Fall 2010

Test-Driven Web Application Development: Increasing the Quality of Web Development By Providing Framework with an Emphasis On Test-Driven Design and Development Methodologies

Jason Hall
Regis University

Regis University
College for Professional Studies Graduate Programs
Final Project/Thesis

Disclaimer

TEST-DRIVEN WEB APPLICATION DEVELOPMENT: INCREASING THE

QUALITY OF WEB DEVELOPMENT BY PROVIDING A FRAMEWORK WITH AN

EMPHASIS ON TEST-DRIVEN DESIGN AND DEVELOPMENT METHODOLOGIES

A THESIS

SUBMITTED ON 10 OF DECEMBER, 2010

TO THE DEPARTMENT OF INFORMATION TECHNOLOGY

OF THE SCHOOL OF COMPUTER & INFORMATION SCIENCES

OF REGIS UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF MASTER OF SCIENCE IN

SOFTWARE ENGINEEING AND DATABASE TECHNOLOGIES

BY

Jason Hall

APPROVALS
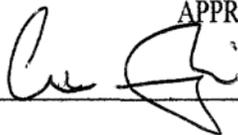
Charles N. Thies, Thesis Advisor

Denise Duncan, Asst. Professor

Nancy Birkenheuer, Asst. Professor

Abstract

Web applications – especially those based on interpreted programming languages – are quickly becoming more utilized and more commonplace than traditional client applications. Despite this growth, no open solution has yet fulfilled the need of a risk-reducing development framework that supports test-driven methodologies and tools designed to coordinate the resources responsible for the most effective development of web applications based on interpreted programming languages. This research paper presents a test-driven development framework consisting of openly available components that can be used by development teams writing web applications based on interpreted programming languages based on the methodologies and tools used by traditional software development teams using compiled programming languages.

Acknowledgements

I am indebted to my many fellow members of the Ignite 360 development team and support staff for their time, hard work, and support, all of which was instrumental for me to complete this significant milestone in my life.

I would like to particularly thank Charles Thies for providing me with the direction and motivation I needed throughout this capstone project.

Lastly, and without a doubt most importantly, words cannot express the thanks I have for my family – especially my wife Terri – for being so understanding and for unconditionally giving me their love and support.

Table of Contents

List of Figures

Chapter 1 - Introduction

Statement of Problem:

The constantly growing demand for better software developed faster (and cheaper) has spurred the development of many frameworks, tools, and programming methodologies that greatly enhance the development efforts of any team developing desktop or client/server software based on compiled programming languages while reducing the risk associated with faster and cheaper development. Web applications are quickly becoming more utilized and more commonplace than traditional client applications – including email applications, streaming media services, social media, collaboration tools, customer relationship management tools, accounting software, and even office productivity software. Despite this growth, no open solution has yet fulfilled the same need of a risk-reducing development framework that supports test-driven methodologies and tools designed to coordinate the resources responsible for the most effective development of web applications (especially enterprise-level web applications capable of hosting many tenants) based on interpreted programming languages. The intent of this research is to design a web application development framework for interpreted programming languages based on test-driven development methodologies that have proven successful in compiled software development.

Statement of Goals & Objectives

The primary objective of this research thesis is to analyze the varying development methodologies and tools that are in place in traditional software development environments based on compiled programming languages in an effort to gain an understanding of best practices

that can then be taken and molded into a framework for developing multi-tenant web applications based on interpreted programming languages using test-driven and other risk-reducing technologies. As a member of the web application development community and responsible for the development, deployment, and maintenance of multi-tenant web applications based on interpreted programming languages, the researcher will be responsible for conducting (and actively applying) a qualitative research study using a case study research approach for the purpose of developing a test-driven web application development framework.

Chapter 2 – Review of Literature and Research

Background & Significance

The development of programs as web applications that were once considered the exclusive realm of desktop applications is becoming only more common. The use of the Internet browser as a thin client to provide access to the wealth of applications and data stored in the Internet as opposed to a local computer or networked server has enabled people all over the world to retrieve, manipulate, save, and share data from any place with an Internet connection (and to some degree, HTML5, various browser extensions, and some software applications allow for limited offline manipulation of that data). These thin client, web-based, applications are allowing for the development of a new generation of low cost devices with limited resources (mobile phones, tablets, and netbooks to name a few) that give users access to whatever they need at just about any time. The software-as-a-service business model is designed around the advantage of lower implementation costs due to the fact that a web application only has to be developed for a single software client (the browser) while being able to provide services to clients that are completely independent of what operating system or hardware the client is using and (as long as Internet access is available) independent of where they may be located.

Software engineering blogger Jeff Atwood summed it up best when he coined "Atwood's Law" by stating that "any application that can be written in JavaScript, will eventually be written in JavaScript" (Atwood, 2009). It makes logical sense that this trend is occurring, since the World Wide Web basically has become the de facto distribution method for software (whether or not it's a web application). Mobile platforms such as Apple's iOS, Google's Android, and Microsoft's Windows Phone accelerate this trend by providing local storefronts for hosting and

maintaining third-party web applications that run on those devices (both phone and tablet). Mozilla and Google have both announced the intention of providing the same for their respective browsers. Atwood continues to describe this transition by stating,

"If you want your software to be experienced by as many users as possible, there is absolutely no better route than a web app. The web is the most efficient, most pervasive, most immediate distribution network for software ever created. Any user with an internet connection and a browser, anywhere in the world, is two clicks away from interacting with the software you wrote. The audience and reach of even the crappiest web application is astonishing, and getting larger every day." (Atwood, 2009).

He follows it up with "Writing Photoshop, Word, or Excel in JavaScript makes zero engineering sense, but it's inevitable. It will happen. In fact, it's already happening. Just look around you." (Atwood, 2009). Proof of that statement is found in the release of traditionally desktop software such as office productivity products in web application form - just look at Google Docs, Zoho, Microsoft Office Live, and other web-based alternatives to the traditional installed office productivity suite.

That being said, due to the quick change of technology on an almost daily basis, the development world of web applications is still a bit like the Wild West. The diversity of web application technologies and general lack of enforced standards has led to an environment where a multitude of different methods, tools, and challenges must be overcome for web application providers to implement their applications. Additionally, the constant change in web technologies and the devices that can access these web applications demands a fast turnaround of their development and an even faster turnaround for adding additional features and updates. This

demand for quick software release cycles with cheaper costs leads to development processes that

release code that is often only 90% complete, with future updates expected to provide the

missing functionality and correcting the various bugs that were released in the initial version

(often while actively developing the next web application product). The end results are web

applications that are in a state of perpetual beta. While this attitude of web development

probably began in the open source development world, it has spread to major corporations - just

look at many of Google's products. Tim O'Reilly, in 2005, stated the following about this trend:

> "The open source dictum, 'release early and release often' in fact has morphed into an
>
> even more radical position, 'the perpetual beta,' in which the product is developed in the
>
> open, with new features slipstreamed in on a monthly, weekly, or even daily basis. It's no
>
> accident that services such as Gmail, Google Maps, Flickr, del.icio.us, and the like may
>
> be expected to bear a 'Beta' logo for years at a time." (O'Reilly, 2005).

There are both advantages and disadvantages to this idea of treating the users of a web

application as its co-developers. To its credit, this mindset does provide users with a constantly

improving product where new features and updates are added directly into the service - taking

full advantage of using the web as a deployment medium. Additionally the user is not required

to actively update or download the application to have access to these new features and updates.

Interest in the product may remain strong if updates with new features are consistently being

added to the system. The problem that arises is that it often becomes an excuse for releasing an

incomplete product (there is a fine line between offering an incomplete product with the missing

functionality being offered in later versions and a finished product with new features being

released in later versions). So the solution is to implement a development process for web

applications that provides the kind of rapid development and deployment required to meet the

demands of the market and users while not remaining in a form of "never quite there" beta.

Part of what creates the environment described above is the many different interpretations

of the idea of rapid development. Steve McConnell explains it as

"To some people, rapid development consists of the application of a single pet tool or

method. To the hacker, rapid development is coding for 36 hours at a stretch. To the

information engineer, it's RAD - a combination of CASE tools, intensive user

involvement, and tight timeboxes. To the vertical-market programmer, it's rapid

prototyping using the latest version of Microsoft Visual Basic or Delphi. To the manager

desperate to shorten a schedule, it's whatever practice was highlighted in the most recent

issue of Business Week." (McConnell, 1996).

Most software development teams have examples of each of the mindsets, tools, and methods

listed above to one degree or another and each one has the ability to actively contribute to faster

development speeds. That being said, no single one of these is a one-size-fits-all tool that can be

universally applied to every project a development team faces (no matter what platform they use

for application development). For that matter, generally speaking, no single one of these can be

universally applied to even the entirety of a single project and still achieve the optimum use of

the available resources and time. To truly design a development process that achieves a state of

rapid development, each of these must be integrated together as part of an overall development

strategy, in light of the resources, projects, and business rules that drive all that the organization

is responsible for.

The use of orthogonal development design and programming practices, design patterns, continuous integration, and unit testing are all components of iterative software development which are designed to reduce the amount of missing features or bugs by constantly testing the programming code as it is developed, and by forcing the software engineers responsible for the development of the application to have a more solid understanding of the functionality needed before they even begin programming. Instead of waiting till all individual components have been completed and then integrating them into a software build for testing (which often leads to time intensive delays caused by debugging), these methodologies advocate automated testing on each commit of programming code (Duvall, 2007). For the most part, these software development methodologies have primarily emphasized client applications based on code written using compiled programming languages. While a few open-source solutions have developed test-driven development tools, relevant design patterns, continuous integration servers, and unit testing programs for web application development written using interpreted programming languages, there has not been an emphasis on integrating these each of these methodologies and tools into a single web application development framework. So while software application development has made great strides with compiled software in providing and adopting these capabilities, development lags behind in providing a suitable framework of methodologies and tools for ensuring quick, stable, and solid development of web applications.

Orthogonality

The field of mathematics uses the term orthogonality in geometry to represent two lines intersecting perpendicularly (at ninety degrees) - for example, the axes of a graph are orthogonal. In vector physics, orthogonality is used to determine the independence of two lines -

for example, two lines are orthogonal if the change of position on one line has no impact on the position on the other line. Much like its definition in math and physics, orthogonality in software engineering is used to denote an independence of one object from another. In The Pragmatic Programmer, Andrew Hunt and David Thomas state that in a system designed with such independence in mind "the database code will be orthogonal to the user interface: you can change the interface without affecting the database, and swap databases without changing the interface." (Hunt, 2000). The idea of orthogonality in software engineering is designed to reduce and remove interdependent components from within a system.

> "Orthogonality means that features can be used in any combination, that the combinations all make sense, and that the meaning of a given feature is consistent, regardless of the other features with which it is combined. The name is meant to draw an explicit analogy to orthogonal vectors in linear algebra: none of the vectors in an orthogonal set depends on (or can be expressed in terms of) the others, and all are needed in order to describe the vector space as a whole." (Scott, 2009)

When modifying one area of a system has any kind of effect on another area of the system (beneficial or not) the system is not optimized orthogonally. These systems have a number of problems, the primary being that they are very difficult to debug problems, maintain, and add functionality without causing unintended consequences that can have long-ranging effects. Hunt and Thomas describe nonorthogonal systems as "inherently more complex to change and control". If components of a particular system are highly dependent upon each other, then there is no local fix that a developer can quickly implement (Hunt, 2000). Eric Lippert goes

into more detail and provides an example of why systems not built orthogonally will cause effects throughout the system even with a simple, localized, change is made:

> "Nonorthogonal systems are hard to manipulate because it's hard to tweak isolated parts. Consider my fish tank for example. The pH, hardness, oxidation potential, dissolved oxygen content, salinity and conductivity of the water are very nonorthogonal; changing one tends to have an effect on the others, making it sometimes tricky to get the right balance. Even things like changing the light levels can change the bacteria and algae growth cycles causing chemical changes in the water." (Lippert, 2005).

Additionally, the longer one goes between releasing a non-orthogonally programmed system and trying to fix a bug or add a new feature, the more complicated it is to complete the change. The side effects can easily range in level of consequence, and can often not only have an adverse effect on the system itself, but also cause delays and increased costs for the organization developing the application and its end-users. Eric Raymond explain how some of the problems that result from non-orthogonality "when side effects complicate a programmer's or user's mental model, and beg to be forgotten, with results ranging from inconvenient to dire. Even when you do not forget the side effects, you're often forced to do extra work to suppress them or work around them." (Raymond, 2003). On the flip side, designing and developing systems with orthogonality in mind can reduce and even eliminate time-sinks and project implementation problems. It provides everyone involved in the development process with the ability to analyze and modify the system at a much more granular level, ensuring that the system consists of a greater degree of quality while helping the project finish on time and hopefully under-budget. A truly orthogonal system provides code that is fully reusable and each

component or module is independent of every other component or module. In contrast to his explanation of the problems with non-orthogonal systems, Raymond states that "orthogonality reduces test and development time, because it's easier to verify code that neither causes side effects nor depends on side effects from other code — there are fewer combinations to test. If it breaks, orthogonal code is more easily replaced without disturbance to the rest of the system." (Raymond, 2003).

Andrew Hunt and David Thomas suggest that components should be designed so that they are

"self-contained: independent, and with a single, well-defined purpose (what Yourdon and Constantine call cohesion). When components are isolated from one another, you know that you can change one without having to worry about the rest. As long as you don't change that component's external interfaces, you can be comfortable that you won't cause problems that ripple through the entire system. You get two major benefits if you write orthogonal systems: increased productivity and reduced risk." (Hunt, 2000).

No matter what the business rules and logic of a system are and no matter how much or how little resources there are to complete it, any tool, concept, or framework that can increase the productivity of a development team and/or reduce the risks associated with the project has immense advantages. One of the primary reasons productivity is increased through writing orthogonal code when developing a system is the reduced amount of time that developing and testing individual components takes (both when initially programming the component and when changes are made to it later). Orthogonal systems consist of small, granular, modules that are independent of their peers. A smaller scope of work for a particular function that a developer needs to write will lead to a quicker turnaround on the completion of that function. The same

goes for quality control: if the scope of work for an individual component is small, then testing

will take much less time than when compared to a module of code that is responsible for the

functionality of multiple components (especially when those in turn affect other components

found elsewhere in the system). Hunt and Thomas describe these advantages by stating that

"Changes are localized, so development time and testing time are reduced. It is easier to write

relatively small, self-contained components than a single large block of code. Simple

components can be designed, coded, unit tested, and then forgotten - there is no need to keep

changing existing code as you add new code." (Hunt, 2000). By approaching the development of

a system orthogonally, reuse is encouraged: if each component in the system has clearly defined

responsibilities then those components are able to be combined (with each other or with other

new components added later in the life cycle of the product) to accomplish goals that were not

part of the original scope of work or even considered by the developers originally responsible for

implementation. The less dependence components require in a system, the simpler it becomes to

refactor or reengineer those components. Hunt and Thomas go on to say that

> "There is a fairly subtle gain in productivity when you combine orthogonal components.
> Assume that one component does M distinct things and another does N things. If they
> are orthogonal and you combine them, the result does M x N things. However, if the two
> components are not orthogonal, there will be overlap, and the result will do less. You get
> more functionality per unit effort by combining components" (Hunt, 2000).

As Hunt and Thomas point out, when a system is truly made up of orthogonal components, then

no duplication has occurred - the maximum efficiency of the usage of programming code has

been achieved. This means that there is no overlapping functionality in the system. Anytime

multiple functions in a system are designed to achieve the same result, extra work (no matter how simple it was to duplicate) occurred. While this extra work may only lead to a few minutes of work in a small system, when scaled up to enterprise-level systems with hundreds of programmers, this simple "subtle gain" can balloon into enormous amounts of wasted time.

In addition to increasing productivity, using orthogonality to develop systems has the benefit of reducing risks. As discussed above, in a nonorthogonal system, changes made to one component or module of a system has the possibility of impacting different components throughout the system. This can lead to costly debugging where the developers responsible for fixing a problem or adding new functionality have to travel down multiple paths to correct any unintended side effects that occur in other areas of the system just to maintain balance in the system after making changes in one specific area. In an orthogonal system, this problem is avoided: changes can be made to one module of code without fear of unintended side effects occurring throughout the system. Hunt and Thomas illustrate that as "Diseased sections of code are isolated. If a module is sick, it is less likely to spread the symptoms around the rest of the system. It is also easier to slice it out and transplant in something new and healthy." (Hunt, 2000). The end result is that orthogonal systems are stronger and healthier than non-orthogonal systems because any problems created by making a change to a particular area of code will be restricted to that area of code. Automated tests are much simpler to design and implement in an orthogonal system because each test only needs to check the quality of the component in question - there is no reliance on tests of the other components. If a system relies on a particular outside product (be it in-house or third-party), risk is reduced by implementing it orthogonally because this reduces the overall integration of the outside product to the system to a minimum.

This provides the development team with the greatest flexibility if a problem arises due to the outside system or if the outside system itself ends up needing replacement.

