AN ABSTRACT OF THE THESIS OF

Loren Paulsen for the degree of Honors Baccalaureate of Science in Computer Science presented on May 27, 2009. Title: Cortex: Open Source Core Financial Processor.

Abstract approved: ______________________________
Dr. Timothy A. Budd

A financial processor is the most important component of a credit union's IT infrastructure. A database storing member demographic information, account balances, and transaction history, it performs financial calculations, such as interest, dividends, and maturities. It also provides a user interface, allowing tellers and financial service representatives to manage accounts and record transactions. Unfortunately for credit unions, the bargaining power of the financial processor vendors is significant. Commercial financial processors are extremely expensive systems with long-term contracts. Dwindling competition, a lack of innovation, and a substantial barrier to entry characterize the stagnant financial processor market. Our team determined that open source would be an effective development methodology, as well as business model, to develop a new financial processor. Using secure, powerful technologies and frameworks, we were able to design an innovative, modern architecture that meets the business requirements of credit unions better. Our web-based user interface also reflects the internet-driven paradigm with which employees are already comfortable, a dramatic shift from the outdated UI patterns and poor learnability associated with existing financial processors. Our team will continue searching for credit unions interested in forming the community to finish the development and testing necessary for the system to be ready for production use.

Key Words: credit unions, financial processor, core processor, open source
Corresponding e-mail address: loren.paulsen@lifetime.oregonstate.edu

Cortex: Open Source Core Financial Processor

by

Loren Paulsen

A PROJECT

submitted to

Oregon State University

University Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Computer Science (Honors Scholar)

Presented May 27, 2009
Commencement June 2009

Honors Baccalaureate of Science in Computer Science project of Loren Paulsen

presented on May 27, 2009.

APPROVED:

____

Mentor, representing Computer Science

____

Committee Member, representing Computer Science

____

Committee Member, representing Financial Services

____

Chair, School of Electrical Engineering and Computer Science

____

Dean, University Honors College

I understand that my project will become part of the permanent collection of Oregon State University, University Honors College. My signature below authorizes release of my project to any reader upon request.

____

Loren Paulsen, Author

Acknowledgements

I would like to thank the following individuals and groups who have helped make this project a reality:

Dr. Timothy Budd for being my thesis mentor, and for providing expertise, support, and direction throughout the course of this yearlong project.

Dr. Mike Bailey, one of my committee members, for overseeing our senior capstone experience, and helping us do great at the engineering expo.

Robbie Wright, my other committee member and co-worker, for his constant encouragement and for being an invaluable resource for all of the business requirements related to this project.

Loren & Nancy Paulsen, my parents, for encouraging me and supporting me in all aspects of my studies and in my life through all my years.

Bryant Brownell, Stephen Perona, and Weiping He, my tireless team members and friends, for working with me all year developing the system, and supporting all the work I (we) have done so far – and may continue to do in the future.

Brittney Paulsen, my wife, for supporting me through my college experience and beyond.

MaPS Credit Union, for my employment, and its employees for working with me on gathering requirements and use cases for the system.

All of the professors, graduate students, and administrators in the Computer Science department who made my education possible and worthwhile.

The University Honors College for offering me many unique and rewarding learning experiences, and giving me the opportunity to pursue this project further.

Table of Contents

LIST OF FIGURES

INTRODUCTION

A core financial processor is a financial institution's most important technological system. It stores information about customers and keeps track of balances for different types of accounts, including checking, savings, certificates, mortgages, and loans. It performs numerous calculations, such as interest, dividend, and maturity calculations. It also provides a user interface, allowing tellers and financial service representatives to manage accounts and record transactions.

Commercial financial processors are extremely expensive systems. A contributing factor to this problem is that financial processing is a stagnant market segment, characterized by a lack of innovation and dwindling competition. This leads to substantial problems in terms of modernization, flexibility, and interoperability, which burden financial institutions. Modern features, such as online banking, are often third party add-ons to these financial processors, further increasing the expense of running a financial institution. Furthermore, financial processor vendors often restrict the ability to add third party modules, and charge substantial fees to open interfaces to their systems.

When I entered the senior engineering capstone sequence, I proposed building a prototype open source financial processor, meaning that it is free to operate and extend. The prototype is codenamed "Cortex". Today there is no open source core financial processor available to financial institutions. The efficiency that a financial processor built on modern technologies would bring, as well as the explosion of the open source movement in general, have created a great opportunity and need for this project.

Team Formation and Motivation

Members of our team have worked for MaPS Credit Union, a local credit union. While there, these team members discovered firsthand the inefficiency of existing financial processors and the significant costs, both direct and indirect, that they bring to the credit union. While the largest financial institutions develop their own financial processors internally, small to mid-size financial institutions purchase existing systems. Our team came to discover that the majority of small to mid-size credit unions are dissatisfied with their financial processors and are openly calling for alternatives. Unfortunately, this large market opportunity is counterbalanced by a substantial barrier to entry, largely due to the complexity of financial processors.

OSU’s senior engineering capstone provided us with a great opportunity to start analyzing the problem in depth. In the capstone sequence, students form teams to work on a real-world problem yearlong. We decided to build a prototype financial processor, which would grow into an open source project by the end of the year. The system will require a great deal of resources to transform into a fully functional financial processor. A large corporation may have these kinds of resources, but our team does not. The open source methodology overcomes this hurdle, by having a large number of volunteers finish the processor for the common good. The strong open source community at OSU has also provided us with direction, in particular, how a large open source project may be structured, and how an effective business model may incorporate an open source strategy.

The Credit Union Difference

Credit unions are not-for-profit, cooperative financial institutions. Every customer of a credit union is both a member and an owner, each having equal ownership and one vote, regardless of how much money that member has on deposit (CUNA 2). Credit unions use their excess earnings to offer members higher rates of return on their savings, more affordable loans, and lower fees on products and services (WOCCU 3). Leveraging their already strong member connections, credit unions differentiate themselves by offering "member-centric" services, as they are referred to in the industry. Member-centric credit unions are characterized by superior service and member experience (Mertka 2).

Member-centric services are a credit union's greatest competitive advantage against banks as well as other credit unions. Doug True, president of FORUM Solutions, a credit union service organization of FORUM Credit Union says, "One of the advantages credit unions have over regional banks is they listen to their members. If [credit unions] can't offer what [members] are requesting, they're at a disadvantage" (Urrico 45). Member-centric services require a credit union to be agile, responsive, and open to feedback. To deploy these services successfully also requires an effective, flexible financial processor.

Beyond achieving a member-centric focus, credit unions must consider competitive forces that drive all financial institutions. Christine Barry, research director at Aite Group, says that to remain competitive, must operate efficiently—easily and quickly complete transactions, launch new products, and access member information

(Urrico 45). These types of operating efficiency have a direct influence on pricing and member satisfaction, and all involve interactions with a financial processor.

State of the Industry

Aging core processors are abundant throughout the credit union industry, and "A lot of credit unions in the next three to five years will have to make a decision… and they'll have limited choices" (Urrico 46). Credit unions will have limited choices, because, according to Urrico, "What once was a wide-open core processing field has dwindled through mergers and acquisitions" (Urrico 46). This consolidation among vendors has created many problems for credit unions including "less competition, reduced price leverage, forced conversions from phased-out products, and overburdened conversion teams" (Urrico 46). The vendors that remain have grown through consolidation rather than organic growth, and have formed an oligopoly that is harmful to credit unions. Sabeh Samaha, president of Samaha & Associates, says, "Having a lot of choices is a sign of prosperity, and we had that from the mid-1980s to 2000… That was nice for credit unions. Consolidation isn't favorable. I feel somewhat threatened by it" (Urrico 46). As a result, aggressive tactics, such long-term contracts and early termination fees, which lead to significant vendor lock-in, have become the norm. Financial processors are also becoming increasingly cost prohibitive for small credit unions. Mergers and acquisitions also create difficulty for credit unions that wish to avoid vendors they have had bad experiences with in the past. Rob Guilford, executive vice president at Wescom, says, "A lot of credit unions we talk to find out their favorite vendor has been purchased by their least favorite vendor" (Urrico 47). The few remaining vendors often purchase smaller vendors, then, as a result of market forces, have no strong incentive to continue developing the systems they have purchased. This tendency has lead to the stagnant market that is now observed by credit unions.

Monopoly on the Horizon

In 1987, there were 24 major financial processor vendors (Moran 11). Today, 75% of the market share is controlled by the top six vendors, and 55% of the market is controlled by the top three (Moran 11). With more than 2.5x the market share of its nearest competitor, and a 40% win rate on new core processing deals (2.5x the rate of its nearest competitor), Fiserv Inc. is the dominant player in the market (Moran 10). In an industry with renewal rates consistently above 90%, Fiserv's dominance in the market is set to continue growing.

As discussed earlier, Fiserv's growth has been "largely accomplished through the acquisition of entities engaged in businesses which are complementary to its operations" (Fiserv Inc. 53). Since Fiserv's formation in 1984, it has expanded its operations through over 60 acquisitions (Fiserv Inc. 55). Currently, Fiserv "plays the role of steward for the many companies it has acquired over time – allowing these companies to continue much as they were as independent firms" (Capachin and Schenck 2). However, it is not helpful to credit unions for a single vendor to own many different financial processor lines, none of which as aggressively developed, or marketed, as processor developed by an independent vendor would be. Fiserv gains little by having its subsidiaries steal market share from each other.

Moran says, "once a contract has been added, the client relationship tends to be fairly sticky and pricing is reasonably inelastic… leading to a revenue stream that is highly recurring and reasonably predictable" (Moran 8). Moran, who was actually analyzing Fiserv as a potential investment and recommending the purchase of Fiserv stock, recognizes the power Fiserv has over financial institutions in the industry. Moran goes on to say:

> As industry dynamics continue to shift toward an oligopolistic structure, it seems highly likely that smaller competitors will either exit the business or lose clients to larger competitors due to service, product capabilities/breadth, pricing, or some combination thereof. Moreover, consolidating competition has intensified barriers to entry and should lead to higher marginal returns on incremental business for the companies that remain (Moran 11).

