IBIMA Publishing
Communications of the IBIMA
http://www.ibimapublishing.com/journals/CIBIMA/cibima.html
Vol. 2012 (2012), Article ID 532248, 9 pages
DOI: 10.5171/2012.532248

Lean Integration Using Open Source Based Infrastructure and an Integration Competency Center Approach

Benneth Christiansson

Redpill Linpro AB, Sweden and Department of Information Systems and Project Management, Karlstad University, Karlstad, Sweden

Abstract

In this paper a set of successful Enterprise Application Integration projects are analyzed and common characteristics are identified. These characteristics are then distilled into a set of key success factors. The empirical foundation for this study is the active participation in a number of successful EAI projects, as a consequence thereof we classify the research method as action research. We can note that common key success factors are the use of common canonical formats for data exchange, open source and ESB for EAI infrastructure, enterprise integration patterns, and the establishment of an integration competency center. The possibility to enable successful and maintainable EAI should be of crucial interest for basically all of todays businesses, where we only see an increased need for successful EAI both in a business as well as in a business to business perspective.

Keywords: EAI, Open Source, Lean Integration.

Introduction

This paper describes a set of key success factors identified as common denominators in several studied Enterprise Application Integration (EAI) projects. The possibility to enable successful and maintainable EAI should be of crucial interest for basically all of todays businesses. Wether we talk about Service Oriented Architecture (SOA) or EAI we see an increased need for successful EAI both in a business as well as Business to Business (B2B) perspective. In this paper we address key success factors we would classify as being part of the traditional software engineering discipline (Pressman, 1992; Brooks Jr, 1987) adapted to the special case of integration/service development. Gu and Lago (2009, p. 5) defines this specific subdiscipline of software engineering as Service-Oriented system engingeering (SOSE):

"...addresses systematic, disciplined and quantifiable approaches to develop service-oriented systems. A common concept shared among these approaches is software being used as a service for consumption."

The ultimate goal of service technologies is an IT infrastructure consisting of cooperating services where application components are assembled with little effort into a network of services that can be loosely coupled to create dynamic business processes and agile applications that span organizations and computing platforms (Leymann, 2005). Working with service development and service maintenance does however place new demands on businesses regarding organization, skills and methodology (Papazoglou et al., 2007; Rosen et al., 2008).

Integration Competency Center

An Integration Competency Center (ICC) can be defined as a management, coordination and operational team, responsible for assisting other projects with

 Contact author: Benneth Christiansson
E-mail: Benneth.Christiansson@kau.se

their EAI needs throughout the whole organization (Schmidt, 2009; Informatica Inc., 2005; Stelzer, 2010). The team will be responsible for development and government of specific integrations offered as consumable services to the organization. The ICC will also possess the key knowledge required for the effective development of successful maintainable integrations between oftentimes disparate business areas.One of the keys to effective development is the ability to (re)use existing services (Christiansson and Christiansson, 2006; Hohpe and Easy, 2007; Rosen et al., 2008). ICC provides the direction, control, optimization, consistency and long-term focus that successful and maintainable integration projects require (Jotham and Toivanen, 2010; Schmidt and Lyle, 2010).

Enterprise Application Integration

Enterprise Application Integration (EAI) is a business need to enable heterogenous applications, in one business or between several businesses, to communicate and exchange data to achieve business objectives (Cummins, 2002; Land and Crnkovic, 2004). The requirements for these integrations are that they need to be built in a seamless reliable fashion irrespective of platform and geographical location of these applications. EAI can be viewed as messages passed between heteogenous applications where messages are delivered, accepted, rejected, transformed translated and routed in accordance with requirements defined in business processes. Usually messages transportation is asynchronous but if a business need requires it it can be synchronous as well. There are two basic architectures to achieve EAI, 1) hub/spoke and 2) bus architecture. Both of these can be used to develop integrations oftentimes the integrations are referred to as services. Such a service can be viewed as a part of a SOA (Goel, 2007; Rosen et al., 2008). In our experience EAI is one of the major driving forces to implement a SOA in a business. In our experience the SOA approach does not need to reach further into the IT infrastructure to be successful, in fact we would argue that taking the larger leap and restructuring the whole infrastructure to SOA is attached with larger risks and the potential of failure is dramatically increased. *1. Hub/Spoke architecture* uses a centralized broker a so called "Hub" and adapters the "Spokes" which connect applications to the Hub. Spokes connect to applications and convert application data format to a message which the Hub understands and vice versa. The Hub brokers all messages and takes care of content transformation/translation of the incoming messages into formats that the receiving application understands. The Hub is also responsible for the routing of the messages, i.e. making sure the message is delivered to the correct Spoke. *2. Bus architecture* uses a central messaging backbone a so called "Bus" for message propagation. Applications publish messages to the bus using adapters. These messages are routed to subscribing applications using the Bus. See Figure 1 for an example of a Bus architecture.

