Software Development: Informing Sciences Perspective

Boris Roussev and Yvonna Rousseva
University of the Virgin Islands, St. Croix, USA

brousse@uvi.edu yrousse@uvi.edu

Abstract

This work examines the process of software development from Informing Sciences (IS) point of view. We explore how the three IS precursor theories--Shannon and Weaver's Model of the communication process, Leavitt's Change-equilibrium model, and the "Meta-approach" to modeling--justify a model-driven approach to software development and we argue that modeling through abstraction, metaphoric mapping and metaphoric comparison is the link holding together the three cornerstone theories of IS.

Keywords: Informing Sciences, metaphor, abstraction, modeling, software development, model-driven approach.

Introduction

Eli Cohen (1999) defines Informing Sciences (IS) as "the field of inquiry that attempts to provide the business client with information in a form, format, and schedule that maximizes its effectiveness." This definition implicitly and consistently links IS to all its reference fields: business (management, accounting, and economics), computing (technology), and systems (organization). Cohen expands the definition to form a framework for the study of IS by relating it to three theories: Shannon and Weaver's Model of the communication process (1949), Leavitt's Change-equilibrium model (1965), and the "Meta-approach" to modeling (Cohen, 1999), all seen as IS precursors. In this work, we examine the process of software development from IS point of view. We explore how the three IS cornerstones justify a model-driven approach to software development and we argue that modeling through abstraction, metaphoric mapping and metaphoric comparison is the link holding together the three cornerstone theories of IS.

To shed light on the importance of modeling we analyze the relationships between modeling, metaphor, abstraction, and active learning. We will build the argument that since modeling is based on the mechanics of metaphor application and abstraction, modeling evokes the same cognitive process as metaphor. We discuss the role of modeling as a mental tool for understanding system domains and as a creative device through which new knowledge is constructed.

We relate Shannon and Weaver's Model of the communication process to modeling end users' requirements for software intensive systems. This relationship emphasizes the importance of end users' requirements and justifies the need for a requirements-driven software development process.

Then, we show that model transformation is a viable approach to systems development because it preserves the properties of the requirements model, and therefore, it helps decrease the

entropy of the end users. We define the connection between traceability and change-equilibrium. We prove that the change-equilibrium effect affords change management through round-trip engineering. Speaking metaphorically, one can say, that model transformation and traceability are the universal laws of software development.

Finally, we consider two applications of the meta-approach to modeling, the third IS precursor: (1) the definition of precise syntax and semantics of modeling languages; and (2) the creation of meta-level blueprint of the designed system as a viable approach to curbing the inherent complexity of system design. We stress once again that central role of modeling for cost-effective software development.

The rest of the paper is structured as follows. Section 2 presents the notions of metaphor, abstraction and modeling and discusses their role in constructing new knowledge. It shows how models function through metaphoric comparison, metaphoric transfer, and abstraction. Next, Section 3.1 examines the relationship between requirements modeling, traceability and model transformation and Shannon and Weaver's model of the communication process. Then, Section 3.2 relates Leavitt's Change-Equilibrium model to the "4+1 view" software architecture model. Section 3.3 deals with the implications of adopting a "meta-approach" to modeling. The following section discusses the driving forces behind the paradigm shift in software development from lines-of-code to architectural modeling. The final section outlines plans for future work and concludes.

Abstraction, Metaphor and Modeling

What makes a good model or metaphor? In his research on psychoanalysis, Sigmund Freud was inspired by poetry and myths. Poetic language is known to be metaphoric. The classical metaphor example, "My love is a red rose," is infused with a rich semantic potential. It is not a monovalent but rather a polyvalent message. By bringing together in an implicit comparison two seemingly disparate things (we call this modeling) or evoking a subtle analogy (*A* is *B*), the above metaphor conveys the idea that love may be beautiful, passionate, tender, non-possessive, requiring nurture and care, cannot be taken for granted, etc., but also bound to bring on pain and hurt, for roses do have thorns that scratch and cause bleeding. Note the level of abstraction of this signification. The inessential detail is suppressed. Whether the speaker is Dulcinea Del Toboso or Romeo is inconsequential. Note the vast range of meanings, expressed by only two words, that makes for the gain of a new level of intelligibility. This is precisely what we, software developers, need – modeling languages with rich semantics and expressive power – and in all fairness, we do not yet have them to hand.