Moran's independent financial analysis of Fiserv, which reflects the statements made by credit unions, yet comes from an entirely different perspective, shows that credit unions are not overstating the power and influence vendors have in the industry. Moran's analysis implies that Fiserv's competitive position is unusually strong, not just in the financial services industry, but among all publically traded companies.

Fiserv's clients have tended to be "small and mid sized institutions that had limited internal technology groups and contracted for a fairly wide range of its services" (Moran 14). Because most credit unions fall into this category, they are particularly exposed to the issues that arise from Fiserv's growing dominance. Large banks frequently have large enough development teams to develop their own financial processors, and the top 25 banks only buy a handful of Fiserv's services. (Moran 14).

Fiserv's recent acquisitions will soon change that. Acquiring CheckFree, for $4.4 billion in 2008, was an aggressive move into a higher growth business that will affect banks as well. CheckFree is the leading electronic bill payment and online banking provider, and already has 3.5x the market share of its nearest competitor (Moran 13). Like Fiserv’s advantage in the core processor market, CheckFree benefits from high switching costs, and has a greater than 90% renewal rate as well. (Moran 13). Once CheckFree's services are fully integrated with Fiserv's products, we will likely see many more Fiserv products in banks, as an extension of CheckFree's dominance.

The Open Source Advantage

An open source financial processor solves many of the issues discussed above. Most importantly, credit unions have the option of using our system at no cost. Third party customizations are inherently possible, as the system is completely open. The institution creating the data also has full access to their own data, meaning they can use it in any reporting scenario. The issues regarding lack of innovation and necessary features are also handled well by open source development. Development and changes to the database and UI can be made by those who use the system the most. Open source projects allow for personal motivation and needs to drive innovation, rather than market forces.

As a value added reseller (VAR), our company would earn income through support and customization. Our company’s contracts with individual credit unions would be optional and based on the needs of the individual institutions we were working with. In contrast to today's adversarial relationships between financial processor vendors and credit unions, our relationships with credit unions would be supportive. An open source financial processor will also be under heavy pressure to make changeover paths easy for credit unions. This would lead to easier migration paths, and fewer associated costs. Open source projects provide for and make possible constant and organic growth from the bottom up, which bodes well for long-term growth and development.

REQUIREMENTS

Reasons to Replace a Core System

John Best, chief technology officer at Wescom Credit Union says, “Members expect a seamless view in all channels”, but “Unfortunately, what’s happening is that the legacy systems they’re trying to build around don’t necessarily work together” (Urrico 45). Credit unions overall are having difficulty integrating new services, particularly web-based and mobile services, with their aging core systems. Additionally, Barry says, “A primary driver for converting to a new core system is that existing systems become outdated, inefficient, or too expensive to operate” (Urrico 44). The user interfaces for these systems, in particular, often demonstrate outdated, inefficient patterns, which are unfamiliar to new employees, resulting in substantial learning curves, lengthy employee training programs, and increased costs.

Despite the potential benefits, there are reasons credit unions tend to resist upgrading their financial processors. In fact, Meyer says, “There seems to be a lot more emphasis on maintaining the current architecture” (Urrico 48). This inertia “creates the danger of getting caught in the legacy system trap and falling behind the industry at a critical time” (Urrico 48). Core processor changes also increase the risk of interruptions and service issues (Moran 8). Moran says credit unions are "unlikely to leave due to a modestly cheaper or slightly superior product" (Moran 8). Because of the costs and risk associated with switching core processors, this is unsurprising.

Additionally, many credit unions ultimately purchase an unsatisfactory core system. Samaha says, “The biggest mistake is that [credit unions’] search and selection criteria aren’t accurate. Some will [convert] for the wrong reasons… They don’t do proper due diligence. They see the demo and buy the product” (Urrico 45-6). Because financial processor migrations are so inherently costly and risky, a rational approach is necessary. A report by the Aite Group outlines seven key drivers in replacing financial processors: phased-out vendor technologies, outdated technologies, greater focus on business banking, new member-centric strategies, cost reduction, organic growth, and better risk management and compliance (Urrico 47). In Cortex, we focused on three of these seven drivers. Addressing the "outdated technologies" driver was one of our most important architectural strategies. Many existing processors were built on technologies that are now either obsolete or obscure. In contrast, we are building our financial processor on modern, innovative technologies, including the .NET Framework 3.5 and SQL Server 2008. This architecture is discussed in depth later.

Enabling "new member-centric strategies" was another key design goal. Many existing financial processors, rooted in decades-old business requirements and processes, do not include modern features, such as online banking. These features are often third party products. Any new financial processor would naturally be designed with online banking in mind. To be developed efficiently, other emerging features, such as mobile banking and RSS feeds, will require substantially more flexible, open architectures than are currently available, particularly for solutions that are developed in-house.

Lastly, "cost reduction" is addressed primarily through our open source development methodology. Existing financial processors are often million dollar systems, and lock credit unions into 7-15 year contracts with their vendors. Offering an open source financial processor allows credit unions to begin testing our system for free, and begin paying for a support contract when they are ready to bring the new system into production. A credit union with sufficient technical skills may be able to avoid the need for a support contract altogether. This leads to considerably reduced cost and risk associated with migrating to our system compared to an existing alternative.

Lack of Openness

Many core processor vendors claim to offer "open" platforms, but this term is often misleading. Guilford says, “It may have an Oracle database, for instance, but if you can’t touch it, it’s not open… Some systems purport to be open, but they’re closed” (Urrico 48). When vendors use the term "open", it usually does not mean open source, fully open, or even freely open, resulting in limited ability to query data, integrate third party products, or directly connect to the database.

The actual application programming interface (API) for a financial processor is often locked down, requiring a substantial per-application fee to utilize it. Additionally, documentation for these APIs is not usually freely available. This burden affects a credit union's ability to develop custom applications, and even install third-party extensions. As a result, providing modern features not provided with the financial processor, such as online, mobile, and/or telephone banking, can be an expensive, complex endeavor.

Basic Definitions and Operations

The first step in the requirements elicitation process was gaining a good understanding of the common terminology in the industry. As this was established, I worked closely with several key employees from MaPS Credit Union to build a detailed understanding of how a core processing system is used. The following is a high-level overview of credit union operations.

A branch is a physical place of business, and tellers and Financial Service Representatives (FSRs) are the primary users of the system. These employees have network credentials (usernames and passwords), and are associated with branch where they are working. FSRs are generally responsible for creating and managing members' accounts. Tellers generally process transactions, such as deposits and withdrawals. Back-office, administrative staff, are responsible for creating new promotional account types such as new checking account with new interest rate. They are also focused on reporting, financial calculations, and other batch processes performed by the financial processor.

Products represent different tiers or classes of accounts offered by the credit union. Products include multiple types of checking and savings accounts. These products have offer interest rates, require different minimum balances, and have different minimum opening balances. Products are also referred to as "class codes".

A person refers to an individual within our system. For each person we must store his or her name, social security number, email address, physical address, and phone number. Our application must be able to provide this information to FSRs or tellers when they are working with members. This information is also collected to meet strict federal regulations regarding account-opening practices.

Memberships are the most important entities in the system. Memberships have an internal id, a "membership number" that is seen by the member and is printed on checks and account statements, a primary owner (a person), and potentially has many joint owners. Rarely, the display number will need to change. One example of this would be cases of identity theft. An important reason to separate memberships from people is the fact that can be the primary owner of one membership, and a joint owner on potentially many accounts. Being able to link these membership associations to the same person entity is beneficially organizationally.

Accounts are the other most important entities in the system. A single membership is comprised of at least one account, and potentially many accounts. An account is associated with a particular product, but an individual account can have a net interest rate set a certain amount above the base interest rate associated with that product. Account numbers are only unique within a particular membership. Accounts are then referred two by a two-part identifier consisting of membership number and an account number, such as 51809-10. The terminology regarding memberships and accounts differs throughout the industry. In some instances, memberships are referred to as an "accounts" and accounts are simply referred to as "suffixes". Fundamentally, a membership represents a main savings account called a "prime share", which also represents the members' vote, and other accounts, which are conceptually referred to as "sub-shares".

Our prototype will need to support basic transactions, primarily deposits and withdrawals. These transactions have a transaction amount and a transaction type (such as deposit, withdrawal, or payment). Because they are a historical record, they keep historical information about the account at the time of the transaction. This includes the transaction was performed at, the active product on the account, the membership number was at the time of the transaction, and the new total account balance after the transaction is complete. These records are not typically edited or removed, and corrections to accounts are recorded as separate transactions. Transactions can also occur electronically in a number of different ways. Debit card transactions create transactions in the financial processor. Automated payments such as ACH, and transfers and payments made in online banking also create transaction records within the financial processor. Because our initial scope did not include integration with these systems, their impact is not discussed in further detail.

