TOWARDS AGILE SYSTEM ANALYSIS & DESIGN: IMPROVING KNOWLEDGE TRANSFER ACROSS DOMAINS USING LAYERED FRAMEWORK

Amber Lynn McConahy, Pennsylvania State University
Abhijit Dutt, George Mason University

Abstract

An ongoing problem plaguing system developers is the inability to effectively transfer knowledge from domain experts to IT experts. This is due to many factors, such as not having a systematic and standardized methodology in order to conduct system development tasks, tendency to embed business rules and processes in source code without codifying them, and incomplete documentation of software architecture. In order to address this problem, we propose a layered framework in order to encapsulate system development activities and standards the process of migrating from a business problem into the actual implementation of a software solution to this problem. By providing such a systematic approach to the development process, we improve the ability of project participants to make key decisions about the design of the system and it architecture. Utilizing accepted techniques for modeling, this framework better supports the evolution and maintenance of the system by providing multiple layers of abstraction, as well as providing an environment conducive to iterative development.

Keywords*: Framework, SA&D, OOAD, Layered Architecture, Software Architecture, Software Development Methodologies, Modeling Techniques*

INTRODUCTION

Since the 1960s, the use of computers and information systems (IS) in business has steadily increased, featuring more robust and comprehensive solutions. Historically, information technology (IT) was only used for the backend of businesses, which often focused on the storage and management of data resources. In recent years, the widespread use of the Internet has led to the

increased acceptance of electronic commerce and mobile commerce both in business-to-business (B2B) and business-to-consumer (B2C) transactions, whereby employees and customers utilize IT systems in order to automate mundane day-to-day operations (Koufaris, 2002). These business processes typically vary in complexity and require developers to embed both business knowledge and business rules into the underlying source code. As the paradigm shift from single, monolithic solutions to software ecosystems continues, system developers must support increasingly complex systems, often with components from multiple sources while maintaining a high level of usability.

In addition to increased complexity, IT system development relies on two distinct stakeholder groups and is a joint effort between IT experts (system analysts, software engineers, software architects, etc.) and domain experts (people who have domain or background knowledge about the system being developed) (Dennis, Wixom, and Tegarden, 2005; Evans, 2003). Business knowledge must routinely be transferred from the domain experts to the IT experts, and communication challenges between these two stakeholder groups often inhibit the timely and effective transfer of such knowledge. Currently, capturing domain expertise into IT systems remains one of the greatest challenges in IS development. This can be attributed to the fact that there are no formal analysis and design methodologies, which facilitate a systematic and standardized transition from domain expertise into system design (Alter, 2005). In other words, the success of a system development effort is highly correlated with the ability to effectively transfer knowledge from domain experts to IT experts. Discrepancies that arise between these two stakeholder groups ultimately lead to delays, budget overruns, and project failures.

In cases where knowledge transfers are effective, an additional challenge presents itself because these transfers are never formally documented. In other words, knowledge that was transferred either exists as tacit knowledge that is never codified or is embedded in the source code. To further complicate this problem, the system development team is often disbanded upon completion of the project, and developers leave without formally capturing this tacit knowledge and extracting it from the source code, which requires both IT and domain expertise. Furthermore, the design patterns utilized in the implementation typically provide key nonfunctional requirements, such as performance, security, availability, etc., from the source code. Consequently, any third party tasked with extracting knowledge from the source code must understand design patterns and the quality attributes each pattern promotes. This inhibits the ability to make informed architectural decisions about the evolution and maintenance of the system.