Empirical Studies

Stelzer (2010, p. 16) have performed a literature review covering research production in the Enterprise Application Integration area. In this study one of the findings is the lack of documentation of results from empirical research:

"Compared to the considerable amount of publications on enterprise architecture in general, the number of articles presenting research findings on enterprise architecture principles is rather low."

This state of affairs in the research community is further elaborated by the same authors Stelzer (2010) when they claim that:

"Extending the basis of case studies. There are only few publications that describe practical experience with enterprise architecture principles. Since this research field has not yet been explored in detail and theoretical foundations are meager we need more explorative research. More case studies might help to shed light on key issues and success factors when formulating and deploying architecture principles."

Papazoglou et al. (2007, p. 70) describes in their paper "Service-Oriented Computing: State of the Art and Research Challenges" that research covering service-oriented software engineering is a topic in need of focus:

"Engineering of service applications. SOA-based applications require a service-oriented engineering methodology16 that enables modeling the business envi- ronment, including key performance indicators of busi- ness goals and objectives; translates the model into service design; deploys the service system; and tests and manages the deployment."

We have for several years been able to do first hand observations while participating in several large and small scale Business Integration projects in Scandinavia, both in the public as well as in the private sector. All case studies are performed in close cooperation with staff from Redpill Linpro AB. Redpill Linpro AB is the largest provider of professional services for open source based enterprise applications in Scandinavia. During the years 2008 to 2011 we have actively participated in a range of different types of Business Integration projects ranging from large and mature ICCs (over 50 governed integrations and approximately 20 team members in the ICC) to small and freshly started ICCs covering approximately 4 team members.

We classify this research as mainly action research where the authors also are active participators in the actual projects, this implies that we as researchers also affect the outcome of the projects and thereby indirectly affect the outcome we base our conclusions on, the possible effects this can have on the viability in our conclusions is further elaborated in the section called "Methodology" below. During the projects we have actively kept a journal noting the day to day activities performed and by analyzing these entries using a Grounded Theory inspired approach (further elaborated below) together with findings from fellow researchers we believe the foundation for our findings is solid and viable even though action research can biase the empiricism used. The used empirical material is collected during the period august 2008 to october 2010.

The Challenge

Todays businesses find themselves in a situation where several disparate demands forces them to invest in Enterprise Application Integration (nowadays oftentimes disguised as a SOA strategy). Below we present a non exhaustive set of reasons we encounter in our discussions with businesses. The majority of businesses have *existing legacy applications*, developed in different eras using different architectures and technologies. When new needs evolve the businesses usually can not afford to write them off or replace them with new applications, because they are mission critical. Besides the existing applications and new integration needs for them businesses will also need to introduce *new applications* from time to time. New applications are usually based on new architectures, which differ significantly from architectures used by existing legacy applications. These new applications also have to be integrated with existing applications; and existing applications have to be integrated with each other to fulfill the information availability and accessibility goals. Another demand is the increased need to make *inter-business integration* or "business-to-business" (B2B) integrations. Where the actual integration project needs to, besides handling the ordinary ordeals of EAI, also handle stakeholders from different businesses. To the above mentioned ordinary ordeals for integration projects we would like to highlight problems like the lack or outdated documentation of legacy applications. The fact that oftentimes only a few staff members possess competence and knowledge regardin´g legacy applications. The older the application the fewer possess knowledge. Oftentimes only a few staff members possess knowledge regarding the oftentimes unique infrastructure created from the use of different technologies from different eras being mixed in a very specific blend based on the legacy applications once acquired.

We can see that integrating applications is a difficult task, maybe even one of the most difficult tasks facing the software development community. After seeing several very successful EAI projects and noticing they all have a series of common characteristics, we decided to document them in this paper, calling them "key success factors". These findings should be of value for a large audience including stakeholders in ongoing or planned EAI projects.