ER Diagram

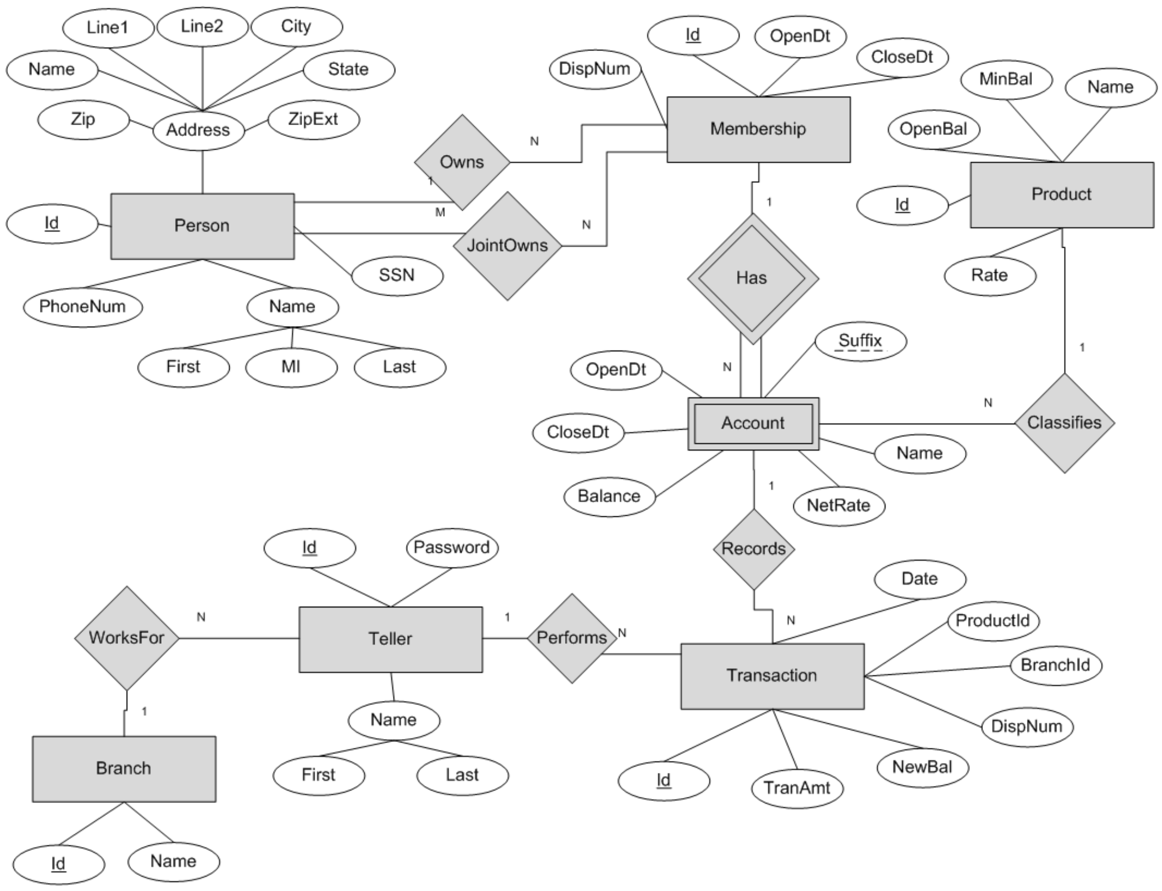


Figure 1: Entity Relationship Diagram

Working with MaPS Credit Union, we created an Entity Relationship (ER) diagram to represent the requirements gathered for our initial scope. ER diagrams are useful in the requirements elicitation process because they provide high-level abstractions of databases, while still representing business requirements well. Figure 1 shows this initial diagram. We found it helpful in clarifying the relationships between entities in the system. For example, the fact that a membership has a single owner, but can have potentially many joint owners, can be clearly seen in the diagram. After this diagram was completed, and the requirements elicitation process was initially complete, we were ready to begin designing our application architecture.

ARCHITECTURE

The first step in designing the system architecture was to survey high-level architectural failures observable in existing systems. The most easily observable problem was the relative obscurity of the platforms hosting financial processors. These platforms, such as the OpenVMS operating system running on DEC Alpha servers, or IBM AS/400 mainframes, may have been optimal historically, but cause problems for credit unions today. Skills and training on these systems are not nearly as common as they are for Windows or Linux.

Authentication to the financial processor was another particular weakness we observed. Often, employees have network credentials, provided by a Windows Active Directory service, which they use to log onto their computers, but use separate credentials to log onto the financial processor. From an IT security standpoint, this can be suboptimal because this leaves two separate lists of credentials to maintain. Often managed less rigorously than Active Directory credentials, financial processor credentials can be a security problem for credit unions. Inefficiencies can also develop as it becomes difficult to synchronize user accounts between, or even determine which pair of accounts represents a single employee.

We designed our system utilizing the powerful, secure Microsoft .NET Framework, which runs on the Microsoft Windows operating system. In most credit unions, Windows is common on desktops as well as servers, so our system will easily integrate with existing IT infrastructures. Windows Integrated Authentication also securely, automatically logs users into the financial processor using their Active Directory credentials.

Microsoft .NET Framework

The Microsoft .NET Framework has two main components: the common language runtime (CLR) and the .NET Framework class library. The CLR is architecturally similar to Java, in that code is compiled into an intermediate language (bytecode) known as the Common Intermediate Language (CIL), which is executed on a virtual machine. Code that targets the CLR (or any virtual machine) is referred to as managed code, while code that does not target the runtime is referred to as unmanaged code. Interestingly, .NET inherently a programming language-neutral and multiple programming languages can be used together to develop CLR applications. Also similar to Java, .NET is type-safe, uses references instead of pointers, and provides automatic garbage collection. Finally, the class library provides an extensive set of reusable types that contribute to rapid application development (Microsoft Corporation 3).

C#

The C# language has many innovative features that substantially increased our team's productivity. One of the most important features is Language Integrated Queries (LINQ). LINQ provides both declarative and functional query syntaxes that can perform SQL-like queries against any collection type (Box and Hejlsberg). Figure 2 demonstrates a simple C# program and a declarative LINQ query.

```
void Main()
{
   string[] teamMembers = {"Loren", "Stephen", "Bryant", "Weiping"};

   var query = from s in teamMembers
                     where s.Length > 5
                     orderby s
                     select s.ToUpper();

   foreach (string name in query)
         Console.WriteLine(name);
}

/* output:
BRYANT
STEPHEN
WEIPING
*/
```

Figure 2: Basic Declarative LINQ Query

LINQ queries written in the functional style, also referred to as method-based queries, build on C#'s support for Lambda Expressions and Expression Trees. As LINQ queries are particularly well suited for the filtering, projection, and key extraction functional patterns, these queries are concise. The query from Figure 2, written as a method-based query, is demonstrated in Figure 3.

```
var query = teamMembers.Where(s => s.Length > 5)
                       .OrderBy(s => s)
                       .Select(s => s.ToUpper());
```

Figure 3: Method-Based LINQ Query

Finally, extension methods are another important piece of the query architecture, and an interesting language feature in general. These methods "combine the flexibility of 'duck typing' made popular in dynamic languages with the performance and compile-time validation of statically-typed languages" (Box and Hejlsberg). Extension methods are becoming increasingly prominent in .NET programs, as they are a convenient way to add functionality to existing classes, without modifying the code of the original classes, or sub-classing them.

ADO.NET Entity Framework

The ADO.NET Entity Framework (EF), first released in August 2008, is major advancement in data access on the .NET platform. On the surface, the EF is an object relational mapping (ORM), which developers to interact with data in terms of strongly typed objects, rather than through standard SQL queries. EF services are based on the Entity Data Model (EDM), which is a medium for defining data models (Flasko 23). The EF is inherently provider-independent, meaning that EDMs can be build for SQL Server, Oracle, MySQL, and other leading databases. A key advantage of the EF is LINQ to Entities, which enables developers to query their database using LINQ instead of SQL.

```
CortexEntities cortex = new CortexEntities();

// get membership account details
var membership =
   cortex.Membership.First(m => m.DisplayNumber == activeMembershipId);
```

Figure 4: LINQ-To-Entities Query

Figure 4 is a LINQ to SQL query. CortexEntities is our EDM. An instance of this class manages the connection to the database, interprets LINQ queries, and sends them to the database. The EDM first translates the LINQ query to Entity SQL, a database neutral SQL language, and finally to the native SQL language of the underlying database. The intermediate Entity SQL is integral to maintaining the database independent query architecture. This query also demonstrates the functional paradigm, by passing the filter as a lambda expression. Finally, the "var" data type declaration utilizes C#'s type inference.

```
CortexEntities cortex = new CortexEntities();

// get membership account details
var suffixes = (from a in cortex.Account
                where a.Membership.DisplayNumber == activeMembershipId
                select a.Suffix);
```

Figure 5: Associations in the EF

The EDM brings many additional benefits to data access. Foreign key relationships in the database can be represented as associations. Associations representing one-to-one (1:1), one-to-many (1:N), many-to-many (N:M) relationships are implicitly understood. One-to-one associations are accessible through standard C# properties, and one-to-many relationships are accessible through C# collection properties. Figure 5 shows how an association can be followed from an account entity to a membership entity, which would a join to accomplish in standard SQL. Additionally, despite the fact that many-to-many relationships require intermediate join-tables to be represented in a SQL database, they are supported directly the in EF. Because this makes join-tables conceptually unnecessary, they are abstracted away and need not be even represented in the EDM as real entities. Finally, multiple tables can be abstracted into a single virtual table, or a single table can be split into multiple virtual tables, and additional annotations are added to the diagram representing additional relationships that cannot be precisely represented in standard SQL, such as table inheritance.

ASP.NET Application Services

ASP.NET provides an extensible set of application services, which are extremely useful in rapid application development. These application services provide a foundation for exception management and logging, user authentication and authorization, session management, user profiles, and personalization.

The membership provider brings built-in support for forms authentication, which is very common on the internet. It also provides secure storage of user credentials, manages the authentication process, and provides session management. This provider also supports Windows authentication, which is different because the user names and passwords are physically stored on domain controllers in Active Directory, rather than in an application's database. This means that even though our application does not store or maintain these credentials, they are used to authenticate to the system, and the system can recognize the individual users.

The profile provider attaches additional fields of information to user accounts, and is used to create and maintain user profiles, whether users log on using forms authentication, or Windows authentication. We used the profile provider to provide a customized user experience for employees. For example, we record all activities, and provide easy access to the most frequently used activities, similar to how the frequently used programs list works in the Windows start menu.

The role provider provides levels of user access to the system. The role provider also supports both forms authentication and windows authentication. In conjunction with Windows authentication, the role provider can pull the groups a network user is a member of, for example “domain admins”, “marketing staff”, or “users”. This functionality provides a great base for user permissions in our system, while meeting the needs of the IT department by allowing these permissions to be centrally managed in their existing Active Directories.

The health monitoring provider is an essential part of our exception management and monitoring architecture. Exceptions are automatically logged, and the user has the option of sending these logs to the Windows application log, flat log files, a logging database, or to have error reports automatically emailed.