Due to the vast problems in the transfer of knowledge between stakeholder groups, we propose a comprehensive framework for system development. Utilizing a layered structure that examines the system at different levels of

abstraction, our framework standardizes the underlying system development process in a systematic manner that provides the following improvements when compared to current methodologies:

- Standardizes knowledge transfer between domain experts and system developers
- Records knowledge transferred between domain experts and system developers
- Minimizes reliance on source code to extract knowledge
- Promotes modifiability and extensibility to support changing business needs
- Improves productivity of the system development team by increasing efficiency while facilitating robustness
- Enables agile development through ability to utilize framework for iterative development

The remainder of this paper is organized as follows. First, we will examine related work and review some of the important issues in software development. Second, we will look at software architecture in practice. Third, we shall describe our framework in detail. Next, we will discuss a test project utilizing the framework. Finally, we shall discuss our plans for future research.

RELATED WORK

Software Development Methodologies & Modeling

IS development has received attention from researchers as well as from practitioners. Four different phases could be identified in an IS development project: requirement elicitation (analysis), design, implementation (coding), and testing. The importance of such models during system development has been recognized since the 1960s. Consequently, several modeling methods have been developed, such as Chen's Entity-Relationship (ER) diagrams (Chen,1977) and Unified Modeling Language (UML), but these modeling techniques have not been very effective in eliciting requirements during the analysis phase due to a disconnect between the models built and final system code (Wand and Weber, 2002). This disconnect can be attributed to two primary reasons. First, UML is an IT-oriented modeling technique, which makes its use challenging and prone to errors when utilized by domain experts due to their lack of comprehensive IT knowledge. One the other hand, domain experts are generally more accustomed to Business Process Modeling and Notation (BPMN), and although BPMN is similar to UML's activity diagrams, it lacks the robustness and versatility necessary to document all necessary perspectives needed to fully model the systems architecture. In other words, the current system development methodology lacks a standard and efficient modeling technique that enables the transfer of knowledge across domains. To further complicate this, current outsourcing trends often separate the domain experts from the IT experts causing additional knowledge transfer difficulties due to cultural differences among globally dispersed team members (Krishna, Sahay and Walsham, 2004).

Secondly, many system development projects still adhere to antiquated waterfall models that lack an iterative process that better supports changes to the requirements. In such cases, last minute changes are generally integrated into the source code without updating the associated models and documentation, which leads to a disconnect between the design documentation and the actual implementation.

Due to these problems, it can be concluded that there is an immense need for a holistic formal framework that enables domain expert to effectively capture business rules and knowledge in a format that can be easily translated into a system design. Such a framework would standardize the transfer of knowledge from domain experts to IT experts thus enabling domain experts to better elicit the necessary requirements and translate these with the IT experts into a reasonable design for the system.

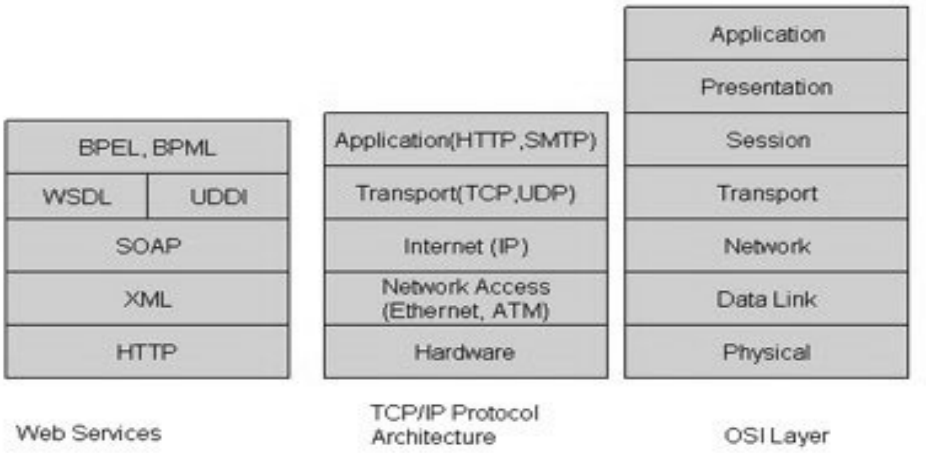
Recently, Hevner, March, Park and Ram (2004) discussed the importance of conducting design science research, and they observed that research into design methodologies was lacking in the MIS community. Additionally, Bajaj, Batra, Hevner, Parsons and Siau (2005) showed that although "System Analysis and Design" (SA & D) appears in almost every IS curriculum where it is considered a core IS course, only 3% of research articles in IS journals are devoted to SA & D. The observation of this teaching-research gap by Bajaj et al. causes an inability to keep SA & D courses up to date. Therefore, our research attempts to fill the aforementioned gap by providing research into SA & D using a decision science approach.