Methodology

We have used a straight forward approach to our empirical studies, using action research as defined in The SAGE handbook of action research: Participative inquiry and practice (Reason and Bradbury, 2008). We believe that the following quotation describes our research approach (Reason and Bradbury, 2008, p. 4)

"..action research is a participatory process concerned with developing practical knowing in the pursuit of worthwhile human purposes. It seeks to bring together action and reflection, theory and practice, in participation with others, in the pursuit of practical solutions to issues of pressing concerns to people..."

We have paid close attention to as many as possible of the drawbacks in using action research as research methodology but would also like to state that it is very well suited for gaining access and insight into "real world projects" that is the foundation for this paper. During the action research process we used a journal approach to keep track of insights and knowledge gained. This journal have then been analyzed during the writing phase this paper represents. We have used a slightly modified version of the original Grounded Theory approach Glaser (1978) when analyzing the Journal. Basically Grounded Theory can be defined as:

"...grounded theory methods are a set of flexible analytic guidelines that enable researchers to focus their data collection and to build inductive middlerange theories through successive levels of data analysis and conceptual development."

The approach we have used when analyzing the journal can very shortly be described as a three step process: *Step 1* Open coding, by analysing the terms used in the journal, common categories or patterns are identified. *Step 2* Axial coding identified common categories are related to eachother in conceptual graphs. *Step 3* Selective coding, main categories are selected and the other minor categories are related to them in conceptual graphs. This approach to Grounded Theory is called Multi Grounded Theory (Goldkuhl and Cronholm, 2003).

An alternative research method we used to a lesser extent can be called "participant observation", as described by Seaman (1999, p. 5):

"...research that involves social interaction between the researcher and informants in the milieu of the latter, during which data are systematically and unobtrusively collected. The idea is to capture firsthand behaviors and interactions that might not be noticed otherwise. Although the name is misleading, participant observation does not necessarily imply that the observer is engaged in the activity being observed, only that the observer is visibly present and is collecting data with the knowledge of those being observed."

The Key Success Factors

Using Common Canonical Formats for Data Exchange

The integration of existing applications and data is probably one of the more complex and challenging tasks facing enterprise IT organizations (Rosen, Lublinsky, Smith, and Balcer, 2008). To be able to integrate and exchange data between heterogenous systems the organization needs to have an enterprise-wide, common semantic model. This common semantic model can be used to identify and specify necessary data transformations for integrating heterogeneous applications. To setup an enterprise common semantic data model, the organization needs tools to enable semantic as well as syntactical transformations between data

representation in existing applications and the enterprise-wide common semantic model.

The introduction of a canonical information format where each information provider and consumer is responsible for normalizing the information into a canonical format prior to providing the information will enable a common semantic data model (Patrick, 2005; Chen, 2003). This will however introduce new requirements on existing applications. Due to the fact that few applications are delivered with source code this can only be handled in the form of adapters and wrappers doing the transformations outside of the actual applications. This conveys the need to govern each adapter and wrapper as unique artifacts leading to an added strain on Operations staff and software maintenance. The inclusion of an Enterprise Service Bus (ESB) (presented in next section) has the potential to remedy this well known issue with EAI. The transformations and required logic are not separate entities but rather part of services contained within the realm of the ESB.

The creation and usage of a canonical information formats for heterogeneous applications, in combination with using an ESB centric approach to integration is one of the identified success factors. Another success factor is to use the eXtensible Markup Language (XML) for defining canonical formats. This does not imply that all heterogenous applications need to provide or process XML formatted data but rather it is a requirement when creating a canonical information format, performed by the services deployed to the ESB. XML consists of a set of rules for defining and representing information as XML documents where information structures are indicated by explicit markup. The markup vocabulary and the structures specified for a particular domain create an XML application, a formal language for representing a common semantic data model.Power (2005, p. 34) claims that XML offers a flexible way to exchange information:

"The advantage that XML offers is a flexible standard for the exchange of information between trading partners via the internet."

One important note regarding an enterprise-wide common semantic information model is that just introducing a homogenous way of data representation (e.g., XML) does not imply common semantics. The notion of "customer" can have many different semantics, connotations, constraints, and assumptions in each participating system. Resolving semantic differences between systems proves to be a particularly difficult and time-consuming task because it involves significant business and technical decisions (Hohpe and Woolf, 2003). To reach common semantics we need to introduce one enterprise-wide common semantic information model defined with homogenous canonical information formats.