Windows Communication Foundation

Windows Communication Foundation (WCF) encompasses a broad set of messaging technologies, ranging from web services to inter-process communication. It represents the unification of many previous distributed computing technologies including Enterprise Services, Net Peer, .NET Remoting, COM+, MSMQ, and ASMX (Microsoft Corporation). WCF also provides a layer of abstraction, managing the bindings between methods and communication technologies.

Most relevant to our project is the fact that WCF is an extensive, flexible base for a service layer. Because web services are an industry-wide standard, our service layer will be accessible to cross-vendor platforms, such as Java-based J2EE applications. Other bindings, such as JSON and REST enable our web service to be called by PHP, Ruby on Rails, or directly from JavaScript. Our service layer hosts the API of the application. Some functionality, such as statement generation, is implemented as an add-in, targeting our API. Developing key functionalities on top of our own API will help promote the design of an effective interface, and maximize interoperability between our platform and third party systems.

Kerberos

Windows Integrated Authentication provides authentication for Cortex. It utilizes the same Kerberos subsystem as Active Directory logons. Because Windows Integrated Authentication is a feature of the Microsoft Internet Information Services (IIS) web server and part of Windows Server, we were not required to perform any development work to support it, other than enabling it in our application's configuration. We designed the architecture assuming that tellers and FSR's would be on the same intranet as the financial processor itself, which would correspond with a simple authentication scenario, but remote scenarios utilizing Kerberos delegation will also be possible.

The basic Kerberos authentication scenario involves a direct connection to an Active Directory domain controller. The client first requests a service ticket from the KDC, which is hosted on the domain controller. When the KDC returns the service ticket, the client generates the authenticator, which is sent, along with the service ticket, in an HTTP request to the web server. Finally, the web server determines the identity of the client, and whether or not to grant the client access to the requested web resource (Schaefer 3).

Over the internet, direct Kerberos connections are often blocked by firewalls, but Kerberos delegation and protocol transition overcome this problem. Kerberos delegation allows the web server to authenticate on behalf of the user. This also allows the user to authenticate to the SQL server, without requiring a direct connection to it. Without delegation, the user would encounter the "double hop" problem, where the user's Kerberos service ticket would not be valid for connecting to the SQL server. Protocol transition allows the user to authenticate to the web server using non-Kerberos authentication, but the web service, in turn, authenticates the user against the Active Directory using Kerberos. Internet scenarios were outside of our initial scope, but knowing this infrastructure is available was an important factor in planning for future growth.

Security

Web applications have a unique set of security considerations, and the .NET Framework provides effective tools to address these concerns. One of these is an innovative permissions model called Code Access Security (CAS). This technology addresses two key scenarios. First, programs usually have access to any resources available to the user account they are running under. CAS gives users granular control over managed application permissions. For example, an unknown executable can be restricted from being able to send emails, upload files to web sites, or create files, even if it is ran as an administrator. This level of control can also be used to prevent known executables from having access to more resources than they require. If an application were compromised, CAS could limit the extent to which the application could be used as an attack vector (Northrup 447). Second, application developers can also using CAS to restrict how external callers can their code, and to limit their application to a restricted permission set.

Sensitive configuration information, such as database connection strings, is also particularly vulnerable in web applications. The .NET Framework mitigates this issue by allowing RSA encrypted connections strings. The encrypted connection string could not be used to reveal the user name and password used to connect to the database, even if the source code or, more commonly, the XML configuration files were compromised.

SQL Injection Attacks

SQL injection attacks are widespread throughout the internet, and allow the execution of arbitrary code SQL servers (Shabat 8). The central vulnerability in these SQL injection attacks is the ability to execute user input. Cerrudo says, "SQL is a general purpose language. No sane person would ever give any rights to run arbitrary C++ or Visual Basic on any server. It's exactly the same with SQL. No sane person should ever accept user input and execute it" (Cerrudo 13). Figure 6 demonstrates a potential SQL injection attack. If the variable "activeMembershipId" represented arbitrary user input, then a malicious user could enter "0 OR 1=1" as their input, and because "1=1" will always evaluate to true, they would retrieve every record even if they are not supposed to.

```
string query = @"SELECT a.Suffix
                FROM Account a
                INNER JOIN Membership m ON
                      a.MembershipId = m.MembershipId
                WHERE
                      m.DisplayNumber =" + activeMembershipId;
```

Figure 6: SQL Injection Attack

The Entity Framework prevents SQL Injection attacks by bringing type safety, proper parameterization, and abstraction to SQL. Because LINQ queries are syntactically part of C#, and parameters are strongly-typed, user input will not be falsely interpreted as query keywords or commands.

Authorization and Authentication Risks

Failures in authentication could allow users to reveal credentials or bypass authentication altogether. Common authentication deficiencies include lack of a password policy, SQL injection vulnerabilities within the login mechanism, using cookies or insecure means to store credentials, and passing user names and passwords in plain text. Utilizing existing, strong authentication mechanisms substantially reduces uncertainty and risk. Again, Windows Integrated Authentication uses the same Kerberos subsystem as Active Directory. For authentication to fail in our application would mean a failure in Kerberos, which would be an industry-wide crisis, and would quickly be resolved. Brute force attacks involve guessing user credentials, and there are several mechanisms to combat this issue as well. The most common method is to lock out a user after several failed login attempts. Another unique approach is introducing a delay of several seconds after a failed login attempt, this time delay makes brute force attacks infeasible.

Possible deficiencies in authorization include inconsistent checks for user authorization on every request, lack of validation and trusting data submitted by users through cookies, hidden fields, URL parameters, etc, which are easily tampered with. The ASP.NET Role provider mitigates these risks by not depending on these weak forms of authorization. Deficiencies in data validation often include the forms of data tampering mentions above, as well as cross-site scripting (XSS), HTTP response splitting, SQL injection (once again), and other malicious input, such as invalid or extended characters.

<u>Performance, and Reliability</u>

A strong bias exists in the IT industry, where Microsoft Windows and SQL Server are often perceived as incapable of hosting mission critical systems, from both a performance and a reliability standpoint. Trends in the financial sector tell a much different story.

Public Perception and the LSE

The London Stock Exchange (LSE) has recently migrated its trading platform to TradElec, a new .NET based system running on Windows and SQL Server. Public reaction and anecdotal evidence presented paint a telling picture of the biases in the IT industry. Prominent bloggers, such as Stephen J. Vaughan-Nichols of Computerworld.com, have expressed scathing criticisms, saying, “The programmers and serious database administrators in the audience can already see where this is going. Sorry, Microsoft, .NET Framework is simply incapable of performing this kind of work, and SQL Server 2000, or any version of SQL Server really, can't possibly handle the world's number three stock exchange's transaction load on a consistent basis” (Vaughan-Nichols 7). He goes on to say, “Even setting aside my feelings for Linux, there's simply no way I'd recommend Server 2003, .NET and SQL Server for a job even a tenth this size. If a customer of mine insisted that they didn't want open source - more fool them - I'd recommended Sun Solaris, JEE (Java Enterprise Edition) and Oracle or IBM AIX or z/OS, WebSphere and DB2" (Vaughan-Nichols 9). Anonymous comments posted beneath the post also represent an interesting cross-section of public opinion. Many agree with comments such as, "If you want a system that never crashes there is one and only one, and everybody knows it. That is the mainframe", while others object to the post entirely: "You just generally attack [TradElec] because it uses a certain technology. It hard for a technologist to find you creditable and see you more than a religious zealot". Before presenting Cortex, and seeing this attitude firsthand, we were not aware that this attitude was so pervasive.

Citi

In a case study provided by Microsoft, Citi's Lava Technology team chose to build a new compliance solution for the U.S. Securities and Exchange Commission's Regulation NMS. The requirements were demanding, 200,000 updates per second, across more than 100 million rows, while supporting query response times of less than two seconds. Rocco Mangino, Senior Vice President Product Development, Lava Technology, Citi, says, "To be quite honest, when this project started we really didn't even consider Microsoft because providing Reg NMS compliance is so demanding" (Microsoft Corporation 13). Jeff Hays, Vice President Product Management, Lava Technology, Citi, says that ultimately they "held a bake-off against other technology vendors and Microsoft won—on performance… The bonus was that it was the lowest cost solution and provided better flexibility as well". (Microsoft Corporation 16).

Clearly, this case study provides comes from a biased source (Microsoft itself), however, it is still highly relevant because it provides another example of the anti-

Microsoft bias we have personally encountered, as well as another strong example of Microsoft technology being used in a critical position within the financial services industry.

NASDAQ

The largest U.S. electronic stock market, NASDAQ trades more shares per day, on average, than any other U.S. market. NASDAQ chose SQL Server 2005 to support its Market Data Dissemination System (MDDS), a system through which every trade processed in the NASDAQ marketplace goes through. Ken Richmond, Vice President for Software Engineering, Market Information Systems at NASDAQ, says, “For years, we used large mainframe computers because of their reputation for reliability… The fact that we can move mission-critical applications from large mainframe computers to SQL Server 2005 and Intel-based servers shows how both Microsoft and Intel are creating enterprise-grade solutions” (Microsoft Corporation 26). This opinion is important because it represents an industry-wide shift from reliance on traditional mainframe systems.

TPC-E Benchmark

The Transaction Processing Performance Council is a non-profit corporation whose mission is to “define transaction processing and database benchmarks and to disseminate objective, verifiable TPC performance data to the industry” (TPC 1). The

TPC has a long history, and is well respected in the industry. The TPC performs several different benchmarks. One of these is the TPC Benchmark E:

> The TPC-E benchmark simulates the OLTP workload of a brokerage firm. The focus of the benchmark is the central database that executes transactions related to the firm's customer accounts. Although the underlying business model of TPC-E is a brokerage firm, the database schema, data population, transactions, and implementation rules have been designed to be broadly representative of modern OLTP systems. (TPC 1)

SQL Server 2008 ranks number one on the TPC-E benchmark results, ranked either by performance or by price/performance. It should be noted that SQL Server does not hold the top position on the older TPC-C metric. IBM DB2 ranks highest on this metric in terms of performance, and Oracle Database 11g ranks highest in terms of price/performance, however, the TPC-C, developed in 1992, is a much older standard than the TPC-E and may no longer reflect real world usages of SQL. In particular, the benchmark does not contain check constraints, referential integrity, or reliable storage, features now extremely common in production environments (Powell 5). Additionally, as of SQL Server 2008, Microsoft no longer publishes TPC-C benchmarks.