Unlike some methodologies, processes, or tools, orthogonality is an idea that is applicable from the top to the bottom of the development of a system - and not just in terms of the responsible parties involved in the development process. It also is a concept that is applicable to the various stages of the development process. Jeff Atwood describes orthogonality as "a powerful concept that applies at every level of coding, from the architecture astronaut to the lowest level code monkey. If modifying item #1 results in unexpected behavior in item #2, you have a major problem -- that's a form of unwanted coupling." (Atwood, 2009).

To start with, a system must be designed with orthogonality in mind. The architect responsible for the design of a system must keep in mind that every component of the system must be considered as an independent unit, designed to operate within itself and not dependent upon any other component to do any of the work it is responsible for. There is no part of orthogonality that is necessarily easy to implement and it is very easy to lose orthogonality in a project. The system should consist of a series of independently functioning components that work together. In larger systems these components may end up designed as structured layers where each component represents a specific abstraction and can only use the abstractions provided by the layers below. This can lead to a powerful and flexible architecture that helps reduce non-orthogonal dependencies between components in a system. Andrew Hunt and David Thomas state the following about how a designer can test for orthogonality:

"There is an easy test for orthogonal design. Once you have your components mapped out, ask yourself: If I dramatically change the requirements behind a particular function,

how many modules are affected? In an orthogonal system, the answer should be "one."

Moving a button on a GUI panel should not require a change in the database schema.

Adding context-sensitive help should not change the billing subsystem." (Hunt, 2000).

Of course, in reality, this test is a bit naive. It is highly unlikely that any real-world change made to the business rules that drive a system will only affect a single component or module of the system. It is quite likely that several components will have to be modified if the requirements are changed. That being said, if the business rules that drive a particular function within the system are changes, in an orthogonally implemented system, no other changes should need to be made elsewhere in the system: each change should only affect one module. Through the use of specific design patterns such as the Model-View-Controller (MVC) design pattern (design patterns in general and the MVC pattern specifically will be discussed in further detail), a designer can help enforce orthogonality throughout a system. Part of the design process is determining which tools such as programming libraries are going to be used in the system. Hunt and Thomas warn of the danger of losing orthogonality in a system when third-party tools are introduced into a system:

"We once worked on a project that required that a certain body of Java code run both locally on a server machine and remotely on a client machine. The alternatives for distributing classes this way were RMI and CORBA. If a class were made remotely accessible using RMI, every call to a remote method in that class could potentially throw an exception, which means that a naive implementation would require us to handle the exception whenever our remote classes were used. Using RMI here is clearly not orthogonal: code calling our remote classes should not have to be aware of their

locations. The alternative - using CORBA- did not impose that restriction: we could

write code that was unaware of our classes' locations." (Hunt, 2000).

Whenever a developer or designer is considering the usage of a particular tool or function, no

matter if the source is third-party or from within the development team, the same questions need

to be asked as if the developer or designer was writing the code for that functionality from

scratch. Analysis of the impact of the code and its input and output must occur to guarantee that

orthogonality has not been lost. For example, if the design of an object's persistence requires the

developer to access objects in a way that is unique to this design, the design is not orthogonal.

Hunt and Thomas illustrate this using Enterprise Java Beans and stating that it has an

> "interesting example of orthogonality. In most transaction-oriented systems, the
>
> application code has to delineate the start and end of each transaction. With EJB, this
>
> information is expressed declaratively as metadata, outside any code. The same
>
> application code can run in different EJB transaction environments with no change. This
>
> is likely to be a model for many future environments." (Hunt, 2000).

As previously stated, when a third-party tool must be used in a system, implementing it

orthogonally to minimize the integration of that tool will reduce the risk involved in using it.

At the team and individual levels, developers must be diligent to remember that every

time they sit down to generate code for a system there is the risk of losing orthogonality in a

system. The mindset of avoiding overlapping functionality and interdependent modules must be

constant, as does an awareness of the purpose of both the system in general and the components

being modified. Part of the individual (and peer, for that matter) code review process should be

to be on the watch for this very problem. There are several techniques that can be used for preserving (and testing for) orthogonality. One of the easiest is to adhere to the Law of Demeter (also known as the Principle of Least Knowledge) which states that one should never talk to strangers. Specifically, it states that each component should have the most limited possible knowledge of its neighboring components in a system (Lieberherr, 1997). By writing "shy code" a developer can prevent any components from revealing anything unnecessary to other components and thus prevent a non-orthogonal reliance upon another component's implementation. Additionally a developer can avoid the use of identical or similar components which might lead to dependence upon each other (or lead to changes to one component needing to be made on all similar components). Lastly, a developer can preserve orthogonality by limiting (ideally avoiding entirely) globally stored data. Hunt and Thomas state that "every time your code references global data, it ties itself into the other components that share that data. Even globals that you intend only to read can lead to trouble (for example, if you suddenly need to change your code to be multithreaded)." (Hunt, 2000). Components that are clearly passed any contextual information required for operation lead to code that is much simpler for developers to maintain over the life cycle of a system.

Once a system has been developed, if it has been built orthogonally, then as mentioned above, quality control becomes easier and often quicker. Quality control can be done at a more granular level - unit (function) testing - and should whenever possible be done as each granular component is completed and integrated into the system (and ideally automated).

"An orthogonally designed and implemented system is easier to test. Because the interactions between the system's components are formalized and limited, more of the

system testing can be performed at the individual module level. This is good news, because module level (or unit) testing is considerably easier to specify and perform than integration testing. In fact, we suggest that every module have its own unit test built into its code, and that these tests be performed automatically as part of the regular build process." (Hunt, 2000).

They go on to recommend that developers take the time to integrate unit testing into each function as they are developed, instead of waiting until later in the development process to introduce and create individual unit tests. An additional advantage of unit testing is the fact that it can be included (and automated) in a continuous integration process. When a repetitive process can be automated it frees up time and resources, allowing those resources to be used in more productive areas. "This is a good opportunity to bring automation to bear. If you use a source code control system, tag bug fixes when you check the code back in after testing. You can then run monthly reports analyzing trends in the number of source files affected by each bug fix." (Hunt, 2000).

Orthogonality should be adhered to in the project resource makeup, and not just in how code is designed and implemented. Wherever possible a development team should be built with orthogonality in mind. Avoid responsibility overlap, and a team will spend less time debugging problems that occur from functionality being affected outside its respective component and also avoids function duplication. Both lead to costly increases in the time spent developing a system and can cause frustration and confusion among the team members. Project and team managers must be diligent to work with the architect of the system when planning out the resources available for implementing the system.

"When teams are organized with lots of overlap, members are confused about responsibilities. Every change needs a meeting of the entire team, because any one of them might be affected. How do you organized teams into groups with well-defined responsibilities and minimal overlap? There's no simple answer. It depends partly on the project and your analysis of the areas of potential change. It also depends on the people you have available. Our preference is to start by separating infrastructure from application. Each major infrastructure component (database, communications interface, middleware layer, and so on) gets its own subteam. Each obvious division of application functionality is similarly divided. Then we look at the people we have (or plan to have) and adjust the groupings accordingly." (Hunt, 2000).

Lastly, but not least, orthogonality does not guarantee the success of a project, or its ability to be completed on time. While an orthogonal system has been proven to provide a software development team with the ability to produce a system that is much easier to maintain and upgrade and the ability to produce it on time with a minimal amount of risk, as Brandon Byars states, "orthogonality doesn't guarantee good code. However, it allows the language to be used in unanticipated ways, which is A Good Thing. Moreover, since everything is an expression, you can put expressions where you wouldn't normally expect them." (Byars, 2008). Hunt and Thomas go on to say that orthogonality "may be a clumsy word, but if you use the principle of orthogonality ... you'll find that the systems you develop are more flexible, more understandable, and easier to debug, test, and maintain." (Hunt, 2000).

Design Patterns

One of the key aspects preached by experienced architects, developers, and designers of object-oriented code is reuse - be that in the form of code reuse, design reuse, process reuse, etc. When looking at the design of a software system, the phrase designers are taught to remember is "don't reinvent the wheel" (or "if it ain't broke, don't fix it"). Design patterns are the concept of taking proven solutions to existing recurring problems and applying that to the current project. The Gang of Four (an name the software engineering industry has affectionately termed for the four developers who first wrote about the use of design patterns in software development - Gamma, Helm, Johnson, and Vlissides), in Design Patterns: Elements of Reusable Object-Oriented Software, described the general problem that software system designers deal with every time they begin work on a new project as follows:

"Designing object-oriented software is hard, and designing reusable object-oriented software is even harder. You must find pertinent objects, factor them into classes at the right granularity, define class interfaces and inheritance hierarchies, and establish key relationships among them. Your design should be specific to the problem at hand but also general enough to address future problems and requirements. You also want to avoid redesign, or at least minimize it. Experienced object-oriented designers will tell you that a reusable and flexible design is difficult if not impossible to get "right" the first time. Before a design is finished, they usually try to reuse it several times, modifying it each time." (Gamma, 1995).

They then go on to say that the best software designers learn to reuse the solutions to problems they come across instead of re-inventing the wheel: "One thing expert designers know not to do

is solve every problem from first principles. Rather, they reuse solutions that have worked for them in the past. When they find a good solution, they use it again and again." (Gamma, 1995). The Gang of Four drew inspiration for the application of design patterns in software engineering from Christopher Alexander, a building architect who was the first to define design patterns (in architecture, but the concept applies to software engineering as well as other professions) as "a solution to a problem in a context" after which he then went on to say that "Each pattern describes a problem which occurs over and over again in our environment and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice." (Alexander, 1977).

In software engineering, design patterns are not specific implementations of programming code - nor are design patterns only applicable to a particular combination of hardware, operating systems, or a given programming language. The Gang of Four describes what design patterns are and are not as

"Design patterns are not about designs such as linked lists and hash tables that can be encoded in classes and reused as is. Nor are they complex, domain-specific designs for an entire application or subsystem. The design patterns ... are descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context. A design pattern names, abstracts, and identifies the key aspects of a common design structure that make it useful for creating a reusable object-oriented design." (Gamma, 1995).

Design patterns, in general, consist of eight specific features combined into four general components. The eight specific features that make up the whole of all design patterns are the

following: name, intent, problem, solution, participants, consequences, implementation, and generic structure. The name, obviously, is a unique word that identifies the pattern. Some examples of names used for software design patterns include the Model-View-Controller (MVC) pattern, the Bridge pattern, and the Observer pattern. The intent is a statement of the purpose of the design pattern and the problem is the statement used to describe the problem the design pattern is supposed to solve. The solution is a description of how the pattern provides the solution to the problem. Participants are the programming entities that are used in the design pattern. The cause and effect relationships of those entities are described in the consequences of the design pattern while the implementation statement is a description of how to apply the solution to the problem. Lastly, the generic structure is a simple diagram of the entities used and the relationships they share with each other - basically a visual representation of the design pattern (Shalloway, 2005).

At a less granular level a design pattern consists of the following four components:

1. The first component essential to a design pattern is its name - which should, ideally, describe the problem, solution, and consequences in just a couple of words at most. Giving a design pattern a name increases the level of abstraction at which a developer can operate when designing a system.

2. A design pattern should have a statement that consists of both a description of the intent of the design pattern and a description of the problem that it is supposed to solve.

3. Next, a design pattern should describe the solution to the problem and the details of how this can be accomplished. This should include the participating entities of the design pattern and the generic structure.

4. Lastly, the design pattern needs to include the consequences involved in using the design pattern. In software engineering, the consequences are the cause and effect relationships of the use of the entities involved in the design pattern. (Gamma, 1995).

There are three basic benefits to the usage of design patterns in software engineering design. The first is the one most often described: the ability to reduce the resources spent on the design of a solution to a problem. Shalloway explains it as "by reusing already established designs, I get a head start on my problems and avoid gotchas. I get the benefit of learning from the experience of others. I do not have to reinvent solutions for commonly recurring problems." (Shalloway, 2005). Developers using design patterns are able to save time in the design process by using already existing patterns instead of spending valuable time creating their own (which likely will be at most a variation of an already existing pattern). By documenting these proven patterns and making them available to any developer, the Gang of Four points out that "Design patterns make it easier to reuse successful designs and architectures. Expressing proven techniques as design patterns makes them more accessible to developers of new systems." (Gamma, 1995).

A second benefit to the use of design patterns is the ability to increase the re-usability of the programming code used in the implementation of that pattern and decrease the difficulty required in the maintenance of any system using that pattern. So not only can time be saved in the design process, but it can also be saved in the implementation (and later maintenance) cycles of the development process. Shalloway states

"Most design patterns also make software more modifiable and maintainable. The reason for this is that they are time-tested solutions. Therefore, they have evolved into structures

that can handle change more readily than what often first comes to mind as a solution. They also handle this with easier to understand code - making it easier to maintain." (Shalloway, 2005).

Lastly, design patterns provide a common language for developers to use in communication. This can save time in all stages of a system's development. According to Shalloway, establishing a shared vocabulary is required for any team to have effective communication and establish efficient teamwork. It provides team members with a common viewpoint of the system, the problem it is attempting to solve, and the solution required to fulfill it. Design patterns contribute a point of reference that is understood by everyone that is usable throughout the development cycle of a system (from analysis and design through to implementation and quality control) (Shalloway, 2005). That same benefit extends to documentation which then provides ongoing benefits for everyone from end-users to developers who may work on this project in the future. That same common point of reference provides a clear description of both the intentions of the objects used and of the interactions that occur between those objects. Shalloway goes on to describe a personal observation from the use of design patterns and its benefits on a development team:

"My experience with development groups working with design patterns is that design patterns helped both individual learning and team development. This occurred because the more junior team members saw that the senior developers who knew design patterns had something of value and these junior members wanted it. This provided motivation for them to learn some of these powerful concepts." (Shalloway, 2005).

Continuous Integration

Continuous integration is the process of applying quality control to a tightly controlled granular level implemented throughout the development process instead of following the traditional model of applying it after the product is considered finished. Martin Fowler, the originator of the practice of continuous integration, describes it as

> "Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly." (Fowler, 2006).

In the development of software applications based on compiled programming languages, continuous integration is a key component to successfully achieving rapid development (especially when using agile processes). One of the key aspects of continuous integration is the ability to reduce the chances of problems (both small and large) from hindering the release of a project. One of the biggest problems associated with the traditional method of developing using deferred integration (which does not apply quality control until after the project is initially built) is that everyone from project management and the client all the way down to the individual members of the development team really have no idea just how much more time is really going to be required before the project is truly completed. Continuous integration removes the long integration period with a blind spot of unknown time from the end of a project. In an ideal world, automated processes such as continuous integration would fix bugs as well as detecting

them, but continuous integration does the next best thing. By integrating after each commit, the exact set of changes made are known, so therefore when the automated tests find a bug the developer is notified and they know exactly which small set of files are host to the cause of the bug. Basically continuous integration provides almost instant feedback and a least common denominator of changes to review to debug any problems. "With CI, you make small changes to the source code and integrate these changes with the rest of the code base on a regular basis. If there are any problems, the project members are informed and the fixed [files] are applied to the software immediately." (Duvall, 2007). Tools such as 'diff' (ideally combined with a version control system) can be used to compare the files in the latest commit to those in the last commit to successfully pass the integration test to further simplify the developer's effort to correct the mistake in the shortest period of time possible (because they notified within minutes of committing the build, this additionally has the advantage of not requiring the developer to have to go back and remember what thought processes they were going through when they initially developed the code). Because continuous integration enforces self-testing in every build of the project which occurs on every commit of code, bugs are kept to a minimum, which reduces or even removes the complications that stem from having bugs that interact and affect other bugs. The short version is that "By integrating many times a day, you can reduce risks on your project. Doing so facilitates the detection of defects, the measurement of software health, and a reduction of assumptions." (Duvall, 2007). Continuous integration also helps keep an orthogonally designed system in that state, as developers can ensure that the automated tests and the components being tested are independent of any other component.

A second key aspect of continuous integration is the ability to much more easily deploy versions of the project on a frequent basis. This allows the end-users of the project to experience

new features (or fixes to existing problems) without having to wait for the next monthly, quarterly, yearly, etc. release. It additionally allows for more frequent feedback from end-users as they experience these new features and find areas for improvement or usability bugs. It allows for a more collaborative experience where the end-user is more involved in the development process. "Continuous integration can enable you to release deployable software at any point in time. From an outside perspective, this is the most obvious benefit of CI. We could take endlessly about improved software quality and reduced risks, but deployable software is the most tangible asset to "outsiders" such as clients or users." (Duvall, 2007). Development projects that follow a deferred integration practice instead of one based on a form of continuous integration usually end up with one of two problems: a delayed release due to fixing bugs found or a rushed quality control process where bugs are only found after the system is released to the client. Either problem can lead to anywhere from a failed launch to a failed project depending on the scope of the problems.

