# INTEGRATING PROCESS- AND OBJECT-APPROACHES
## *An ontological imperative*

Shivraj Kanungo
*Department of Management Science, The George Washington University, 2115 G Street NW, Washington DC, USA*

Keywords: object, process, structured, ontology, methodology

Abstract: There is an emerging belief about the virtually unanimous agreement that the object-oriented paradigm is superior to the classical (structured) paradigm. We do not accept such unqualified judgments. In this paper, we address the differences from the ontological perspective. We adopt a discursive approach to analysing and discussing the differences, similarities and resolution approaches. We accept the position that object-oriented programming is here to stay and is one of the legitimate silver bullets. Once we contrast the two approaches, we explain how the consumer of the approach perceives its utility. By employing this approach, we highlight the end-user and developer perspectives. We conclude the paper by restoring some perspective on the uncontested superiority of the object paradigm over the classical paradigm. Lastly, we highlight research and pedagogical issues regarding contemporary treatment of structured and object-oriented approaches.

## 1 INTRODUCTION

This paper addresses the relationship between structured or process-oriented and object-oriented approaches, in the context of information systems development. The ontological imperative for addressing this issue lies in the fact that both approaches are important and fundamental ways of viewing the world. One is inherently static and the other dynamic. Both are ontologically independent in that one cannot subsume the other, or be represented in terms of the other, gracefully. Given that some saw the "future of the two paradigms to be settled by experimentation (de Champeaux et al., 1990a)" and given further that questions have begun to emerge over the assumed or imputed primacy of the OO approach (Coad and Yourdon, 1991; Rumbaugh et al., 1991; Jacobson et al., 1992; Schlaer and Mellor, 1992; Booch at al., 1996), it is important that we revisit this issue. The conceptual tool we employ in this research is that of ontological differences that exist between process-based and object-based approaches. The question we address is the following: Is the traditional process of "understand → conceptualise → build" (P1) giving way to "conceptualise → understand → build" (P2)? There is an assumption in P1 that systems can indeed be specified. There is also an assumption that once we understand what to build, we can think of how to build the system (the design process). This assumption implies the decoupling between design and analysis and seems to be premised on the belief that de-linking design and analysis should lead to better conceptual models.

The problem that we describe has turned out to be persistent. Programmers want crisp program specifications and users prefer to communicate in their natural language. The approach that has been adopted all along has been to inject some formalism, standardization and precision into how users communicate. The general idea is that if precision is injected upstream, it will be reflected in the downstream activity of programming.

We start by giving an overview of process-oriented (PO) and object-oriented (OO) approaches to system development. Both have advantages and disadvantages. However, the object-based approach has been entrenched solidly in the programmer's domain. Given this, the essential question that remains is that should this reality constrain the end users' conceptualisation? We then present the importance of the ontological perspective when viewing these approaches. Following that we analyse the PO and OO approaches for their differences, similarities and overlaps.



Kanungo S. (2004).
INTEGRATING PROCESS- AND OBJECT-APPROACHES - An ontological imperative.
In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, pages 237-244
DOI: 10.5220/0002623402370244

# 2 PROCESS AND OBJECT-ORIENTED APPROACHES

System development methodologies can be generally divided into several categories. Two major methodological approaches are structured analysis (SA) also referred to as process-oriented by some authors (Agarwal et al., 1999); and object-oriented (OO) approaches. There are other methodological approaches also. However, they are not widespread. They include data modelling and behavioural modelling. While there is a huge push to adopt OO approaches, the rate of absorption has not been high. This has been attributed to perceptions and organizational and social issues in the larger software development environment (Perry et al., 1994). We describe the two approaches in following sections. While there are many variants of these two approaches, we will concentrate on the archetypal characteristics of these approaches and avoid delving into the finer grained differences that exist within each.