By challenging and contesting preconceived notions and worn-out concepts, metaphors (models) impart freshness and newness to the familiar and can, therefore, alter our way of looking at things. From metaphoric language "we receive a new way of being in the world, of orienting ourselves in this world" (Valdés, 1991). Metaphors can be seen then as agents (re)activating cognition. According to Ricoeur the purpose of metaphorical language is neither to improve or facilitate communication, nor to ensure clarity of argument, "but to shatter and to increase our sense of reality by shattering and increasing our language" (Valdés, 1991). Susanne Langer has said that metaphor is the fundamental process by which language grows and adapts itself to the changing world. To Embler (1966) metaphors are the principle organization by means of which we sort out perceptions, make evaluations, and guide our purpose (p. 27). We see metaphors to be at the heart of the process of (re)cognition.

There is general understanding that the notion of metaphor can be brought to bear on the question of how prior knowledge organizes the comprehension and the process of exploring new objects of knowledge, i.e., how it unlocks the floodgates of knowledge. Carroll and Mach (1985) define metaphor as a kernel comparison statement whose primary function in exploring a new domain is

to stimulate active explorer-initiated thought-processes. By jolting perception out of the old cognitive framework, metaphors incite thought, i.e., create a new condition for knowledge. *Metaphor* is a comparison exhibiting a high degree of relational similarity with little attributal similarity (Gentner & Markman, 1997). Holyoak (1985) developed the so called pragmatic account of metaphor. This approach emphasizes the role of pragmatics in analogical transfer, i.e., how current goals guide the interpretation of a metaphor.

Metaphors are employed in analogical reasoning as well as in analogical problem solving. In reasoning, analogical transfer is by inference, the so-called explanatory metaphors (Gentner, 1983). In contrast, in problem solving, analogical transfer is by adaptation (Keane, 1996). It has been found out that in understanding a new object of knowledge, people tend to resort to less conclusive and less constrained abductive and adductive processes (Lewis & Mack, 1982; Carroll & Mack, 1985) rather than to the more rigorous inductive and deductive processes. This could be explained with the fact that abduction opens up the possibility of discovering knowledge on the basis of limited information.

Metaphors facilitate active studying by providing vital clues for abductive and adductive inferences through which people construct knowledge. One begins with a metaphorical statement, "A is B", shown in Figure 1, where B is an object of pre-existing knowledge. From this, a person can generate and test a variety of hypothesis about A, that is, she/he can study actively A. Metaphor, in this respect, is an innate manifestation of constructing new knowledge as an active process. Carroll and Mack (1985) argue that in metaphoric comparisons the structural mappings between the target and the source domains afford cognitively constructive processes which can lead to new knowledge as problems people pose to themselves. To sum up, in constructing new knowledge, metaphors are important in the transfer of knowledge and of inferences across different concepts or domains. Metaphors serve as mental models for understanding new domains. Therefore, metaphors enable cognition.