The age old adage "work smarter, not harder" is the simplified way of stating the fact that achieving effective practices that help a development team achieve the objectives that have been set before it are the crux of rapid development. One component of working smarter is the reduction (or outright removal where possible) of any process that is repeated on a regular basis. Due to the nature of systems development, repetitive tasks cannot always be outright removed, but they can often be automated, which achieves the same result. Continuous integration is designed around the idea of automating the build processes and much of the quality control processes that occur throughout development. "Reducing repetitive processes saves time, costs, and effort. This sounds straightforward, doesn't it? These repetitive processes can occur across all project activities, including code compilation, database integration, testing, inspection,

deployment, and feedback." (Duvall, 2007). Implementing an automated continuous integration

system provides for a stable and consistent quality control process (i.e. the tests are run the same

way every time, eliminating missed bugs due to inconsistent testing). This also allows the

development team to test the quality of the code (code inspection, documentation inspection)

before running the automated tests. And of course, this allows every single commit to result in a

full test of the system, which, as previously mentioned, minimizes the impact of any bugs found

to the least common denominator of changes. These advantages free up development resources

as it reduces the amount of work required for repetitive processes to a minimum, which then

allows developers to spend their time on much more valuable work (Duvall, 2007).

Effective decisions cannot be made without the foundation of solid information to back

them up. In the corporate world, at the executive level, information system tools such as

decision support systems provide support for those responsible for decision-making and provide

them with the reports necessary to make the best possible choice. It only makes sense to apply

this to the development world as it can provide help at all levels: engineers, developers, project

managers, etc. can all benefit. Continuous integration tools can provide the information needed

for each of these roles to make smarter decisions throughout the project. Duvall continues to

elaborate on this advantage of stating that continuous integration "provides the ability to notice

trends and make effective decisions, and it helps provide the courage to innovate new

improvements. Projects suffer when there is no real or recent data to support decisions, so

everyone offers their best guesses." (Duvall, 2007). When this data is not able to be collected

and analyzed automatically, it requires developers and managers to manually acquire and

organize the data. Often, due to the sheer amount of time required to manually collect the data

and build the report when compared to the time left until the project's deadline, the collection of

this data never occurs (or is out of date by the time it is completed). This leaves the entire development team without the information on trends that could have provided great benefit to decision-making at every level of the process. A continuous integration server such as CruiseControl, Hudson, or Bamboo all provide reports that can help reduce mistakes, increase efficiencies, and reinforce processes that will help a development team improve their own capabilities which in turn will lead to better results, happier end-users, and more confident developers.

> "Overall, effective application of CI practices can provide greater confidence in producing a software product. With every build, your team knows that tests are run against the software to verify behavior, that project coding and design standards are met, and that the result is a functionally testable product. Without frequent integrations, some teams may feel stifled because they don't know the impacts of their code changes. Since a CI system can inform you when something goes wrong, developers and other team members have more confidence in making changes. Because CI encourages a single-source point from which all software assets are built, there is great confidence in its accuracy." (Duvall, 2007).

Implementing continuous integration is not a simple process though, and especially in a corporate environment where products must continually be finished on time and under budget to meet business rules and objectives, must be done incrementally. Broken down into its simplest components, there are four steps to implementing continuous integration into a project. Duvall lists them as Identify, Build, Share, and Make it Continuous (he uses the mnemonic device "I Build So Consistently"). The first step is to identify all of the processes (across all areas of the

development process, from database integration to code compiling to documentation inspections) that can be automated. Secondly, a master build script must be written that should consist of all the processes being run. Thirdly, using a centralized or distributed version control system (such as Git, Mercurial, CVS, and Subversion) allows for the automated processes and build script (in addition to the project's code) to be shared among the entire development team. The last step to implementing continuous integration is tying the first three steps together into a system that runs with every commit of programming code to the repository (ideally this becomes an automated process). Duvall goes on to say

> "Aim for incremental growth in your CI system. This is simple to implement, the team gets more motivated as each new item is added, and you can better plan what you need next based on what's working so far. Often, attempting to throw everything into a CI system immediately can be a bad move, just like refactoring a lot of code at once isn't the best approach when writing software. Get it to work first, get developers using it, and then add other automated processes as needed based on the project risks." (Duvall, 2007).

To be successful, any development methodology or framework must be adopted and used at every level of a development team. Developers, engineers, quality assurance, managers, and even clients and end-users must all be on the same page. This is just as important in the big picture (the overall framework) as it is at any given component of that framework, such as continuous integration. Duvall states it simply when he says "CI is not just a technical implementation; it is also an organizational and cultural implementation." (Duvall, 2007). Without cooperation and understanding from everyone involved, continuous integration is nothing more than a complicated Rube Goldberg device in the development process. For

example, in continuous integration, the ability to integrate successfully and find all possible bugs

in a build requires a complete test suite that covers all areas of the project. If a test suite only

tests certain components of the project, or does not do so thoroughly, or the build script only

partially builds the system, then frequent integration is not going to provide much of a benefit

over deferred integration. When changes are made to the business logic that drives a system, or

when new features are added to that system, then new tests or updated build scripts need to be

added to the continuous integration process to keep the development process as efficient as

possible. This does mean that there is some overhead to using continuous integration, though the

time saved by using it correctly should easily eclipse the time taken to perform quality control

processes when deferred integration is used. Within the realm of building the best possible

approach that will meet schedules, costs, quality, performance, and any other goals expectations

must be understood. Developers, engineers, and managers must know what is possible and

quality assurance, managers, and clients need to know what is realistically possible to be

accomplished. Continuous integration (any form of rapid development) requires that a team

choose both "effective practices rather than ineffective practices" and "practices that are oriented

specifically toward achieving your schedule objectives." (McConnell, 1996).

Unit Testing

Quality control in software should ideally be an ongoing process, and not a one-time

occurrence. The computer hardware industry has for some time now employed built-in testing

on many systems and the advantages of being able to test the quality and functionality of a

processor or chipset on demand or when changes are made has not been lost of the software

engineering industry. Hunt and Thomas point how the software engineering industry has begun

imitating the same type of testing that chip manufacturers employ when launching new processors:

> "Chips are designed to be tested - not just at the factory, not just when they are installed, but also in the field when they are deployed. More complex chips and systems may have a full Built-In Self Test (BIST) feature that runs some base-level diagnostics internally, or a Test Access Mechanism (TAM) that provides a test harness that allows the external environment to provide stimuli and collect responses from the chip. We can do the same thing in software. Like our hardware colleagues, we need to build testability into the software from the very beginning, and test each piece thoroughly before trying to wire them together." (Hunt, 2000).

The primary aspect of test-driven software application development in general, and continuous integration specifically, that integrates quality control into every commit made to a system's code repository, is the use of unit tests. Unit testing consists of applying pre-built individual tests to the smallest possible parts (aptly named units) of code (such as a specific function in procedural programming or a class in object-oriented programming) that checks to make sure that the unit of code is functioning correctly and returning the expected results. These tests, just like the code that they test, should be written orthogonally so that each unit test is created independent of any other unit test and independent of any other unit of code. This method of verifying the quality of the programming code submitted validates its usability in the system being modified. According to Microsoft's Software Development Network,

> "The primary goal of unit testing is to take the smallest piece of testable software in the application, isolate it from the remainder of the code, and determine whether it behaves

exactly as you expect. Each unit is tested separately before integrating them into modules

to test the interfaces between modules. Unit testing has proven its value in that a large

percentage of defects are identified during its use." (MSDN, 2010).

By applying this level of testing at the most granular level possible (individual units of

programming code), when a unit test fails to correctly pass, the programmer responsible for

quality control is able to know exactly where to debug and fix the system. In contrast, when

integrating and applying quality control at the very end of a system's development instead of

throughout the development, the programmer or engineer responsible for correcting the problem

must spend an unknown amount of time determining the cause of the problem and where it is

located in the code (and if the system was not written orthogonally, this can lead to an

exponential increase in the time required to resolve the problem, as the problem can be linked to

code throughout the system with far reaching effects).

> "The key aspect for unit tests is having no reliance on outside dependencies such as
>
> databases, which have the tendency to increase the amount of time it takes to set up and
>
> run tests. Unit tests can be created and run early in the development cycle (i.e., day one).
>
> Because of the rapid time between coding and testing the results, unit tests are an
>
> efficient way of debugging." (Duvall, 2007).

In all practicality, it is likely that some unit tests will be tightly linked to specific functions or

classes that are linked with other functions or classes. To maintain the highest level of

orthogonality, these should be kept to a minimum, but when it occurs, Duvall mentions the

following, "Occasionally, unit tests even employ mocks, which are simple objects that substitute

for real, more complicated objects. If a dependent object itself does depend on an outside entity

like a file system or database and isn't mocked, the test becomes a component test." (Duvall, 2007). Therefore a unit test, when the unit it is testing depends upon a different unit or component, should be designed and written so that the test itself can simulate the functionality that the unit depends upon and thus remove the requirement for that additional unit to be active within the system. This does require the test to be modified should the unit being simulated change, but it removes the need for that unit to be included and accessible to the test in order for it to be executed.

Unit testing provides several benefits for a system - whether the system is currently in development or being maintained after its initial release. One of the benefits is the fact that the code has accountability built in and removes the need for any one developer to "own" the programming code of a particular unit of functionality (be that for testing or for making changes). Don Wells states that unit tests "enable collective ownership. When you create unit tests you guard your functionality from being accidentally harmed. Requiring all code to pass all unit tests before it can be released ensures all functionality always works." (Wells, 1999). The use of unit testing to preserve the intent of a particular unit of code removes the need for a development team to establish individual code ownership.

Unit testing also provides developers with the ability to show the progress of a system as it is being developed. In a system being developed using deferred integration, it is difficult (and sometimes impossible) to demonstrate to the end-user or client the progress that has been made over the course of a period of time. Even if a critical component to the system is missing (such as a database), unit testing will allow the development team to show input and output and working functionality. Timothy King emphasizes this when he says that unit tests "demonstrate

concrete progress. You don't have to wait a month for all the pieces of the system to come together. You can show progress even without a working system." followed by "Not only can you say you've written the code, you can actually demonstrate success. Of course, this is another distinction that traditional programming teaches us to ignore." (King, 2006).

One of the most common arguments against the use of unit testing in software development is the time associated with the development of each and every unit test ("The biggest resistance to dedicating this amount of time to unit tests is a fast approaching deadline." (Wells, 1999)). There is no doubt that there is merit in the argument that it takes time away from pure functional development of the system but the time spent developing these unit tests and using them versus the great unknown of debugging that occurs when deferred integration (and thus deferred testing) occurs can result in a significant amount of time saved. Wells goes on to illustrate this when he states that over the course of entire software development life cycle of a project "an automated test can save you a hundred times the cost to create it by finding and guarding against bugs. The harder the test is to write the more you need it because the greater your savings will be. Automated unit tests offer a payback far greater than the cost of creation." (Wells, 1999)." In all truth, the use of unit tests should reduce the total time of development for a system instead of increase it. Additionally, this provides future developers who are working in the system with a greater ability to understand the purpose of any given unit of code, which will save time in any future programming of that code.

"Test-first reduces the cost of bugs. Bugs detected earlier are easier to fix. Bugs detected later are usually the result of many changes, and we don't know which one caused the bug. So first we have to hunt for and find the bug. Then we have to refresh our memories

on how the code is supposed to work, because we haven't seen it for months. Then finally we understand enough to propose a solution. Anything that reduces the time between when we code the bug and when we detect it seems like a obvious win. We consider ourselves lucky to find out about bugs within a few days, before the code is shipped to SQA or to customers. But how about catching them within a few minutes? That's what test-first accomplishes with the bugs it catches." (King, 2006).

Quality control is going to happen on a system. As Andrew Hunt and David Thomas state, "All software you write will be tested - if not by you and your team, then by the eventual users - so you might as well plan on testing it thoroughly. A little forethought can go a long way toward minimizing maintenance costs and help-desk calls." (Hunt, 2000).

Chapter 3 – Research Methodology

Background

The researcher's original plan was to make use of a grounded theory approach in order to achieve the objectives planned for this research. Grounded theory research makes use of a set of specific steps applied in a repetitive pattern for collecting data and analyzing it to develop a framework based on the data collected. It was determined that after spending more time in a thorough analysis of applying grounded theory as a methodology to this research that this research did not provide the correct background or setting for the use of grounded theory research, due the resources available to the researcher and the specifics of the research area being investigated.

In grounded theory research, data is analyzed and documented in a four step process as outlined by Leedy and Ormrod. The first step is open coding, where the data that has been collected thus far is broken down into categories based on common themes. Categories are then broken down into more granular attributes called properties. In short, open coding is the process of breaking down the data collected into a series of classifications based on least common denominators. The second step is called axial coding and the emphasis is determining the connections that exist between items of both levels (categories and properties) (Leedy, 2005). The first two steps are the most intertwined with data collection as additional data collected is used to flesh out the categories, properties, and the connections that exist throughout. The third stage is selective coding. Selective coding is the building of a "story line" of what happened throughout the data collection process. The final stage in data analysis is to gather all of the analysis from the previous three steps and develop the theory and framework that will make up

the research project (Leedy, 2005). In grounded theory data collection and data analysis are performed hand-in-hand, with seamless movement occurring between the two throughout the process.

After the initial research of available literature regarding software development frameworks used in the development of software applications based on compiled programming languages, the researcher came to the conclusion that the initial starting point for a framework for test-driven web application development using interpreted programming languages would be based on the methodologies and tools discovered to have been adopted for use with compiled programming languages. One of the primary reasons grounded theory was abandoned as the primary research method was because, according to Leedy and Ormrod, "a grounded theory study is the one least likely to begin from a particular theoretical framework." (Leedy, 2005). Grounded theory research is designed to derive a framework from the analysis of the data collected. This research project is about the application of proven methodologies used with compiled programming languages to the development of web applications based on interpreted programming languages. In addition, because the purpose of this research is the evaluation of the application of known methodologies in one area of software engineering to a different area, the researcher reached the conclusion that the time required to complete the almost recursive nature of grounded theory research (looping between data collection and data analysis) was more than the time available to actually complete the research.

Overview

Instead of using grounded theory research as the framework for this study, the researcher chose to use single case study research as the research methodology. According to Lee and Rine,

one of the most difficult problems researchers face when proving a software engineering

methodology or framework is how to accurately access the collection, presentation, and analysis

of the data. In the software engineering realm, additional complications arise because software

engineering methodologies (SEM) "involves the use of human knowledge in its methods

(phases)." (Lee, 2004). The question that then arises is how to collect that knowledge and make

it available for analysis and therefore substantiate the framework being tested. Lee and Rine go

on to say that "measuring such knowledge is hard, but we can benefit from the 'case study

research design' which is a valuable and an important empirical research alternative in designing

a research plan that establishes a logical link from the data to be collected to the initial questions

of the study." (Lee, 2004). A case study is the detailed study of a person(s), group(s), event, or

series of events over a specified time frame and is a proven qualitative research methodology. It

is described by Robert Yin as an "empirical inquiry that investigates a contemporary

phenomenon within its real-life context, especially when the boundaries between phenomenon

and context are not clearly evident." He goes on to say that one would "use the case study

method because you wanted to understand a real-life phenomenon in depth, but such

understanding encompassed important contextual conditions – because they were highly

pertinent to your phenomenon of study." (Yin, 2009).

To best understand the software development methodologies used in the development of

applications based on compiled programming languages, a review of relevant online and offline

literature (including, but not limited to, vendor whitepapers, peer-reviewed documentation

including blogs and journals, and books written by professionals in the software engineering

field) was carried out. In addition, this review of materials offered the researcher the necessary

understanding to properly select the use of a case study over grounded theory as the ideal

research methodology for this thesis. As an empirical evaluation research methodology applied

to a technological framework (such as the invented test-driven web application development

framework that is the focus of this research) a case study research design builds a basis for valid

conclusions to be drawn from both specific data collected and observations of outcomes from

events in the case study (Lee, 2004). By itself, a case study will not provide statistically

significant conclusions, but "many different kinds of evidence, figures, statements, documents,

are linked together to support a strong and relevant conclusion." (Runeson, 2009).

Case Study Research Framework Components

Based on the presentations of the application of case study research (applied to software

engineering where possible) of Robert Yin, Seok Won Lee, David C. Rine, Per Runeson, and

Martin Host, there are there are five major steps to performing case study research:

1.) Design: objectives are defined and the case study is planned.

2.) Prepare: procedures and protocols for data collection are defined.

3.) Collect: execution with data collection on the studied case.

4.) Analyze: analysis of collected data.

5.) Report: presentation of findings.

These operate in a fairly linear order, going from step to step, while allowing for some

iterative development to occur as changes to the design may be necessary due to events that

occur during the collection and analysis stages and for communication to occur as necessary

(such as sharing and reporting on important information gathered while preparing the case study)

(Yin, 2009). As such, these five steps were utilized as the framework to be used for the

evaluation to validate the test-driven web application development framework for the rest of this research.

Design and Prepare

Central to the design of research based on case studies is the objective statement of the research – a description of what the case study is expected to be accomplish. Further refining of the objective breaks it apart into a series of research questions that make up the source of what is to be answered with the conclusion of the analysis of the case study events (Runeson, 2009). The primary objective of this research was to analyze the varying software development methodologies that are in place in the environments responsible for the development of applications using traditional compiled programming languages in an effort to gain an understanding of best practices that could then be taken and molded into a framework for developing web applications using interpreted programming languages. Following the review of related works, both online and offline, breaking down this objective into its most basic components created a series of propositions that can best be summarized through the following research questions (which then provide links to the data collected during the course of the case study):

1.) How significant (and how advantageous) are the software engineering frameworks and methodologies used by development teams that primarily use compiled programming languages to the software development life cycle when used with interpreted programming languages for the development of web applications?