Researchers as well as practitioners recognize the importance of both (process and object) constructs. For instance one of the most prominent proponents of recognizing the importance of both constructs are Dori and Reinhartz-Berger (2003) who clearly borrow from the simplicity of the dataflow diagramming and propose that UML needs to be simplified.

## 2.1 Structured Analysis and Design Methodology

Structured analysis and design is often referred to as the data flow modelling methodology. Structure analysis and design is one of the frequently used approaches for system development. It focuses on the flow of data and it's processing. The set of modelling tools includes dataflow diagram, data dictionary, process specification, entity-relationship diagrams, state-transition diagrams, and structure chart. These tools allow system analysts to: focus on important system features while downplaying less important features; discuss changes and corrections to the user's requirement with low cost and minimal risk; verify that the systems analyst correctly understands the user's environment and has documented it in such a way that the system designers and programmers can build the system.

## 2.2 Object-Oriented Methodology

Object Oriented analysis uses the partitioning of the problem with respect to objects when analysing the problem domain. The concept of object orientation in software development has it roots in Smalltalk, which is a purely OO programming language. Problem analysis is the analysis and description of the real world, its entities, their attributes and their relationships. Therefore, it makes sense to use OO concepts in problem analysis. The primary motivation for object orientation is that as a system evolves, its functions tend to change, but its objects remain unchanged. Thus, a system built using object-oriented techniques may be inherently more maintainable than one built using more conventional functional approaches (Davis 1993).

Object-oriented methodology (OOM) emerged in the 1980s with the popularity of object-oriented languages. OOM uses abstraction as a method of isolating properties that are not relevant to a particular function from those that are. These functions are then distributed among components, or encapsulated. Each software component isolates a single function, therefore hiding that function to users of the object. One of the most important attributes of these components is referred to as polymorphism that is, the fact that a component can take on different forms based on the conditions in which it is operating. This allows greater reuse of objects, whether by sharing, copying or cloning, and adjusting them.

It should be clear from the short descriptions of each methodology that the primary focus for each approach is different. For structured approach it is the process and for the object-oriented approach it is the object. The importance of these differences arises when these conceptualisations start influencing the way an observer (be it an end user, an analyst or a programmer) views the world. There are, in our opinion, ontological differences between the two approaches. These differences, when understood, can account for the meaningful and judicious use of both approaches. We now move on to understand the meaning and role of ontology in the context of system development approaches.

While Dori's approach to integrate these approaches as the OPM is commendable, we develop an argument for a looser coupling between methodology and ontology. We do so because approaches like Dori's OPM may end up suffering from similar problem that UML is facing today (being monolithic as a methodology and hence unwieldy). We also agree with Hughes and Wood-Harper (1999) that most modelling is predicated on the assumption that the system development process is rational and predominantly technical. It is also important to understand that purity of any methodology cannot be guaranteed at the time of practice. As a result methodologies are necessary but not sufficient to ensure the development of valid and

robust information systems. This is because developers possess varying levels of expertise. Yet as Hughes and Wood-Harper observe, "as developer experience increases, there is a real danger that the methodology becomes a fetish at the expense of personal and critical reflection. Moreover, developers form mental constructs to influence the use of methodologies. These mental constructs influence their own sense-making and decision-making activities and explain the ways in which a methodology is used differently by one person as compared with another, and indeed as compared with the original proponents of the methodology (p. 1182)."

# 3 ONTOLOGY AND THE LINK WITH SYSTEM DEVELOPMENT APPROACHES

Ontology is the study of the nature of being. The term "ontology" comes from "ontos," the Greek word for being, and so literally means the study of being. The subject of ontology is the study of the categories of things that exist or may exist in some domain. The product of such a study, called an "ontology," is a catalogue of the types of things that are assumed to exist in a domain of interest D from the perspective of a person who uses a language L for the purpose of talking about D. This paper is about the language and schema associated with abstractions used to support conceptual modelling of real systems in the process of information system development. We are not concerned with developing ontologies for specific domains (Gruninger and Lee, 2002).