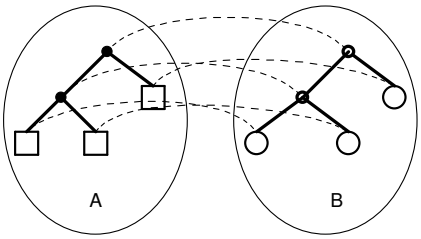

Figure 1. Structure mapping in metaphorical comparison

Abstraction is a fundamental principle in systems development. The general idea is to hide the details of a complex system while providing a simpler "cover story" (model) to explain the system's aspects of interest (Bucci, Long, & Weide, 2001). The system model abstracts from (in the sense of suppress) irrelevant real world details, as well as from implementation details as shown in Figure 2 in an effort to convert the original problem/system to a simpler problem/model. When done properly, abstraction blazes the trail for new ways of thinking about the system being abstracted. The abstract representation enables us to think about the external behavior of the system through the use of terms that are fundamentally different from those in which its internal details are explained (Norman 1994). Norman's ideas can be seen as directly foreshadowing the work of Zave and Jackson (1997), linking systems models to user requirements.

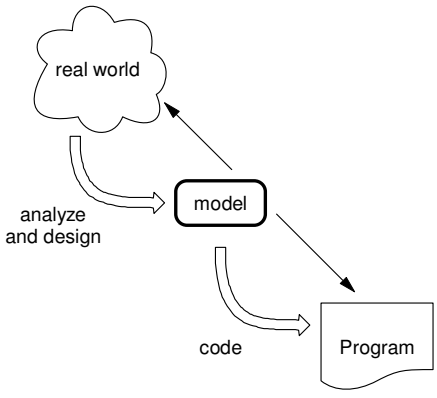

Figure 2. Role of abstraction in systems development

A *model* is a simplified representation of a real world domain and, as such, a model includes only those aspects of the domain that are of interest to the designer creating it. For example, a statechart model of a class (Jacobson, Booch, & Rumbaugh, 1999) allows software developers to analyze only the dynamic behavior of the class. The attribute structure, serialization, and concurrency do not affect the dynamic properties of the class. The reduction in scale and complexity achieved by statechart modeling affords the analysis of properties such as homing sequence, reachability, and reversibility. This reduction in scale and complexity is made possible through abstraction. A model is related to the real world domain it represents through an abstraction function. The predictive power and validity of the model is based on a kernel metaphor, mapping the modeling elements to its real world counterparts. The model, then, is an epistemological metaphor. Creating models deepens our understanding of the designed system. Modeling allows developers to capture subtle nuances before they have turned into bugs (Mellor & Balcer, 2002). The distinction between models and metaphors resides chiefly in the open-endedness, incompleteness, and inconsistent validity of metaphoric comparisons versus the explicitness, comprehensiveness and validity of models. However, this contrast is not a dichotomy. Models are always grounded in a kernel metaphor, or metaphors, and hence stimulate many of the same cognitive processes.

The rose metaphor could be seen to be both a model and analogue of love, exemplary of love, both love's model and representation, and an instance of it. There is some correspondence, similarity, kinship between the two and a transfer across sites, a shift to a different order. This also holds good for software models, e.g. UML. The analysis level model of a system is both a model and an integral part of the system's blueprint, i.e., an instance of the system.

Relation between IS Precursors and Modeling

In this section, we examine the relationship between modeling and the three cornerstone theories of IS.

Shannon and Weaver's Model of the Communication Process

The first theory, considered by Cohen, is Shannon and Weaver's model of the communication process. Shannon and Weaver (1949) define information as a reduction in uncertainty. For a system to be of value to its users, it has to reduce the user's (we equate receiver with user) level of uncertainty. The implications for systems development are profound and, unfortunately, only recently well understood.

Modern software development processes, e.g. the Unified Process (UP), are driven by user requirements (Jacobson et al., 1999). The goal of UP is to guide the developers in developing in cost-effective way systems that meet customer needs. Customer requirements are modeled as use cases. The use case model describes the complete functionality of the system. In UP, the software architecture of the system includes the most important static and dynamic aspects of the system. The architecture grows out of the use case model iteratively, each iteration resulting in an increment.