Impact on Cortex

Microsoft Windows and SQL Server are highly criticized in terms of performance and reliability. Historically these arguments may have been valid, but they are unfounded today. SQL Server 2008 holds the record in data load performance, being capable of extracting, transforming, and loading 1TB of data from flat files in less than 30 minutes (Wyatt, Shea and Powell). As mentioned earlier, the presence of Windows in enterprise server scenarios is strong, and appears to be increasing.

Existing financial processors at credit unions are usually not high-availability systems, and have many single points of failure. It is also common for financial processors to be brought entirely offline to install regular updates, if not taken offline each night for hours long batch processing. Disaster recovery and business continuity plans are common, but automatic failover to secondary, live databases is not. As a result, tellers and FSRs are trained to work around occasional downtime in the financial processor by being able to perform transactions manually and even offer handwritten receipts. This backlog of manual transactions is then input when the financial processor comes back online.

The negative perception of Windows in mission-critical scenarios comes from a variety of sources. Several individuals have shared their negative personal experiences with Windows Vista, and extrapolated those experiences to form their impression of Windows Server. While Windows Vista and Windows Server 2008 share a common foundation, Windows Server 2008 is preconfigured to be substantially more stable. It should be mentioned that over 70% of Windows system crashes are due to poorly written drivers (Ganapathi, Ganapathi and Patterson). Unstable third-party applications and services make up a large percentage of the remaining crashes. A Windows server, running a supported hardware configuration and stable applications, can achieve much greater reliability than an average desktop installation.

DESIGN

High Level Architecture

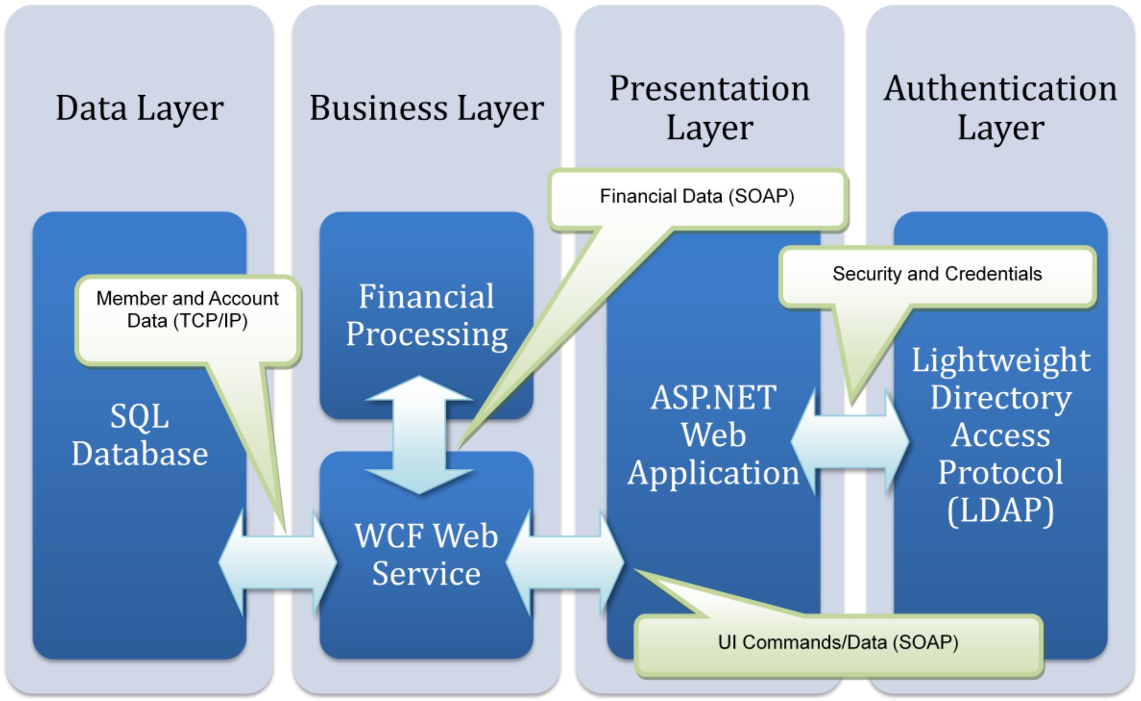


Figure 7: Block Diagram

Now that we have established the framework and technologies that were utilized in designing Cortex, it is possible to discuss how these distinct pieces fit together. The data layer consists of a SQL Server database hosting all system data. The business layer houses all of the business logic, performs the financial processing, and hosts the service layer and service interface. The service layer is implemented as WCF web service. This web service provides a service interface, which acts as the gateway to the business layer. The service layer abstracts the business rules and validation of the system, while ensuring that these are not bypassed by the upper layers of the application.

The presentation layer consists of the web-based UI and is hosted by IIS. The web application itself operates as a client to the service layer, using the service interface to pull data in and out of the business layer. User accounts are stored in a Windows Active Directory, and Windows Integrated Authentication for our application is provided by IIS.

Entity Framework Data Model

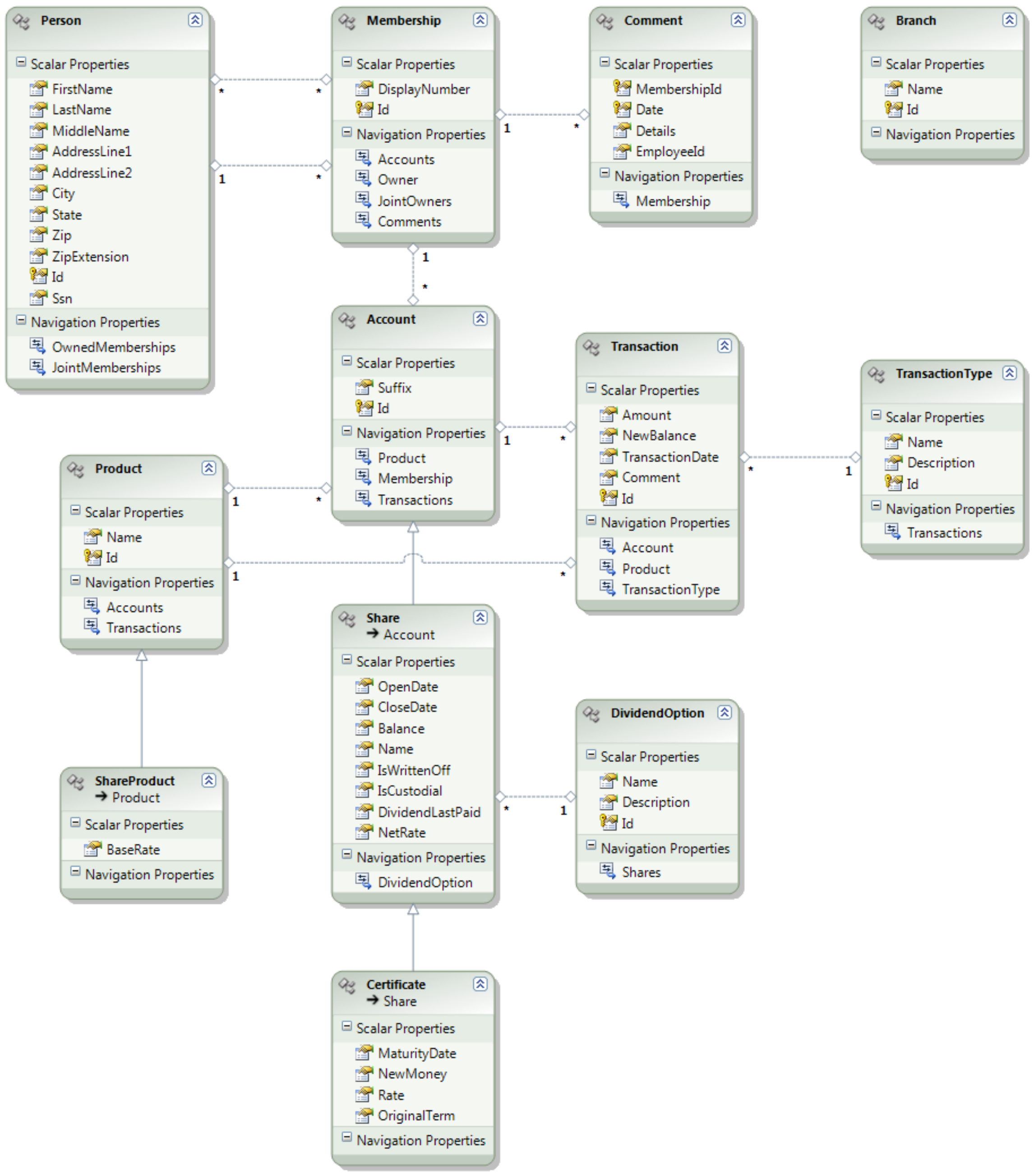


Figure 8: EF Data Model

Designing the database was the most complex task of the project. The ADO.NET Entity Framework, discussed earlier, was one of our team's greatest advantages, because it dramatically simplified database development. The Entity Data Model (EDM) is represented as a diagram similar to a UML class diagram (Figure 8). The EDM, again, provides the mapping between the database tables and the classes that will be used to represent them.

The EF promotes an object-oriented approach to database design. With this approach, relationships between data tables are more concrete, intelligent, and diverse than is intrinsic to SQL databases. Data is also stored in ways more intuitive and intrinsic to credit union operations and thought processes, which makes it easier for administrators/support staff to learn the database structure and report on data trends. Object-oriented database design also allows better modeling of real-world credit union requirements and practices, which are more nuanced and complex than most databases are capable of modeling.

Problems Being Addressed

Existing financial processors are commonly built on obscure database platforms, leaving credit unions unable to directly access their data as they wish, and causing incompatibility with other systems within the credit union. Other specific database issues present in existing processors include slow query performance, poor administrative usability and learnability, and redundant, inconsistent data storage practices within the database.

