Regis University

# ePublications at Regis University

Fall 2006

# Unlocking Test-Driven Development

Chris H. Knapp
*Regis University*

Follow this and additional works at: https://epublications.regis.edu/theses

Part of the Computer Sciences Commons

## Recommended Citation

Knapp, Chris H., "Unlocking Test-Driven Development" (2006). *Regis University Student Publications (comprehensive collection)*. 389.
https://epublications.regis.edu/theses/389

# Regis University
School for Professional Studies Graduate Programs
**Final Project/Thesis**

## Disclaimer

# Regis University

## School for Professional Studies

**Master of Science
In
Computer Information Technology**

Unlocking Test-Driven Development

**Thesis**

**Version 1.0**

**Chris H. Knapp**

**August 9, 2006**

# Certification of Authorship

**Regis University**
**School for Professional Studies**
**MSCIT Program**

<u>Certification of Authorship of Professional Project Work</u>

Submitted to: **Tim McKenzie**

Student's Name: **Chris Knapp**

Date of Submission: **July 30th, 2006**

Title of Submission: **Unlocking Test-Driven Development**

Certification of Authorship: I hereby certify that I am the author of this document and that any assistance I received in its preparation is fully acknowledged and disclosed in the document. I have also cited all sources from which I obtained data, ideas or words that are copied directly or paraphrased in the document. Sources are properly credited according to accepted standards for professional publications. I also certify that this paper was prepared by me for the purpose of partial fulfillment of requirements for the MSC 696 or MSC 696B course.

Student's Signature: _____

# 696B Faculty and Advisor Approval page

**Regis University**
**School for Professional Studies**
**MSCIT Program**

**Advisor/MSC 696 or MSC 696B Faculty Approval Form**

Student's Name: **Chris Knapp**

Professional Project Title: **Unlocking Test-Driven Development**

Advisor, MSC 696 and MSC 696B Faculty Declaration: I have advised this student through the Professional Project Process and approve of the final document as acceptable to be submitted as fulfillment of partial completion of requirements for the MSC 696 or MSC 696B course. The student has received project approval from MSC 696A faculty and has followed due process in the completion of the project and subsequent documentation.
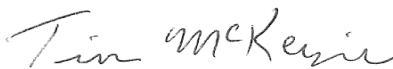
Advisor

| Michael Nims | | 08-01-2006 |
|---|---|---|
| Name | Signature | Date |

MSC 696 or MSC 696B Faculty Approval

| Tim McKenzie | | August 6, 2006 |
|---|---|---|
| Name | Signature | Date |

# Revision History

| Revision Number | Date | Key Changes |
|---|---|---|
| 0.0 | 03/14/2005 | Initial draft from the 696A class. |
| 0.1 | 05/10/2006 | Continued to flush out the draft. |
| 0.2 | 07/05/2006 | Draft submitted during week 1 of 696B class |
| 0.3 | 07/16/2006 | Draft submitted during week 2 of 696B class<br>The following changes were made:<br>• Wrote missing/incomplete sections in Lessons Learned.<br>• Moved the cipher research from in Lessons Learned to Methodology.<br>• Dropped the project design chapter since it was for the initial approach of using a waterfall SDLC flow.<br>• Dropped the project testing chapter – it's redundant with another chapter.<br>• Added legends to all figures that required them. |
| 0.4 | 07/30/2006 | Revision submitted during week 5 of 696B class<br>• Updated the Acknowledgement.<br>• Edited the Abstract and the introduction.<br>• Updated the style used for heading one, which affects the chapter heading.<br>• Moved the crypto research into the research/methodology chapter.<br>• Move the project management text into the lessons learned.<br>• General editing of all text.<br>• Rewrote the conclusion chapter. |
| 1.0 | 08/09/2006 | • Added in signatures from advisor and 696B instructor.<br>• Minor formatting changes to fix issues when converting this file over to PDF format. |

# Acknowledgements

The student acknowledges his loving and patient wife Cara. She generously sacrificed her time with her husband. This gave him more time for writing the thesis and for developing Women Partnering's software system.

# Abstract

Women Partnering is non-profit organization that helps women who are financially vulnerable. This organization establishes relationships with the women and connects them to support services. This project created a software system to support Women Partnering's daily operations and reporting needs, which replaced the previous manually intensive, paper-based system. There were many problems with the previous paper-based system including the following: data duplication, data not readily available, and lack of a reporting capability. Besides these problems, the previous system was not expected to support anticipated growth.

The student followed a Test-Driven Development Methodology while building the software system. This is the first time that the student has used Test-Driven Development on a project. To help with his understanding, he compared and contrasted this methodology to the Zachman Framework Methodology. The student knew that he also had to secure the application, so he researched the Rijndael cipher.

The analysis, design, and testing is handled differently in Test-Driven Development. Testing happens first, and the design captures the requirements. The student found Test-Driven Development lacking in a few areas, so he used other tools that are not part of the methodology like entity relationship diagrams and a data dictionary. Since the student was new to Test-Driven Development, he shares his many lessons on this project in hopes to helping others to avoid

the same pitfalls. The project's next steps include getting help integrated and possible integration with other support agencies.

Test-Driven Development is not a tool that should be used on all development projects. Rather, Test-Driven Development works best when the requirements are not clear, when the development team is smaller, and when the requirements are changing frequently. Most importantly, this methodology works well when the users are willing and able to participate throughout the entire project. The student suggests that software developers remain flexible in their tool choice in order to better serve their projects and avoid project failure.

# Table of Contents

# List of Figures

# List of Tables

# Chapter I.    Introduction

## A.  Problem statement

Women Partnering is a non-profit organization, which has been created through an endowment of the Sisters of St. Francis of Colorado Springs. Women Partnering helps women who are financially at risk. For example, if a women partner is about to loose her job because her car is broken down, then Women Partnering helps her get her car repaired. While Women Partnering directly helps some women partners out financially, this is not their main goal. Rather, they establish relationships with the women partners and connect them with support agencies. Their goal is to build long term relationships with the women in order to address their basic needs, to help them become self-sufficient, and to enrich their lives spiritually.

Women Partnering interacts with volunteers, donors, apartment managers, and support agencies in order to help women. When the student first interviewed Women Partnering, there were 100 women partners, 60 different support agencies, numerous donors and apartment managers, and a handful of volunteers. Women Partnering ran their organization primarily on Excel Spreadsheets and paper forms. Given the volume of women partners and support agencies alone, Women Partnering benefited by automating their data collection and other management activities.

## B.  Review of Previous Situation

Women Partnering used a few Excel spreadsheets and many paper forms to run their operations. This mostly paper-based system quickly became inadequate as the number of women partners, volunteers, donors, and support agencies

increased. The paper-based system was very manual and introduced errors to include data duplication, lack of timely retrieval of the data, and instances of paper forms getting filled out more than once for the same women partner, donor, support agencies, etc.

Women Partnering actively pursues funding through grants. As such, they supply reports with their grant applications. Likewise, some grants require periodic reports to be submitted. The grant reporting was difficult for Women Partnering to produce because the data was not easy to compile. Staff members and volunteers scanned all file folders and Excel Spreadsheets to compile the statistics needed for the grants. It was possible for the paper forms to be missed altogether or to be counted multiple times. Also, the women partner information in the file folders and in the Excel Spreadsheets was sometimes out of sync with each other. The volume of data, along with how the data was recorded, became a hindrance to applying for grants.

Women Partnering encountered the traditional problems when working with the paper forms, which were stored in filing cabinets. The staff could not readily locate the information when needed. The staff returned many calls because the information was filed away. Sometimes, the files were misfiled or were left on someone else's desk. Additionally, the staff was not able to easily identify new women partner contacts from the existing ones. When this occurred, a new file folder was created and personal information was collected again from the women partner.  Later, it may have been discovered that the women partner was not a

new partner at all, but rather an existing one. In the end, the staff was spending time filing, recreating, and locating paperwork instead of helping their partners.

The previous situation faced by Women Partnering did not enforce any business rules. Business rules are important to the business and to the information systems that may implement them. As defined by the Business Rule Group, "a business rule is a statement that defines or constrains some aspect of the business. It is intended to assert business structure, or to control, or influence the behavior of the business" (5). In many cases, the Women Partnering's forms were partially filled in. In several cases, the basic information about a women partner was missing like their name. All Women Partnering knew was that they helped someone out, but could not really say who they were helping. This missing information could not be used for grant reporting.

Paper-based systems do not enforce business rules. Women Partnering had a few complex business rules. For example, some support agencies offered support to certain ethnicities, had limits on number of times they would help, and/or had income limits. These rules could not be enforced by the paper-based system. Thus, the staff had to be memorized them.  Sometimes the staff mistakenly sent partners to agencies where they were not eligible to receive support.

## C.  Review of the Previous Automation Attempt

Women Partnering had was a previous attempt at solving them problems. The previous attempt was built by another individual. The previous attempt was an Access database application. Apparently, the database application was never completed. Thus, Women Partnering never used it. The Access database

contained only a few useless test records. Even though the previous attempt did not directly relate to the newer system, it was useful to understand the issues that Women Partnering experienced with it. This way the issues are avoided in the new system. The automation attempt had data entry, business rule enforcement, and relationship management issues.

A brief synopsis of the data entry issues follows. The flow of the data entry forms made it awkward to use. Pressing the tab key will move focus to the next control, which it did in the previous automation attempt. However, the next control that received focus did not always make sense. In a column of 3 controls, the first, third, and then the second column's control received focus. Likewise, there were a couple of cases were the focus jumped up to a control on top of the screen after leaving a control on the bottom of the screen. Then, the focus would return back to a control on the bottom of the screen. This jumping around made the system awkward to use.

Another data entry issue dealt with required fields, which could have been calculated. For example, the main form required the user to enter the number of children in one place and then enter the actual children's information elsewhere. Thus, it was possible to tell the previous system that a partner had 34 children, yet only have the 2 of the children's information entered. Worse, yet, the system saved the data this way. The number of children can be calculated based upon all of the children's information entered. Further, the count of children would never disagree with the actual number of children entered into the system.

The worse kinds of data entry issues are ones that corrupt or destroy the data. The previous attempt was plagued with data corruption issues. Agencies that the women partner previous sought out help from could be entered into the system, but was never saved in the database. Likewise, the partner's marital status retained the previously viewed record's value, which would then be saved. This would leave the wrong marital status stored in the database.

## D.   Goals of project

The project's goal was to address the business problems by creating an integrated, computer-based system for Women Partnering. Further, Women Partnering's issues with the previous automation attempt were to be avoided. The key deliverables for this project included a computer application and a networked database. The new system gave Women Partnering a system managing for the various interactions between the women partners, volunteers, donors, and support agencies. Instead of recording the data on paper forms, this project centralized all the data into one repository – the database. The database accommodates anticipated growth better than the previous paper-based system. With the new system, time spent tracking information about partners and other entities will decrease and shift over to time spent on helping the women partners. Also, the staff will become more productive when first learning the system. The system enforced the business rules instead of having the staff memorize them. While not a Women Partnering goal, the student had a goal to incorporate one cipher algorithm. By doing so, the student hoped to become more familiar with cryptography and its use in a computer system.

The project's success is measured by the quality of the system. Women Partnering believes that a high quality system will be easy to use. Yet, quality is subjective. However, the student approached the quality concern by placing an emphasis on testing. The student feels that quality is a concern of Women Partnering because there was a previous failed attempt at automating Women Partnering. Women Partnering never used the previous automation attempt because of the quality issues. The student agrees with Women Partnering that a quality system is one that will be useful to them.

## E. Barriers and/or issues

Women Partnering is a non-profit organization. As such, funds available to this project were non-existent. There were no time constraints imposed by Women Partnering. In fact, they preferred to implement this project slowly even though they are experiencing exponential growth. However, the student planned to have a technical solution in place by the end of September 2006. This time constraint was self-imposed to be able to complete academic requirements for graduation.