Software Architecture

When documenting the software architecture, four types of architectural drivers are typically defined that correspond to the system's requirements: functional requirements, nonfunctional requirements, business constraints, and technical constraints. Functional requirements are generally the easiest to elicit from stakeholders and refer to the overall functionality of the system. For example, a e-commerce system might include functional requirements, such as catalog of products searchable by consumers, a shopping cart, shipping calculator, and payment processing. The business constraints refer to the requirements imposed by the organization, which includes budget, delivery time, number of developers, documentation policies, compliance requirements, etc. The technical constraints refer to any technology that must be used or supported by the system, including hardware support, programming languages, software support, etc. For the e-commerce solution, perhaps the system must be deployed on a Tomcat web server and implemented using a Java frontend combined with a MySQL backend. The nonfunctional requirements or quality attributes are the most challenging of these architectural drivers and refer to the properties that the system must have, including performance, security, usability, availability, extensibility, modifiability, etc. Although domain experts may understand that the system

needs to be secure, it is often challenging to express the level of security in a manner that is quantifiable and testable. For instance, the security may be defined as the number of unauthorized accesses to the system in a period of time, such as no more than 1 unauthorized access per month. Establishing reasonable nonfunctional requirements and translating these into the design of the system is extremely challenging.

FIGURE 1
Examples of Layered Architectures in Networking and Web Services



Due to the complexity of software architectural documentation and the need to ensure its completeness, there is yet another layer of disconnect between the domain experts and IT experts. Business processes and views are typically only evaluated by domain experts from a dynamic perspective, and they generally do not concern themselves with how the system is implemented or what hardware is needed to support the system. As a result, lack of knowledge about static and physical perspectives hinder the ability of the domain experts to comprehensively identify the necessary components of the system. Furthermore, the difficulty in establishing nonfunctional requirements compounds this problem. As a result, it is frequently necessary to change a system, which leads to cost and budget overruns. Additionally, changes that must be made to the system during later phases of the development lifecycle cost exponentially more than changes made during earlier phases (Boehm, 1981).

METHODOLOGY

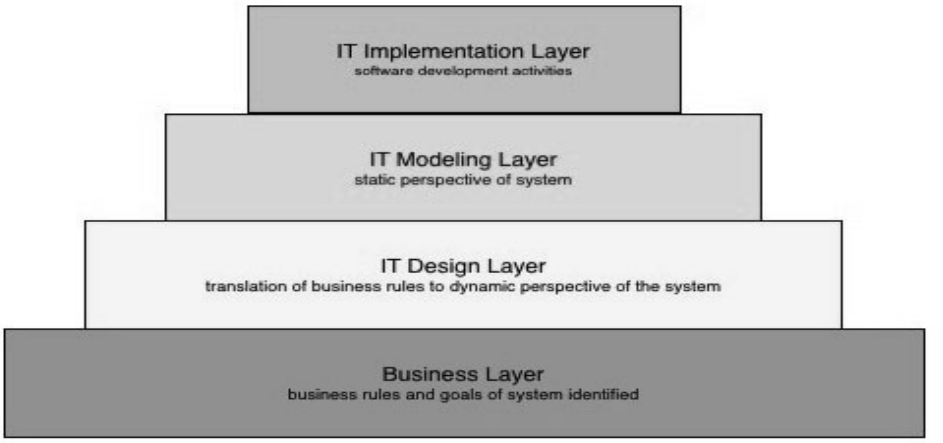
During the last forty years, many different system development methodologies have been adopted, such as structured analysis, object-oriented analysis and design (OOAD), component based software development, etc. (Blahaand Premerlani, 1998; Vitharana, 2003). Although there are important differences