In contrast, traditional structured methodologies (Fichman & Kemerer, 1992) promote the theory that the database is the backbone of the system and everything revolves around that database (Naiburg & Maksimchuk, 2002). Structured analysis views a software system as a collection of data and separate functions that operate on the data. While it is true that much of the information lives within a database, the database cannot stand on its own. Without customers and transactions, there would be no information for the database. Without a business, there would be no reason to have a database. A software system is brought into existence to serve its users. And it serves its users only when it reduces their entropy. How to guarantee a software system bringing value to its users?

First, the system's functional description (requirements model) must be sufficiently user-friendly to all stakeholders and it must facilitate effective communication among them. Adequate and precise user requirements can be elicited only when end users are actively engaged in the software process. The requirements model will be user-friendly if the modeling language employed is expressive, precise and understandable to all stakeholders. The modeling language must allow for describing stakeholder terminology, concepts, viewpoints and goals. The requirements should contain nothing but information about the (user) environment (Zave & Jackson, 1997). To say anything about the system is regarded as implementation bias. If a statement is implementation-biased—it cannot be validated, i.e., proven to be an accurate account of stakeholder requirements—the resulting information system might not decrease the uncertainty of its users. In this respect, it is of utmost importance to specify at interface level and use declarative specifications, e.g., use cases specified as pre-conditions, post-conditions and invariants (Roussev, 2003), rather than programming constructs. Programming is implementation-biased (by virtue of being an implementation) and it produces artifacts that cannot be proven correct with respect to the language of the user.

Second, the developer has to show that the developed software system satisfies the user requirements. Ideally, one would like to prove the implementation model (source code) and the requirements model homomorphic, i.e., related in a mathematically rigorous way. A proven technique for bridging the gap between two models is to use a series of refinements and/or transformations preserving the properties of interest, e.g., bi-simulation and observational equivalence (Milner 1989, p. 106) or UML model transformation and integration (Araújo, 2002). A case in point are Jacobson's work (1992, Jacobson et al., 1999), which is the basis of UP, and Executable UML (xUML) (Mellor & Balcer, 2002). The success of UP and xUML is predicated on the fact that they are defined as model refinement/transformation from more abstract to more detailed and precise models, see Figure 3. Model refinement/transformations allow for user requirements to be traced to the modules of the implemented system, ensuring that user needs are met. In UP, user requirements, modeled as use cases, drive the system design, implementation and test, i.e., they

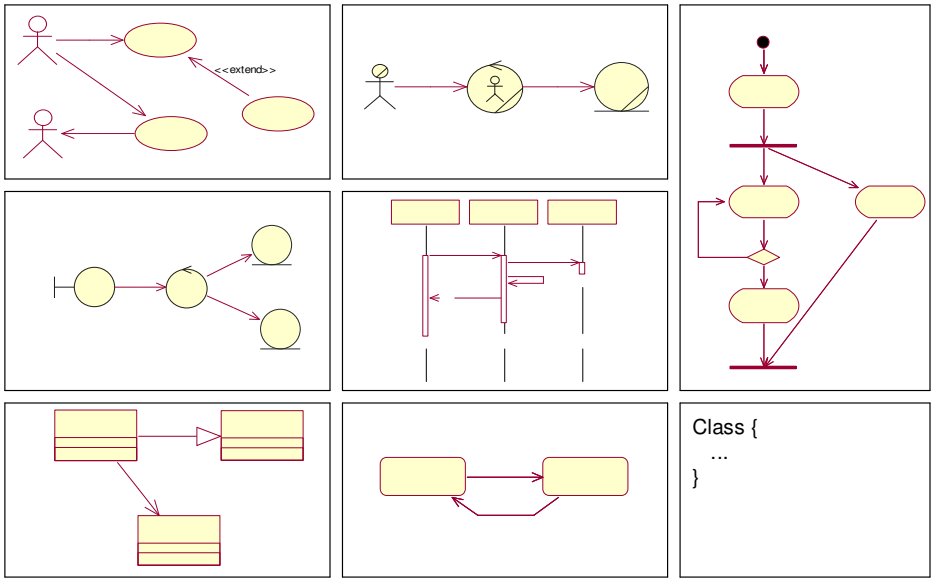

Figure 3. UML and xUML models are related rigorously through morphisms. In UP, design is viewed as refinement/transformation from more abstract models (top left) to more precise and detailed models (bottom right).

drive the whole development process. In short, software must be designed before it is programmed in order to decrease the entropy of its users.

