point out and discuss changes that need to be made to the database. This may seem like a waste of time because surely not all changes will affect the database. This may be correct, but at the same time the result of making a bad database decision is that you will live with the bad design throughout the life of the project.

Pairing
Pairing is applicable only when developing code in the database, namely during development of stored procedures and views. The two who should pair are the developer responsible for the card and the DBA responsible for making the needed database changes for that card. This is unconventional pair programming because we have two different skill sets working to solve one problem. Applying a test-first strategy, the DBA should work with the developer when writing the unit tests giving input based on the requirements that were agreed upon in the planning game. Afterwards the developer should pair with the DBA and allow the DBA to drive in writing the stored procedure. This type of pair programming will need a developer who is familiar with the database and a DBA who is familiar with the application programming language. The real gain from this is increased communication between the two teams and the melting of the teams into one to write one application[1].

All story cards must recognize DB tasks and list constraints that will be enforced after the card is finished

[1] In fact, on our particular project, we have rotated developers into and out of the DBA team.

(this includes PK, FK, RI, unique, and not nulls constraints). This means a story card is not finished until all of its related constraints are enabled and not simply that the code passes all of the unit/function tests.

Collective ownership
All code that was developed by pairing a DBA with a developer should be jointly owned. This code is now part of the code base and the unit tests are present as a safety net to keep the code on track. This should help to resolve the problem of DB changes being a bottleneck in the development process. This also would help join the two teams into one development team.

However, we should distinguish between creating the new database code and applying this new code to all of the environments that need it. This is due to the situation when some environments require this change immediately, other environments require one week delay on code and DB changes, some other environments require two week delay and some environments have code and DB changes applied asynchronously (on request of the owner). It is an open question whether to have exclusively DBAs perform the task of applying/maintaining the database code changes or to allow developers to apply them once they are ready to do so.

Testing
Database related bugs can be very subtle and discovered very late. Having unit tests for all DB components will help to discover many issues earlier. These unit tests can be unit tests for the application code that uses DB code. They will be part of the whole testing suite for the application and will allow leveraging of the usually more powerful development language.

To synchronize our database model and the code's view of the database (code model) we had manual runs of utilities to find: forgotten-to-remove tables and columns, data type mismatching, and required column constraints. These runs helped to bring the code into synch with the database. This very useful process can easily be automated and incorporated into the testing suite.

4 REFACTORING
The fact is that in our experience, bad database design decisions are rarely fixed. They are tolerated and are heavy weights that we carry throughout the life of the project because they are seen as a low priority because the code is working. DBAs should be allowed/encouraged to add refactoring tasks in appropriate story cards.

Many common database refactorings are not cataloged. It might be useful if developers would become familiar with such refactorings. One example is an upgrade of one-to-one (1-1) and one-to-many (1-M) to many-to-many (M-M) relationships. Missing this refactoring may cause an ill/inefficient database design.

5 CONCLUSION
XP with database administration is a part of XP that has not been extensively discussed. Many issues came up; we tried some solutions and we would like to try others whenever we do this again.

Database Administration in conjunction with XP is not a frequently addressed subject. We have presented our experience with a large XP project that included a DBA team and have enumerated our pains. All of these pains seem to originate from the fact that in reality we had two separate teams with different methodologies and poor communication between the teams. This resulted in the code and database design getting out of synch, code that is ultimately part of the application (stored procedures and views) did not have unit tests, there was little refactoring of the database, and other problems. Then we presented some solutions to these problems by increasing the involvement of the DBA team in the development process, and increasing the involvement of the development team in the database administration process. We do not have all of the answers to the problems that we encountered; there are many issues left to be addressed. Where do we draw the line – if any – on code ownership? When should we refactor? What is the quality of the database design of an XP driven project compared to other databases with an upfront design? What we do know is that many of the pains that we encountered could have been reduced by the introduction of testing, pairing, DBA involvement in the planning game, and more collective ownership.

REFERENCES

[1] Elssamadisy, A. XP On A Large Project: A Developer's View, http://www.thoughtworks.com/library/index.html

[2] Elssamadisy, A. and Schalliol, G. Recognizing and Responding to "Bad Smells" in Extreme Programming, http://www.thoughtworks.com/library/index.html

[3] Schuh, P., and Punke, S. ObjectMother: Easing Test Object Creation In XP, http://www.thoughtworks.com/library/index.html