among the aforementioned methodologies, concepts, such as information hiding, modularity, reuse of code, and adoption of architectural design patterns remain common features in most methodologies (McConnel, 2000; Wasserman, 1996). Consequently, we recognize that any system development framework should be based on those concepts. Furthermore, layered structures are common in most companies. For example, consider the management structure of an organization, where employees can be organized in layers that correspond to their job responsibilities. For example, the CEO takes an overall view of the company, whereas lower level employees concentrate on specific jobs. Similarly, layered structures have been successful in other areas, including networking, database design, web services, protocols, operating systems, etc. Consequently, we reviewed three areas where the concept of layering has improved understanding and enabled the construction of effective and interoperable systems.

First, in the networking world, the seven-layer OSI model and the practical four-layer TCP/IP model have enhanced the understanding of networking concepts through the separation of concerns. This allows system developers to focus on smaller pieces of overall problems and encapsulates implementation details. For instance, when developing components belonging to the Internet layer, developers can develop applications that utilize packets for data transmission without knowledge of the how each bit is transmitted. Alternatively, developers working in the hardware layer implement the efficient transmission of bits without a need to consider the implementation of the user interface. In the web services domain, utilizing different levels of abstraction to conceptualize the service enables developers to identify the business needs efficiently, while simultaneously ensuring that individual components of the system can adopt different protocols without cumbersome integration issues.

In our framework, we propose a layered model of system development. Layering provides the following advantages in system development. First, it simplifies the complexity of system development by using information hiding. This is achieved through the adherence to the principles of layered architecture where a given layer can only access components within its own layer or from the layer immediately below it. All other layers are invisible and their implementation does not matter to the current layer. By providing this structure, a system developer or a domain expert focuses only on a small subset of the system and can complete the required activities without needing to worry about the overall system. Second, dependencies among layers are minimized and the design is modular and supports reusability. The framework can be utilized in an iterative fashion, which better supports agile methodologies. Finally, the system development process is completed in a systematic and standardized fashion. The four layers are shown in Figure 2. The framework is composed of the following layers:

FIGURE 2
Layered Framework for System Analysis & Design



Business Layer: The business problem is identified and described allowing a domain expert to represent business logic, business rules, and business processes unambiguously. This requires the identification of all functional requirements. Additionally, any business constraints will be identified. Activities in this phase will be driven by the domain expert with the support of the IT expert will also be involved.

IT Design Layer: Using the information from the business layer, the IT expert will translate the business processes and rules into a dynamic perspective of the system. This requires the identification of nonfunctional requirements, and feasible tests to ensure adherence to these. Assuming the use of UML, this would include the construction of use cases and activity diagrams or sequence diagrams. Additionally, the identification of persistent elements would be completed and an initial ER diagram constructed. Activities in this phase will be driven by the IT expert with the support of the domain expert, and most of the knowledge transfer from the domain expert to system analyst will happen here.

IT Modeling Layer: Using the information generated as part of the IT modeling layer, the detailed design of the system is made by system developers. This includes a static perspective of the system as well as the modification of any dynamic perspectives. Some of the IT artifacts produced in this layer would be class diagrams, package diagrams, data dictionaries, and database schemas. Activities in this layer are driven by IT experts.

IT Implementation Layer: This layer depicts the actual implementation of the system. Technical constraints must be identified here, and a physical perspective of the hardware needed for the system must be completed. Some of the artifacts which belong to this layer are source code, executable, external libraries,

software dependencies, and hardware. Activities in this layer are driven by IT experts, specifically software developers.