Leavitt's Change-Equilibrium Model

Leavitt's (1965) Change-equilibrium model posits that in order to understand organizational change, four interrelated elements must be considered: the task, technology, structure, and people. This is a clear manifestation of the Newtonian approach (Prigogine & Stengers, 1984). In Newtonian parlance, studying the state of a system means determining the various forces acting on the bodies making up the system. The forces are not orthogonal, and thus vary at each instant as a result of the motion they themselves produce. This problem is expressed as a set of differential equations. The description of a system has two elements: the positions and velocities of each of the bodies in the initial instant and the "universal" laws of motions. The key point here is that the accelerations of the bodies are interdependent, and thus a change in one affects all the rest. In Newtonian dynamics, a system can be studied from different points of view. All these points of view are equivalent in the sense that we can go from one to another by a transformation (a change of variables). The task is to find a representation that is the easiest to solve. We shall draw an analogy to software systems development.

Kruchten (1995) noted that there is no single, elementary view (model) of a system sufficient to describe its architecture; rather, different stakeholders best view a complex system from different perspectives. Kruchten suggests growing a system by advancing each view simultaneously, i.e., developing concurrently the different models (static, dynamic, at different level of abstraction) representing points of entry for different sets of stakeholders, see Figure 3. However, for a system implementation to be feasible, the different models must be consistent and not-contradictory, i.e., the viewpoints must seamlessly relate to each other. Therefore, model integration in general and diagram interchange in particular must be at the core of a successful approach to resolving the tensions among the competing interests of the different stakeholders.

Traceability is defined as an abstraction relationship that relates two models representing the same concept either at different levels of abstraction or from different viewpoints. Traceability is mainly used for tracking requirements and changes across models. In Newtonian models, change is w.r.t. time. In systems development, change is w.r.t. any subset of task, technology, structure, and people. The paramount question is how to deal with change. Change in complex systems cannot be predicted. To predict a change implies solving the paradox "expect the unexpected." Jacobson introduced the Boundary-Controller-Entity architectural pattern to contain changeability through localization of change (Jacobson, 1992). He assumes that changes inevitably originate from system users, and as such, they tend to affect mainly boundary classes. Be that as it may, changes do occur in technology and tasks. However, these changes have different lines of action. Defining a system's blueprint as a set of coherent models, see Figure 3, allows for round-trip engineering, thus making it possible to trace the effect of changes across models both forward and in reverse. To sum up, the major implications of the Change equilibrium model are: (1) model transformation; and (2) traceability. In a way, (1) and (2) are our universal laws of motion.

Meta-Approach to Modeling

The third conceptual development, from which IS derives, is the "meta-approach" to modeling (Cohen, 1999). On the one hand, by defining multiple levels of abstraction, a meta-level architecture curbs the complexity of information systems. On the other hand, the meta-level architecture has emerged as the only satisfactory approach to defining the precise semantics required by complex models. In a meta-level architecture, the semantic constructs of each layer are recursively defined by the layer above it, i.e., one can consider each layer as a semantic refinement of the layer above; conversely, each layer can be considered a semantic abstraction of the layer below.

The main advantages associated with the meta-level approach to modeling are semantic and syntactic rigor in defining modeling elements and heavyweight and lightweight extension mechanisms, which facilitate model transformation and exchange of models across tools. It is not coincidental that the main critique of UML dwells on its semiformal syntax and semantics. Modeling from different viewpoints, successive refinements (cf. Figure 3) and the absence of formal semantics and of formal mapping between UML models give rise to vertical, horizontal, syntactic and semantic consistency problems among models (Engels, Kuster, Heckel, & Groenewegen, 2001).