2.) What currently used compiled programming language-based software engineering methodologies are the most effective in interpreted language-based web application

development? Why? How can these methodologies be best applied to the

development of interpreted language-based web applications?

3.) What impact does the introduction of the most effective methodologies taken from

the development of applications using compiled programming languages using

currently available tools have on the overall web application development process?

Yin defines a case as anything that is a "contemporary phenomenon in its real-life

context" (Yin, 2009). In the real-life context of software engineering, the most straightforward

choice for a case may be a software development project, framework, or methodology (though of

course a case is not limited to these) (Runeson, 2009). An internationally distributed software

engineering team (working for an Austin, TX based multi-tenant web application provider)

specializing in software-as-a-service web applications was chosen as the case studied for this

research.

The team, which was organized in 2006, consisted of up to sixteen developers divided

into three project teams, two systems engineers, three quality assurance specialists, and one

project manager. The developers were categorized by one (or more) of three Subject Matter

Expertise (SME): front-end (programmers who specialized in JavaScript, CSS, and client-side

web programming), core-structure (programmers who specialized in PL/SQL and database

management – including server-side code responsible for interactions with the database), and

logic-code (programmers who specialized in PHP and server-side web programming). All but a

couple of the developers were from Russia and Ukraine, while the engineering, quality control,

management, and a couple of the development staff were located in the United States. All

members of the team ranged from one to fifteen years of experience working with development

teams. From 2008 till the time of this research, the team had been focused on the design and

implementation of several software-as-a-service web applications for both consumer and

business clients. The three primary products that were in development by the project sub-teams

and were therefore involved in this case study were a small/medium business Customer

Relationship Management portal (IgniteCRM), a social network portal specialized for multi-

level-marketing consultants (360Central), and a document and file sharing application (Ignite

Share). The team primarily used the following tools to develop and maintain its projects: Linux-

based clustered servers: web servers running Apache, database servers running PostreSQL, a

centralized version control system (Subversion), a centralized project and issue management

system (Trac), and primarily wrote software using PHP, XML/XHTML, JavaScript, CSS, and

SQL. Due to the international nature of the team, communication primarily occurred in English

over company hosted XMPP-based chat rooms with support by email and Skype.

The combination of using orthogonality, design patterns, continuous integration, and unit

testing were implemented as successful components of the test-driven development of

applications when used with compiled programming languages to the case study's web

application development process. Orthogonality is a mindset of writing units of code that are

independent of other units of code. In this case study, this was handled at the component level as

it directly applied to the design pattern chosen. Wherever possible it was also handled at the

function (method) level so that it would directly apply to unit testing also. Design patterns are

the concept of taking proven solutions to existing recurring problems and applying that to the

current project. In this case study, the Model-View-Controller design pattern (which encourages

orthogonality) was chosen and implemented using the object-oriented Kohana PHP5 framework

(by default PHP is not object-oriented). All programming was written using Kohana version

2.3.4. Unit testing was deployed using PHPUnit with a module specifically written for use with this version of Kohana. Lastly, continuous integration was managed using the Hudson Continuous Integration server with individual application builds handled through a custom-written build script (including the creation of a test database) and automated testing was handled using Phing calls of the PHPUnit binary.

Collecting Data

In case study research, it is important to use several data sources in order to reduce the effects of a single interpretation of only one source of data. "If the same conclusion can be drawn from several sources of information, i.e. triangulation, this conclusion is stronger than a conclusion based on a single source." (Runeson, 2009). Data collection can be categorized in three levels based on the degree by which the researcher is involved with the case study sources responsible for data collection: first degree (where the researcher gathers data via direct contact with the case study participants and therefore collects data as it is generated), second degree (where the researcher gathers data via indirect methods such as the viewing of remote/automated recordings of the case study events), and third degree (where the researcher gathers data via analyzing available documentation such as requirements specifications or reporting databases) (Lethbridge, 2005). In this case study, as a participant and observer the researcher was able to gather data in the form of both first and third degree data resources. Participant observation and conversations with members of the development, engineering, and quality control staff provided first degree data collection sources while and both qualitative and quantitative document analysis of prior projects, requirement specifications, and quality assurance reports provided third degree data.

First degree data collection was conducted in the form of participant observations by the researcher as one method of data collection. Participant observation as a form of data collection provided the researcher with a unique perspective into the implementation of developing web applications using the test-drive framework derived from research into methodologies used by development teams that use compiled programming languages. For the most part, observations occurred with a high degree of interaction by the researcher but a fairly low awareness of being observed of the developers and staff involved. Data collected consisted of primarily chat room logs, logs from individual conversations, notes taken from relevant meetings and discussions, and field notes recorded by the researcher.

Participant surveys were conducted by the researcher as another method of data collection. Developers, engineers, and quality assurance staff of the development team were contacted by the researcher and asked to participate in the data collection. Data collected consisted of a questionnaire given to all members of the development team that consisted of interview questions and a series of modified likert-scale statements designed to collect data focused specifically on the objectives of the research which were based on the review and analysis of related works. The questionnaire was processed through the use of web-based forms where the collected data was stored in a database.

The questionnaire requested the development team staff respond to the following four open-ended questions and 10 likert-scale statements:

1.) How does the lack of standardized test-driven methodologies in the web application industry affect your ability to develop on-time and on-budget web applications with a high degree of quality?

2.) How would you describe the environment (specifically in regards to quality control, communication, and development tools) of the development team before the implementation of the full test-driven web application development framework? Why?

3.) How would you describe the environment (specifically in regards to quality control, communication, and development tools) of the development team after the implementation of the full test-driven web application development framework? Why?

4.) What components of the test-driven web application development framework currently in use by the development team have you found to be the most effective? Least effective?

Evaluate and respond to the following ten statements using this scale (1. Strongly disagree; 2. Disagree; 3. Neither agree nor disagree; 4. Agree; 5. Strong agree; 6. Not applicable):

1.) Orthogonal design and programming is important for the on-time and on-budget delivery of quality web applications based on interpreted programming languages.

2.) Using Design Patterns and the common language for components (an example of this consistent vocabulary in general would be the primary terms of the MVC design pattern where "model" represents accessing/storing data, "controller" represents handling input and processing data, and "view" represents rendering the data and input/output into a user interface) is important for increasing the effectiveness of

communication across all members (from QC to software analyst to developer) of a development team.

3.) Continuous integration systems like Hudson are important for the on-time and on-budget delivery of quality web applications based on interpreted programming languages.

4.) Unit testing systems like PHPUnit are important for the on-time and on-budget delivery of quality web applications based on interpreted programming languages.

5.) Consistent use of programming code units and components that are written orthogonally (independent of changes made to other code units and components) is important for providing all members of the development team the capability to read and understand the purpose of a function or component.

6.) Consistent use of well-established solutions involving consistent vocabulary to existing problems in the form of design patterns and is important for increasing efficiency and communication among members of web development teams.

7.) Consistent feedback in the form of unit testing and continuous integration is important for building trust in both the programming code and development team members.

8.) The continuous integration and unit testing systems, along with the design pattern framework were reliable and consistent through the software development lifecycle.

9.) Development teams responsible for web applications based on interpreted programming languages would benefit from the implementation of a reporting system based on providing feedback from tests run on each committed change in code of a particular application.

10.) Development teams responsible for web applications based on interpreted languages should regularly review and evaluable the need to incorporate or adapt the software methodologies of other types of development teams (such as those responsible for desktop applications based on compiled languages).

Appendix A contains an example questionnaire.

Third degree data collection was conducted in the form of document analysis of prior projects, requirement specifications, and quality assurance reports. Data from four different milestones from three separate projects were analyzed in retrospect (each of which was conducted prior to the start of this case study). The archival data was used broadly throughout the analysis of all collected data as complementary sources of information to both compare and contrast the impact of employing test-driven development methodologies in web application development.

Analyzing Data

Collected data from this emancipatory (or improving) case study was analyzed using hypothesis confirmation techniques with a focus on using triangulation and negative case analysis (researching possible alternate explanations of case study events that reject the proposed hypothesis) to ensure the validity of the research study. The hypothesis generated in the course of this research was, specifically, that the application of test-driven software development methodologies proven successful with development teams that use compiled programming languages can be applied to web application development teams that use interpreted programming languages in order to fulfill the need for a risk-reducing design framework to

coordinate the resources responsible for the most effective development of web applications. The use of an open-source qualitative data analysis program (Weft QDA) alongside the use of standard office productivity software (specifically Microsoft Word and Excel) was used to support the analysis of the data collected.

As is necessary when conducting qualitative research, the analysis of the data collected was approached in parallel with the collection of the data in order to provide the required flexibility needed for this type of research (to provide the ability to adapt to new insights and information and accumulate additional data). Analysis was therefore accomplished in a series of stages where initial data was collected based on the review of related works, collated into categorical classifications (codes), and used to form an initial hypothesis. Further research involving the case study was then conducted and analysis was repeated. This pattern continued until generalized conclusions could be formulated. Analysis was conducted with a semi-formal (editing) approach where only a few codes used are based on the hypothesis and most are generated based on the results of the researcher's analysis of data collected.

Reporting Results

According to Runeson, "An empirical study cannot be distinguished from its reporting. The report communicates the findings of the study, but is also the main source of information for judging the quality of the study." (Runeson, 2009). In general, a case study report consists of the following five attributes:

1.) State the objectives of the case study.

2.) Clearly describe the studied case.

3.) Provide a "history of the inquiry" in detail so the reader can see who did what and when it was done.

4.) Include a "chain of evidence" of basic data including any categories used in the classification of the data so the reader arrives at the same conclusions as the researcher.

5.) Communicate the researcher's conclusions and ensure they are set into the context that they affect. (Runeson, 2009)

The conclusions of this case study research are being communicated in the standard linear-analytic report structure. This is the traditional structure for research reports made up of the following components in order: a description of the problems found in current web application development, a review of related work, an explanation of the data collection methods used, analysis of the data collected, and a presentation of the conclusions reached and their respective context.

Chapter 4 – Data Analysis and Results

Overview

The purpose of this research was the evaluation of the methodologies and tools that software development teams working with compiled programming languages use when applied to a software web application development environment for the purpose of designing a suitable test-driven framework based on working with interpreted programming languages. Through the process of reviewing available literature the researcher became aware of a specific set of components (orthogonality, design patterns, continuous integration, and unit testing) that were used by many successful development teams that primarily used compiled programming languages. These directed the development of the three research questions that guided the remainder of the research (data collection, analysis, and reporting) and became the key components used in the design of the test-driven web application development framework.

Evidence was collected from multiple sources using first and third degree data collection techniques. A review of literature and available research, participant observation, and participant surveys provided the majority of the data collected. Data collected via participant observation consisted chat room logs, logs from individual conversations, notes taken from meetings, and the researcher's field notes. While this data collection occurred with a high level of researcher interaction a low level of observational awareness was maintained with the case study participants who were being observed. This data provided the researcher with a greater understanding of the impact of the day-to-day usage of the various components of the test-driven framework through the eyes of the team members. Data collected via participant survey processed through the use of a web-based form consisted of a 14 question survey that was given

to all members of the case study. The data collected by the case study participants who completed the survey provided the researcher with a better understanding of the overall impression made on the case study participants of the effectiveness of the test-driven framework's overall implementation.

The analysis of the data collected from the case study of the test-driven web application development framework was structured around the research questions that best summarized the objective of this research:

1.) How significant (and how advantageous) are the software engineering frameworks and methodologies used by development teams that primarily use compiled programming languages to the software development life cycle when used with interpreted programming languages for the development of web applications?

2.) What currently used compiled programming language-based software engineering methodologies are the most effective in interpreted language-based web application development? Why? How can these methodologies be best applied to the development of interpreted language-based web applications?

3.) What impact does the introduction of the most effective methodologies taken from the development of applications using compiled programming languages using currently available tools have on the overall web application development process?

Research Question 1

The development team members in this case study were all skilled PHP web programmers with varying levels of experience in the use of test-driven methodologies (mostly

from prior experience working with compiled applications based on Java). PHP as a language is

not object-oriented, and the majority of the web applications written by members of the team

prior to this case study were written using the native implementation of PHP in a procedural (or

structured) manner. When asked about the lack of test-drive methodologies in web software

development, the majority of the developers acknowledged that web development (based on

interpreted programming languages such as PHP) is a step behind software development based

on compiled programming languages such as Java. Some of the more experienced developers

pointed out that implementing the tools was only going to be the first step of building a test-

driven culture in a development team primarily designed around the use of PHP. Consistent

work was also going to be required to change the mind of web application developers into

accepting the advantages of applying test-driven methodologies to the development of web

applications (as can be seen in some of the following survey results – often the majority of the

developers agreed with the survey statements while one or two would regularly disagree because

that is not how they were used to developing code). Because the majority of the development

team was inexperienced in the use of design patterns and orthogonal programming, the

researcher chose to introduce both of these concepts to the team using the Model-View-

Controller design pattern. This specific design pattern was chosen and implemented for this case

study in part because it established a simple, but effective, common vocabulary and solution for

the members of the team and its implementation encourages orthogonal programming. The

teams were divided into smaller project teams which were in turn divided by subject matter

expertise. This organization method existed prior to the introduction of this research, but was

modified so that terminology from the common vocabulary introduced by the Model-View-

Controller design pattern was used (for example, developers whose subject matter expertise was

working with CSS, JavaScript, and XHTML were considered to have the "Interface" subject

matter expertise). This implementation was accomplished using the object-oriented Kohana

PHP5 MVC (Model View Controller) framework. While several versions of the Kohana

framework were available at the time of this case study, all development was accomplished using

version 2.3.4 of the Kohana framework. When asked to respond to the survey statement

"Consistent use of programming code units and components that are written orthogonally

(independent of changes made to other code units and components) is important for providing all

members of the development team the capacity to read and understand the purpose of a function

or component.", 82% of the seventeen case study participants who answered the question agreed,

12% neither agreed nor disagreed, and 6% disagreed.

Consistent use of programming code units and components that are written orthogonally (independent of changes made to other code units and components) is important for providing all members of the development team the capability to read and understand the purpose of a function or component.
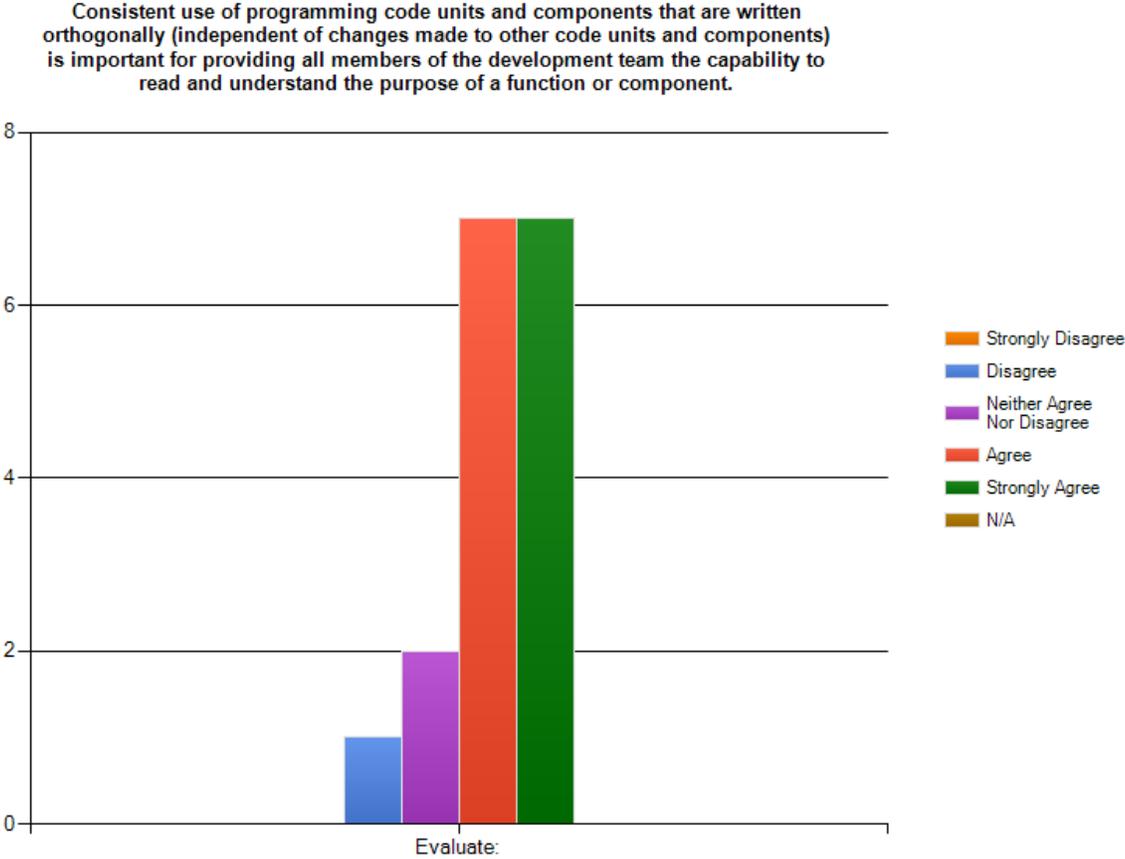
Additionally, no members of the case study disagreed with the survey statement "Consistent use of well-established solutions involving consistent vocabulary to existing problems in the form of design patterns is important for increasing efficiency and communication among members of web development teams.". Of the seventeen case study participants who responded, 82% agreed, 12% neither agreed nor disagreed, and 6% felt the answer was not applicable to their involvement in the study.