## F. Project Scope

The project ended when the following criteria were satisfied:

| Deliverable | Criterion Description |
|---|---|
| Academic | Lessons learned |
| Academic | Published Thesis |
| Academic | Thesis Presentation |
| Technical Solution | At least 80% of the Women Partnering staff is trained |
| Technical Solution | Future project ideas |
| Technical Solution | Future Project Ideas turned over to Women Partnering |
| Technical Solution | Programmer's Manual turned over to Women Partnering |
| Technical Solution | User's Manual turned over to Women Partnering |
| Technical Solution | Working system installed at Women Partnering |

## G. Definition of terms

The various terms used throughout this document are defined in alphabetical

order in Table 1 – Definitions.

**Table 1 – Definitions**

| Term | Definition |
|---|---|
| AES | Advanced Encryption Standard |
| DES | Data Encryption Standard |
| Entity Relationship Diagram | A diagram that is used as a communication device. The diagram presents entities and the various attributes associated with the entities. Additionally, an entity relationship diagram shows how the various entities relate to each other. |
| ERD | An abbreviation for an entity relationship diagram.  See Entity Relationship Diagram. |
| NUnit | A software program that is used to automate the running of unit tests. |
| PKI | Public Key Infrastructure |
| TDD | An abbreviation for Test-Driven Development. |
| Woman Partner | A woman who is financially vulnerable. |
| XP | Extreme Programming. |

## Chapter II.    Methodology Research

Software development means several things to different people. To some, software development is an art form. By applying creativity and ingenuity, a developer can create the next big software title. In this case, the developer feels that an engineering-like approach to software development can be too confining. Yet to others, an engineering-like approach is exactly what software development is suppose to be – following strict processes is the only way to build software systems. Sometimes this makes sense. For example, creating software that helps fly the space shuttle has to work flawlessly. In this case, there are millions of dollars at stake plus lives depending on the software working correctly. However, in the end, neither approach is right for all software development projects. Both approaches have numerous successes as well as numerous failures. The underlying problem here is software development is just not easy. What works for one situation does not work for all situations. There are many factors that influence the outcome of your software development project. Besides people, the software development life cycle that you follow is one of the biggest decisions that you can make on the project. Choose wisely.

Two software development life cycles will be analyzed in this paper. They are the Zachman Framework and Test-Driven Development (TDD). The goal is to highlight the strengths and weaknesses. What modern software developers must understand is that one has to be insightful and flexible enough to adapt the software development processes to the situation at hand. However, in order to adapt the software development processes used, one must first understand their

strengths and weaknesses. Only then can the developer steer their project away from crashing into the rocks of failure.

## A.  Zachman Framework

First off, the Zachman Framework is considered. While most other software development life cycles are split up into phases and then further broken up into steps, the Zachman Framework views software development from a different point of view. Here, the Zachman Framework considers the perspective of those involved and topic areas (Hay "Requirement", 1). In fact, the grid used to describe the Zachman Framework is laid out with perspectives on one axis and the topic areas in the other. The Zachman Framework is shown in the Chapter VII.D – The Zachman Framework. The topic areas contain more areas than are traditionally considered during software development. Software developers tend to focus in on the functionality provided by the system and the data that is to be processed. So, then, the Zachman Framework helps remind us that the where, who, when, and why are also important when building software. At the intersections between the perspectives and the topic areas are the building blocks of an information system (Whitten, 52).  These building blocks become more detailed as you move closer to the bottom of the framework. Thus, the Zachman Framework offers many of us a natural way of thinking about information systems.

Besides defining a framework to organize our thinking about an information system, the Zachman Framework defines an enterprise-wide architecture as described by one author:

The architecture serves as an "enterprise blueprint." It is a repository for designs and specifications of physical data structures and applications, as well as business plans, data models, and process models. Furthermore, it serves as a map of all the linkages among business initiatives, data required to support those initiatives, business processes that use the data, and physical information systems that support data requirements and processes. (Perkins, 8)

The enterprise-wide approach provides a holistic view of a business and its information systems. It is comprehensive and rigorous whereby a full set of plans and documentation are produced (Wikipedia Enterprise, 1). Thus, the Zachman Framework is a process-heavy and a documentation-heavy software development life cycle.  The planner's perspective is the top layer within the architecture. Plans are created and become more detailed and technical as the plans proceed from the top perspective down to the bottom one. Another way of looking at it is that the planners plan, the business owners provide requirements, which then are translated into the architecture view by performing requirements analysis and so on until all the details of the system are captured in documentation. Then, the system can be built. So, the Zachman Framework follows a waterfall type of flow through the software development life cycle. The main difference from the traditional waterfall software development life cycle is that the Zachman Framework addresses an enterprise-wide view and not just an individual project.

## 1.  Zachman Framework Benefits

The Zachman framework benefits from the emphasis on perspectives. Perspectives are important, but are sometimes ignored! For example, it is just silly to write a paper without knowing who the audience is. Likewise, this can be a

problem with software development. The various models and diagrams are created throughout the process of building software. Presenting an entity-relationship diagram to top-level executives just does not make any sense. With the Zachman framework, identifying the intended audience is exactly where the perspectives come into play. The top-level executives will not understand the entity-relationship diagram, but the database designers will. So, the perspectives help make sure that the software building block is directed towards the correct audience. With the correct audience, the software developer is able to the clearly understand the processes, data, and interfaces, which the system must contend with by being able to effectively communicate with the project stakeholders.

With the Zachman Framework addressing an enterprise-wide view of the information and systems, it should be worked by larger software development teams. It does not rely on the tacit knowledge of the team members. Instead, it relies on the knowledge captured in the form of plans and diagrams. If a key team member leaves the company, then the knowledge pool is still intact. Thus, the Zachman Framework is not affected by employee turnover, which can hurt agile teams. Further, a new employee can quickly come up to speed and be a valuable team member quicker by reading the documentation. On the flip side, the Zachman Framework does not seem to be viable software development life cycle when the team is small and there is a large backlog of projects. In this case, the team will spend all of its time documenting changes instead of delivering projects.

Another benefit for the Zachman Framework is it is less likely to create duplication in data and in systems than an agile method would. The enterprise-wide view of systems and information offer a single top-down view. This prevents duplication of information and systems from getting built. The Zachman Framework creates a master set of documentation that incorporates all information and systems. If there is a question about a particular topic area, then the answers can be ascertained by consulting the next layer up in the framework. This removes assumptions from the project and removes the guesswork that leads to duplication.

When there is a potential for loss of life or where a significant amount of money is at stake, the Zachman Framework is better choice for a software development life cycle. For example, software that sends someone to the moon, software running in a satellite, or software running a life-support system in a hospital environment has to work. The practitioners produce documentation, review it, and double-check it for any errors.  Further, they build contingency plans to address project risks.

Another area considered is team size. With the Zachman Framework being a process-heavy software development life cycle, there are many documents created. At the very least there is one document per system building block, which means that there are at least sixty documents that are maintained. Why is the answer not thirty documents since there are thirty building blocks? Well, it is true that there are at least thirty documents, but there are two copies of each document– one is for "as is" system; another is for the "to be" system. The

other assumption made is that there is only one piece of documentation per building block, which is unlikely. It would be impossible to describe the information system for the enterprise in just one document per building block. Even if you could, the document would be voluminous. One advantage of having all of this documentation is that project communications are easier. The need for face-to-face communication is reduced when the knowledge contained within the documents can be shared with whoever needs the information. Thus, the Zachman Framework can easily support larger team sizes, but may over tax a smaller team especially if they are working in a rapidly changing environment where the requirements are changing. They would do nothing but changing the documents.

## 2. Zachman Framework Issues

The Zachman Framework may failure in dynamic environments. In a dynamic environment, the business changes may cause the requirements to change rapidly. With this situation, the Zachman Framework documentation is always in flux. The team may not be able to keep up with the changes. Keeping the documentation current, the team's need for discipline gets in the way of keeping up with the shifting business directions. However, the Zachman Framework is perfectly suited in environments where this is not rapid and dynamic changes.

As stated before, the Zachman framework is a very appealing approach because it offers a natural way of thinking about information and software development.  However, as Simsion points out, there are several issues with it: 1) lacks pursuit of alternative classification of data by practitioners; 2) a tactical

approach can be more successful than an enterprise approach; and 3) where is the evidence that the framework really works? (8). Even though the Zachman framework is a viable software development life cycle, it is becoming dated. Zachman first conceived the framework back in 1987. As such, the student had hard time finding current information about the Zachman Framework. It is getting overshadowed by more recent approaches to software development – namely, extreme programming, which is touched upon next.

## B. Test-Driven Development

Test-Driven Development (TDD) is considered next. It is an agile approach to developing software. Agile programming is also known as extreme programming. Contrary to the Zachman Framework, the agile software development life cycles are not documentation based. Instead, they focused on getting the software in the hands of the users. "Agile methods are an outgrowth of rapid prototyping and rapid development experiences as well as the resurgence of the philosophy that programming is a craft rather than an industrial process" (Boehm, 16). Using the Test-Driven Development approach to developing software, the testing comes first. This seems a little backward at first. How can you test the system if you have not gone through the traditional waterfall phases of analysis, design, and code? Well, Doshi points out that Test-Driven Development is not about testing – it is about "evolving the design to meet the requirements" (1).

So, how does Test-Driven Development work? Well, there are a few easy steps one must follow: 1) write a test; 2) write code to pass the test; and 3) refractor the code to remove duplication to make it simpler, more flexible, and

easier to understand (Stott, 2). Sometimes others split up the second step into

two parts: write just enough code so that everything complies, but the test fails

(Wikipedia Test, 10) and then one should finish the code getting the test to pass.

### 1. __Test-Driven Development Benefits__

Test-Driven Development (TDD) creates a prototype. Prototyping has many

benefits. First of all, they can help with clarifying and completing the

requirements, exploring design alternatives, and implementing layers

progressively (Wiegers, 234). The use of prototypes has direct relationships with

many of the agile concepts. First, Wiegers states, "Envisioning a future software

system and articulating its requirements is hard to do" (233). Building a prototype

helps figure out what the system is to do.  With the agile approach, it also

recognizes that users may not know what they want until they see it. Using a

simple design, quickly getting the system into the hands of the customer, and

recognizing that the requirements may change is much like prototyping. Both

prototyping and the agile approach try to engage the users early to elicit their

input. Effort should be minimized when creating a prototype, which supports the

agile concept of fast delivery cycles.  Additionally, a prototype can be elaborated

into the final system through multiple iterations. This is just like the agile concepts

of fast cycle/frequent deliveries.

As with other extreme programming techniques, TDD identifies quality

attributes. Users and system builders tend to focus in on what the system is to do

(Wiegers, 216). They overlook the quality attributes of availability, efficiency,

flexibility, integrity, interoperability, reliability, robustness, usability,

maintainability, portability, reusability, and testability. Further, the quality attributes can distinguish between a mediocre system and a great system. The student does not see any direct relationship between the non-functional requirements and the agile concepts, which are embrace change, fast cycle/frequent delivery, simple design, refactoring, pair programming, retrospective, tacit knowledge, and test-driven development. However, the student can infer some relationships. First, "Quality attributes are difficult to define" (Wiegers, 216). Therefore, by following the agile concept of fast cycle/frequent delivery, one can uncover missing quality attributes early and reduce the risk of delivering a mediocre system in the end. Second, following the agile concept of simple design directly supports the quality attribute of maintainability. However, maintainability might not be a priority to the users. If the priority is robustness, portability, or flexibility, then the simple design will not support the user's requirements.  So, the agile concepts are sometimes in alignment with the quality attributes.

In TDD, assigning priorities to each requirement is important. This helps the system get implemented when there are limited resources. The requirement priorities integrate well with the test-driven development. Higher priority requirements will be implemented first. This gives the users the greatest benefit at the beginning of the project. Test-Driven Development is indifferent to shifting priorities. The newer set of priorities will be included in the next iteration. This is one of the agile concepts of adaptability.