Using Open Source and Enterprise Service Bus

There are several well defined advantages in using open source based software, but also a set of disadvantages (Ven et al., 2008; Wheeler, 2007; Lakhani and Von Hippel, 2003). In our experience, an open source approach to developing and governing an ESB infrastructure has proven advantageous in several ways. First, the rapid turn-around time on feature requests and bug fixes has encouraged early adoption and the growth of a community that both provides feedback and now supports itself, in addition to several large scale companies backing up the projects. Second, the lack of a licensing fee encourages the use of open source software in organizations seeking numerous deployments. Finally, the reuse of existing open source components allows the development teams to focus their efforts on improving the aspects of the integration solutions rather than the underlying mechanisms that make it possible. This in combination with the obvious promotion of open standards usages enables for a flexible and highly reusable architecture with a potentially high population of stakeholders. We have found similar experiences from other projects with similar circumstances like for instance Bortis (2008).

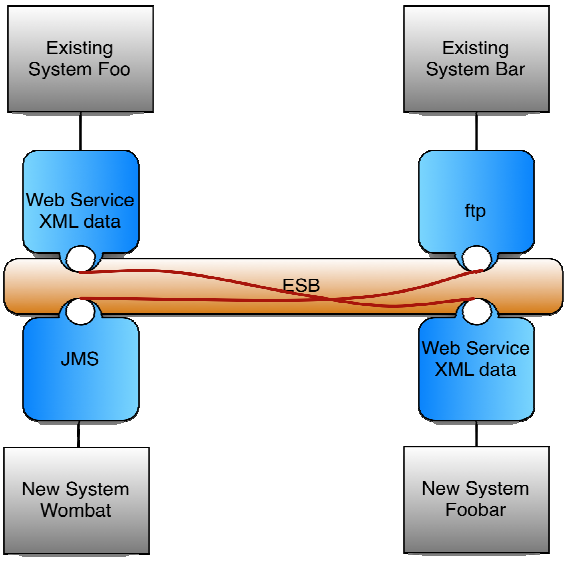

Figure 1: An Enterprise Service Bus with Flexible and Extendable Ways of Connecting Applications

An extension of the basic EAI strategies is the introduction of an ESB or an ESB-centric integration approach, using the previously mentioned Bus approach to EAI. Using an ESB can provide the flexibility to allow data transformations and other services to be "plugged" into the bus and then be reused by any number of different services and application components (Patrick, 2005). An ESB is an enterprise-wide extendable middleware infrastructure providing virtualization and management of service interactions, including support for the communication, mediation, transformation, and integration technologies required by services (Rosen, Lublinsky, Smith, and Balcer, 2008). This is illustrated in Figure 1 where the black boxes represent existing and future applications, blue boxes represent ESB connectivity adapters. The red wavy lines indicates implemented integrations deployed to (plugged into) the ESB.

Using Enterprise Integration Patterns

Unlike most other engineering disciplines IT development and software engineering is still characterized by the lack of a precise vocabulary (Hohpe and Easy, 2007). While computer science has established solid theoretical foundations, designing complex software systems tends to be a much less structured activity than designing buildings or machines (Hohpe and Easy, 2007; Pressman, 1992). Patterns constitutes one way of handling the lack of precise vocabulary by establishing a common language in terms of "patterns" for common used abstractions regarding designing and development of software artifacts. Since patterns are not meant to be precise definitions and do not have to map into an overarching meta-model, they can also be "soft" around the edges, conveying knowledge without necessarily being regarded as specifications (Hohpe and Easy, 2007). Software developers tend to start with a blank slate and are oftentimes constrained by very few factors beyond the language syntax. In such an environment patterns can aid the designer as opposed to prescribing and forcing a generic solution based on required specifications. Design patterns can be helpful if they represent field-tested solutions to common design problem. This means that for an individual situation we only need to identify which pattern to use and the quality of the solution will be "guaranteed" through the previous real life usage. Furthermore patterns can assist IT development by organizing design intelligence into standardized and labeled

recipes/descriptions. By introducing and consistently reusing the same set of patterns we increase our possibility to ensure consistency in how systems are designed and built. Patterns are however not something to be enforced or made mandatory when doing design but rather flexible and optional tools to be used.

A specific type of patterns are defined as "Enterprise Integration Patterns" these patterns are part of the foundation for several ESB solutions, with maybe the Apache Camel project as the leading example (Apache Foundation, 2012). EAI needs to provide efficient, reliable, and secure data exchange between multiple enterprise applications, and enterprise integration patterns distill a set of best practices for accomplishing this. According to Hohpe and Woolf (2003) the patterns are targeted at providing solutions for the constraints of developing integration solutions where the limited amount of control the integration developers typically have over the participating applications is a fact. Another constraint is the lack of interoperability between "standards-compliant" products. We have identified the use of selected enterprise integration patterns together with an ESB-centric integration approach to be one of our key success factors.