USING THE FRAMEWORK

Business Layer

In this section, we discuss the details of the business layer of our framework. This layer is not technology dependent, and it captures the business processes that we wish to solve through system development. One of the main functions of business layer is to identify and document the purpose for building an IS as well as the full functionality of the IS. It also is necessary to capture all the business rules, which determine the workflow of the business process we are trying to map through this system.

The framework consists of six concepts, and no hierarchy exists among these artifacts since each concept belongs to the same layer. However, there are interdependencies among these concepts. These six concepts and their dependencies should be documented in a manner that is conducive to migration into written use cases. The six concepts that compose the business layer are discussed below.

Business Work: Since most IT systems are designed in order to automate or solve a specific business problem, we define business work as an overall view of the business problem described using plain English. Business work helps users understand an overview of the system being developed. As an example, business work could be "Ticket Purchase."

Business Process: A business process is defined as a clearly identifiable workflow, which has a specific business meaning. In most businesses, a workflow evolves as a set of well-defined steps for achieving an objective. As we discussed earlier in this paper, there is often variability in business processes. As an example, let us consider an organizational process "Ticket Purchase". A ticket could be purchased through various ways such as the Internet, window, phone, etc. Each of these correspond to a different business process yet achieve the same overall goal of purchasing a ticket. Although there is similarity among those business processes, there are also be important differences. As a result, in order to describe business processes, it is important to first quantify the variability in the process using the methodology suggested by Pentland (2003). The business processes, which are variable, are complex, and hence, those processes need to be modeled more carefully. For representing business processes, several methods are available, but Business Process Modeling Notation (BPMN) appears to be most popular and very effective. BPMN is one of the three specifications that Business Process Management Initiative (BPMI) has developed. The other two are Business Process Modeling Language (BPML)

the standard business execution language and Business Process Query Language (BPQL) a standard management initiative.

Activity: An activity is a simple atomic task or a collection of tasks and is defined as a series of steps needed for accomplishing a business process. If a person is purchasing ticket using the Internet, an activity involved is "Processing Credit Card Transaction". An activity will be described using plain English, and it should be easily understandable. Although there are instances where an activity is not related to a business process, the activities should be identified by first inspecting the business processes. Then, one should look outside the process for additional activities. In many cases, activities can be broken down into sub-activities. In such cases, it is up to the modeler to determine the level of detail needed to sufficiently document an activity.

Actor: An actor is either a person or a device (equipment), and it plays an active role in a business process. An actor initiates or participates in or reacts to an activity. An example of an actor is a customer of a business or a truck, which transports goods in a "Supply Chain". An actor could have many instances and usually has information associated with it.

Event: An event is a significant occurrence in time or space (Eriksson and Penker, 2000) or in other words, a particular, specific, and unique instance of an activity. The main difference between an event and an activity is that there is only one specific instance of an event, whereas there can be many instances of an activity. In most cases, an event will have specific start time and specific end time. Hence, making a backup of database is an activity; however, making a backup of a database on Friday at 5:00 PM is an event.

Business Objects: Business objects are defined as either concepts or documents that are used for conducting business. Business objects cannot initiate or be active participants in an activity; however, they can be used in business processes. Some examples are tickets, invoices, purchase orders, etc. Business objects contain the necessary information pertaining to physical objects without containing any operation or activity.

Once the aforementioned 6 concepts are identified, a business model of the system using BPMN is constructed.

Transformation to IT Design Layer

In order for this framework to be effective, it should be straightforward to move from a lower layer to higher layer. Knowledge is transferred from domain experts to IT experts, and a dynamic view of the system is generated. The IT design layer transforms the artifacts from the business layer into IT design layer artifacts as follows:

- Inspect all actors and identify the actors, which are outside the system. These actors will map to the actors in the use cases.
- For each of the actors, identify the activities they participate in.
- Using the above information, draw use case diagrams, and complete written use cases.
- Map activities and business processes to a dynamic perspective of the system. Identify nonfunctional requirements and adopt design patterns that are conducive to these quality attributes. If using an object-oriented approach, any objects must be identified and the relationship between these objects.
- Map BPMN diagrams to UML activity diagrams using the standard procedure.
- Identify, business rules embedded in activities. Incorporate business rules into the UML diagrams already drawn and ensure that design adheres to nonfunctional requirements.
- Inspect all the business objects, activities, and connections between activities and actors. Then consolidate the business objects and the activities into objects and classes and then draw the UML sequence diagrams. Some of these objects are persistent, and for those objects, create ER diagrams in order to begin database design.