Discussion

Our analysis of Cohen's evolutionary approach to framing the field of IS sheds light on the shift of the primary focus of software development from lines-of-code to coarser-grained architectural elements, their modeling, their high-level interactions, and their overall interconnection structure. The basic promise of the software architecture-based approach is that better software systems can result from modeling their important architectural aspects early on in the development lifecycle (Medvidovic, Rosenblum, Redmiles, & Robbins, 2002). Taken to the extreme, this approach transcends to model-driven architecture and xUML, where implementation issues are entirely dedicated to target-specific compilers and programming is deemed obsolete and irrelevant.

The paradigm shift from lines-of-code to architecture modeling is based on abstraction and metaphorical comparison. Software development has been redefined as creation of modeling artifacts. A successful software development process starts with requirements modeling whose artifacts ensure that the designed system reduces the entropy of its end users. To guarantee that the captured requirements result in a system meeting its specification, the analysis, design and implementation models must be derived through model transformation from the user requirements, thus creating traceability relationships mitigating the competing view points and models. Traceability allows proving that the system implementation and the system requirements are homomorphic, thus achieving the desired change-equilibrium effect. The ubiquitous "change" is contained by the change-equilibrium effect. For example, a change in the technology can be reverse engineered along the traceability axes, and its effect on the end user requirements or the system's dynamic behavior automatically determined. In order to have the change-equilibrium effect, the different modeling languages employed must have precise syntax and semantics. Precise syntax and semantics are achievable through adopting a meta-level architecture to modeling language specification. Furthermore, modeling artifacts are partitioned into sets of models at different levels of abstraction, thus curbing the software blueprint complexity.

Devlin (2000) points up the significance of math education for software developers. Devlin argues that it is not what is taught in a math class that is crucially important but rather the fact that what is taught is mathematical. It is the process rather than the content that matters in a math class. He goes on to say that mathematics is the sole notable subject congenial to giving experience in reasoning not about the physical or social world but about the abstract world that is uniquely the mind's exemplary conjecture and creation. Well, it may turn out that math would no longer be the only noteworthy contestant in this respect. Although far from yet being a mature discipline, software development has established a solid apparatus conducive to teaching abstract thinking about abstract entities through modeling.

Conclusion

Our analysis of the evolutionary approach to framing the field of IS explained the shift of the primary focus of software development from lines-of-code to architecture modeling. We showed that the modeling movement drives this paradigm shift. We demonstrated that modeling brings and links together the three IS cornerstone theories because it is based on metaphor application, and thus, facilitates active learning about the domain of interest.

Research results from the following three areas will further enhance our results. The forthcoming specification of UML2 is expected to streamline and define more precisely the syntax and semantics for the various UML modeling languages. Adding rigor to the specification will further empower model transformation as a method for software development. The role of end users in software development will grow as more advanced techniques for modeling and validation of user requirements emerge. Last but not least, the integration of economic considerations into software project management, as well as the departure from traditional cost-based metrics and orientation towards models and methods viewing design as an investment activity, whose fundamental goal is to create maximal value added for any given investment, will only help decrease the entropy of the end users (Boehm & Sullivan ,2000).

References

Araújo, J. (Ed.) (2002). Workshop on integration and transformation of UML models. *ECOOP*, Málaga, Spain, June, 2002.

Boehm, B. & Sullivan, K. (2000). Software economics: A roadmap. *Proceedings of ACM Future of Software Engineering*, Limerick Ireland.

Booch, G., Rumbaugh, J., & Jacobson, I. (1999). *The unified modeling language.* Boston: Addison-Wesley.

Bucci, P., Long, T. J., & Weide, B.W. (2001). Do we really teach abstraction? *Proceedings of the 32nd SIGCSE Technical Symposium on Computer Science Education*, ACM Press, 2001, 26-30.