Establishing Integration Competency Centers

According to Wikipedia the term Integration Competency Center and its acronym ICC was initially defined by Roy Schulte of Gartner (Wikipedia Foundation, 2012). The ICC is an enterprise shared service for performing systematic application integration. Introducing an ICC will always require an initial organizational effort but will once its done (correctly) lead to reduced integration costs. An ICC can accomplish this by enforcing standards, using standardized and well defined processes, and driving software and data reuse throughout all integration projects. The result can lead to less development effort, reduced need for extensive testing, and lower support costs.

The introduction of an ICC will be a step in creating an adaptive enterprise to allow the business to rapidly change as the market changes. The ICC does this by allowing individual applications to be loosely coupled so that they can change independently yet still be tightly integrated to enable efficient business processes (Informatica Inc., 2005). In our empirical studies we also identified that including lean principles into the day to day tasks of the ICC to be another success factor. Lean Integration can be summarized as a set of principles similar to Lean manufacturing and Lean software development (Schmidt and Lyle, 2010). The first principle *eliminate waste* is targeted to the obvious task of always scrutinizing every activity to make sure that it adds value to the deliverable at hand and otherwise reject the activity. The next principle is to *sustain knowledge* one of the big challenges with IT development in general and integration in particular is the demands on knowledge and skills it imposes on the practitioners. This can be met by working with elaborate individual competence enhancement plans, defined processes and requirements for documentation, throughout all phases of the integration development projects, and through a standardized and centralized information hub (e.g. integration wiki) to place central documentation for easy access and usage. The following three principles *plan for change, deliver fast* and *empower the team* are included in the set of processes and methodologies we have seen incorporated into the ICCs covered in the empirical studies. The major part is done through the use of Scrum as a project management method. Scrum is an iterative, incremental methodology for project management often seen in agile software development. Although Scrum was intended for management of software development projects, it is used to run both software maintenance teams, and general project/program management approaches (Schwaber and Corporation, 2004).

In our empirical studies we have identified Scrum as the preferred methodology to run and manage the day to day operations of ICCs including development and governance

of integrations. Another key success factor is that the ICC's all include a standardized set of tools for enhancing both quality and speed in delivering integrations. These tools include support for thorough testing, continuos integration, intelligent handling of software builds and source code versioning. One interesting note is that the entire tool suites is built with pure open source products.

Conclusions

EAI is a challenging and crucial task that the majority of todays businesses needs to handle. One could probably claim that the level of success can, at least for some businesses, be measured in conjunction with their success in EAI. We have participated in several successful EAI projects for over a two year period and have during this time identified a set of common characteristics. These common characteristics are:

1. Using Common Canonical formats for data exchange. Through the use of canonical formats and a uniform way of defining them (XML) we create the basis for enabling common semantics within a business. This is crucial for successful EAI.

2. Using open source and ESB for EAI infrastructure. The use of open source have several benefits beside the obvious of no licensing costs. We have also found that an ESB-centric approach to EAI provides several of the needed characteristics for creating maintainable EAI.

3. Using enterprise integration patterns. In all studied EAI projects a common denominator is the use of Enterprise Integration patterns. This is even observable in the inner workings of several open source based ESB solutions where several enterprise integration patterns are included as major parts of the ESB basic functionality.

4. Establishing ICCs, an ICC provides the benefits of increased quality through controlled processes and standardized tooling operated by competent and experienced staff using best practices. We also notice cost savings through reusability, faster development and easier maintenance.

References

Apache Foundation (2012). "Apache Camel," Wikipedia. [Online], [Retrieved March 4, 2012], http://camel.apache.org/.

Bortis, G. (2008). "Experiences with Mirth: An Open Source Health Care Integration Engine," Proceedings of the 30th International Conference on Software Engineering, *ACM,* 649–652.

Brooks Jr, F. P. (1987). "No Silver Bullet Essence and Accidents of Software Engineering," *Computer,* 20 (4), 10–19.

Chen, M. (2003). "Factors Affecting the Adoption and Diffusion of XML and Web Services Standards for E-Business Systems," *International Journal of Human-Computer Studies,* 58 (3), 259–279.