Another important problem many credit unions face is the lack of proper historical records within the database. For example, if a member’s demographic information is changed, the database may only store the new value, rather than keeping a history of changes. Some credit unions extract nightly flat-file snapshots of their database so that previous versions can be cross-referenced. With these extracts, they would have no way of knowing exactly what their database looked like in any particular time in the past. In this respect, these databases are purely meant to act as OLTP systems, without any OLAP functionality.

Improvements in Our Design

As previously discussed, our database is built on a common, industry standard database platform: SQL Server common to credit unions already. Our database design features a normalized, optimized schema, which eliminates duplication within the database and increases performance speed and usability. We also record of historical changes (previous "versions" indexed by time). Historically, due to storage limitations, this may not have been feasible, but advancements in storage technology have enabled systems to store tremendous amounts of history and statistical information. Wikis, for example, are commonly able to store the entire revision history of each document, so it is reasonable to expect a financial processor to record the changes made to accounts.

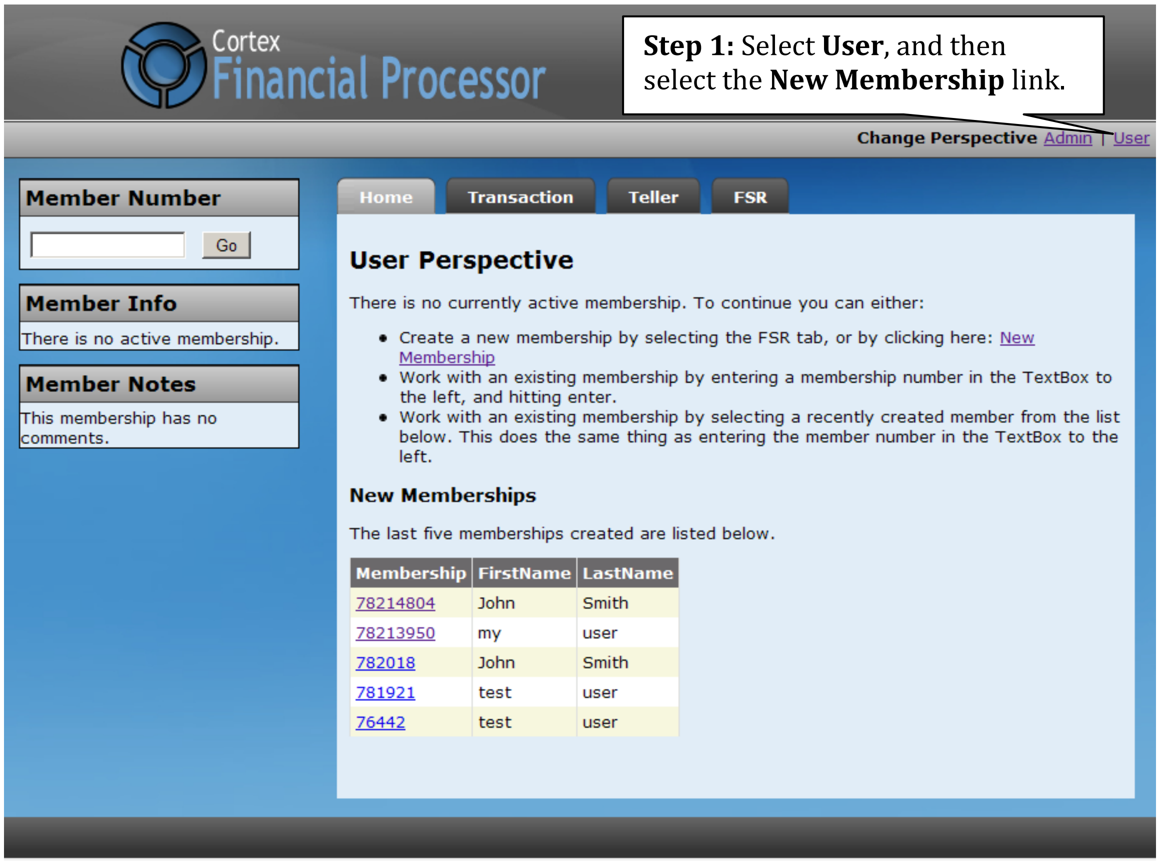
User Interface

Scott Hodgins, senior director of Cornerstone Advisors, advises credit unions to look for functionalities including, “automated work flow, navigation, and ease of use that promote better and more efficient transaction processing and account opening, as well as more commercial offerings. Financial institutions want easier access to information, plus nice, logical screen flow” (Urrico 46). As discussed earlier, our primary categories of users are tellers, FSRs, and back office administrative staff. To make the interface clean, intuitive, and well organized, we divided the website into an *Admin Perspective* and a *User Perspective*. We also designed a tab-based navigation for role-based functionality based on credit union employee divisions, which reduces visual clutter and long menus, and provides a focused, logical navigation for our system, and more efficient transactions, requiring less typing and clicking. Finally, our modern, web-based UI takes into account widespread user familiarity with web-based applications. We improved upon the installation experience provided by other financial processors by requiring no installation at all on the client, other than a web browser.

Design Overview

The prototype design is divided in three sections: Header, Body, and Footer. The Header section contains the logo and code name for this project. The Body section contains the major functionality of the website. The Footer section will contain other information such as copyrights. The Body section contains a brief summary about the customer on the left. The textbox in the Member Number section will allow the FSR or teller to enter the membership number of a member. Once entered, the application will display information pertaining to a member, including primary and joint owner names, home and work phone numbers, and the address of the member. The account notes section, which resides below the Member Info section, indicates any special notes or status of the member's account. The center section of the Body will be change based on the current perspective: either teller FSR, or User. The users will be able to navigate between these perspectives using the tab-based navigation.

Steps to Create a New Membership


Cortex
Financial Processor
Step 1: Select **User**, and then select the **New Membership** link.
Change Perspective Admin | User
Member Number
Go
Member Info
Member Notes
Home
Transaction
Teller
FSR

There is no active membership.

This membership has no comments.

User Perspective

There is no currently active membership. To continue you can either:

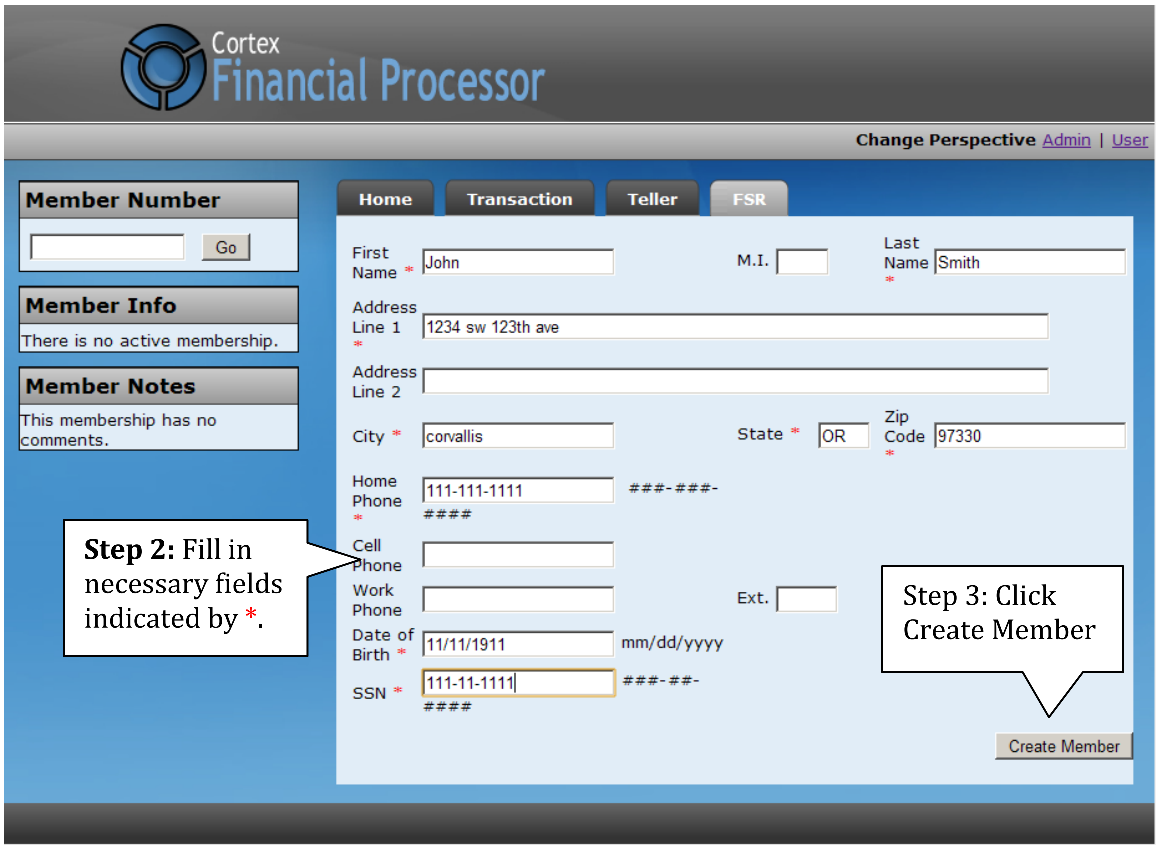
- Create a new membership by selecting the FSR tab, or by clicking here: New Membership
- Work with an existing membership by entering a membership number in the TextBox to the left, and hitting enter.
- Work with an existing membership by selecting a recently created member from the list below. This does the same thing as entering the member number in the TextBox to the left.

New Memberships

The last five memberships created are listed below.

Membership	FirstName	LastName
78214804	John	Smith
78213950	my	user
782018	John	Smith
781921	test	user
76442	test	user

Figure 9: Membership Creation Step 1



Member Number

Member Info

There is no active membership.