Transformation to IT Modeling Layer

Once the IT design layer artifacts are generated, we transition to the IT modeling layer. This involves using the dynamic perspectives and persistent models from the IT design layer to construct static perspectives of the system and update current dynamic models. In order to transition from the IT design layer to the IT modeling layer, the following tasks are completed:

- Objects identified in IT design layer are formalized into classes. Class hierarchies are established, and instance variables and methods are defined.
- Using dynamic perspectives, create static perspectives of the system, including class diagrams.
- Adopt design patterns that ensure adherence to nonfunctional requirements for static perspectives.
- Use ER diagram to construct data dictionary and database schema
- Adjust dynamic perspectives if needed to support static perspectives.

Transformation to IT Implementation Layer

In order to complete the system, we must transition from the IT modeling layer to the IT implementation layer. In this layer, we must identify all technical constraints and implement the actual system. In order to transition from the IT modeling layer to the IT implementation layer, the following tasks are completed:

- Technical constraints are identified, including programming language, IDE, operating system, libraries used, software dependencies, and hardware.
- Physical perspective constructed that depicts the hardware aspects of the system.
- Using the models completed during the IT modeling layer, source code is written.
- Unit testing is completed.
- Integration testing is completed.
- Completed system is deployed.

TEST CASE: CONFIGURATOR

Project Setup

In order to demonstrate the utility of the framework, we tested the framework on a real software project. The project was completed for a company who sells electrical components and control systems. The ultimate goal of the project was to rebuild their online product configurator. For each product offered by the company, there are up to ten configuration choices that must be selected in order to build the appropriate product. For example, to configure a push-to-test pilot light, the user must select the operator, operator type, voltage, lamp type/color, clamp ring, lens type, lens color, and options. Each one of these component selections, effects the available component choices for the other options, thus requiring some way to manage rules governing the availability of component options.

The project team was composed of 18 team members, and the estimated time to complete a working prototype within 16 weeks from the start date. In order to accommodate these requirements, we utilized a modified Scrum methodology to complete the project as follows:

- 2 sprints of 2 weeks for the Business Layer
- 2 sprints of 2 weeks for the Design Layer
- 1 sprint of 2 weeks for the Modeling Layer
- 3 sprints of 2 weeks for the Implementation Layer

The project kickoff focused on the business layer, and the team gathered requirements through interviews with employees and an onsite visit to the company as part of the first 2-week sprint. Once prioritized requirements were elicited, the team completed the second 2-week sprint, which focused on identifying the six concepts of the business layer, including business work, business process, activities, actors, events, and business objects. These concepts were then used to construct a BPMN model of the configurator, which in conjunction with the concepts served as the interface connecting the business and design layers.

As part of the design layer, the BPMN was mapped into written use cases and use case diagrams as part of the first 2-week sprint. As part of the second 2-week sprint, the team constructed the dynamic perspective of the system as well as an initial ER diagram showing the construction of the database. As part of the dynamic perspective, the team generated several activity and sequence diagrams, which in conjunction with the ER diagram served as the interface between the design and modeling layer.

Using the dynamic views from the design layer, the modeling layer was initiated and completed as part of a single 2-week sprint. The team used these artifacts to generate the static perspective of the system. The deliverables for this sprint included a class diagram of the system, data dictionary, and database schema. The team also chose to adopt the Model View Controller design pattern in order to ensure that nonfunctional requirements were met. The deliverables from this layer constitute the interface connecting the modeling layer to the implementation layer.