As for ideal team size, test-driven development favors smaller teams. Since test-driven development is an extreme programming software development life cycle, a lightweight process is emphasized. This means there is little to no documentation. Besides, why write documentation when you are going to have to maintain and no one is really going to read it any way? At least, that is what the extreme programmer thinks. Now, with that being said, the student believes that the test-driven development can be supported in larger teams because the system design is documented in unit tests. So, test-driven development fairs better than other extreme software development life cycles in larger teams. It relies on communications between the team members to be more face-to-face. This means that test-driven development works great for small to medium sized teams. The number of communication points between all members team grows exponentially for each team member added. On a large team, the number of communication points will be large.

The test driven development is made possible only through the use of automated unit test program like JUnit or NUnit. See Chapter VII.C – Example NUnit Screen for an example. Most of the unit test tools are freely available on the Internet, so no additional funds are needed by the project. These test tools are able to run an entire suite of unit tests and report back any encountered errors. The student used NUnit as the unit test tool for his project, which is written in C#. NUnit uses the red, yellow, and green colors to indicate the status of the tests. Red is failure; yellow is an ignored test; and green for a properly working tests. The ability to run the unit tests frequency and quickly is a plus for test

driven development. Any code changes that break the functionality are caught within minutes of the code changes. The developers making the change can the fix the error while the code change is still fresh in his/her thoughts.

Stott also points out that "the long gaps between the design, coding, and testing phases are gone, thus making for a much better learning environment" (3). This quick feed back from the unit test tool has several benefits. First, the development gains confidence in the changes knowing that the changes will not break the overall design of the system. This is especially helpful to a new software developer joining a team. Second, the software developer is able to make changes to the code (think – refactor the code) to make the improvements and remove duplications knowing that he or she has not broken the interface. Thirdly, the unit tests are accumulated over time to create a regression test bed. Currently, the student has 3300 unit tests that run in about 10 minutes. The unit tests can be organized into suites of unit tests. Further, the unit tests can be place into different categories. The software developer has the option to run all the unit tests, a certain suit of tests, or any combinations of unit test categories. Combinations of tests that the student created are unit tests for all user interfaces, business rules, and database transactions. These unit tests consist of 57,102 lines of code. This student disagrees with Doshi in that a unit test can interact with files and databases. Doshi's point of view states: "A test that does not operate in isolation is not a unit test. It is safe to assume that a test that connects to the network or a database or a real file is not a unit test" (1). The student has designed techniques for working with the database whereby the

database is returned back to an initial before the next unit test runs. Thus, the unit tests are isolated from one another and the unit tests works even though it connects to a database.

Contrary to Zachman Framework, test-driven development is not a process-heavy software development life cycle. Instead, it is based upon a lightweight process known as extreme programming. In extreme programming, the knowledge is tactical. The knowledge lies in the brains of the development staff and not on paper. Besides, why waste time documenting the system requirements and designs if they are going to change? Why not just plan on them changing? Test-driven development emphases an interactive process of writing the unit test, write the code, and refactor the code all along you are running the unit test at each step.

Contrary to the Zachman Framework, Test-Driven Development is well suited for dynamic environments where the requirements are changing quickly. Since there is little or no documentation, the test-driven practitioners can quickly adjust direction with minimal impact. Further, some users are unable to fully describe what they want system to do. They may not know what they want until they are able to see the system in action. Because the test-driven development uses an iterative process, changes and user feedback can be fed into the next interaction of development.

## 2. <u>Test-Driven Development Issues</u>

The extra lines of code written for unit tests are overhead. They will never get deployed into a production environment, nor do they satisfy any functional needs

of the users. After all, they are unit tests. In total, the student's project had

102,643 lines of code of which 57,102 of them are unit tests. Thus, the project's

unit test overhead is 56%. While this might seem excessive, the student's unit

test overhead is a little over what is considered normal. Jon Udell states, "The

overhead can be substantial, however, because the test framework that ensures

a program's correctness may require as many lines of code as the program itself"

(2). Even the test-driven development approach founder, Kent Beck, had a 50%

overhead of functional code and unit test code for a large project he worked on

(Ambler "Introduction", 20). In a world where being first to market can make or

break a business, having an additional 50% lines of code just does not make

sense at first. However, looking at what the 50% lines of code offers in terms of

benefits, and then it does not look so bad. One just has to take into account in

the project plan that TDD will result in more lines of code generated than using

other methodologies.

Besides the additional lines-of-code overhead, there are a couple of other

issues with test driven development. This student has spent years developing

programs using object-orientated techniques and structured, top-down

techniques. What the student found is that he tended to focus on building the

system bottom-up while following the test-driven development approach.

Meaning, he was stuck in the details. Later, he would discover that the functional

code did not make sense into the overall solution, yet it was tested thoroughly!

For example, he created unit tests for a dataset on a particular table and wrote all

the functional code working for that table. Later, he went to integrate that dataset

into the final solution and discovered that it was not needed. Part of a day was lost working on the unit tests and the functional code. Like Doshi states, "Each unit test corresponds to a single requirement that the code must satisfy" (1). However, when the requirements are not clear, then there is a chance that you are writing throwaway code. This is where the student should have followed is one of the extreme programming concept of "you aren't going to need it" or the YAGNI (Boehm, 41). Just like the student, any software developer can get distracted with the problem at hand only to find out later that the wrong problem was solved. This is why confirming the requirements as you go are important in TDD.

A possible weakness of the test-driven development is that it does not take an enterprise-wide view of the information and systems. Instead, the development cycle is focused on just one project, which can lead to the system being built in its own "silo." The silo effect means that there can be duplication of functionality and data between the various systems within an organization. Yes, being that it may, this weakness can be turned into strength as compared to the Zachman Framework. "A tactical approach to data management, based on individual high-value initiatives, is likely to be more successful than one centered on an enterprise architecture" (Simsion, 9). Test-driven development definitely supports tactical approaches.

Another minor weakness of the test-driven development approach is it not based on documentation. Thus if a key member leaves, his knowledge leaves with him. Or does it? Yes, it is true, that when the team member leaves the

group, his knowledge leaves with him. However, provided that this team member has followed the test-driven approach, he has written unit tests, which captures his knowledge in the form of unit tests. So, his knowledge about the next steps and his business knowledge are no longer available. However, at least he has written tests that a new employee is able to run. Further, the unit tests capture the requirements of the system. When the new team member has changed the code, he/she is given immediate feedback if something was adversely affected by the changes.

With test-driven development, everything cannot be tested. For example, user interfaces are a stumbling point. Visual aspects about the screen layout require a human verification. A unit test cannot tell if the zip code field on the screen is too small and is not displaying the complete contents of the field. Further, there are other concepts that defy unit testing. For example, does the tab order make sense? What should the tool tip say when the mouse hovers over a control? Are the report contents correct? Some believe that the user interface is completely un-testable by the unit tests. However, there are some aspects that lend themselves to unit testing. For example, when populating the search field with a valid value and then pressing the search button -- the screen should display the correct data. All of these actions, even the pressing of the search button, can be put into a unit test which can be written so that the correct data is displayed back after the search button has been pressed.

Seeing the actual user interfaces as the unit test tool runs the tests is a challenge, which the student was able to overcome. Without doing anything

special, you will see the form partially painted on the screen as the test runs. See

Chapter VII.F – Form Incorrectly Painted for an example. In order to see the user

interface fully painted on the screen, two tasks are required. First, run the user

interface in a separate thread. This will allow the user interface to properly draw

itself. If you stop here, you will get random errors as the unit tests run. After much

research and headaches, the student discovered that the unit test tool, running in

a separate thread from the form, should not execute any methods on the form

directly. "Never directly access a property or invoke a method of a

System.Windows.Forms.Control object or any object that inherits from this class

if there is any chance that the code running in a thread different from the thread

that created the control" (Balena, 332). Under the covers, the issue is that the

form is not thread safe. The student did not know this, so the random errors

plagued the student's unit tests for months. The second task is to use the Form's

Invoke method and pass in the delegate to the method that you wish to execute.

See Figure 1 – Marshalling a Method Call onto another Thread for an example

for how this is done. Once you implement the two steps as outlined, you will see

a fully painted user interfaces as the unit test runs. See Chapter VII.G – Fully

Painted User Interface for an example of this.

**Figure 1 – Marshalling a Method Call onto another Thread**

```
#region SetField
/// <summary>
/// This method is used to set a field's data on the form running on
a different thread.
/// </summary>
/// <param name="setter">The setter method</param>
/// <param name="new_data">The new data</param>
protected void SetField(SetFieldData setter, string new_data)
{
   object [] args = new object[] {new_data};
   runnableForm.Invoke( setter, args);
}
#endregion SetField
```

## C. Securing the System

Regardless of the exact methodology followed by a software developer, one must apply measures to secure the system. The system that is not secure is open to attack, which can lead to loss of sensitive data, corruption of data, and loss of system availability. While system security was not a direct requirement levied against this project by Women Partnering, the student understood the importance of making sure that defensive measures were needed in their system.

The student knew that one form of defensive measure was to encrypt data within the system. Then, the student considered what ciphers were available. A cipher is a form of cryptography that is used to encrypt and decrypt messages. For this project's purposes, the messages are a few of the data elements passed between the application and the database. A few of the fields in the database are stored encrypted, so that the data cannot be ascertained by running a query against the database.

Cipher algorithms are classified into asymmetric or symmetric ciphers (Cross, 499-500). Further, symmetric ciphers are subdivided into steam ciphers and block ciphers.

## 1. **Asymmetric Cipher Algorithms**

Asymmetric algorithm requires two keys. The two keys are mathematically related and usually involve very large prime integers. A message encrypted with one key can only be decrypted with the other key. Asymmetric algorithms are used primarily in public key infrastructures (PKI). One of the two keys is considered the private key. Private keys should be secured and not to be disclosed to anyone else. The other key is the public, which is available to anyone who needs to communicate with the private key owner. Asymmetric algorithms are considerably slower than symmetric algorithms especially when the message sizes are larger.

## 2. **Symmetric Cipher Algorithms**

Besides asymmetric algorithms, there are symmetric algorithms. There are more symmetric algorithms than there are asymmetric algorithms. The reason is symmetric algorithms are faster than asymmetric algorithms and because symmetric algorithms only require a single key. Thus, symmetric algorithms are simpler to develop. The single key in symmetric algorithms is called the secret key, which is used to encipher (encryption) and decipher (decryption). One challenge with symmetric algorithms is how to securely share the secret key between the message sender and the message receiver. The pro for symmetric

algorithms is speed; while the con is that they are vulnerable to brute-force attacks (Cross, 500).

Symmetric encryption algorithms can be further sub divided into stream ciphers and block ciphers. Regardless of this sub-division, the symmetric encryption algorithms still require one key for the encrypting and decrypting the message.

First, stream ciphers process small individual units, usually bits, during the encipher/decipher cycle.  Because stream ciphers process small pieces of data, they are faster than block ciphers (Cross, 506). In stream ciphers, a key is combined with the plain text to produce the cipher text. It is interesting to note that any particular plaintext will be encrypted differently depending its location within the plaintext (RSA Stream, 1).  This is not the case with block ciphers. The same plaintext message in block 1 and block 2003 will have the same cipher text! One desirable property of the stream cipher is one-time pads. A one-time pad means that the secret key is used once and then is discarded (RSA Stream, 3). With each new plaintext to be enciphered, a random secret key will be used. The one-time pad helps to defend against statistical attacks. Stream ciphers using a constant secret key are vulnerable to statistical attacks (Cross, 103). One example of a stream cipher is the RC4 encryption algorithm, which is used in the Wireless Encryption Protocol.