Christiansson, M.- T. & Christiansson, B. (2006). Mötet Mellan Process Och Komponent: Mot Ett Ramverk för En Verksamhetsnära Kravspecifikation Vid Anskaffning Av Komponentbaserade Informationssystem, *Institutionen för Datavetenskap, Linköpings Universitet,* Doctoral Thesis, in Swedish.

Cummins, F. A. (2002). Enterprise Integration: An Architecture for Enterprise Application and Systems Integration, *John Wiley & Sons Inc.,* New York.

Glaser, B. G. (1978). Theoretical Sensitivity: Advances in the Methodology of Grounded Theory, *Sociology Press,* Volume 2.

Goel, A. (2006). "Enterprise Integration EAI vs. SOA vs. ESB," *Infosys Technologies White Paper.*

Goldkuhl, G. & Cronholm, S. (2003). Multi-Grounded Theory Adding Theoretical Grounding to Grounded Theory, 2nd European Conference on Research Methods in Business.

Gu, Q. & Lago, P. (2009). "Exploring Service-Oriented System Engineering Challenges: A Systematic Literature Review," *Service Oriented Computing and Applications,* 3 (3), 171–188.

Hohpe, G. & Easy, C. (2007). "SOA Patterns–New Insights or Recycled Knowledge," *Enterpise Integration Patterns.* [Retrieved March 4, 2012] http://www.eaipatterns.com/docs/SoaPatterns.pdf.

Hohpe, G. & Woolf, B. (2003). Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions, *Pearson Education, Inc.*

Informatica Inc., (2005). 'Information Competency Center Seize the Integration Advantage,' *Informatica Corporation, White Paper.*

Jotham, G. & Toivanen, A. (2010). "Integration Competency Center," *Frends Technology, White Paper.*

Lakhani, K. R. & von Hippel, E. (2003). "How Open Source Software Works: "free" user-to-user assistance," *Research Policy,* 32 (6), 923–943.

Land, R. & Crnkovic, I. (2004). "Existing Approaches to Software Integration – And a Challenge for the Future," *Proceedings of Software Engineering Research and Practice in Sweden (SERPS),* Linkoping University, Sweden, 4, October 2004.

Leymann, F. (2005). "The (Service) Bus: Services Penetrate Everyday Life," *Service-Oriented Computing-ICSOC 2005,* 12–20.

Papazoglou, M. P., Traverso, P., Dustdar, S. & Leymann, F. (2007). "Service-Oriented Computing: State of the Art and Research Challenges," *Computer,* 40 (11), 38–45.

Patrick, P. (2005). "Impact of SOA on Enterprise Information Architectures," Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, *ACM,* 844–848.

Power, D. (2005). "Supply Chain Management Integration and Implementation: A Literature Review," *Supply Chain Management: An International Journal,* 10 (4), 252–263.

Pressman, R. (1992). 'Software Engineering-A Practitioner's Approach,' *Mcgraw Hill.*

Reason, P. & Bradbury, H. (2008). The SAGE Handbook of Action Research: Participative Inquiry and Practice, *Sage Publications Ltd.*

Rosen, M., Lublinsky, B., Smith, K. T. & Balcer, M. J. (2008). Applied SOA: Service-Oriented Architecture and Design Strategies, *Wiley Publishing, Inc.*

Schmidt, J. (2009). "Lean Integration," *Infortmatica Corporation, White Paper.*

Schmidt, J. G. & Lyle, D. (2010). Lean Integration an Integration Factory Approach to Business Agility, *Addison-Wesley Professional.*

Schwaber, K. & Corporation, M. (2004). "Agile Project Management With Scrum," *Microsoft Press, Redmond.*

Seaman, C. B. (1999). "Qualitative Methods in Empirical Studies of Software Engineering," *Software Engineering, IEEE Transactions on*, 25 (4), 557–572.

Stelzer, D. (2010). "Enterprise Architecture Principles: Literature Review and Research Directions," *Service-Oriented Computing. ICSOC/Servicewave 2009 Workshops,* 6275, 12–21.

Ven, K., Verelst, I. & Mannaert, H. (2008). "Should You Adopt Open Source Software?," *Software, IEEE,* 25 (3), 54–59.

Wheeler, D. (2007). "Why Open Source Software / Free Software (OSS/FS, FLOSS, or FOSS)? Look at the Numbers!," [Online], [Retrieved March 4, 2012], http://www.dwheeler.com/oss_fs_why.html.

Wikipedia Foundation (2012). "Integration Competency Center," [Online], [Retrieved March 4, 2012], http://en.wikipedia.org/wiki/Integration_Competency_Center