Consistent use of well-established solutions involving consistent vocabulary to existing problems in the form of design patterns and is important for increasing efficiency and communication among members of web development teams.
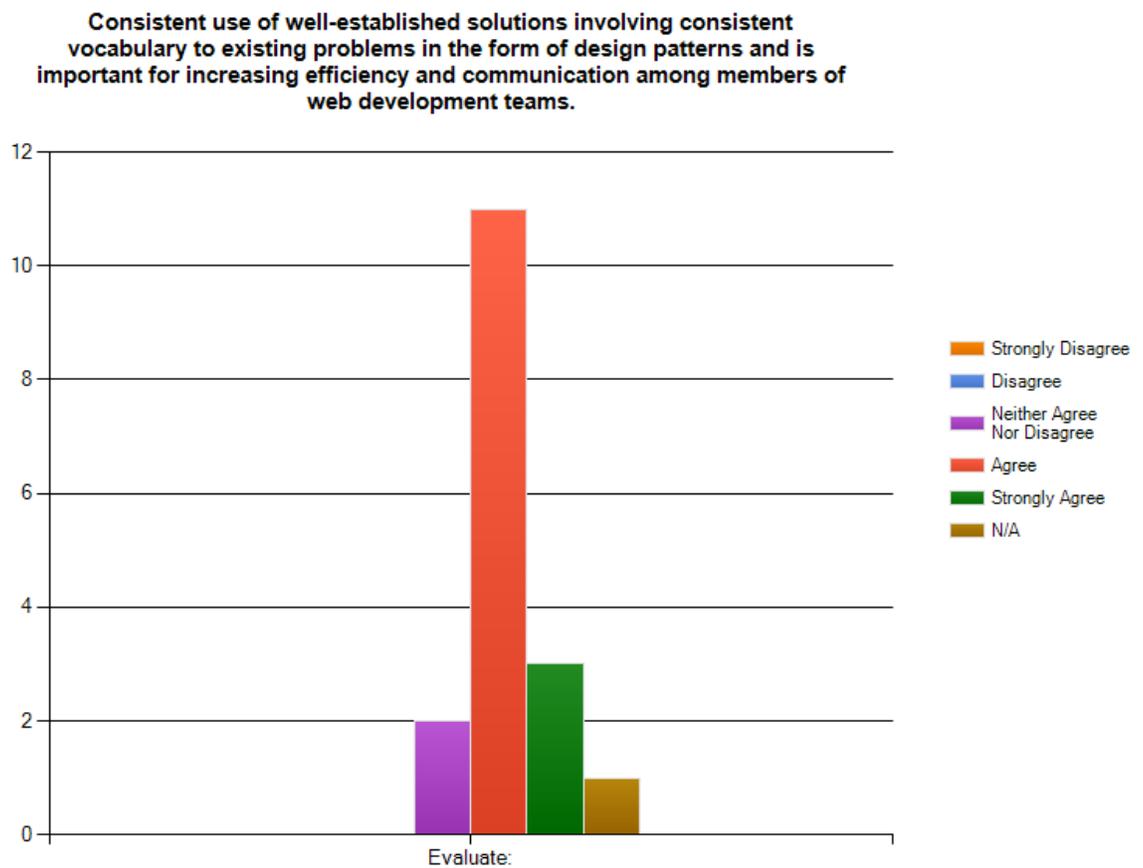
While several members of the development team had heard of the terms "continuous integration" and "unit testing" almost none of them had any experience actually using these technologies (using interpreted programming languages such as PHP or using compiled

programming languages such as Java). When questioned about the lack of methodologies and

tools for applying continuous integration and unit testing to web application development, quite a

few developers indicated surprise that the two terms were, in fact, different. The common

misconception among the developers of this team was that continuous integration was just

automated unit testing instead of unit testing being one component of continuous integration.

Quite a few continuous integration tools were available at the time of this case study but almost

all of the available options were designed for compiled programming languages and not for any

type of web applications (the exception seemed to be compiled Java applications designed to run

as a web applet). The researcher decided to make use of the Hudson continuous integration

server primarily because it offered a wide variety of customizability for tailoring to specific

development environment paired with a fairly easy to use web administration interface. Since at

the time of this research there were no dedicated, mature, continuous integration servers

designed for interpreted programming languages such as PHP (especially when using an object-

oriented framework such as Kohana which required a bootstrap file to initiate), Hudson's

flexibility allowed the engineers working with the development team to build a custom

continuous integration solution. Hudson was responsible for monitoring the execution of each

integration of a project and running each step of the build process. Figures 1, 2, and 3 show

example screens from the development team projects, build history, and build configuration.
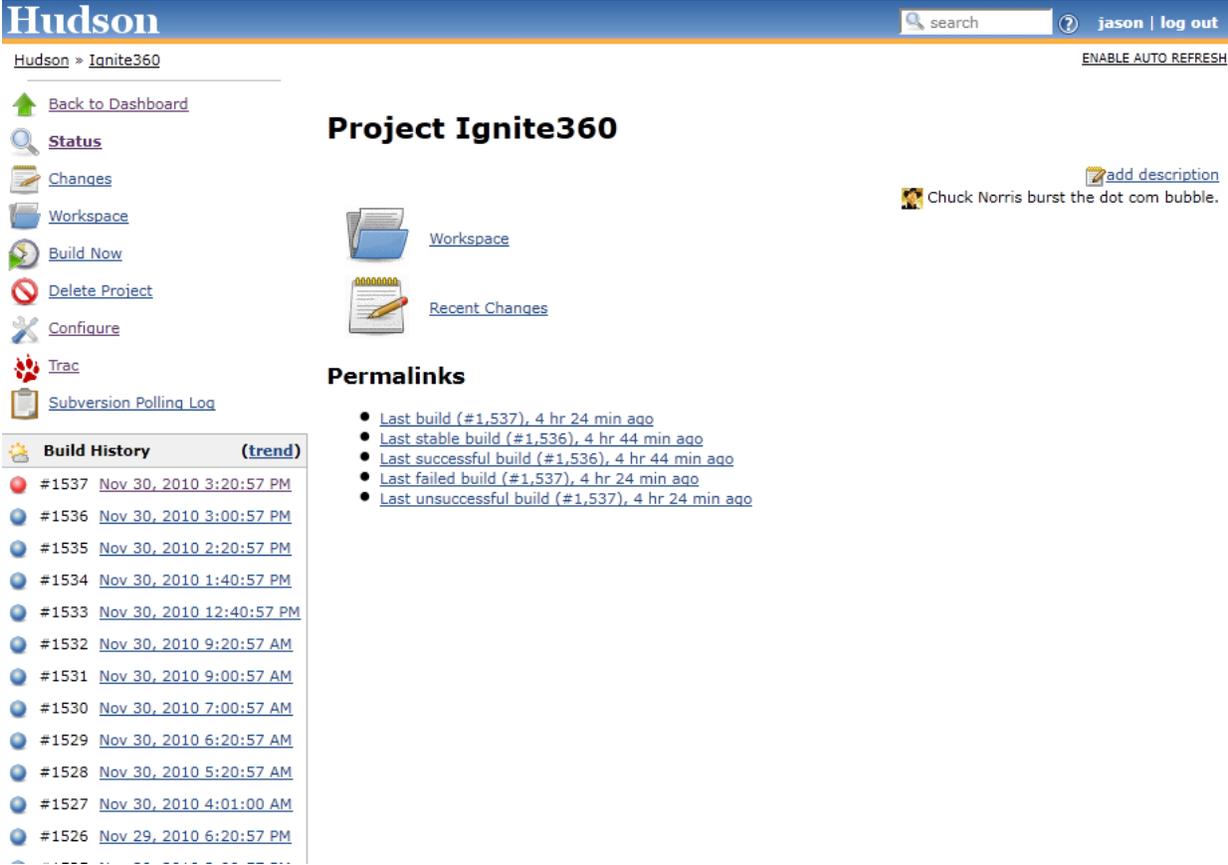
Figure 1. Hudson screen showing project status and recent build history.

Figure 2. Hudson screen showing project build details.

Figure 3. Hudson screen showing project configuration and build setup.

Phing was chosen as the software responsible for actually building each release of the development team's projects. Phing is a build system designed specifically for PHP but based on Apache Ant (a build system commonly used to build and test applications developed in Java (and other compiled languages) which is often paired with the Cruise Control continuous integration server). Phing build scripts are designed using XML which made it a good starting ground for developing (and introducing) each step in the continuous integration process. PHPUnit was the tool chosen for unit testing each application – primarily because of plugins that allow it to work

natively with the Kohana PHP framework and because when combined with Phing, PHPUnit can

be used to run a batch of unit tests instead of being run one by one. Figure 4 shows an example

of using Phing to call a series of PHPUnit tests associated with the IgniteCRM project.

Figure 4. SSH screenshot of a manual execution of Phing to call PHPUnit.

When asked to respond to the survey statement "Consistent feedback in the form of unit

testing and continuous integration is important for building trust in both the programming code

and development team members.", 70% of the seventeen case study participants who answered

the question agreed, 18% neither agreed nor disagreed, 6% disagreed, and 6% felt the answer

was not applicable to their involvement in the study.

Consistent feedback in the form of unit testing and continuous integration is important for building trust in both the programming code and development team members.

Evaluate:

- Strongly Disagree
- Disagree
- Neither Agree Nor Disagree
- Agree
- Strongly Agree
- N/A
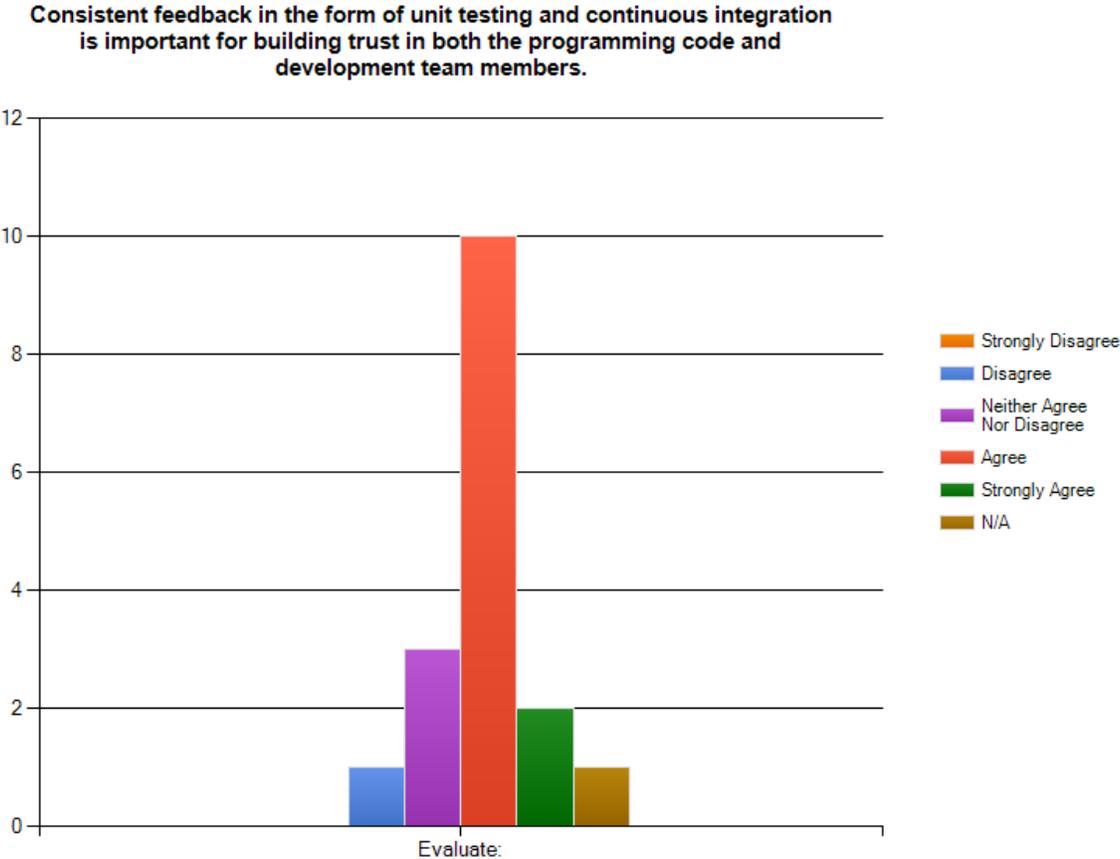
The development team used an XMPP-based chat system (Openfire) that not only

provides instant-messaging capability but also provides the ability to host multiple-user chat

rooms – more specifically chat rooms were created for each project so that each project team

could work independent of distractions from project-specific communications from the other

teams. These chat rooms became the focal points for collaboration across the entire distributed

development team (augmented by voice calls using Skype and shared desktop tools such as

LogMeIn and Skype). The Hudson continuous integration server provided several methods for

providing feedback to the development team when each build of a project was tested. Whenever

the Hudson server built and tested a project, it would announce in the Openfire chat rooms to the

entire development team the success or failure of that build. It would additionally announce the

revisions numbers of the repository commits that were specifically tested along with the name of

the developers who commited those changes. Lastly, if a project failed to pass a build test, an

email would be sent to each of the developers responsible for the changes made. When a build

then successfully passed after a previous failure, in addition to the announcement made in the

project chat room, an email would be sent out announcing that the project's status was returned

back to normal. 82% of the seventeen case study participants who answered the question agreed

with the survey statement "Development teams responsible for web applications based on

interpreted programming languages would benefit from the implementation of a reporting system

based on providing feedback from tests run on each committed change in code of a particular

application." (12% neither agreed nor disagreed, and 6% disagreed).

Development teams responsible for web applications based on interpreted programming languages would benefit from the implementation of a reporting system based on providing feedback from tests run on each committed change in code of a particular application.
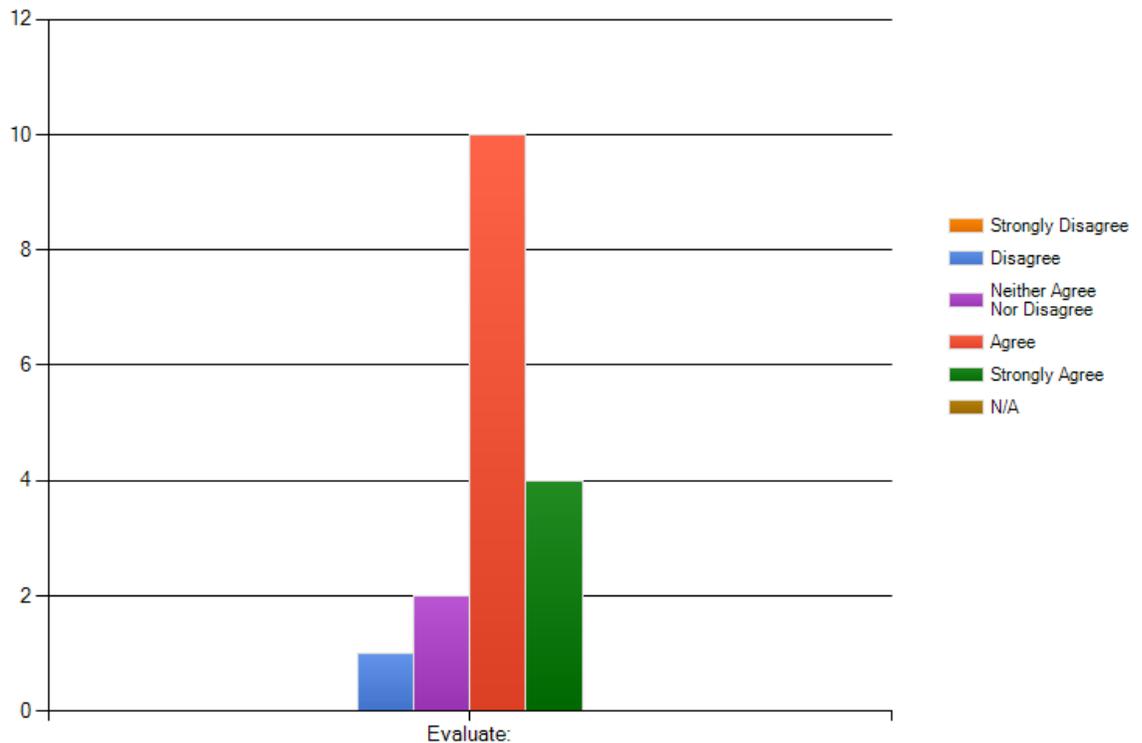
Research Question 2

The members of the development team that had previous experience in or knowledge of the methodologies used in software development based on compiled programming languages had a better grasp of what was and what was not considered test-driven. Many of the more inexperienced developers (or those that had only worked with interpreted programming languages) struggled with the application of the concepts of orthogonality, design patterns, and continuous integration. That said, all of the developers – no matter how much experience they had – picked up on the application of using unit tests fairly quickly. When responding to questions of which components of the test-driven web application development framework were considered the least and most effective, the responses paralleled the comfort level each developer felt using those components.