Block ciphers differ from stream ciphers in that they manipulate a large block of data. The block itself can be variable length. However, once a block length is chosen, it is used throughout the entire encipher/decipher process. Each block is

processed using the same algorithm and the same key. However, the key is usually broken into pieces and each part is used during the iterations. The key to be applied during each interaction is called the key schedule. The encipher/decipher process within block ciphers can be iterated, which means that the process is repeated a number of times (RSA Iterated, 1). When iterations are involved, the block cipher is called an iterated block cipher (RSA Iterated, 1). Regardless of the key schedule used during each round, the block length remains fixed.

### 3. **Rijndael Cipher**

The student wanted to explore the Rijndael algorithm. Two Belgian cryptographers named Joan Daemen and Vincent Rijmen created the Rijndael algorithm. The Rijndael algorithm was submitted and eventually approved for the United States Government's Advance Encryption Standard (AES) in November of 2001. The creators had three goals in mind when creating the Rijndael cipher: resist against all known attacks, speed and code compactness, and design simplicity (Daemen, 8). The Rijndael cipher can be implemented in software and hardware including devices that lack processing power like smartcards.

As for how the Rijndael algorithm is classified, it is considered to be a symmetric algorithm (Wikipedia "Block", 1). One key is used for ciphering and deciphering the message. The Rijndael algorithm is further classified as being a block cipher. While Rijndael supports larger block sizes and key sizes, AES confines the block sizes to 128 bits (Wikipedia "Advanced", 5). Each block is represented as a matrix. The number of rows in each block is fixed to be 4 rows.

As for the number of columns, the exact number depends on the block size divided by 32 (Daemen, 8). So, under AES, the number of columns equals 4. Additionally, the cipher key is also represented as a block. Again, the number of rows in the cipher key block is fixed at 4 rows. Just like the cipher data block, the number of columns is calculated. The number of columns is equal to the key size divided by 32 (Daemen, 9). Under AES, the key sizes can be 128, 192, and 256 bits (Wikipedia "Advanced", 5). The three key sizes are known as AES-128, AES-192, and AES-256. The numbers of columns in the key cipher blocks are 4, 6, or 8 under the AES specification.

Once the cipher data block and the key cipher block have been determined, the data is loaded into the blocks and the cipher process starts. The Rijndael cipher processes a number of rounds depending on the key size. So, in addition to be being a block cipher, the Rijndael cipher is considered to be an iterated block cipher. In AES, 10 rounds are used for the 128-bit key, 12 rounds for the 192-bit keys, and 14 rounds of the 256-bit keys (Wikipedia "Advanced", 16). Each round, except the last, consists of 4 steps: subbytes, shiftrows, mixcolumns, and addroundkey. By design, the last round omits the addroundkey step.

"[Ferguson, Schroeppel, and Whiting] know of no other 'serious' block cipher that has an algebraic description that is anywhere near as simple as the one for Rijndael" (6). So, what does this all mean? The answer is simply that the Rijndael is simple to implement (following the 4 steps in each round) and can be expressed via a simple algebraic formula. However, Rijndael is a very hard-core cipher! Under the AES implementation of Rijndael, there are 3 key sizes: 128,

192, and 256 bits. Therefore, there are 3.4 x $10^{38}$ possible keys using a 128-bit key, 6.2 x $10^{57}$ possible keys using a 192-bit key, and 1.1 x $10^{77}$ possible keys using a 256-bit key (Computer, 15). To put another way, consider the following:

> Assuming that one could build a machine that could recover a DES key in a second (i.e., try $2^{55}$ keys per second), then it would take that machine approximately 149 thousand-billion (149 trillion) years to crack a 128-bit AES key. To put that into perspective, the universe is believed to be less than 20 billion years old (Computer, 16).

The above takes into account the smallest key size as specified by the AES.

With the cipher complexity and number of possible keys in AES's version of the Rijndael cipher, it is expected to have a useful life of twenty year's time (Computer, 18). This of course assumes that the only attack possible is a brute force attack. Additionally, it does not take into account any further advances in CPU processor speeds. However, the student thinks that the next twenty year's worth of CPU processor increases will do little to reduce the brute-force timeframe of 149 trillion years by any significant measurable amount.

Since the Rijndael cipher was proposed to National Institute of Standards and Technology (NIST) for the AES standard back in 1996, the Rijndael cipher has been under review by crypto analyst around the world. Crypto analyst considers a cipher break as any technique that is faster than the brute force approach (Wikipedia Advanced, 15). "As of 2005, no successful attacks against AES have been recognized" (Wikipedia "Advanced", 13). However, there has been a claim made that there is a break, but this claim failed to be verified. This so-called attack was called the XSL attack. Time will tell if Rijndael is a viable cipher for the next twenty years.

Initially, the Rijndael cipher was only to be used for the US Government's non-classified data. However, in 2003, the Rijndael cipher can be used for classified data at all key lengths (Wikipedia, Advanced, 11). Further, it can be used for securing top-secret data as long as the 192-bit or 256-bit key lengths are used. The Rijndael cipher seems very secure. Besides being a government standard, it may gain enough momentum to be considered the worldwide standard for data encryption.

## D. Methodology Research Conclusion

This chapter looked at two different approaches to software development life cycles. First, the Zachman Framework was looked it. It offered a comprehensive view of the business and of its information systems. The architecture of enterprise can be broken down into different perspectives and into various topic areas. Just like the traditional waterfall software development life cycle, the Zachman Framework's holistic view of software development follows the same flow. The Framework forces the software developers to view more than processes and data. It also looks to other concerns like when, where, and why. The Zachman Framework is a very natural approach to viewing and building software for the enterprise.

As a newer software development life cycle, test-driven development has its roots in extreme program. Here, the focus is writing a test case for the software – even before the software has been written. This is awkward to get use to, but the benefits are many. Creating the unit tests first, this forces one to think through the interface first before writing the actual software. Further, unit tests provide quick feedback to the developers when they have negatively impacted

the design of the overall system. This is especially helpful to newer team members. Overtime, the unit tests become a regression test bed for the entire system. Using a unit test tool like NUnit automates the execution of the unit tests.

This chapter also looked at the Rijndael crypto. It is classified as a symmetrical crypto meaning that it uses one key for both encrypting and decrypting the message. Further, the Rijndael is a block cipher that is very secure and is resistant to all known attacks. The Rijndael cipher is secure enough and simple enough to be used in the overall implementation of the Women Partnering's system.

## E. Contributions Made

This project contributes to the industry by having a student new to Test-Driven Development follow the process to create a small system for a non-profit organization. The strengths and weaknesses of the methodology are pointed out. With Test-Driven Development being relatively new, it is compared and contrasted against the older Zachman Framework to see if how it measures up. Further, the student recounts many of the lessons that he learned along the away. This way, the student hopes that the reader will avoid the same pitfalls encountered and will be able to further build upon the student's experiences and advance TDD.

## F. Planned Methodology

Initially, the student had planned on following a waterfall software development life cycle as identified below.

## 1.  Analysis Phase

During the analysis phase, the existing system was to be studied. Input from Women Partnering is critical during the analysis phase. Planned activities included reviewing of the existing system, reviewing existing forms, conducting interviews, and observing current business activities. Throughout the analysis phase, understanding of the existing business and problems was to be documented.  The requirements for the new system will be captured in a requirement specification document. It is expected that Women Partnering approved the requirement specification document before continuing on with the next phase of the project. Upon signoff, the feasibility criteria would be defined. These criteria would help identify a viable candidate solution.

## 2.  Design Phase

The recommended candidate solution along with the requirement specification is the inputs in to the design phase. The design's goal is to create a system blueprint. The design phase's key deliverables are a network design specification and an application design specification. Women Partnering would need to approve the design specifications before proceeding to the Construction phase.

## 3.  Construction Phase

The construction phase executes all plans. The system is constructed, the database is defined, and any changes applied to the network. Also during this phase, a programmer's manual and network documentation would be written and turned over to Women Partnering before the end of the Construction phase.

### 4.  Testing Phase

Concurrent to the Construction phase, unit and system testing would be conducted. The Testing Phase would produce a test plan and test results. Testing ensures that all requirements have been included in the system and properly work. Women Partnering would need to approve the test plan before finishing the testing phase.

### 5.  Implementation Phase

The implementation phase is where the new application, database, and network changes were made available to Women Partnering. Prior to the actual implementation, Women Partnering would need to approve the implementation plan that was written during this phase. As part of the implementation phase, training materials would be written. Also, training sessions for the staff would occur.

## G.  Actual Methodology

As some point during the initial analysis, the student became aware that it was very hard for Women Partnering to express the requirements of the system upfront. Generally, they would know what they wanted when they saw it. So, the initially planned methodology was not going to work on this project. Rather, the student needed to follow a methodology that got the system in the hands of the users, so that the requirements could be confirmed. Further, because of the emphasis of quality as a system goal that was expressed by Women Partnering, the student discovered that Test-Driven Development would be a better methodology and would be able to meet the needs of this project. Test-Driven

Development (TDD) consists of iterations of 3 steps as described in the following paragraphs. Further, the 3 steps are repeated until the system is done.

### 1.  Write Unit Test

 Before writing any functional code in the system, the developer has to write the unit test for a requirement first. This is a different mindset that many developers are use to – testing is the last thing you do before the system gets implemented into a production environment. By writing the unit test first, this forces the developer to think through the interface. Interfaces are the means by which the system building blocks work together to satisfy the needs of the users, yet they are not emphasized in other methodologies as they are in TDD. Instead, the code behind the interface draws the developer's attention first. TDD emphasizes the interface by having the developer write a unit test to test the interface before anything else is written. At this point the unit test should fail. In fact, the unit test should not even compile because there is no functional code written yet. If desired, the functional code can be written as a stub, which means the interface exists but there is no code written beyond that. By writing a stub, this gets the system to the point where it can at least compile without errors.

### 2.  Write Functional Code

 Once the test has been written, the functional code should be written. This step is pretty basic – get the interface operational. At this point, the quality of the code is not important. Rather, achieving the desired outcome is important. The desire outcome of this step is getting the unit test to a passed status.

## 3.  <u>Refactor</u>

One of the neatest aspects with TDD occurs in the refactor step. In this step, the developer's attention turns to cleaning up the code that he/she just wrote. In other methodologies, this step is non-existent. The refactor steps forces the developer to consider the design of the functional code and make changes to enhance the quality of the code, the efficiency of the code, and the robustness of the code. Regardless of the changes made during the refactor step, the developer can run (and should frequently run) the unit test that he/she wrote in the first step to validate that the test still produces the intended results.

## Chapter III.   Test Driving Test-Driven Development

### A.  Project Analysis

Analysis is the study of the existing system and problem domain. Without analysis, the problem domain would not be fully understood. The information technology industry is plagued with failed attempts at solving the business problem. "And some three quarters of all large systems are 'operating failures' that either do not function as intended or are not used at all" (Gibbs, 43). To compound this problem even further is the fact that "Systems have become larger and more complex than ever before" (Christensen, 5). Today's environment also demands that these larger and more complex systems get created faster and faster to keep up the increased levels of competition.

Analysis is also important because it lays the foundation for the rest of the software development life cycle processes. Under the Zachman Framework, the requirement specification is produced during the analysis phase. "Nowhere more than in the requirements process do the interests of all the stakeholders in a software or system project intersect" (Wiegers, 4). All stakeholders use the requirements specification to build, to test, to design, to market, to write user documentation, etc. Any problems introduced during the analysis phase will cause potential rework in later phases or cause the project to cancel. Of course, the rework will cause the schedule to slip, demand extra resources, and/or changes to the project scope. Further, the rework can have a cascading affect on the rest of the system – just like tossing a stone into a lake can cause a rippling effect throughout the entire lake.

Once the problem is fully understood, only then can a clear and simple solution be realized. Yet, analysis can be very hard to perform. Different stakeholders have conflicting options on the business problems and the intended solution. Further, the business problem evolves as the analysis is performed. This is exactly what happened in this project. Women Partnering launched a series of classes called Spiritual Networking right in the middle of the analysis.