Using the interface components from the modeling layer, the implementation layer was completed. As part of the first 2-week sprint, the team identified the technical constraints and generated the physical perspective of the system. The project would be written in PHP and Python in order to accommodate the company's reliance on Wordpress. Several rule engines were tested, and the decision to utilize Pyke was eventually reached. During the second sprint, the source code for the system was written. Since there was a backlog from the previous sprint, the last sprint focused on the completion of the backlogged source code implementation as well as testing.

Project Evaluation

Upon completion of the project, we asked both the team members as well as key contacts within the company that the team developed the configurator system for to complete a brief survey. The survey consisted of the following statements whereby respondents answered using a 5 item likert scale consisting of Strongly Agree, Agree, Neutral, Disagree, Strongly Disagree:

1. The team fully understood the underlying domain concepts.
2. The prototype meets requirements and accomplishes goals.
3. The framework was easier to utilize than previous development methodologies.
4. The company was able to efficiently and effectively transfer business knowledge to the team.
5. I would utilize the framework for future projects.

The responses to the survey are summarized in Table 1. As one can see, the responses were positive. None of the responses were marked as Strongly Disagree, and very few Disagree responses were seen. Although this is a rather subjective evaluation, it lends credibility to the framework.

TABLE 1
Summary of Survey Responses from Configurator Team and Company Contacts Evaluating Framework

Summary of Survey Responses					
	Strongly Agree	**Agree**	**Neutral**	**Disagree**	**Strongly Disagree**
Question 1	12	6	2	0	0
Question 2	15	5	0	0	0
Question 3	10	8	1	1	0
Question 4	8	12	0	0	0
Question 5	10	7	2	1	0

To further understand the utility of the framework, we also spoke with the company executives and asked for feedback. One company contact responded that they were "quite impressed with the project overall." Another said that they would "very much like to utilize the team for future projects." They also commented on the level of understanding that the team was able to accomplish in a very short period of time. These comments all lend credit to the framework.

FUTURE WORK

This work will be extended in the following way. First, a more comprehensive model for a system using the framework will be developed. Second, we shall do another laboratory experiment in order to test the usability of the new framework. Another interesting area where this work could be extended relates to the work done on patterns. Gamma, Helm, Johnson, and Vlissides (1994) introduced the concept of "Design Patterns" and solved common problems using robust solutions based on patterns. Fowler (1997) identified similar patterns, which occur in the analysis phase of software development, and he named them "Analysis Patterns." Similarly, further research could be undertaken in our "Business Layer" to uncover business patterns in a similar way, which would assist domain experts in modeling the business layer of an IS project.

CONCLUSION

In this paper, we investigated a problem frequently faced by domain experts. Specifically, domain experts need to participate in system development, but there is neither a guideline nor a methodology to effectively permit their participation. Our work adds to the current literature in the several ways. First, it identifies disconnect between domain experts and IT experts as an interesting problem. In addition, it proposes a comprehensive layered development framework, which approaches the system development process using different levels of abstractions. Furthermore, the proposed framework incorporates common technologies, such as BPMN and UML. Therefore, it does not require the use of any unfamiliar or

new modeling tools, but rather it shows how the existing modeling tools, such as BPMN and UML can be used to better model a system. Similarly, it uses the existing facility to convert BPMN to UML. Finally, it proposes how to convert business models into an actual IS implementation.

REFERENCES

Alter, S. 2005. The Work System Method: Confronting a Void in Systems Analysis and Design. In*Proceedings of the Fourth Annual Symposium on Research in Systems Analysis and Design*, April 23-24, Cincinnati, Ohio.

Bajaj, A, Batra, D., Hevner, A., Parsons,J., and Siau, K. 2005. Information Technology and Systems – I Systems Analysis and Design: Should We Be Researching What We Teach? *Communications of the Association for Information Systems*, (15:1), 478-493.