Carroll, J. M., & Mack, R. L. (1985). Metaphors, computing systems, and active learning. *International Journal of Human-Computer Studies*, *22*, 39-57.

Cohen, E. (1999). Reconceptualizing information systems as a field of transdiscipline informing science. *Journal of Computing and Information Technology*, *7* (3), 213-219.

Devlin, K. (2000). *The math gene: How mathematical thinking evolved and why numbers are like gossips*. Basic Books.

Engels, G., Kuster, J., Heckel, R., & Groenewegen, L. (2001). A methodology for specifying and analyzing consistency of object-oriented behavioral models. *ACM ESEC/FSE*, Vienna, Austria.

Fichman, R. & Kemerer, C. (1992). Object-oriented and conventional analysis and design methodologies. *Computer*, *25*, 22-39.

Gentner, D. (1983). Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, *7*, 155-170.

Gentner, D., & Markman, A. B. (1997). Structure mapping in analogy and similarity. *American Psychologist*, *52* (1), 45-56.

Holyoak, K. J. (1985). The pragmatics of analogical transfer. In G. H. Bower (Ed.), *The psychology of learning and motivation: Advances in research and theory* (pp. 59-87). New York: Academic Press.

Jacobson, I. (1992). *Object-oriented software engineering: A use case driven approach.* Boston: Addison-Wesley.

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *The Unified software development process*, Bostson: Addison-Wesley.

Kruchten, P. (1995). The 4+1 view model of software architecture. *IEEE Software*, *12* (6), 42-50.

Leavitt, H. J. (1965). Applying organizational change in industry: Structural, technological, and humanistic approaches. In J.G. March (Ed.), *Handbook of Organizations.* Chicago: R. McNally.

Lewis, C. & Mack, R. L. (1982). The Role of abduction in learning to use text-processing systems. In *Proceedings of the Annual Meeting of the American Educational Research Association*, 19-24, New York.

Medvidovic, N., Rosenblum, D. S., Redmiles, D. F., & Robbins, J. E. (2002). Modeling software architectures in the UML. *ACM Transactions on Software Engineering and Methodology*, *11* (1).

Mellor, S. J. & Balcer, M. J. (2002). *Executable UML: A foundation for model driven architecture.* Boston: Addison Wesley Professional.

Milner, R. (1989). *Communication and concurrency*. Hertfordshire, UK: Prentice Hall.

Naiburg, E. J. & Maksimchuk, R. A. (2002). *UML for database design*. Boston: Addison-Wesley.

Norman, D. A. (1994). *Things that make us smart: Defending human attributes in the age of the machine*. Perseus Publishing.

Prigogine, I. & Stengers, I. (1984). *Order out of chaos*: *Man's new dialogue with nature*. Toronto: Bantam books.

Roussev, B. (2003). Generating OCL specifications and class diagrams from use cases: A Newtonian approach. *36th Hawaii Int'l Conference on System and Sciences*, *HICSS-36*, Hawaii.

Valdés, M. J. (1991). *A Ricoeur reader*. Toronto: University of Toronto Press.

Shannon, C. E. & Weaver, W. (1949). *The mathematical theory of communications*, Urbana, IL: University of Illinois Press.

Zave, P. & Jackson, M. (1997). Four dark corners of requirements engineering. *ACM Transactions on Software Engineering and Methodology*, *6* (1), 1-30.

Biographies

Borislav Roussev is an Associate Professor of Information Systems at the University of the Virgin Islands. He was educated in Bulgaria, and was previously a faculty member in higher education in South Africa. Dr. Roussev's research interests are in the areas of object-oriented modeling and software engineering.

Yvonna Rousseva is a Visiting Assistant Professor at the University of the Virgin Islands. She graduated from St. St. Cyril and Methodius University, and was previously a faculty member in higher education in Bulgaria. Professor Rousseva's research interests are in the areas of linguistics, poetry, metaphor, and active learning.