The Zachman Framework's approach of completing the analysis before proceeding with the system design would not have worked on this project. The requirements were not well understood and consequently were hard for Women Partnering to express. With the TDD approach, the requirements are captured in unit tests as the system is built. This form of prototyping allowed for the requirements to be validated as the system was being built. Further, any requirement conflicts were flushed out as soon as they were implemented into unit tests.
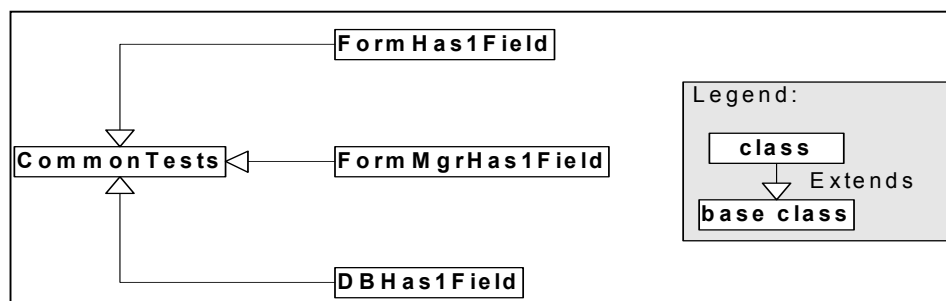
## B. Handling of the Design

As mentioned before, Test-Driven Development is an extreme programming technique for developing software. It is a lightweight process where the emphasis is on speed and getting the software in the hands of the customer quickly. Test-Driven Development captures the design of the system in the tests, which are written before the functional code is written. This forces the developer to hone in on the interface first. After the intent of the interface is captured in the test, the developer will then write the code that implements the interface.

As an example of how the design is captured in the tests, the student followed several of one author's suggestions to improve the overall C# design. In

particular, Wagner suggests that one should always provide ToString() method to help with others being able to understand the contents of your types (38). So, the student created an interface call IToString, which all tests dealing with user-defined types implemented. This interface ensured that the ToString method was tested. Further, making sure that it is tested means that all user-defined types had to follow the guideline of always providing the ToString method on all user-defined types. Later, the student combined the IToString interface with inheritance. The student found that all presentation layer tests could inherit from a base presentation layer tests class whereby the IToString interface was implemented. This was also true for the business logic layer tests and all database access layer tests. So, it made sense to extract all the common tests like the test that made sure the ToString method was provided into a common tests class as shown in Figure 2 – Common Tests.

**Figure 2 – Common Tests**



As for the design within the Zachman Framework, it is more documentation based. The design is not activity used to make sure the system is functioning correctly. Further, with the design based in documentation, it can easily become out of sync with the system. The student has experienced many projects where

the design documentation could not be trusted 100% of the time. Many

developers resorted to trusting the actual code in lieu of reading the design

documentation. The student works in an environment where many systems

evolve through hundreds of projects throughout the years. The design

documentation is specific to each project. In the end, there is not one complete

view of the system design. Further, referencing a design published one year ago

may not represent an accurate picture of the system today. Was there another

project that changed the design between this older design and what is there

today? This question plagues the approach of having the designs documented.

Because of the possible staleness of the design documentation, this highlights

the beauty of the test-driven development. Remember that the unit tests capture

the design. The tests are created over time. At any point in time, the entire test

suite can be executed to ensure that the design is intact and is valid.

## C.  When Testing Occurs

Testing is the biggest difference between Test-Driven Development and

Zachman Frame methodologies. The testing of the system occurs throughout the

entire life cycle of the system when using Test-Driven Development. This pay as

you go approach to testing identifies errors at the point in time when the error is

introduced. This has the benefit of having the coding change still being fresh in

his/her thoughts. It is well known that testing improves the quality of the system

(Smith, 1) (Murphy, 1). "By reducing the feedback loop, the time between

creating something and validating it, you will clearly reduce the cost of change"

(Ambler "Examining", 3). Further, the feedback loop is compressed because in

test-driven development, only small steps are taken (Ambler "Introduction", 3). It is interesting to note that the Zachman Framework does not explicitly define when or even if testing should occur. Instead, it focuses on the analysis and design of the system. However, given that the Zachman Framework focuses on an enterprise-wide view of the business captured in models, the student infers that testing would occur at the end of the project. This means that testing does not follow the pay as you go model. As Ambler points out, the danger of this is that the cost of change grows exponentially as the project progresses when following a waterfall type of software development ("Examining", 3).

## D. Business Rules

As defined by the Business Rule Group, "a business rule is a statement that defines or constrains some aspect of the business. It is intended to assert business structure, or to control, or influence the behavior of the business" (Business Rule Group, 5).  One cannot ignore the business rules and still be successful. Yet, under the TDD approach, business rules are not formally addressed by the methodology. It is the student's belief that business rules are to be expressed as requirements, which are then later transformed into unit tests.

As for business rules under the Zachman Framework, entity relationship diagrams (ERDs) start to capture the business rules. An ERD shows the various data entities and how they relate to each other. However, they do not describe everything that is needed to know about the entities existence. In fact, data models like ERDs depict structure of the data, but they do not depict how or when the entities are to be used (Hay "What", 1). In other words, data models fail

to depict business rules.  At least the Zachman Framework does account for the "how" and "when" aspects in other topic areas within the Framework.

### E.  Data Dictionary

The student created a data dictionary for this project. This proved to be useful, and provided many benefits to this project. Besides helping the student learn about Women Partnering's data, the data dictionary laid the foundation for creating the database. "Usually [data dictionary] means a table in a database that stores the names, field types, length, and other characteristics of the fields in the database tables" (Foldoc, 3). However, manually created data dictionaries work just as well. Even before having a database and tables, a data dictionary can help with user-to-developer communications and help with many of the other software development processes.

This project used a data dictionary to help with the project analysis. A data dictionary is "a shared repository that defines the meaning, data type, length, format, necessary precision, and allowed range or list of values for all data elements or attributes used in an application" (Wiegers, 190).  While there seems to be no industry standard for creating a data dictionary, Wiegers describes a data dictionary syntax that is able to account for primitive data elements, composition, iterations, and selections (190-191).  See Table 2 – Data Dictionary Excerpt for a few examples. Any definition that includes "= * text *" identifies a primitive data element. As for composition entry, see the "Budget Worksheet" entry in Table 2 – Data Dictionary Excerpt. Here, the budget worksheet consists of multiple elements: current budget, proposed budget, budget recommendation, budget prepared date, and budget other information.  Further, there can be 1, 2,

or 3 budget recommendations. The Min: Max {data element} notation represents iteration or multiple instances of a data element. The budget worksheet entry also contains an optional element – budget other information. Any element delimited by parentheses indicates that the data element is optional. The last class of entry is a selection entry. Here, there is a fixed list of possible data values. A selection entry is formatted as follows: [possible value 1 | possible value 2 | possible value 3]. In Table 2 – Data Dictionary Excerpt, "Ethnicity" can take on any one of the listed values.

**Table 2 – Data Dictionary Excerpt**

| Dictionary Entry | Definition | Where Referenced |
|---|---|---|
| Budget Recommendation | = * Consist of free form text up to 500 characters. * | • Budget Worksheet |
| Budget Worksheet | = Current Budget<br>+ Proposed Budget<br>+ 1-3 {Budget Recommendation}<br>+ Budget Prepared Date<br>+ (Budget Other Information) | |
| Ethnicity | = [African American \| Asian \| Caucasian \| Hispanic \| Mixed \| Samoan \| West Indies \| Native American] | • Application<br>• Child Ethnicity<br>• Partner Ethnicity<br>• Phone intake form<br>• Women Partner Profile |
| Zip | = * The postal code, which is a 5 or 9 numeric digits number. May have a "-" character between the 5th and 6th digit. * | • Address |

The data dictionary helped the student become familiar with the data used by Women Partnering. Additionally, the data dictionary forced the student to make sure that he fully understood what each data element was and where it was used. The student extended the Data Dictionary to also include the existing forms and spreadsheets used by Women Partnering. This was helpful when the

business process flow was explained to the student. The student was able to more clearly understand the process flow when referring to the data dictionary to see what data was being worked on. The data dictionary was also helpful to point out inconsistencies. Various data elements were recorded as being a check number while on other forms the same data element was recorded as dollar amounts. By sitting down with the data dictionary and the various stakeholders at Women Partnering, the inconsistencies were resolved. Further, the data dictionary helped point out synonyms used by Women Partnering. For example, employment was recorded on the budget worksheet as being a "salary from the employer" while employment was recorded on the application form as being the "name of the employer" that the women partner worked for. For another example, on some forms "disabled" was used while on others "handicapped" was used. Additionally, the Data Dictionary helped the student seek out and understand the acronyms used by Women Partnering. For example, "SSD" was used on several forms, but referred to as "Social Security Disability" during interviews. As Wiegers points out, "the data dictionary should define items from the problem domain to facilitate communications between the customers and the development team" (61). The usefulness of the Data Dictionary to this project was remarkable. One added benefit that a data dictionary provides is documenting the data definitions, which "sometimes lead to functional requirements that the user community did not request directly" (Wiegers, 124).

Using a data dictionary was very useful on this project. The student was able to discover associations, synonyms, and homonyms within the data elements

that would have gone undiscovered if the student followed a pure TDD approach. If the Zachman Framework was followed, the student believes that the same discoveries would have been made.

The data dictionary was very useful for the student to get immersed into all the various data elements that Women Partnering tracks. In total, there were 35 forms, spreadsheets, reports, and pamphlets that were inspected to locate the data elements. While the process of going through the 35 separate artifacts of information was time consuming, it was at least thorough. The data dictionary was created through this inspection process.  Quickly, the student became aware of a few data elements that were called one thing on one form and then called something else on another form. For example, the terms "salary" and "income" were confused. During an interview with one of the Women Partnering staff members, income is defined as salary, food stamps, child support, etc. While in another case, income is defined as funds received from a place of employment. On the Partner Profile form, salary was mention when the correct term should have been income.

Another inconsistency that the data dictionary helped to uncover is the use of age versus date of birth. Some forms asked for age while other forms asked for date of birth. The problem with using age is that it is temporal – it is accurate only for the current year. Often times, Women Partnering is asked to report statistics when perusing grant money. Part of the statistics includes age breakdown of the women helped.  This means that the age recorded by Women Partnering produces erroneous statistics. Women Partnering has since converted over to

tracking the partner's date of birth instead of age. Through a simple calculation, the age statistics will now be accurate.

There are two things that went wrong with the data dictionary. First, the student sorted all the data dictionary entries. This made it very cumbersome for the walkthrough with the users. They were familiar with the existing forms and the contents of the forms.  Discussing the data elements out of context made it hard to for the users to describe the data elements. The student changed the data dictionary to include the various forms, spreadsheets, and other artifacts with drill down capability. This made it easier for the users to describe each of the data elements by having the context included in the data dictionary. The second challenge with using the data dictionary is that the student started off trying to abstract granular data elements into larger structures. These larger structures were named and where not familiar to the users. However, the larger structures were a step towards data normalization.

## F.  Application Construction Challenges

Since the student was most familiar with C++, he started construction of the system in the C++ language. However, the student's C++ experience was on a UNIX server.  Women Partnering's new system was Windows-based and not UNIX based. The significance of this is that a Windows-based programming was unfamiliar to the student. With his C++ skills, the student sat down to learn how to do C++ programming in Windows.

This proved to be very difficult for this student who had little Window's programming experience. First of all, the student had to learn a different mindset for programming in an event-based model.  With Window's programming, the

programmer writes code within the various control's events. For example, the programmer needs to write code to respond to button clicks, form loads, mouse moves, etc.  The exact timing of knowing the window events and when the events fired is crucial to being an effective Windows programmer.  The student's Windows experience was dated. Previously, the student did a few projects working with Visual Basic about 5 years prior to attempting this project.