Blaha, M. and Premerlani, W. 1998. *Object-Oriented Modeling and Design for Database Applications*, PrenticeHall, Upper Saddle River, NJ.

Boehm, B.W. 1981. *Software Engineering Economics*, PrenticeHall, Upper Saddle River, NJ.

Chen, P. 1977. The Entity Relationship Model: A basis for the Enterprise View of Data. In *Proceedings of National Computer Conference*, AFIPS Press, 77-84

Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Merson, P., Nord, R., and Stafford, J. 2011. *Documenting Software Architectures: Views and Beyond (Second Edition)*, Addison-Wesley, Boston, MA.

Eriksson, H. and Penker, M. 2000. *Business modeling with UML: Business Patterns at Work*, John Wiley & Sons, New York, NY.

Dennis, A., Wixom,B.H., and Tegarden, D. 2005. *System Analysis and Design: An Object-Oriented Approach with UML*, Wiley, New York, NY.

Evans, E. 2003. *Domain-Driven Design: Tackling Complexity in the Heart of Software*, Addison-Wesley, Boston, MA.

Fowler, M. 1997. *Analysis Patterns: Reusable Object Models*, Addison-Wesley, Boston, MA.

Frankel, D.S. 2003. *Model Driven Architecture: Applying MDA to Enterprise Computing*, Wiley, Indianapolis, IN.

Gamma, E., Helm, R., Johnson,R., and Vlissides, J. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Boston, MA.

Hevner, A., March, S.T., Park, J., and Ram, S. 2004. Design Science in Information Systems Research.*MIS Quarterly*(28:1), 75-104.

Koufaris, M. 2002. Applying the technology acceptance model and flow theory to online consumer behavior.*Information Systems Research*(13:2), 205-223.

Krishna, S., Sahay,S., and Walsham, G. 2004. Managing Cross-Cultural Issues In Global Software Outsourcing.*Communications of the ACM* (47:4), 62-66

McConnell, S. 2000. The Best Influences on Software Engineering. *IEEE Software*(17:1), 10-17.
Pentland, B.T. 2003. Conceptualizing and Measuring Variety in Organizational Work Processes.*Management Science*(49:7), 857-870.
Vitharana, P. 2003. Risks and challenges of Component-Bases Software Development.*Communications of the ACM* (46:8), 67-72.
Wand, Y. and Weber, R. 2002. Research Commentary: Information Systems and Conceptual Modeling – A Research Agenda.*Information System Research* (13:4), 363-376.
Wasserman, A.I. 1996. Toward a Discipline of Software Engineering.*IEEE Software* (13:6), 23-31.

About the Authors

Amber Lynn McConahy is an Instructor of Information Science and Technology at Penn State Beaver. Prior to arrival at Penn State Beaver, Amber taught graduate level courses as an adjunct at Carnegie Mellon University. Although she has taught a wide variety of courses in both computer science and information systems, her specialty is software architecture, specifically platform-based socio-technical ecosystems. She was awarded with highest distinction a Masters in Information Technology from Carnegie Mellon University. She also has a Bachelor's of Science in Information Science and Technology with a focus in software design and development and an Associate's of Science in Information Science and Technology with a focus on networking. Both degrees were awarded with highest distinction from Penn State University. In addition, she served as a research scientist at Bosch giving her experience in industry as well as academia. Currently, she is working on research on platform identification in emergent ecosystems as well as ubiquitous systems.

Abhijit Dutt is an Assistant Professor of Information Systems at School of Business, George Mason University. He has a PhD in Management Information Systems from University of Wisconsin-Milwaukee. Before coming to academia, he worked for more than ten years in companies such as Motorola, Tellabs and Cabletron; he developed software for different networking devices. He was also an investigator of a NSF grant on Enhancing Distance Learning in Information Security. Dr. Dutt has presented many research papers in peer reviewed IS conferences such as ICIS, AMCIS etc. Currently he is working on research papers on managerial issues in cloud computing.