All of the developers had experience writing in the PHP programming language, but only a few of them had any experience writing PHP in an object-oriented manner. In addition, all but just a couple members of the development team had no experience (or knowledge) of programming orthogonally. The quality control and management members of the development team indicated strong agreement with Hunt that productivity gains and risk reductions are two of the primary advantages of writing programming code orthogonally (Hunt, 2000). The developers all signified that the combination of orthogonal programming and the use of design patterns made it much simpler for them to modify programming code in a project that they had not written due to the common vocabulary and reduced chance of affecting different areas of the system. The Kohana PHP framework chosen to implement the Model-View-Controller design pattern (and encourage the use of orthogonality when programming) is an object-oriented

framework and required a change in programming methods used by the development team. Once the developers learned how to use and program in Kohana, though, all members of the case study agreed that the quality of the projects (especially the quality of the code written) was improved. By the end of the case study, the majority of the case study participants who responded to the survey felt that the use of orthogonality and design patterns were effective means to increasing the effectiveness of web application development teams. One of the primary advantages of using design patterns, according to Shalloway, is the establishment of a shared vocabulary which is required to establish efficient teamwork by providing team members with a common viewpoint of the system, the problem it is attempting to solve, and the solution required to fulfill it (Shalloway, 2005). When asked to respond to the survey statement "Orthogonal design and programming is important for the on-time and on-budget delivery of quality web applications based on interpreted programming languages.", 88% of the seventeen case study participants who answered the question agreed, 6% neither agreed nor disagreed, and 6% felt the answer was not applicable to their involvement in the study.

Orthogonal design and programming is important for the on-time and on-budget delivery of quality web applications based on interpreted programming languages.
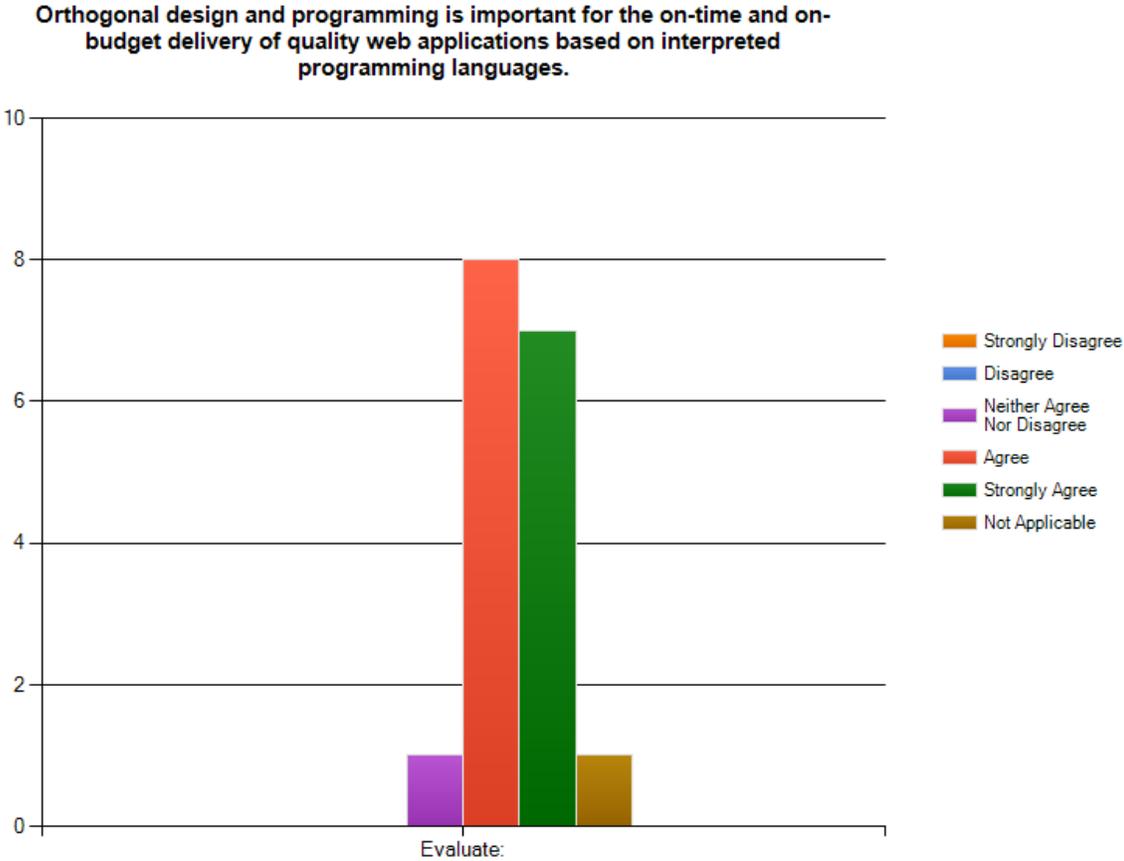
The response to the survey statement "Using Design Patterns and the common language for components (an example of this consistent vocabular in general would be the primary terms of the MVC design pattern where "model" represents accessing/storing data, "controller" represents handling input and processing data, and "view" represents rendering the data and input/output into a user interface) is important for increasing the effectiveness of communication acorss all members (from QC to software analyst to developer) of a development team.", showed that just how clear the memebers of the development team felt the advantages of using design patterns were - 88% of the seventeen case study participants who answered the question agreed (in fact 70% of the participants strongly agreed), 6% neither agreed nor disagreed, and 6% disagreed.

Using Design Patterns and the common language for components (an example of this consistent vocabulary in general would be the primary terms of the MVC design pattern where "model" represents accessing/storing data, "controller" represents handling input and processing data, and "view" represents rendering the data and input/output into a user interface) is important for increasing the effectiveness of communication across all members (from QC to software analyst to developer) of a development team.
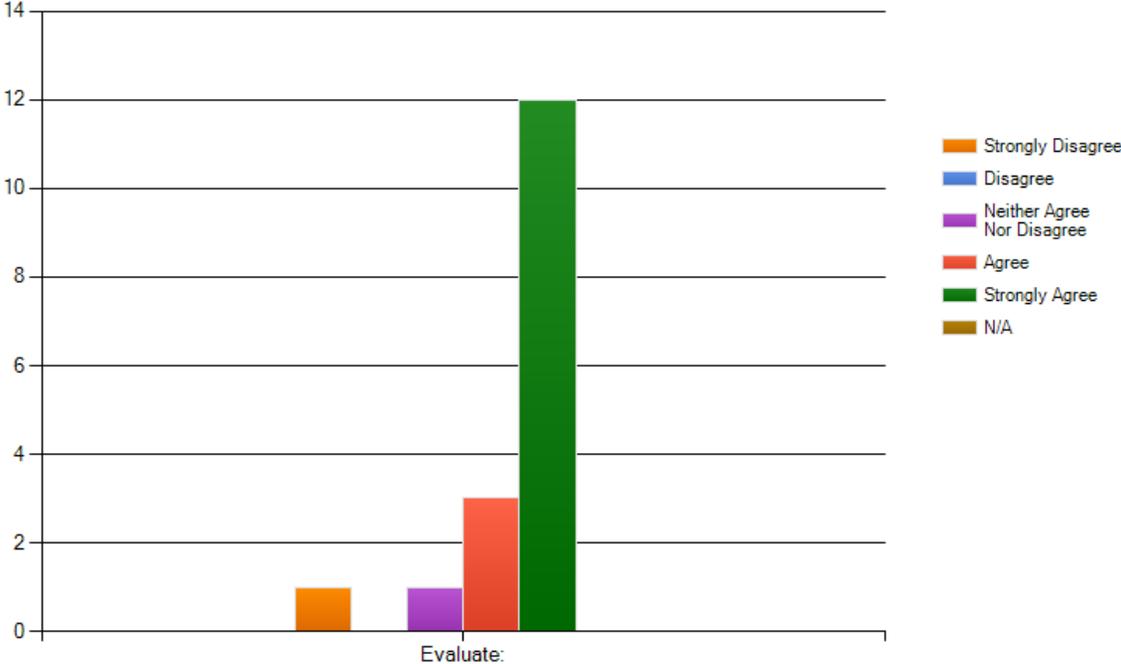
Unit testing was the single component of the test-driven web application development framework that all of the members of the development team involved in the case study grasped quickly and enthusiastically. Within hours of introducing Phing and a Kohana-compatible version of PHPUnit as the medium for creating, building, and testing code quality with unit testing the majority of the developers (even those who were fairly inexperienced) were grasping how to create unit tests and how to test their code using them. When questioned about it, the majority opinion was that unit testing provided them with the capability to take ownership of the quality of their code (in contrast to the other components which were controlled (or designed) by outside parties – for example, the continuous integration server was managed by the systems engineers and the Kohana framework was designed and written by an outside community of

developers). Additionally unit testing was the only component in the framework that consisted of physically programming code (orthogonality and design patterns are concepts that are applied to (or with) programming). The programming members of the development team are generally enthusiastic about their craft, and the feedback received from conversations among the team members and through the survey confirmed that unit testing was one of the more effective components of the test-driven web application framework.

The only aspect of writing unit tests that proved to be problematic was the emphasis on keeping a unit test orthogonal in addition to writing the original code orthogonally. Often developers lumped several units of code together with a single unit test, thus breaking the rules of orthogonality. As they began to grasp the concepts of orthgonality though, members of the team took the time to go and start decoupling unit tests. The application of using unit tests to improve the quality of the web applications being developed was the only component of the test-driven web application development framework that no members of the case study felt was ineffective. 82% of the seventeen case study participants who responded to the survey statement "Unit testing systems like PHPUnit are important for the on-time and on-budget delivery of quality web applications based on interpreted programming languages." agreed and 18% neither agreed nor disagreed.

Unit testing systems like PHPUnit are important for the on-time and on-budget delivery of quality web applications based on interpreted programming languages.
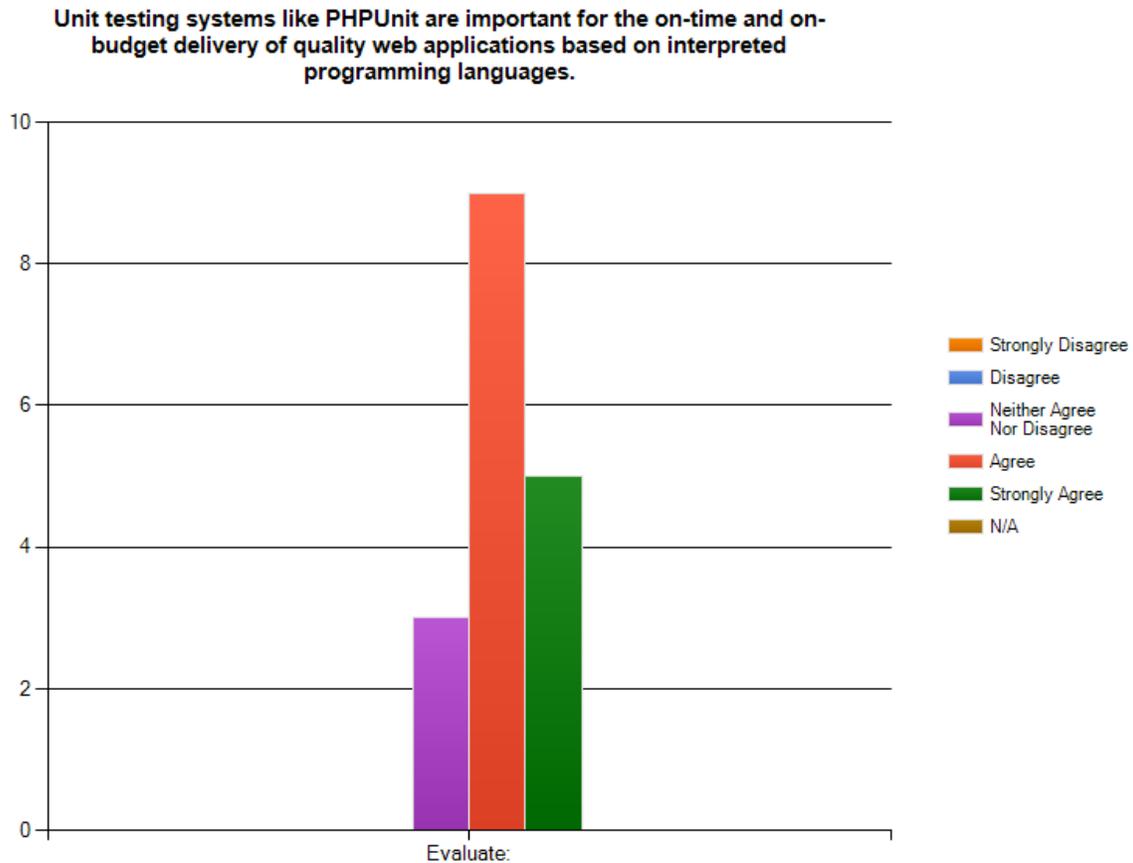
While the overwhelming majority of the development team considered the use of design patterns, orthogonality, and unit testing to be effective measures for increasing the quality of the web applications being developed, the use of continuous integration was considered a less effective measure. When asked to respond to the survey statement "Continuous integration systems like Hudson are important for the on-time and on-budget delivery of quality web applications based on interpreted programming languages.", 64% of the seventeen case study participants who answered the question agreed, 12% neither agreed nor disagreed, 6% disagreed, and 18% felt the answer was not applicable to their involvement in the study.

Continuous integration systems like Hudson are important for the on-time and on-budget delivery of quality web applications based on interpreted programming languages.
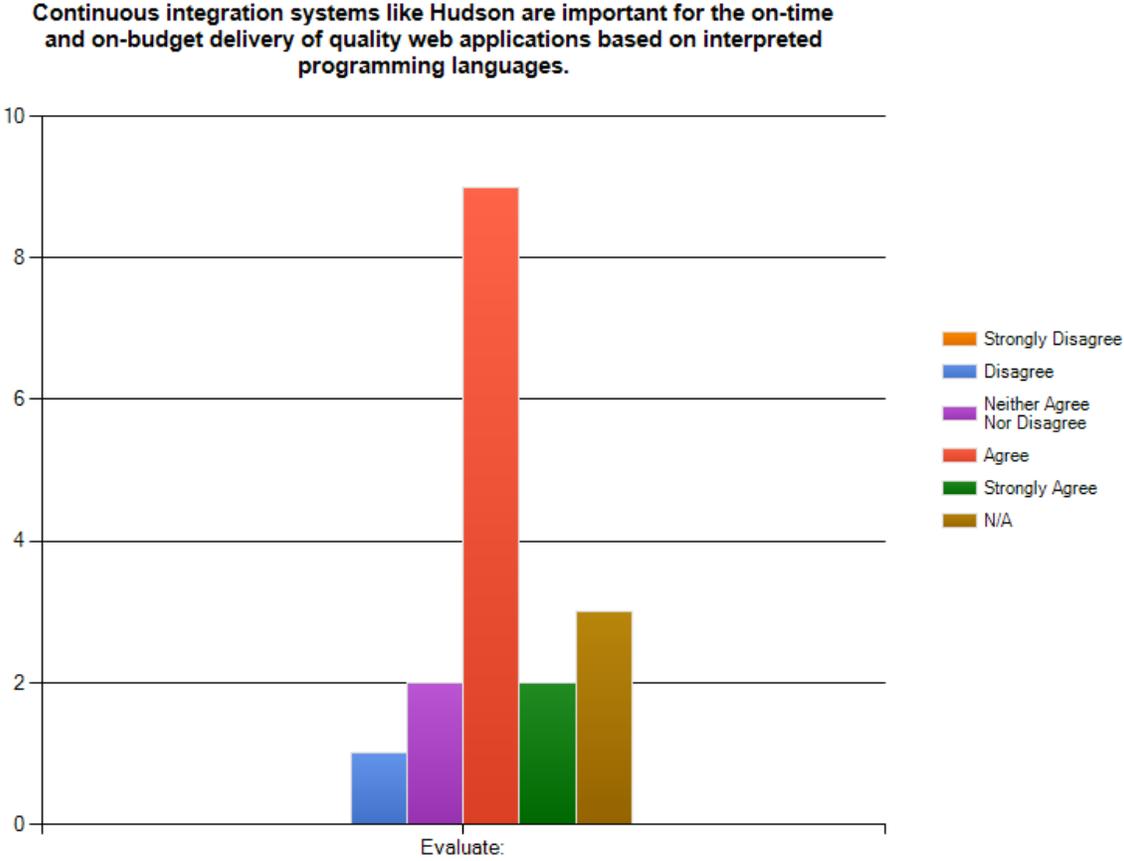
These results were in part due to the opinion among the members of the development

server that while the Hudson continuous integration server reliably built and tested projects,

occassional problems caused inaccurate results with builds due to mistakes made with creating

unit tests. In other words, the continuous integration server could consider a build to have failed

testing when there were no problems with the project's programming code itself if a unit test was

flawed in its design or implementation. Those problems occurred less often once the majority of

the team had a solid understanding of how to create unit tests although build errors due to unit

test flaws continued to be a risk – especially when new developers joined the team and had to

learn the test-driven development framework. More information about the performance of

Hudson and the impact it had on the results of this research can be reviewed in the analysis of

Research Question 3 that follows later in this chapter.