Second of all, debugging a Windows-based program proved to be challenging especially coming from a non-event based model.  In a non-event based model, as in an UNIX environment, the program overall structure is easier to understand. Primarily this is because you can see the lines of code being executed from beginning to end. In Windows, your program becomes an extension to the Windows operating system. Moving your mouse or clicking on an item is first passed to the Window's operating system where it is converted into an event. A Windows program identifies the events that it wishes to subscribe to.  With each event, custom code is written to respond to the event. Once finished with the custom code, the Window's operating system takes back control until it passes another event to your program. In short, if you watched your program run from beginning to end, you would only see bits and pieces of your program run.

The third challenge encountered was that the C++ for Windows has a robust, low-level application level interface (API) that proved to be difficult to learn. When the student wanted Windows to perform a task, he had to figure out which function to call and to properly format the augments to the function call. This

sounds pretty basic, but often times the function calls required pointers to functions. The function pointers are difficult to work with.

With these challenges in mind, the student wanted to be able to finish this project without having to go through a significant learning curve. So, the student explored using C++ for the .NET environment instead of using C++ for Windows. The student found the C++ /.NET combination easier to use. Yet, other challenges were encountered. The biggest challenge was that coding examples for the .NET almost always were for the C# language. When it wasn't for C#, the coding examples were in Visual Basis. The student noticed that the C# examples were close enough to C++ that he was able to read and understand enough to proceed with coding. The student found that Microsoft extended the C++ language to work specifically the .NET environment. This confused the student. Further, the student became flustered with understanding the C# examples and trying to find the C++ equivalent syntax.  In the end, the student switched over to using C# on the .NET platform for this project.

The student noticed that intellisense did not work for the C++ language. See Chapter VII.B – Intellisense Not Working in C++. Without the intellisense, the student had to rely on the help and the index to complete the programming statements.  See Chapter VII.A – Intellisense Works in C# for an example of showing Intellisense helping the student with the parameters of the oleDbDataAdapter's Fill method when programming in C#.

## G.  Application Construction

Because of all the construction issues overcome by the student, the student was very glad to use TDD as his methodology. The student was able to quickly

adapt the design of the system as the student learned more about Windows programming without having to redo any documentation.  In the past, the student's design experiences had shown him that the more he knew about the target tool set, the better he could tailor design. Now, reflecting back on his almost complete lack on knowledge for the Windows programming, the student feels that his Zachman Framework's designs (assuming that he followed the Zachman Framework instead of TDD) would have been inadequate and would have been scrapped several times. This would have resulted in more time lost redoing documentation.

# Chapter IV.   Lessons Learned

This chapter outlines lessons learned by the student throughout the project. Learning from mistakes and issues encountered in the past is a great way of avoiding them in the future. The student also hopes to share the lessons learned, so that others can avoid the same mistakes and advance the information technology body of knowledge.

## A.   The Infamous Note Field

During development of the project, the student ran into an issue that took an hour to figure out. The symptom was that all database SQL statements issued against the partner_note table always returned with a syntax error. The first thought that the student had was that there was a spelling error or some other syntax error like a stray punctuation mark embedded in the SQL statements. Table 3 – partner_note Database Table shows the columns and data types of the various fields that made up the partner_note table. The student ascertained that the column "note" was causing the syntax error. The student was using OLEDB connection to interact with the database. While "note" was allowed as a valid field name, it caused syntax errors when the SQL statement was passed through the OLEDB connection. The student proved this by renaming the "note" column to "message." After this change, the symptom disappeared.

**Table 3 – partner_note Database Table**

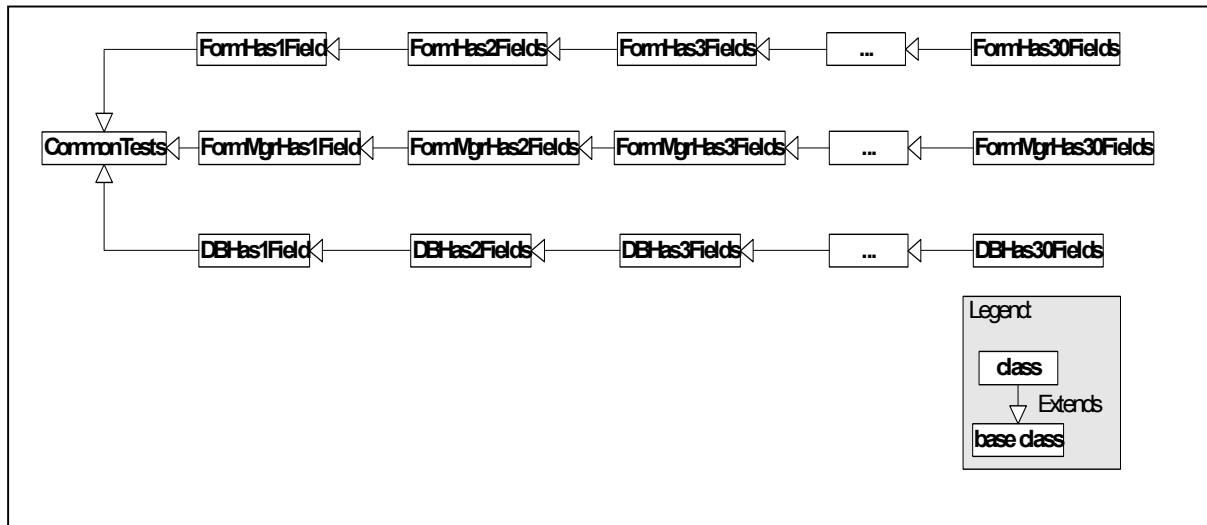| Column Name | Data Type |
|---|---|
| Prtnr_id | Number |
| Staff_signon | Text |
| Note_date | Date/time |
| Note | Text |
| Note_id | Autonumber |
| Updated_by | Text |
| Updated_on | Date/time |

## B. The Need for Good Test Design

How you design your unit tests can make test-driven development a pain or a

pleasure. Look at the form that is shown in Chapter VII.J – Staff Form. This form

is relatively simple – there are only a couple of data entry fields. The student

proceeded to create a suite of tests to exercise the user interface, the business

logic layer supporting this form, and the dataset implementing the data access

layer. All told, the student had 142 unit tests for the three layers (database layer,

business-rule layer, and the user-interface layer). The student was new to test-

driven development at the time the 142 unit tests were created. The student was

very content with the unit tests. He was content until he realized that he had

another 20 forms that needed almost identical unit tests. The issue was that the

142 tests were not reusable. So, do not forget to refactor your unit test code as

well to avoid the brute force approach of unit testing.

Eventually, the student created a set of classes and interfaces whereby any

user interface, business logic layer, and data access layer can be quickly

incorporated into the unit test bed. See Figure 3 – Test Infrastructure. The

CommonTests class defines a common set of unit tests that all layers have in common. It also defines a set of routines that facilitate testing.

**Figure 3 – Test Infrastructure**



The next layer of child classes is important as well to good unit test design. This next layer includes common utilities and tests for testing a user interface, the business rule layer, and the database layer. Then, from there are child classes that implement the tests for one field, two fields, three fields, etc. The student agrees with Balena and Dimauro's suggestion that you avoid deep class inheritance structures (58). The NUnit infrastructure prevents a cleaner solution to the problem of being able to create tests for any number of fields. The issue with the NUnit infrastructure is that it uses attributes to determine which class methods to invoke to run the tests. NUnit discovers the attributes by using reflection into the .NET assemblies. Attributes are defined as part of the method signature in the C# code as shown in Figure 4 – Method that has a "Test" attribute on line 5. This design does not allow for more robust unit test designs using interfaces.

Figure 4 – Method that has a "Test" attribute

```
1. #region SetField9Test
2. /// <summary>
3. /// This method tests the setter for field 9.
4. /// </summary>
5. [Test]
6. [Category("Accessor/Mutators Tests")]
7. public void SetField9Test()
8. {
9. SetTest(this.MetadataAttr.FieldGoodData, new
   GetFieldData(this.GetField9Data), new
   SetFieldData(this.SetField9Data), this.MetadataAttr.FieldName,
   this.MetadataAttr.FieldReadOnly);
10.        }
11. #endregion SetField9Test
```

## C.  The Ins and Outs of Data Binding

There are two techniques that one can follow to move data between the

application and the database. First, you can write the code to move the data, but

this is repetitive. The second technique offered is to use data binding. Data

binding "maps a property of an object to a property in the control" (Wagner, 218).

Wagner suggests using data binding over hand writing the code (217-225). The

student agrees with Wagner – let .NET worry about moving the data. Letting the

.NET libraries move the data for you is much easier and saves time by not having

to write the code yourself. However, there are a few pitfalls lurking in data

binding.

First of all, when one encounters a problem with data binding, the error is

hard to debug. Data binding occurs automatically and the details are hidden from

view because the .NET libraries control the moving of the data. This makes it

impossible to debug. One cannot step through the .NET library code. One

common symptom is where a control fails to receive any data from the database.

**Figure 5 – oleDbDataAdapter_RowUpdating**

```
private void oleDbDataAdapter_RowUpdating(object sender,
                                      OleDbRowUpdatingEventArgs e)
{
      if ( e.StatementType == StatementType.Insert ||
            e.StatementType == StatementType.Update )
      {
          if ( e.Row[IntakeDetail.EthnicityColNm].ToString() == "-1")
          {
             ethnicity_is_null = true;
             e.Row[IntakeDetail.EthnicityColNm] = System.DBNull.Value;
          }
          else
          {
              ethnicity_is_null = false;
          }

          if ( e.Row[IntakeDetail.LivingColNm].ToString() == "-1")
          {
              arrangement_is_null = true;
              e.Row[IntakeDetail.LivingColNm] = System.DBNull.Value;
          }
          else
          {
              arrangement_is_null = false;
            }
      }
}
```

When this occurs, the student found that a null value may have caused data

binding to fail. Here is the situation. The database column was defined to allow

nulls. Further, the data column in the .NET data set also allowed null values.

Next, the student bound the data column to a control. The control stopped

working at this point. The solution was to create two event handlers for handling

the row updating (see Figure 5 – oleDbDataAdapter_RowUpdating) and row

updated (see Figure 6 – oleDbDataAdapter_RowUpdated).  Plus, the list box

control had to be updated to plug in a "-1" value when a null value was expected.

These changes allowed the student to set a null value (really a "-1" in the control)

and allow a null value to be inserted into the database. Note: the student had

spent more time figuring out what was required for these two methods than he would like to have spent.

**Figure 6 – oleDbDataAdapter_RowUpdated**

```
#region oleDbDataAdapter_RowUpdated
private void oleDbDataAdapter_RowUpdated(object sender,
OleDbRowUpdatedEventArgs e)
{
      if (e.StatementType == StatementType.Insert ||
           e.StatementType == StatementType.Update )
      {
           if ( ethnicity_is_null )
           {
                 e.Row[IntakeDetail.EthnicityColNm] = -1;
           }

           if ( arrangement_is_null )
           {
                 e.Row[IntakeDetail.LivingColNm] = -1;
           }

           e.Row.AcceptChanges();
      }
}
#endregion oleDbDataAdapter_RowUpdated
```

Another area that caused the student to stumble with data binding was the student's custom controls. The student created custom controls for check boxes, combo boxes, text boxes, group/radio button control, and a date-time picker. Each of these controls facilitated data binding. The technique that the student followed was to create a hidden text box, which is where the data binding property was bound. Then, changes to the text box would be propagated out to the primary control(s). For example, take the group box/radio buttons as shown in Figure 7 – Group Box with Radio Buttons. Behind the scenes there is a text box. When the contents of the text box changes, one of the radio buttons needs to be checked. Likewise, clicking on one of the radio buttons needs to update the

value stored in the text box since it is the control that is bound to the database. Once again the student fell into a data-binding trap. Initially, the student set the text box's visible property to false, which in essence turned off data binding on the control. As a solution, the control's visible property had to be set to true, yet place the control behind another control on the screen. In essences, the control was not visible. This allowed data binding to be turned on, which allowed the data to flow between the control and the database.