Member Notes

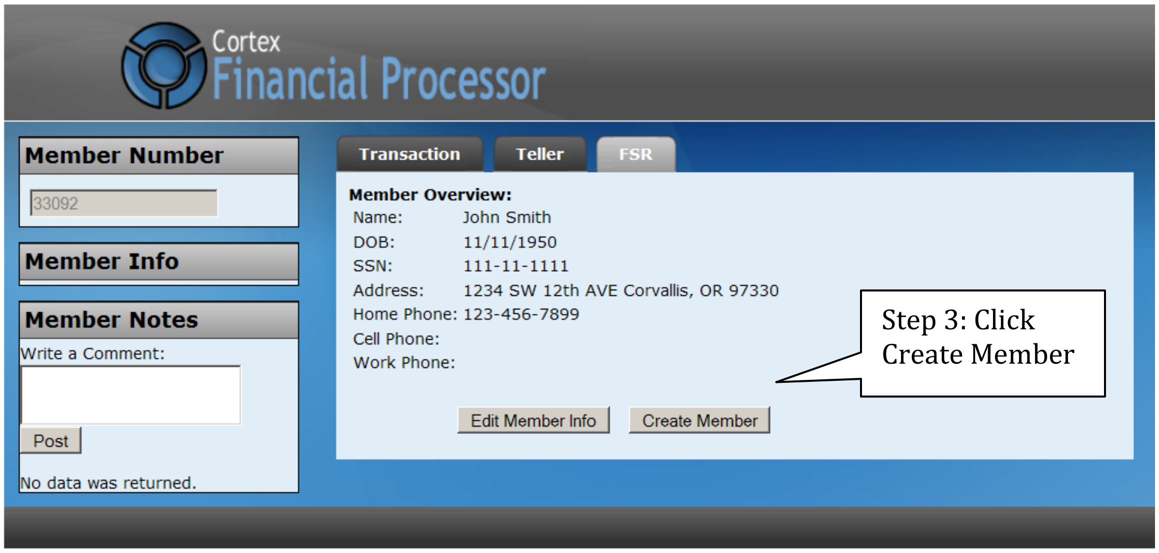
This membership has no comments.

First Name *	John	M.I.		Last Name *	Smith
Address Line 1 *	1234 sw 123th ave				
Address Line 2					
City *	corvallis	State *	OR	Zip Code *	97330
Home Phone *	111-111-1111	###-###-####			
Cell Phone					
Work Phone		Ext.			
Date of Birth *	11/11/1911	mm/dd/yyyy			
SSN *	111-11-1111	###-##-####			

Step 2: Fill in necessary fields indicated by *.

Step 3: Click Create Member

Figure 10: Membership Creation Step 2



Member Number

33092

Member Info

Member Notes

Write a Comment:

No data was returned.

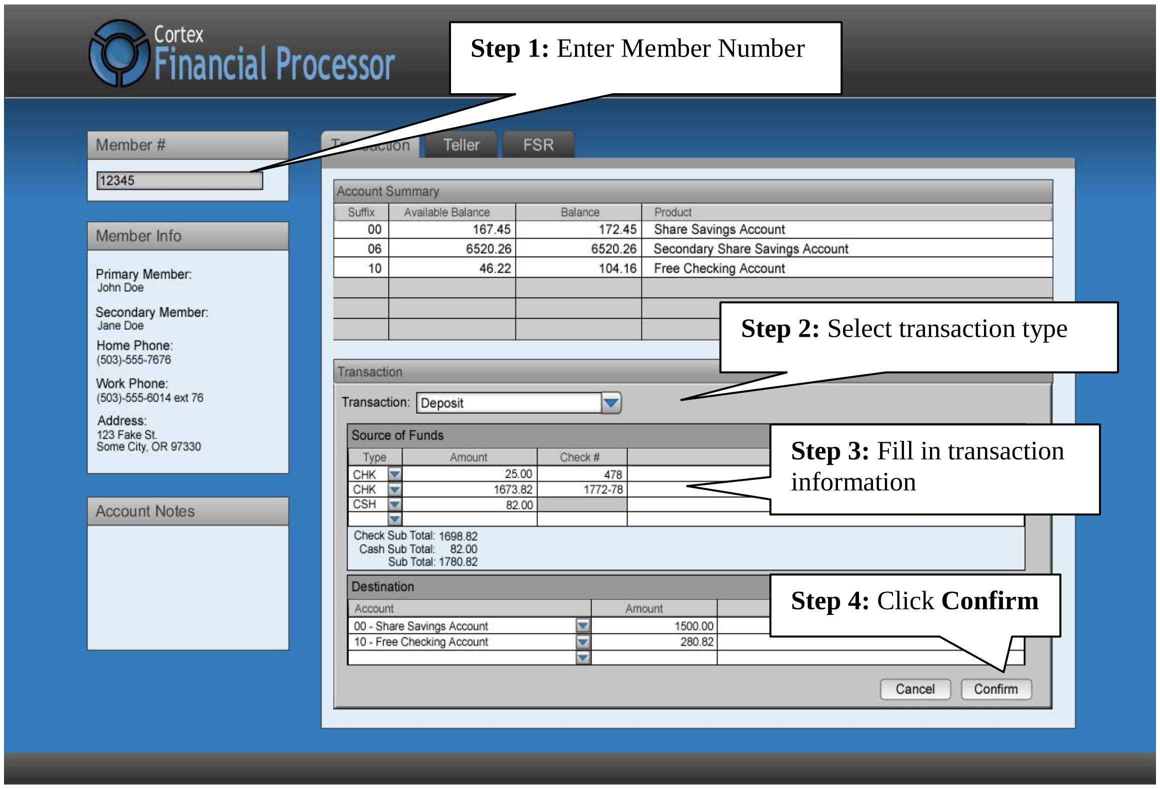
Member Overview:

Name:	John Smith
DOB:	11/11/1950
SSN:	111-11-1111
Address:	1234 SW 12th AVE Corvallis, OR 97330
Home Phone:	123-456-7899
Cell Phone:	
Work Phone:	

Step 3: Click Create Member

Figure 11: Membership Creation Step 3

Steps to Record a Transaction


Cortex
Financial Processor
Step 1: Enter Member Number
Step 2: Select transaction type
Step 3: Fill in transaction information
Step 4: Click **Confirm**
Transaction
Teller
FSR
Cancel
Confirm

Member #: 12345

Member Info

Primary Member:
John Doe

Secondary Member:
Jane Doe

Home Phone:
(503)-555-7676

Work Phone:
(503)-555-6014 ext 76

Address:
123 Fake St.
Some City, OR 97330

Account Notes

Account Summary

Suffix	Available Balance	Balance	Product
00	167.45	172.45	Share Savings Account
06	6520.26	6520.26	Secondary Share Savings Account
10	46.22	104.16	Free Checking Account

Transaction

Transaction: Deposit

Source of Funds

Type	Amount	Check #
CHK	25.00	478
CHK	1673.82	1772-78
CSH	82.00	

Check Sub Total: 1698.82
Cash Sub Total: 82.00
Sub Total: 1780.82

Destination

Account	Amount
00 - Share Savings Account	1500.00
10 - Free Checking Account	280.82

Figure 12: Transaction Entry

Other Functionality and UI Walkthrough

The following as a walkthrough of the UI, demonstrating some of its key functionality:

1. Users are authenticated through Windows Authentication: employees login with their normal network username and password to use the system.
2. FSRs can create a new member profile by visiting User Perspective > New Membership, filling in each form field, and submitting the information.
3. Tellers perform transactions by visiting User Perspective, entering the Member Number, navigating to the Transactions tab and filing in each form field, selecting the transaction method, and submitting the information.
4. FSRs edit member account information by visiting User Perspective, entering the Member Number, selecting Edit Details under the Member Info section, then edit the fields and submit the new information.

5. FSRs add new share accounts to an existing membership by navigating to the FSR tab, selecting Add Share Account, filing in the form fields, and submitting the information.
6. Back office users add new products by navigating to the Admin Perspective, selecting Add Product, filling in the necessary form fields, and submitting the information.
7. Back office users edit products by navigating to the Admin Perspective, selecting Edit Product, filling in the necessary form fields, an submitting the information.
8. Back office users add a new branch by navigating to the Admin Perspective page, selecting Add Branch, filling in the necessary form fields, an submitting the information.

Business Model

Open source business models are relatively new, but evolving. We first began by examining whether an open source model would be effective for this system. Raymond says, "open source has a high payoff where (a) reliability, stability, and scalability are critical, and (b) correctness of design and implementation is not readily verified by means other than independent peer review" (Raymond 172-3). While most non-trivial applications share these characteristics, this description sounds as if it were written for financial processing.

He goes on to say, "A customer's rational desire to avoid being locked into a monopoly supplier will increase its interest in open source… Thus, another criterion (c) pushes towards open source when the software is a business-critical capital good" (Raymond 173). Finally, he notes, "open source seems to be most successful in generating greater returns than closed source in software that (d) establishes or enables a common computing and communications infrastructure" (Raymond 174). This leads to strong indication that a financial processor—a component of a credit union's critical infrastructure—could benefit from open source development, and that an open source business model could be effective in this market.

PERCEPTIONS

Perceptions the General Public

As we began presenting our application to the public, the response was generally positive, yet misconceptions about open source affected some individuals' enthusiasm towards our project. The most common concern expressed was whether open source software is inherently insecure, as any hacker has access to its source code. We were typically able to lessen this concern by explaining that our system was not designed on the "security by obscurity" principle, that is, we designed our system to be secure regardless of whether its source code is publically available.

Another common class of concerns relate to the balance between the numbers of developers on the core team vs. potential hackers. The assumption is that on such an unexciting, yet high-value system would be attractive to hackers. Two factors minimize this risk. First, other open source projects, such as Linux, have not demonstrated any important link between the ratio of contributors to hackers and the vulnerability of the system. Second, and more importantly, physical access to financial processing systems is highly restricted. They are not public-facing web servers, for example, but buried deep within institutions' firewall and physically secured within their data center. Existing financial processors are not presumed to be entirely secure to those within the firewall.