Research Question 3

When asked to discuss the describe the environment (specifically in regards to the quality

control, communication, and development tools) of the development team prior to the

implementation of the full test-driven web application development framework, the members of

the case study agreed upon the following:

- Project development was filled with delays at each stage of a release cycle. There
  were no levels of automation in the development process. Each stage – design,
  analysis, implementation, testing, and code releases – was handled manually.

- The development environment was frustrating. Quality control was mediocre at
  best due to to the lack of standardization, lack of communication, and lack of
  understanding expectations. Due to the delays found throughout the deferred
  intergration process, quality control was never given the time necessary to fully
  test and provide feedback on a project.

- Project management was described as a scattered collection of communication
  problems that have to be micromanaged by the project manager and software
  analysts. This led to bottlenecks that were complicated by the lack of automation.

Figure 5 shows the workflow that was used by the development team prior to the implementation

of the test-driven web application development framework. Due to communication problems

and delays at each stage, the workflow was overly complicated and often a point of frustration

for both the development staff and the quality assurance staff. Issues were lost in the shuffle and projects were rarely released on schedule, and often were released to the live production servers with problems still being worked on.
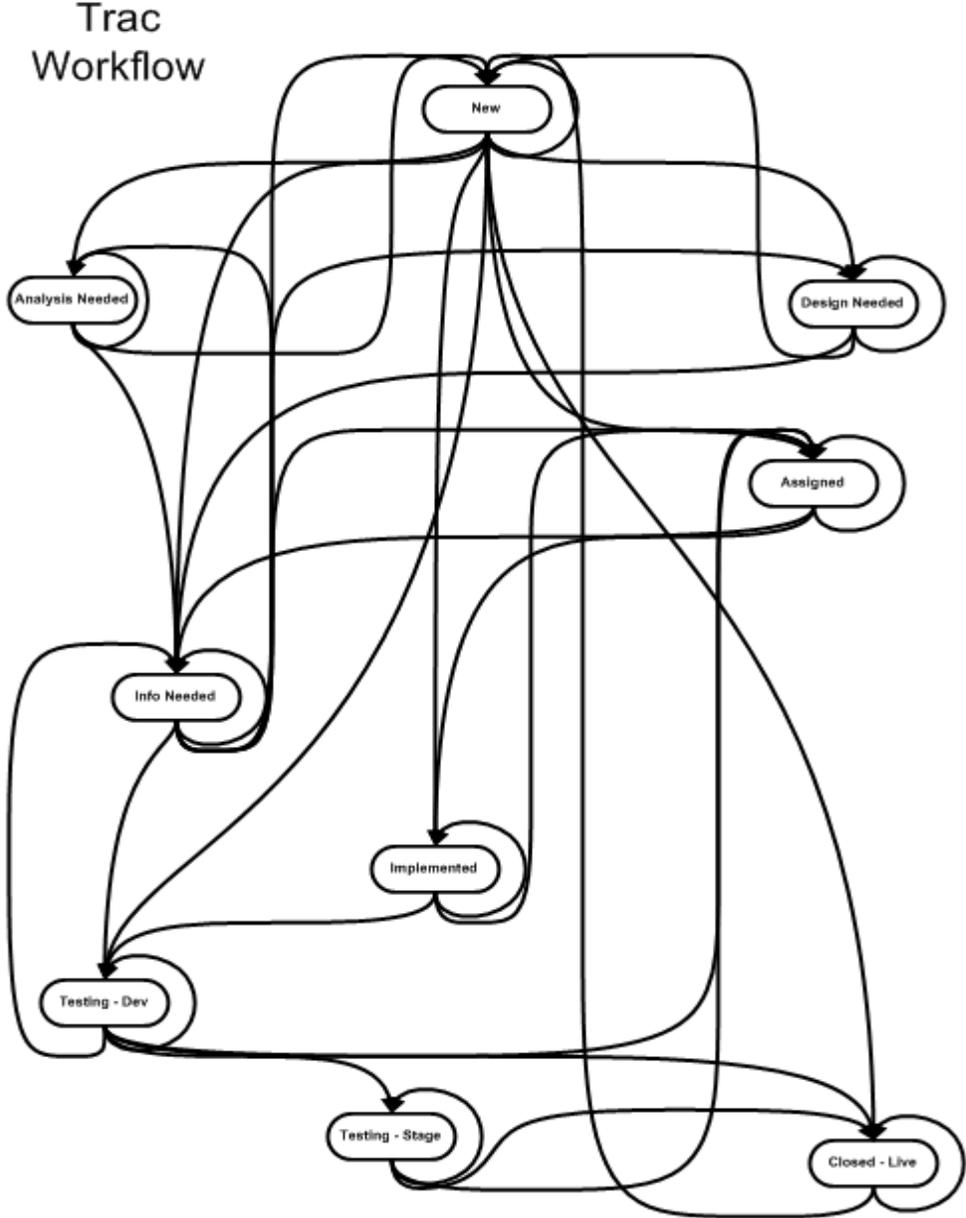
Figure 5: Ticket process workflow diagram before test-driven development framework.

Following the completion of the implemention of the test-driven web application development framework, members of the development team were asked to again discuss the development environment. The majority of the members of the development tea agreed upon the following statements:

- The addition of automation to the development process provided the capability to make changes to projects in a much shorter period of time. This was especially effective when optimizing or refactoring code to provide performance increases.

- Though the learning curve was steep, using design patterns such as Kohana's implementation of the Model-View-Controller pattern made it much easier to debug and track down code causing errors.

- There was a greater emphasis on feedback at all levels of the team. Additionally, there appeared to be greater oversight while still reducing the need to micromanage every level of the development process.

- Unit testing provided fast feedback for both the developers and the quality assurance staff. While it was not possible to detect all of the bugs within a system development proceeded much more smoothly.

- The team discovered that implementing the test-driven development framework was a much simpler process when applied to a new project instead of modifying an existing project not originally built on the framework.

Figure 6 shows the simplified workflow that was able to be used by the development team after the implementation of the test-driven web application development framework. Many of the bottlenecks (communication breakdowns, code release from one stage to the next, etc.)

that occurred throughout the development process were able to be removed, resulting in greatly reduced frustration from all members of the development. Projects were released on-time with a much higher level of quality (both in code quality and adherance to necessary business rules).
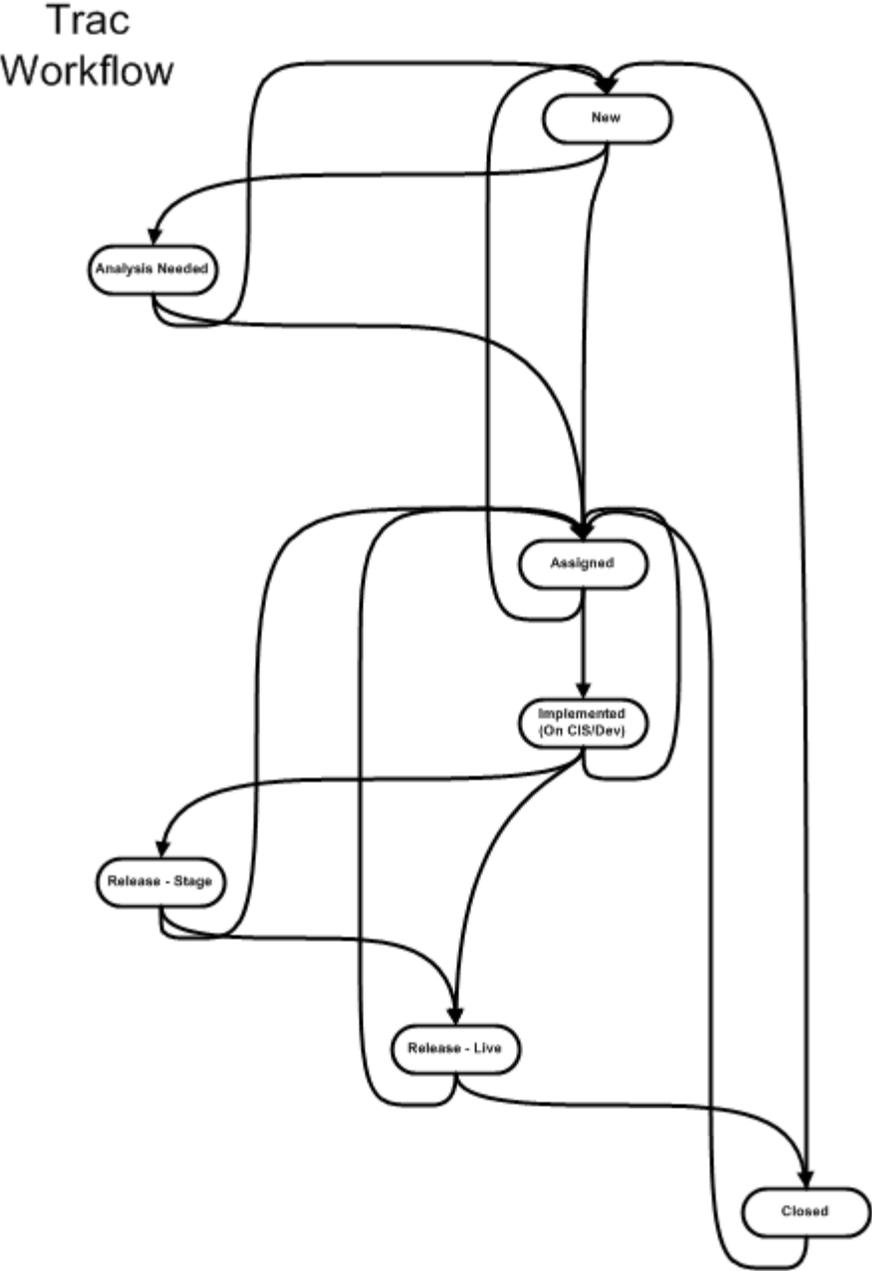
Figure 6: Ticket process workflow diagram after test-driven development framework.

Among the technologies used to implement the test-driven web application development framework, Kohana, PHPUnit, and Phing all worked without any problems. The development team members that interacted with each of those components all agreed that each of these proved to be reliable and consistent throughout the case study. Hudson, the continuous integration server, proved to be a bit more problematic. The engineering staff had to closely monitor the Hudson server processes because of memory leaks that caused instability from time to time. The Hudson development team provided regular and consistent updates to the software which started to help stabilize the majority of the problems that were experienced by the development team.

Additionally, Hudson was agnostic to the reason why a build could fail. While it would provide the details of the failure, making it fairly simple to find the initial cause (such as the execution of the build script, database configuration, or a particular unit test), it could provide inaccurate results with builds due to mistakes made with creating unit tests. This meant that while there were potentially no bugs found in the code of the system in a specific build, an incorrectly written unit test could cause the continuous integration server to consider a build to have failed testing. While this led to some frustration – especially among the developers – overall the team accepted the consensus was that this the correct action for Hudson to take (by nature of how continuous integration works, Hudon had to assume that a unit test was correctly written) because it ensured that not only was the project's code written correctly the automated testing system itself was "self-checking".

When asked to respond to the survey statement "The continuous integration and unit testing systems, along with the design pattern framework were reliable and consistent through the software development lifecycle.", 64% of the seventeen case study participants who

answered the question agreed, 18% neither agreed nor disagreed, 6% disagreed, and 12% felt the

answer was not applicable to their involvement in the study.

The continuous integration and unit testing systems, along with the design pattern framework were reliable and consistent through the software development lifecycle.
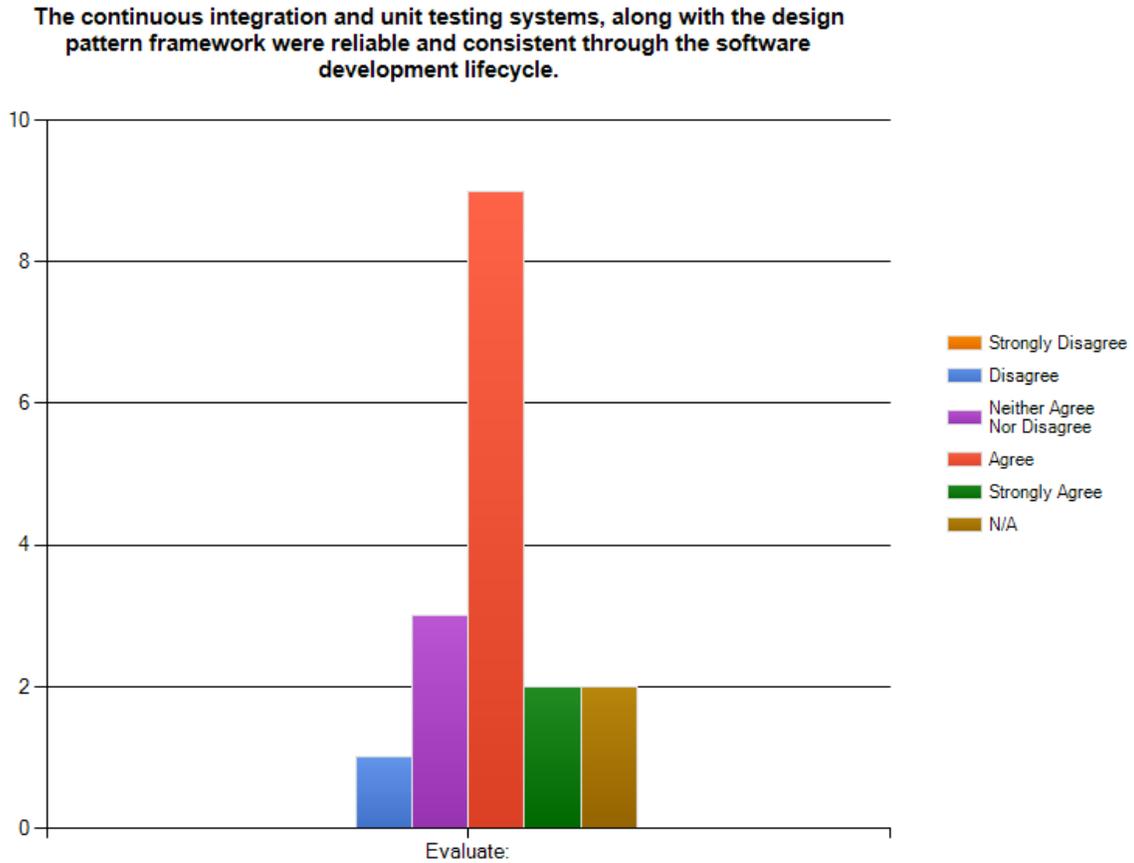
Even after the adoption of the test-driven web application development framework, the

majority of the development team (developers, quality assurance, and management) did not feel

all that strongly about the advantages of reviewing and evaluating the tools and software

methodologies used by development teams that were not using same type of development

environment (programming language, business models, etc.). While the majority did agree that

there were advantages, the overall consensus was fairly indifferent. The engineering staff that

supported the developers were the only members of the team that felt strongly about the

advantages of regularly reviewing the methodologies and tools used by the rest of the

information technology industry. When asked to respond to the survey statement "Development

teams responsible for web applications based on interpreted languages should regularly review

and evaluate the need to incorporate or adapt the software methodologies of other types of

development teams (such as those responsible for desktop applications based on compiled

languages).", 58% of the seventeen case study participants who answered the question agreed,

24% neither agreed nor disagreed, 12% disagreed, and 6% felt the answer was not applicable to

their involvement in the study.