**Figure 7 – Group Box with Radio Buttons**



The student encountered a third data-binding pitfall – data binding did not occur when expected. The student found out the hard way that binding a control to the dataset does not mean that there is going to be data in the control. This is true even if the bound dataset is populated with data. This caused problems with unit testing. All of your unit tests will fail if you only instantiate the form that uses data binding. Why? There is not data in your controls because data binding has not been activated. The student discovered that data binding is turned on only when the form has been loaded. This caused a problem because the student had developed a validation routine that would fire against controls that had no data.

The solution was to disable the validation routines until the form was loaded. However, showing the form, which fires the form-load event, has its own set of issues with unit testing as discussed previously.

## D. Testing in the Weeds

"TDD is performed from the bottom up by sequentially applying a series of simple solutions to small problems that eventually evolves into a design" (Stott, 55). This sounds good, but does it actually work? The student purposely followed the test-driven development mantra of "red-green-refactor" only to find that the student had developing something that was fully tested, but sometimes was not needed. The unit tests exercised a small chunk of code or building block. As more and more of the building blocks are put together, one is suppose to end up with a working system that meets the users' expectations. This student criticizes this approach to system development. The reason being is this: just because you have hundreds of building blocks does not mean that you will end up with a working system. For that matter, you may not even end up with a system! The student found himself making good progress building unit tests and system code only to eventually find out that he wandered off track days beforehand. As you can imagine, this is very frustrating. The student had to remember to step out of the test-driven mindset, look up over the weeds, and consider the big picture. Only by doing this top-down assessment was the student able to stay on course in building a system that was well tested.

## E. Securing the Application

The Rijndael cipher was used to store the encrypted user passwords and women partner's social security numbers in the database. One problem

encountered using the Rijndael cipher pertained to the fact that it is a block

cipher. Errors encountered with the cipher occurred when the plaintext length

passed to the cipher was shorter than the block size. See Table 4 – Padding

Solution for the Rijndael Cipher for how the student solved this issue.

**Table 4 – Padding Solution for the Rijndael Cipher**

| Mode | Code | Explanation |
| --- | --- | --- |
| Encrypt | symmetricKey.Padding = PaddingMode.Zeros; | If the plain text is shorter that the block size, pad the plain text with zeros up to the correct block size. |
| Decrypt | Regex.Replace(plainText, @"\0", ""); | After decrypting the cipher text, make sure to remove the zeros that may have been added when encrypting the plain text. |

## F.  In the Dark with Failed Tests

When the unit tests failed in the NUnit, sometimes the error message was

enough to know what was needed to fix the error. These types of failed tests

were the most desirable ones – ones that can be fixed quickly without hindering

the progress on the project. Further, these types of errors did not require digging

around the code to discover the issue.

The next type of failed tests was more of a nuisance. The student

encountered some failed tests were the fix was not readily apparent. This type of

failed tests required the student to step through code to debug the issue.

Remember: the unit tests are not part of the production code, so one cannot step

through the production application to identify the issue with the test. The student

ended up creating a non-production form, which used a menu bar. The menu

options called the various unit tests. By doing so, the student was able to set the

project containing the unit test forms as the startup project and run the

application, which then loaded the test form.   See Chapter VII.H – Form Used to Organize Unit Tests for a sample screen snapshot of the unit test form. With the form running, the student was then able to walk through the unit test code using the Visual Studio debugger to locate the issue.

The last type of failed tests was the most troublesome. Every once in a while, one or more tests would fail when the student ran the entire suite of unit tests, yet these same tests would pass when ran individually.  The student knew that the unit tests are suppose to be independent from one another, but there is nothing in place to enforce this golden rule of test-driven development. To the student's knowledge, there are no tests that were dependent on one another. However, there were two situations encountered by the student. First of all, there was a dependency within the unit tests and the setup and/or teardown methods. The setup and teardown methods were used to return the test back to an initial state. The student was in the dark when these types of failed unit tests were encountered. Because the combination of interactions was not known, using the previous technique of placing the unit test on the unit test form did not to work. A second situation that appeared regularly occurred where the constructor of the class encountered an exception. When this occurs within NUnit, NUnit attempts to call the constructor again – this causes the exception to be thrown again. In the end, the entire set of tests would fail with the only feedback is that the test had failed.

The biggest discovery made to help combat this last type of failed tests was the ability to attach to a running process. See Chapter VII.I — Attaching to

another Process. The beauty of this is that you can attach to the NUnit process and watch it invoke your tests.  By doing this, you can see the code as it executes and see the order of the various tests as they are called. Before the student discovered this ability to attach to a process, the student would have to guess at the sequence of events that caused the tests to fail. Debugging the NUnit process can be challenging. The best thing to do is to set break points in your code.  Then wait for the debugger to stop in your code to debug your unit tests. The student installed a copy of the NUnit source code and tried to debug the NUnit process. This proved to be very challenging because the NUnit application runs in one application domain while your unit tests runs in another. One benefit of multiple application domains is that the application running in one application domain is completely protected from the other application in the second application domain that may fail (Troelsen, 463). Because of complete isolation, the two applications have to use the .NET remoting protocol in order to communicate back and forth (Troelsen, 463).  Debugging the .NET remoting interaction between NUnit and the unit tests is very difficult. Before abandoning stepping through the NUnit source code in the debugger, the student found that a complete copy of unit tests is created in the temporary directory called the shadow copy. The student found the location for the shadow copy and discovered about 500 copies of the application or about 9 month's worth of unit testing that was sitting on the disk drive. Running the Disk Cleanup process from Microsoft cleaned up these shadowed copies.

With the discovery of being able to attach to a process in order to debug it, the student came into the light and was able to quickly identify all issues with the failed unit tests.

## G. Work That Project!!!

The student's initial project time line included the software development for Women Partnering, the thesis/research, and the two classes required for graduation. A summary view of the initial time line is shown in Figure 8 – Initial Project Schedule. Everything was to be complete by year's end of 2004.

**Figure 8 – Initial Project Schedule**

| Task Name | Duration | % Complete | Start | Finish |
|---|---|---|---|---|
| ⊟ Women Partnering Project/Thesis | 338 days? | 4% | Tue 1/13/04 | Thu 4/28/05 |
| ⊞ Administration | 6 days? | 8% | Tue 1/13/04 | Tue 1/20/04 |
| Select Advisor | 10 days | 0% | Mon 5/3/04 | Fri 5/14/04 |
| ⊞ Proposal Class | 41 days? | 38% | Mon 3/1/04 | Mon 4/26/04 |
| ⊟ Technical Solution | 234 days? | 0% | Tue 1/13/04 | Fri 12/3/04 |
| ⊞ Analysis | 99 days? | 0% | Tue 1/13/04 | Fri 5/28/04 |
| Research | 30 days | 0% | Mon 5/31/04 | Fri 7/9/04 |
| Design | 30 days | 0% | Mon 7/12/04 | Fri 8/20/04 |
| Construction | 60 days | 0% | Mon 8/23/04 | Fri 11/12/04 |
| Testing | 10 days | 0% | Mon 11/15/04 | Fri 11/26/04 |
| Implementation | 5 days | 0% | Mon 11/29/04 | Fri 12/3/04 |
| ⊞ Thesis Class | 32 days? | 0% | Mon 1/17/05 | Tue 3/1/05 |
| ⊞ Presentation Class | 42 days? | 0% | Wed 3/2/05 | Thu 4/28/05 |

The actual project timeline as shown in Figure 9 – Actual Project Schedule tells a completely different story. The project was definitely not a smooth one where everything executes according to plan. On the contrary, this project encountered numerous problems.

Figure 9 – Actual Project Schedule

| Task Name | Duration | Start | Finish |
|---|---|---|---|
| ⊟ Women Parnering Project/Thesis | 694 days | Tue 1/13/04 | Fri 9/8/06 |
|     Administration | 6 days | Tue 1/13/04 | Tue 1/20/04 |
|     Select Advisor | 1 day | Tue 4/13/04 | Tue 4/13/04 |
|   ⊞ Proposal Class | 41 days | Mon 3/1/04 | Mon 4/26/04 |
|   ⊞ Technical Solution | 694 days | Tue 1/13/04 | Fri 9/8/06 |
|   ⊞ Thesis (paper) | 374 days | Mon 3/7/05 | Thu 8/10/06 |
|   ⊞ Thesis Class | 45 days | Mon 7/3/06 | Fri 9/1/06 |
|   ⊞ Presentation Class | 45 days | Mon 7/3/06 | Fri 9/1/06 |

The student wished that he could have worked on this project from beginning to end without having any delays. This project officially started in March 2004 and was planned to finish in September 2006. The project was not too large or too difficult to cause the project to be a 2 and ½ year project. The student's commitment to the project was the one thing that really caused the project to take so long. Not working on the project for a week or two, left the student trying to figure out where he left off. So, the student now knows to take better notes. This is necessary so that if there is a project delay the student can return to the project running and not waste days figuring out where he left off.

Beyond the sometimes spotty effort to keep the project moving, the student's next biggest issue causing delay was the student's lack of experience with programming in a Window's environment. The change to event-driven programming proved to be rather challenging for the student who has over ten years programming in an UNIX server environment. The challenge was in learning about the events and when the events where raised by the Windows environment and/or by the program itself. A further challenge was becoming

immersed into the Active Data Objects .NET. This set of libraries is feature rich and different enough to the student that there was a steep learning curve.

The student initially started with trying to build the system using C++ and the WIN32 API. As this proved to be a significant learning curve, the switch was made to C++ and the .NET environment. Here the learning curve was not so steep. However, because of the lack of documentation and examples of using C++ in the .NET environment, the student again switched to a different set of development tools. This time it was C# and the .NET environment. The student found the transition over to C# from his C++ background was actually pretty easy. More important though was the wealth of documentation and programming examples available to the student. These flatten the learning curve even further. Yet, each time the student switched languages, the project was delayed further.

## Chapter V.    Conclusion

### A.  What Should Have Been Done Differently

The student should have done a couple of things differently on this project.

First, communication between the student and Women Partnering was not

always the greatest. The student would go off and work on the project for weeks

and months at a time without communicating what was going on with the project.

To make matters even worst, there were periods of time that the project was not

actively worked on. When this happened, there were no communications with

Women Partnering at all. The student needs to communicate what is going on

with the project at all times, and communicate what is going on with external

influences that caused delays in the project. The communications with the users

is critical to TDD's success. As an extreme programming methodology, TDD

relies on quick, short iterations, which pulls the users into the process and

flushes out the system that they want. The lack of communications hampered the

student following a pure TDD approach.

A second item that should have been done differently was that the student

should not have initially committed to delivering the system on an aggressive

schedule. This is true especially with the student's lack of experience in a

Window's development. The student is very thankful for Women Partnering's

patience. Women Partnering allowed the student to work through the learning

curve and deliver a system to them long after when first committed. The student

wished he had taken Window's programming classes as part of his course work

to help ease the student into Window's programming. At the very least, the

classes could have helped set a realistic schedule for a Window's based programming project.

In hindsight, coming up with a master plan before wandering off building unit tests would have been wiser. While the unit test is suppose to capture a requirement in TDD, there are still many other design factors that are not covered directly by requirements. For example, one design factor that should have been considered throughout the project was class design. All too often with the student's experience with TDD was that he would just go off and build unit tests and functional code to support the tests – not giving any focus to overall class design. Class reuse, class inheritance, and consideration for class interfaces fell by the wayside because focus was placed on getting the unit tests created. Perhaps, the student should have considered class design more frequently during the refactoring step.

## B. Did the project meet initial expectations?

Expectations in the beginning of a project often times do not match what is built. This is a pretty natural occurrence. As a project starts up, you are working with ideas and visions. Some thoughts may even contradict each other, because input is taken from all stakeholders. As the software development lifecycle progresses, the ideas and visions are transformed into a working system. There are two perspectives if the project meets the initial expectations: 1) from the perspective of Women Partnering; 2) from the perspective of the student.