We also encountered individuals who mistakenly believed that open source software was a free-for-all of contributions. We were asked how we could prevent malicious users from checking in rogue code into our core processor. We explained that open source does not mean that anonymous users have control over the central code base. The core development would be done by a trusted team of developers, who may accept outside patches, but under heavy scrutiny. In open source development, customizations are usually more localized, sometimes taking the form of custom redistributions of the project. This would be perfectly acceptable, however, in this case the financial institution would have to be careful that their customized distribution comes from a trusted source.

Perceptions from the NCUA

The National Credit Union Association (NCUA) governs and supervises credit union operations, and has considerable influence over credit unions. Credit unions have been historically reluctant to embrace any open source technology. This attitude among credit unions, and reflected by the NCUA, is slowly starting to change, but free and open source software (FOSS) is still highly scrutinized. In a recent letter to credit unions, the NCUA expressed the opinion that, "The use of FOSS by financial institutions does not pose risks that are fundamentally different from those presented by the use of proprietary or self-developed software. However, FOSS adoption and usage necessitates some distinctive risk management practices with which institutions must be familiar" (NCUA 9). While the letter promising initially, it actually still reflects a deep-rooted hesitation toward open source.

The NCUA says, "Institutions should identify and consider the legal risks associated with the use of FOSS prior to deployment or development. Key legal risks include licensing, infringement, indemnification, and warranties. In most cases, prior to selecting a FOSS solution, institutions should consult with counsel knowledgeable in the areas of copyright and patent law" (NCUA 7). This places an enormous burden on credit unions. After hearing this, it is not surprising that many credit unions would choose to skip open source altogether, rather than expose themselves to the (perceived) dramatically increased risk.

One main emphasis of the letter relates to increased legal exposure resulting from the use of open source. The NCUA states, "Proprietary software licenses customarily include a warranty that the software will achieve a specified level of performance and an indemnity that the vendor will defend the user in the event of an infringement lawsuit. In contrast, FOSS is customarily licensed 'as is,' without warranty or indemnity" (NCUA 8). The NCUA acknowledges that value added resellers (VARs) do offer indemnity as part of their support contracts, but it questions their ability to provide a robust defense. This concern, however, applies to any small vendor.

The NCUA also cautions credit unions about the possibility of obtaining tampered distributions: "Code integrity is important when institutions adopt and implement FOSS because the source code is widely available and can be distributed by anyone" (NCUA). Lack of adequate documentation is also cited as a cause for concern.

Contingency planning is an important aspect of any credit union's overall business strategy, and the NCUA's attitudes toward open source lead to concerns in this area as well. The NCUA says, "Institutions using FOSS can end up with 'dead-end software' if the development community abandons a product. Other outside forces, such as unexpected litigation, may also compel an institution to terminate its use of a particular FOSS application" (NCUA 6). Forced migrations from "dead-end" financial processors was a specific concern mentioned earlier, so this risk is not isolated to proprietary software. In fact, when proprietary software is abandoned, it is a much more serious situation, as there is no possibility for a different community to take over development, as exists within open source development.

The primary issue with these comments from the NCUA is that, as the NCUA acknowledges, many of these concerns apply to proprietary software as well as open source, but tone of the message indicates that open source is inherently riskier, or at least more of a burden. Our team must effectively design our marketing strategy to effectively address these concerns.

Perceptions from Open Source Enthusiasts

On the opposite end of the spectrum, open source enthusiasts frequently criticize our decision to build the system on top of Windows, instead of Linux. The reality is that accepting a financial processor built on an open source operating system would be even harder for credit unions. Credit unions have been reluctant to even accept open source operating systems in their networks, assuming they are also pose a security risk or impose additional costs. Ultimately, our decision to build an open source financial processor on top of the .NET Framework is reasonable because open source does not always imply Linux, it is the most efficient set of technologies available to build this system, and the required IT infrastructure is already widely available in credit unions.

CONCLUSIONS

This year we designed and implemented an efficient financial processor from the ground up, designed and implemented an innovative user interface, implemented support for basic check/savings accounts, and created a foundation for future components. While Cortex is an effective prototype, it will need further development in many areas before credit unions will be able to put it into production. One important area will be implementing support for additional account types, such as loans. Another area will be adding support for peripheral hardware, such as image scanners and signature pads, which are commonly used by tellers and FSRs.

Implementing Open Financial Exchange support will be an interesting future phase of the project. The OFX protocol is an XML web service through which Microsoft Money and Quicken aggregate account balance. On a related note, implementing an online banking extension is an important long-term strategic goal. Online banking is usually a third-party system added on top of the financial processor. Integrating our online banking directly will be a competitive advantage. Finally, we need to further develop our business model, and create a network for future support.

Credit unions and industry members can contribute to development in four important ways. Future development will also require all of these to varying degrees:

1. Beta test and provide feedback
2. Support development financially
3. Contribute to code development
4. Spread awareness of the project and grow the community

The ideal scenario for future development would be for an interested credit union to run our prototype alongside their production system. This would give our team excellent feedback regarding the usability, reliability, and accuracy of our system. They would also contribute to development financially, if not delegate development work to some of their own developers. We look forward to these opportunities.

BIBLIOGRAPHY

Box, Don and Anders Hejlsberg. "LINQ: .NET Language Integrated Query." February 2007. Microsoft Developer Network. 30 April 2009 <http://msdn.microsoft.com/en-us/library/bb308959.aspx>.

Capachin, Jeanne and Michon Schenck. "Fiserv 2.0." October 2006. Financial Insights. 23 April 2009 <http://www.financial-insights.com/FI/getdoc.jsp?containerId=FIN203856>.

Cerrudo, Cesar. "Manipulating Microsoft SQL Server Using SQL Injection." Application Security Inc. 30 April 2009 <http://www.appsecinc.com/presentations/Manipulating_SQL_Server_Using_SQL_Injection.pdf>.

CUNA. "WHAT IS THE CREDIT UNION DIFFERENCE?" 2009. Credit Union National Association. 23 April 2009 <http://www.cuna.org/gov_affairs/legislative/cu_difference.html>.

Fiserv Inc. "Form S-4 -- For the Registration of Securities Issued in Business Combination Transactions." 14 March 1997. SEC Info. 23 April 2009 <http://www.secinfo.com/dsVQy.8252.htm>.

Flasko, Elisa. "Introducing LINQ to Relational Data." January 2008. Microsoft Developer Network. 30 April 2009 <http://msdn.microsoft.com/en-us/library/cc161164.aspx>.

Ganapathi, Archana, Viji Ganapathi and David Patterson. "Windows XP kernel crash analysis." Proceedings of the 20th conference on Large Installation System Administration (2006): 12.

Mertka, Bill. "Developing A More Member-Centric Credit Union." 6 September 2004. CreditUnions.com. 31 May 2009 <http://www.creditunions.com/article.aspx?articleid=1404>.

Microsoft Corporation. ".NET Framework Conceptual Overview." 2009. Microsoft Developer Network. 30 April 2009 <http://msdn.microsoft.com/en-us/library/zw4w595w.aspx>.

—. "Market Montage Solution Supports 200,000 Updates per Second with SQL Server 2005." 11 January 2008. Microsoft Case Studies. 8 May 2009 <http://www.microsoft.com/casestudies/Case_Study_Detail.aspx?CaseStudyID=4000001199>.

—. "NASDAQ Deploys SQL Server 2005 to Support Real-Time Trade Booking and Queries ." 29 November 2005. Microsoft Case Studies. 15 May 2009 <http://www.microsoft.com/casestudies/Case_Study_Detail.aspx?CaseStudyID=49271>.

—. "What Is Windows Communication Foundation?" .NET Framework Developer Center. 2 March 2009 <http://msdn.microsoft.com/en-us/library/ms731082.aspx>.

Moran, John. "The Competitive Moat of Fiserv." 14 April 2008. Seeking Alpha. 21 April 2009 <http://seekingalpha.com/article/72086-the-competitive-moat-of-fiserv>.

NCUA. "Risk Management of Free and Open Source Software." November 2004. NCUA. 15 May 2009 <http://www.ncua.gov/letters/2004/04-cu-14.pdf>.

Northrup, Tony. MCTS Self-Paced Training Kit (Exam 70-536): Microsoft® .NET Framework Application Development Foundation, Second edition. Redmond: Microsoft Press, 2009.

Powell, David. "SQL Server 2008 is on its way with great performance." 6 August 2008. SQL Server Performance. 15 May 2009 <http://blogs.msdn.com/sqlperf/archive/2008/08/06/sql-server-2008-is-on-its-way-with-great-performance.aspx>.

Raymond, Eric S. The Cathedral & The Bazaar. Sebastopol, CA: O'Reilly & Associates, Inc., 1999.

Schaefer, Ken. "IIS and Kerberos. Part 3 - A simple scenario." 16 January 2007. Ken Schaefer. 30 April 2009 <http://www.adopenstatic.com/cs/blogs/ken/archive/2007/01/16/1054.aspx>.

Shabat, Gil. "Securing ASP.NET Applications ." 8 April 2009. Novologies. 27 April 2009 <http://www.novologies.com/post/2009/04/08/Securing-ASPNET-Applications.aspx>.

TPC. "About the TPC." Transaction Processing Performance Council. 8 May 2009 <http://www.tpc.org/information/about/abouttpc.asp>.

—. "TPC-E OLTP." Transaction Processing Performance Council. 11 May 2009 <http://www.tpc.org/tpce/default.asp>.

Urrico, Roy W. "CUs Brace for Core Changes." Credit Union Magazine December 2007: 44-48.

Vaughan-Nichols, Steven J. "London Stock Exchange suffers .NET Crash." 9 September 2008. Computerworld Blogs. 15 May 2009 <http://blogs.computerworld.com/london_stock_exchange_suffers_net_crash>.

WOCCU. "What is a Credit Union?" World Council of Credit Unions. 24 April 2009 <http://www.woccu.org/about/creditunion>.

Wyatt, Len, Tim Shea and David Powell. "We Loaded 1TB in 30 Minutes with SSIS, and So Can You." March 2009. SQL Developer Center. 13 May 2009 <http://msdn.microsoft.com/en-us/library/dd537533.aspx>.