Development teams responsible for web applications based on interpreted languages should regularly review and evaluable the need to incorporate or adapt the software methodologies of other types of development teams (such as those responsible for desktop applications based on compiled languages).
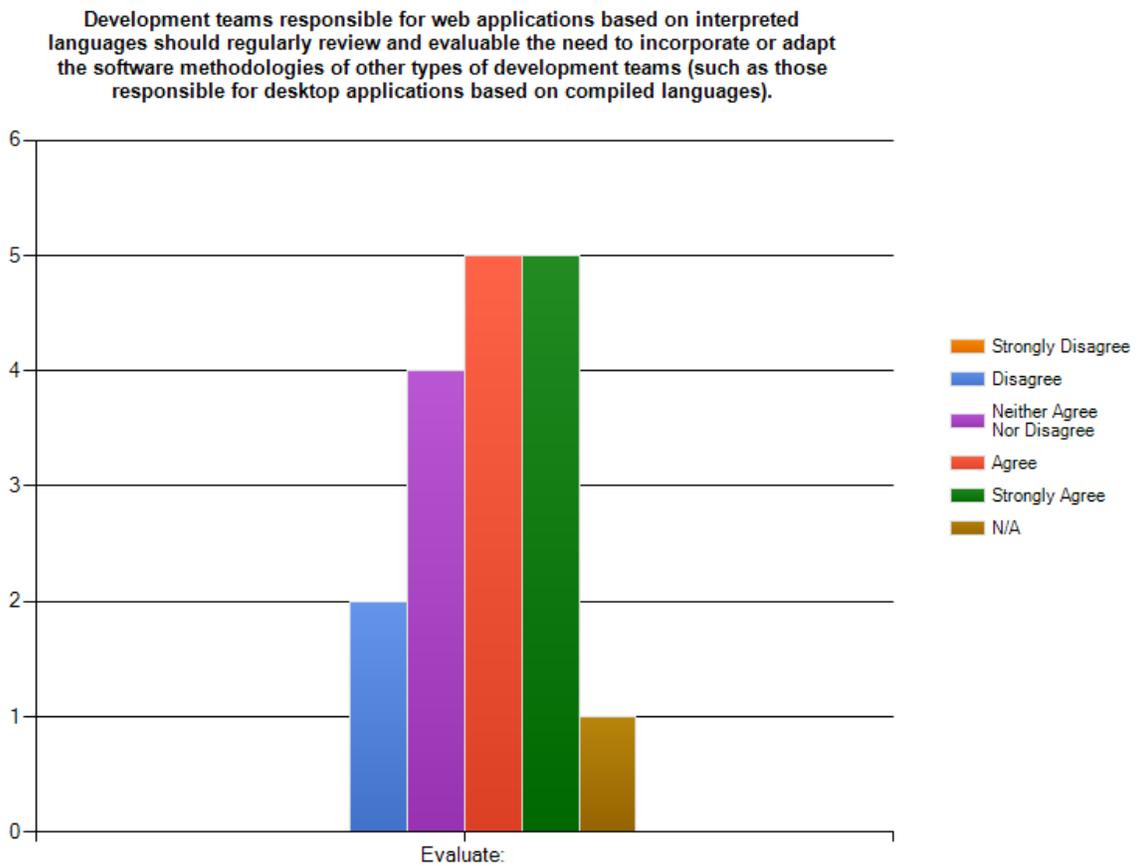
Chapter 5 – Recommendations and Conclusions

Reviewing available literature and research clearly revealed that there were methodologies and tools in use by development teams writing traditional applications using compiled programming languages that reduce risk and increase productivity. The data gained from the evaluation of the available research led to the development of a framework that consisted of four basic components that were considered capable of providing test-driven software development methodologies and tools that support the development of web applications built on interpreted programming languages. Orthogonality, design patterns, continuous integration, and unit testing were all evaluated in order to determine their effectiveness in an environment such as the web application development team chosen for this research case study.

The review of available literature and research convincingly emphasized the value of using orthogonality as a means of reducing risk and increasing productivity in a software development team (Atwood, 2009; Byars, 2008; Hunt & Thomas, 2000; Libbert, 2005; Raymond, 2003; Scott, 2009). 82% of the development team case study participants agreed that writing code orthogonally was an effective tool for providing all members of the team with the capacity to understand the purpose of a unit of code in a project and thus increased programming productivity. Additionally, 88% of the participants agreed that by writing code orthogonally, therefore reducing or eliminating the dependence of any unit of code from any other unit of code, the risks associated with the programming of a particular project were greatly reduced. Along with the use of design patterns, orthogonality was overwhelmingly accepted by all active programming members of the team as a viable tool for web application programming when using interpreted programming languages.

Starting with the words of an architect (Alexander, 1977) followed by the Gang of Four (Gamma et al, 1995), the idea of solution reuse (design patterns) has had widespread success and adoption as both a methodology and available tools among traditional software development teams. Shalloway (2005) emphasized the advantages of using design patterns to not only improve the quality of programming projects but also as a method of improving the skills of development team members. When the Kohana Model-View-Controller PHP framework was used to evaluate the use of design patterns (and orthogonality, by extension of some characteristics of the MVC design pattern), 82% of the development team agreed (and no team members disagreed) that the common vocabulary established through its use for greatly facilitated communication across the entire project. The use of design patterns, much like the application of orthogonal programming, greatly simplified the individual developer's ability to modify code that they had not originally written. 88% of the participants agreed that using design patterns in web application development both increased productivity and reduced risk.

Continuous integration was achieved through the use of the Hudson CI server and Phing as a build tool. While continuous integration is fairly new even in traditional software development, it has grown quickly and the works of Fowler (2006) and Duvall (2007) emphasize the advantages of applying quality control throughout the development process instead of following the traditional methods of deferred integration and quality control. While a majority of the case study participants agreed that by continuously integrating each build of a project they had a higher level of trust in the quality of a project (70%), 18% felt no different about the level of trust they had and 6% disagreed. Only 64% felt that the continuous integration solution implemented for the case study effectively reduced risk and increased productivity. That being said, 82% of the case study participants felt that an effective reporting system based on the

concepts of continuous integration would be beneficial. The tools to apply continuous integration to projects built on interpreted programming languages are still in their infancy, and it is quite clear that a more effective solution for continuous integration needs to be developed.

While continuous integration was not as effective in web application development as orthogonality or the use of design patterns, one of the core components of continuous integration was very successful. One of the core (and most widely adopted) components of test-driven software development is the use of unit tests. Atwood (2006), Duvall (2007); Hunt & Thomas (2000), King (2006), and Wells, (1999) all identify unit testing as one of the most effective methods for ensuring that programming code actually works. PHPUnit as the unit testing medium of choice integrated very well with the Kohana MVC framework. Not a single member of the case study disagreed with the value that unit tests have in reducing risk and increasing productivity. Much like the other components of the test-drive web application development framework, there was a learning curve and some overhead, but 82% of the development team agreed that unit testing was important for the on-time and on-budget delivery of a finished product. Unit tests provided quick and generally accurate feedback on the failure of a project's build and several times prevented the release of bugged code to production systems. With an operational continuous integration solution to provide consistent and accurate feedback on every committed change to a project, unit testing's effectiveness could become even more evident (the engineering team of this case study is currently looking at ways to contribute to the improvements being made to Hudson, along with looking at alternative options to Hudson and even considering developing a custom continuous integration solution) – as mentioned earlier, 82% of the development team participants agreed that a reporting system that regularly reported back the success of building and testing a project against its unit tests would be beneficial.

One side effect of this research for almost 60% of the members of the case study (of which the researcher was both an observer and participant) is the value that a periodic review of what other development teams of different environments are using as both methodologies and tools instead of just what is being used in identical development environments was made clear. The research and evaluation of a test-driven development framework built upon the methodologies and tools that software development teams using compiled programming languages led to a simplified development process and a more responsive development team.

In an industry where turnover, attrition, growth, and rapidly changing priorities all can lead to common situations where developers often are responsible for modifying code they originally did not write, the use of a shared common vocabulary, independently written units of code, and the automated regular execution and testing of project builds with detailed reporting provides for a much more responsive development environment. It is, of course, important that all members of any development team making use of this development framework (from management and quality control to the systems and software engineers) to each have an understanding of each of the components and the benefits of their implementation. The test-driven web application development framework defined in this research brings the advantages of reducing risk while increasing both productivity and quality to software engineering teams responsible for the development of web applications using interpreted programming languages.

Chapter 6 – Areas for Further Research

Additional research that could complement, or even supplement, the research done in this this thesis includes the investigation of adding additional case studies to make the data collected and analyzed more compelling and therefore the research more robust (especially the addition of case studies using alternative interpreted programming languages to PHP). Other areas that could be researched include the inclusion of Acceptance Testing as an additional stage of the test-driven development framework detailed in this research. Variations include User Acceptance testing and Operational Acceptance testing. Software such as the Selenium server and client would be an ideal starting place for investigating the usefulness of Acceptance Testing in this web application development framework. Additionally, applying more collaborative-based version control systems such as Distributed Version Control System (DVCS) instead of using the Centralized Version Control System (Subversion) that was used in the course of this research could provide a more efficient process for web development when used by teams that have remote members. Lastly, applying Exception-Driven Development methodologies to the development of web applications would be a logical follow-up research to this study in the application of test-driven development methodologies in web application development.

References

Alexander, C., Ishikawa, S., and Silverstein, M. (1977). A Pattern Language. New York:

    Oxford University Press.

Atwood, Jeff. (2009). *All Programming is Web Programming.* Coding Horror: Programming and

    Human Factors. Retrieved August 14, 2009 from

    http://www.codinghorror.com/blog/archives/001296.html.

Byars, Brandon (2008). Orthogonality. A Day in the Lyf. Retrieved February 25, 2010 from

    http://brandonbyars.com/blog/articles/2008/07/21/orthogonality.

Duvall, P., Matya, S., & Glover, A. (2007). *Continuous Integration: Improving Software Quality*

    and Reducing Risk. Boston, MA: Pearson Education, Inc.

Fowler, Martin. (2006). *Continuous Integration*. Martin Fowler. Retrieved February 20, 2010

    from http://www.martinfowler.com/articles/continuousIntegration.html.

Fowler, Martin (1999). Refactoring : Improving the Design of Existing Code. Upper Saddle

    River, NJ: Addison-Wesley.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). Design Patterns: Elements of

    Reusable Object-Oriented Software. Indianapolis, IN: Addison-Wesley.

Hunt, Andrew and Thomas, David (2000). The Pragmatic Programmer: From Journeyman to

    Master. Boston, MA: Addison-Wesley.

King, Timothy (2006). Twelve Benefits of Writing Unit Tests First. J. Timothy King's Blog.

    Retrieved March 2, 2010 from http://blog.jtimothyking.com/2006/07/11/twelve-benefits-

    of-writing-unit-tests-first.

Lee, Seok Won and Rine, David C. (2004). *Case Study Methodology Designed Research in*

    Software Engineering Methodology Validation. 16th International Conference on

Software Engineering and Knowledge Engineering. SEKE. Retrieved June 25, 2010

from

http://citeseerx.ist.psu.edu/viewdoc/download&doi&10.1.1.96.7524_rep&rep1_type&pd

f&rct=j&q=case study methodology designed research in software engineering

methodologyvalidation&ei=fTstTIh4wYWdB7aj1PQC&usg=AFQjCNF0N7_nX9mz7_

eVg1ah_mB8XcUBpw&sig2=whsEYMZcxadFOTe92tVzhw.

Leedy, P. & Ormrod, J. (2005). *Practical Research: Planning and Design*. Upper Saddle River,

NJ: Pearson Education, Inc.

Lieberherr, Karl (1997). Law of Demeter. Retrieved February 25, 2010 from

http://www.ccs.neu.edu/home/lieber/LoD.html.

Lippert, Eric (2005). Five-Dollar Words for Programmers, Part Two: Orthogonal. Fabulous

Adventures In Coding. Retrieved February 25, 2010 from

http://blogs.msdn.com/ericlippert/archive/2005/10/28/483905.aspx.

McConnell, Steve. (1996). *Rapid Development: Taming Wild Software Schedules*. Redmond, WA:

Microsoft Press.

MSDN (2010). Visual Studio: Unit Testing. Microsoft Corporation Retrieved March 2, 2010

from http://msdn.microsoft.com/en-us/library/aa292197%28VS.71%29.aspx.

O'Reilly, Tim. (2005). *What is Web 2.0: Design Patterns and Business Models for the Next*

Generation of Software. O'Reilly Media, Inc. Retrieved August 1, 2009 from

http://oreilly.com/pub/a/web2/archive/what-is-web-20.html?page=1.

Perry, D., Sim, S., Easterbrook, S (2004). *Case Studies for Software Engineers*. 26th

International Conference on Software Engineering. IEEE. Retrieved June 25, 2010

from http://www.springerlink.com/content/p015327373105661/.

Raymond, Eric (2003). Compactness and Orthogonality. The Art of Unix Programming.

    Retrieved February 25, 2010 from http://www.faqs.org/docs/artu/index.html.

Runeson, Per and Host, Martin (2009). *Guidelines for conducting and reporting case study*

    research in software engineering. Springer. Retrieved June 25, 2010 from

    http://www.springerlink.com/content/t22r8l65q7h31636/.

Shalloway, Alan and Trott, James (2005). Design Patterns Explained: A New Perspective on

    Object-Oriented Design. Boston, MA: Pearson Education.

Scott, Michael (2009). Programming Language Pragmatics, Third Edition. Burlington, MA:

    Elsevier, Inc.

Vines, Donald. (2008). *Test-driven Development in an SOA Environment*. IBM. Retrieved July

    20, 2009 from

    http://www.ibm.com/developerworks/websphere/techjournal/0812_vines/0812_vines.html.

Wells, Don (1999). Unit Tests. Extreme Programming. Retrieved March 2, 2010 from

    http://www.extremeprogramming.org/rules/unittests.html.

Yin, Robert K. (2009). Case Study Research Design and Methods, Fourth Edition. Thousand

    Oaks, CA: Sage Publications, Inc.

Appendix A

Test-Driven Web Application Development   <u>Exit this survey</u>

1. Open-Ended Questions

Please answer the following 4 open-ended questions listed below.

✱1. How does the lack of standardized test-driven methodologies in the web application industry affect your ability to develop
on-time and on-budget web applications with a high degree of quality?

┌─────────────────────────┐
│   │
│   │
│   │
│   │
│   ∴∴ │
└─────────────────────────┘

✱2. How would you describe the environment (specifically in regards to quality control, communication, and development tools) of
the development team BEFORE the implementation of the full test-driven web application development framework? Why?

┌─────────────────────────┐
│   │
│   │
│   │
│   │
│   ∴∴ │
└─────────────────────────┘

✱3. How would you describe the environment (specifically in regards to quality control, communication, and development tools) of
the development team AFTER the implementation of the full test-driven web application development framework? Why?

┌─────────────────────────┐
│   │
│   │
│   │
│   │
│   ∴∴ │
└─────────────────────────┘

✱4. What components of the test-driven web application development framework currently in use by the development team have
you found to be the most effective? Least effective?

┌─────────────────────────┐
│   │
│   │
│   │
│   │
│   ∴∴ │
└─────────────────────────┘

[Next]

Test-Driven Web Application Development

2. Likert-Scale Statements

Evaluate and respond to the following ten statements:

✱1. Orthogonal design and programming is important for the on-time and on-budget delivery of quality web applications based on interpreted programming languages.

	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree	Not Applicable
Evaluate:	○	○	○	○	○	○

✱2. Using Design Patterns and the common language for components (an example of this consistent vocabulary in general would be the primary terms of the MVC design pattern where "model" represents accessing/storing data, "controller" represents handling input and processing data, and "view" represents rendering the data and input/output into a user interface) is important for increasing the effectiveness of communication across all members (from QC to software analyst to developer) of a development team.

	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree	N/A
Evaluate:	○	○	○	○	○	○

✱3. Continuous integration systems like Hudson are important for the on-time and on-budget delivery of quality web applications based on interpreted programming languages.

	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree	N/A
Evaluate:	○	○	○	○	○	○

✱4. Unit testing systems like PHPUnit are important for the on-time and on-budget delivery of quality web applications based on interpreted programming languages.

	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree	N/A
Evaluate:	○	○	○	○	○	○

✱5. Consistent use of programming code units and components that are written orthogonally (independent of changes made to other code units and components) is important for providing all members of the development team the capability to read and understand the purpose of a function or component.

	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree	N/A
Evaluate:	○	○	○	○	○	○

✱6. Consistent use of well-established solutions involving consistent vocabulary to existing problems in the form of design patterns and is important for increasing efficiency and communication among members of web development teams.

	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree	N/A
Evaluate:	○	○	○	○	○	○

✱7. Consistent feedback in the form of unit testing and continuous integration is important for building trust in both the programming code and development team members.

	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree	N/A
Evaluate:	○	○	○	○	○	○

✱8. The continuous integration and unit testing systems, along with the design pattern framework were reliable and consistent through the software development lifecycle.

	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree	N/A
Evaluate:	○	○	○	○	○	○

✱9. Development teams responsible for web applications based on interpreted programming languages would benefit from the implementation of a reporting system based on providing feedback from tests run on each committed change in code of a particular application.

	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree	N/A
Evaluate:	○	○	○	○	○	○

✱10. Development teams responsible for web applications based on interpreted languages should regularly review and evaluable the need to incorporate or adapt the software methodologies of other types of development teams (such as those responsible for desktop applications based on compiled languages).

	Strongly Disagree	Disagree	Neither Agree Nor Disagree	Agree	Strongly Agree	N/A
Evaluate:	○	○	○	○	○	○

[Prev] [Done]

Appendix B

REGIS UNIVERSITY

Academic Affairs
Academic Grants

3333 Regis Boulevard, H-4
Denver, Colorado 80221-1099

303-458-4206
303-964-3647 FAX
www.regis.edu

IRB – REGIS UNIVERSITY

November 2, 2010

Jason Hall
3000 Colonial Parkway #4203
Cedar Park, TX 78613

RE: IRB #: 149-10

Dear Jason:

Your application to the Regis IRB for your project "Test-Driven Web Application Development," was approved as exempt on November 1, 2010.

The designation of "exempt," means no further IRB review of this project, as it is currently designed, is needed.

If changes are made in the research plan that significantly alter the involvement of human subjects from that which was approved in the named application, the new research plan must be resubmitted to the Regis IRB for approval.

Sincerely,

Edwin May
Director

cc: Charlie Thies