Women Partnering's initial expectations have been met. They wanted to get away from a primarily paper-based system. The built system eliminates many different forms as well as several different spreadsheets. The information needed

by Women Partnering is now readily available to them – making them more efficient in helping women in the community. Further, Women Partnering's expectation for a system that is well tested has been fully met.

As for the student's expectations, the initial expectations contrast significantly to what was delivered. As pointed out before, the student initially started with trying to build the system using C++/WIN32 API, but switch over to C++/.NET and eventually finished the application in C#/.NET. The programming languages prior to C#/.NET proved to have steep learning curves. The student is very thankful for a .NET feature called language independence. This allowed the easy transition from C++ over to C#. The student was able to run C# classes that inherited from C++ classes. Additionally, one .NET assembly written in C++ worked seamlessly with another .NET assembles written in the C# language. Language independence allowed the student to ease over to writing C# without having to completely scrap his previous work in C++.  The student was able to test drive C# little by little until the student was comfortable enough to convert all classes over to the C# language.

**C.  What would be the next stage of evolution for the project if continued?**

The next stage of the project can be to add in the ability to manage the classes offered by Women Partnering and by Sister's of St. Francis of Colorado Springs, Colorado. The classes help the women partners learn skills to better their lives. The student sees opportunity for setting up class schedules combined with teacher schedules and to eventually allow for women to enrollment in the classes by visiting web pages.

Second, the ability to press the F1 key and receive help would be valuable. The student initially planned to deliver help with the project, but this was removed from scope in order to align this project's completion with finishing his Master's degree. So, for a next step in the system, help pages should be integrated with system. The student started to include help pages into the application. The compiled help pages are part of the installation package for the system. There are only 2 entries in the help index, so the bulk of the help pages would have to be flushed out.

Another possible next step would be to network together the various support agencies with which Women Partnering works with. This would allow for information sharing, which would allow the support agencies (including Women Partnering) to be able to help the women faster than they do today. There would be no need for women to fill out applications at each support agency visited. Instead, their information will be available on-line with the description of the help that they require.

## D. Conclusion

Test-Driven Development was used to build a small-to-medium sized system for Women Partnering. The student followed the process of writing the unit tests before any functional code was written. The student admired the focus on the interfaces captured in unit tests. While this process worked great for developing a system, the student deviated from TDD by creating a couple of documents that are usually associated with a waterfall, plan-based methodology. For example, the student deviated from TDD by creating a data dictionary and entity relationship diagrams. These other tools allowed to the student to view the

system from other points of view, which is exactly what the Zachman Framework forces you to do. The student learned aspects from the data dictionary and entity relationship diagrams that may not have been discovered using a pure TDD approach.

It has been refreshing using TDD as the methodology for building Women Partner's system. The student spent more time programming than writing requirement specifications, design documents, and other artifacts associated with the more traditional waterfall methodologies. The student is no different than other programmers in that he prefers programming over writing documentation. However, the student feels that TDD is not the software methodology that should be used on all software development projects. Rather, TDD is just another tool that software developers have available to them. Today's software developers need to be flexible and be able to use the correct tool for the particular project at hand. Using the wrong tool can cause the project to fail. TDD works well where the requirements are not well understood, where the users have a hard time articulating the requirements, where the environment is dynamic with frequently changing requirements, where the team size is smaller, and most importantly where the users are willing to be engaged throughout the entire project.

## E. Recommendation

The student suggests that software developers remain flexible in their tool choice in order to better serve there projects and avoid project failure. Test-Driven Development should not be used on all projects. Similarly, the Zachman Framework should not be used on all projects. Today's software developers must be cross-trained on many methodologies and be able to adapt their approach to

their particular project's needs in order to be successful with today's larger,

complex development projects.

# Chapter VI.   Annotated Bibliography

Ambler, Scott W. "Examining the Agile Cost of Change Curve." Ambysoft. February 12, 2006. Retrieved from http://www.agilemodeling.com/essays/costOfChange.htm on May 3, 2006.

Ambler, Scott W. "Introduction to Test Driven Development (TDD)." AmbySoft. May 4, 2006. Retrieved from http://www.agiledata.org/essays/tdd.html on May 4, 2006.

Balena, Franceso and Giuseppe Dimauro. "Practical Guidelines and Best Practices for Microsoft Visual Basic and Visual C# Developers." Redmond: Microsoft, 2005.

Boehm, Barry and Richard Turner. "Balancing Agility and Discipline: A Guide for the Perplexed." Boston: Pearson 2004.

Business Rule Group. "What is a Business Rule?" 2004. Retrieved from http://www.businessrulesgroup.org/brgdefn.htm on March 16, 2005.

Christensen, Mark J. and Richard H. Thayer. "The Project Manager's Guide to Software Engineering's Best Practices".  Los Alamitos: IEEE Computer Society, 2001

Computer Security Division of the NIST. "Advanced Encryption Standard (AES):  Questions and Answers". January 28, 2002. Retrieved from http://csrc.nist.gov/CryptoToolkit/aes/aesfact.html on April 24, 2005.

Cross, Michael, Norris L. Johnson Jr., Tony Piltzecker, and et. al. "Security+: Study Guide & DVD Training System." Rocklan: Syngress, 2002.

Daemen, Joan, and Vincent Rijmen. "The Rijndael Block Cipher: AES Proposal". March     9, 1999. Retrieved from http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael.pdf on April 24, 2005.

Doshi, Gunjan. "Test-Driven Development Quick Reference Guide." Instrumental Services Inc. 2005-2006. Retrieved from http://www.testdriven.com/files/doshi/TestDrivenDevelopmentReferenceGuide.pdf on April 15, 2006.

Ferguson, Niels, Richard Schroeppel, and Doug Whiting. "A simple algebraic representation of Rijndael." Retrieved from http://www.macfergus.com/pub/rdalgeq.pdf on April 10, 2005.

FOLDOC. "Data Dictionary". April 24, 2001. Retrieved from
http://foldoc.doc.ic.ac.uk/foldoc/foldoc.cgi?query=data+dictionary on
March 14, 2005.

Gibbs, W. W., "Software's Chronic Crisis" from Scientific American. Sept.
1994. pp. 86-95. Reprinted in Software Engineering. Volume 1: The
Development Process. 2nd ed. Editors Thayer, Richard H. and Merlin
Dofrman. Hoboken: John Wiley, 2002.

Hay, David C. "Requirement Analysis: From Business Views to Architecture."
Upper Saddle River: Pearson, 2003.

Hay, David C. "What Data Models Can't Do." Essential Strategies, Inc. 1998.
Retrieved from http://www.essentialstrategies.com/ on March 16, 2005.

Murphy, Craig. "Improving Application Quality Using Test-Driven Development
(TDD). Originally published in the Spring 2005 issue of Methods & Tools.
Retrieved from
http://www.methodsandtools.com/archive/archive.php?id=20 on May 3,
2006.

Perkins, Alan. "Implementing the Zachman Framework for Enterprise
Architecture: Visible Tools and Services Help Implement the Zachman
Framework for Enterprise     Architecture!" Visible Systems Corporation,
1997. Retrieved from
http://www.ies.aust.com/~visible/papers/Zachman.html on April 15, 2006.

RSA Laboratories. "What is a Stream Cipher?" 2004. Retrieved from
http://www.rsasecurity.com/rsalabs/node.asp?id=2174 on April 24, 2005.

RSA Laboratories. "What is an Iterated Block Cipher?" 2004. Retrieved from
http://www.rsasecurity.com/rsalabs/node.asp?id=2169 on April 24, 2005.

Simsion, Graeme, Simsion & Associates / University of Melbourne. "What's
Wrong with     the Zachman Framework?" 2005. Also, published in
TDAN.com January 2005.     Retrieved from
http://www.tdan.com/i031fe02.htm on April 15, 2006.

Smith, Steven A. "Get Test Infected with NUnit: Unit Test Your .Net Data
Access Layer." October 2003. Retrieved from
http://msdn.microsoft.com/library/default.asp?url=/library/en-
us/dnaspp/html/aspnet-testwithnunit.asp on May 3, 2006.

Stott, Will and James Newkirk. "Improve the Design and Flexibility of Your Project with    Extreme Programming Techniques." MSDN Magazine: The Microsoft Journal for    Developers. April 2004. Retrieved from http://msdn.microsoft.com/msdnmag/issues/04/04/ExtremeProgramming/default.aspx on May 1, 2006.

Troelsen, Andrew. "C# and .NET Platform." 2nd ed. New York: Apress, 2003.

Udell, Jon. (Nov 1, 2004) Source code analysis breaks new ground - New tools and  accelerated research bodes well for future software.  In *InfoWorld*, 26, p12.  Retrieved December 13, 2005, from *Computer Database* via Thomson Gale: http://find.galegroup.com/itx/infomark.do?&contentSet=IAC-Documents&type=retrieve&tabID=T002&prodId=CDB&docId=A123871722&source=gale&userGroupName=pike&version=1.0.

Wagner, Bill. "Effective C#: 50 Specific Ways to Improve Your C#." Scott Meyers ed. Boston: Addison, 2005.

Whitten, Jeffrey L, Lonnie D. Bentley, and Kevin C. Dittman. "System Analysis and Design Methods." 5th ed. New York:  McGraw-Hill, 2001.

Wiegers, Karl E. "Software Requirements. Practical Techniques for Gathering and Managing Requirements Throughout the Product Development Cycle". 2nd ed. Redmond: Microsoft Press, 2003.

Wikipedia. "Advanced Encryption Standard." April 7, 2005. Retrieved from http://en.wikipedia.org/wiki/Rijndael on April 10, 2005.

Wikipedia. "Block Cipher." March 26, 2005. Retrieved from http://en.wikipedia.org/wiki/Block_cipher on April 10, 2005.

Wikipedia. "Enterprise Architecture." April 12, 2006. Retrieved from http://en.wikipedia.org/wiki/Enterprise_architecture on April 17, 2006.

Wikipedia. "Test-Driven Development." April 15, 2006. Retrieved from http://en.wikipedia.org/wiki/Test_driven_development on April 15, 2006.

Zachman, John A. "Enterprise Architecture: A Framework."  Retrieved from http://www.zifa.com/ on April 15, 2006.

# Chapter VII.  Appendixes

## A.  Intellisense Works in C#



Here the cursor is on line 69 in a C# source file. Microsoft's Intellisense pops up a tool tip that shows the parameters for the OleDbDataAdapter's Fill method. This is very helpful for someone who doesn't remember or is learning the parameters as they type.

## B. Intellisense Not Working in C++



In this example, the cursor is on line 16 of the C++ file. Microsoft's Intellisense fails to display the parameters for the OleDbDataAdapter's Fill method. See Chapter VII.A – Intellisense Works in C# for an example for where the Intellisense does work.

## C. Example NUnit Screen



NUnit is a unit test tool, which is able to run the unit tests and report the status of the test. Shown here, there is one test that failed. All others have passed.

**D.   The Zachman Framework**



(Zachman)

75

## E.  Testing Status in NUnit



This example screen shows the NUnit screen where there are six failed tests.

## F. Form Incorrectly Painted



Because of issues with events, the form was not properly painted. The text describes how to resolve this issue.

## G.  Fully Painted User Interface



As the NUnit runs your tests, the user forms are displayed with all the fields being populated. Without following the steps as outlined in this paper, the fields are not displayed correctly when the user form is run by the unit test.

## H.  Form Used to Organize Unit Tests



This form was used by the student to organize all unit tests, so that he could run any test without having to run the NUnit tool. This was necessary to be able to debug the unit tests from within the Visual Studio Environment (i.e. without having to attach to the NUnit process).

## I. Attaching to another Process



Sometimes, it was necessary to attach to the NUnit process in order to watch the interactions between unit tests.

**J.   Staff Form**



This form is just one of the forms created by the student.