From a linguistic perspective, object-oriented and process-oriented methodologies provide their specific "linguistic hints (Steels and Kaplan, 1999)" that enable or nudge a user to recognize in an analytical context something to be an "object" or something else to be a "process." In other words, the OO approach encourages users to think in terms of objects while in the PO approach the primary vehicle to understand a system is to capture the flow of information (and even physical flow).

Why is ontology important for us in this context? As different programming styles influence the way we think about the world and therefore, as, programmers want to extend their languages to suit their problem domains, the consequence is that programmers define new languages all the time. In a similar vein, the language for translating users' requirements into a software product keeps evolving. However, to say that object-orientation is inherently better (as a language or approach) may be going too far. This is important given Yourdon's prediction (deChampeaux, 1990b) that people will not change to OO unless they have to. This brings to question as to which people will shift. There is no question that programmers have made the shift to the OO paradigm. We invite caution here because, while most of the popular compilers of today support object-oriented programming, there is no way to assess how many programmers employ OO practices and to what extent.

# 4 PO AND OO AND ONTOLOGY

Fundamental objectives associated with analysis and programming (important and core aspects of any system development methodology) have long been to standardize problem domain terminology and semantics and provide basis for standardized descriptions of problems to be solved in the domain. In this context it is instructive to understand ontology as (terminology + semantics). Ontology of models is a precise syntactic and semantic definition of what is taken for granted, namely the languages available for expressing the structural, behavioural, and functional models). Table 1 shows a comparison between the OO and PO approaches.

If we analyse whether end users feel more comfortable with classes, objects and instances or activities, processes and data flows, it becomes obvious that, it is the latter. However, a software developer tends to be far more comfortable with classes, objects and instances because it may mirror

Table 1: Contrasting OO and PO approaches

| | |
|---|---|
| **Object-Oriented approach**<br>Booch et al. (1996), Coad and Yourdon (1991), Jacobson et al. (1992), Rumbaugh et al. (1991), and Shlaer and Mellor (1992) | **Ontology**: Classes, objects, instances<br>Class – common properties, behaviours.<br>Inheritance, polymorphism, encapsulation |
| **Process oriented approach**<br>Structured Analysis and Design Technique (SADT) (Ross, 1977), Structured analysis (Gane and Sarson, 1979) | **Ontology**: Activities, processes, data, data sources<br>Hierarchy of activities and processes<br>Representation: Visual. |

or be close to the conceptual primitives that he or she deals with on an everyday basis.

The fundamental differences between OO and PO approaches have been captured in a panel discussion in deChampeaux et al. (1990a). Structured analysis cannot be used effectively to produce the requirements for a system that will be designed and implemented in an OO fashion. While some panellists were clear that PO approaches could not be changed or extended to support OO development, some suggested that adding entity-relationship diagramming and extensions would help. The panel also believed that some application domains are better suited for one kind of approach than the other. The fundamental differences between the OO and PO approaches (as also the need to co-operate) are also captured by Lee and Wyner (2003) who lament that for the most part, however, "system behaviour continues to be modelled using traditional tools such as state diagrams and dataflow diagrams, which remain outside the scope of the specialization hierarchy used to such advantage with objects."

# 5 DISCUSSION

It is important to recognize two questions here. First, is the quality of analysis using OO approaches "better" that that using PO approaches? And if so, for whom? Second, is it fair to impose on the end user an ontology that is meaningful to the programmer, but is not consistent with the end users' worldview or vocabulary? With respect to the second question, Alspaugh and Anton (2001) question the desirability of characteristics of requirements and specifications that are consistent with object-orientation.

Take the example of a new customer in a bank that wants to open an account. When we employ an object-oriented approach to analyse and design an information system for the bank, "customer" and "account" are likely to be the two least controversial classes. Opening of an account would likely be modelled as a link between "customer" and "account." While this is a meaningful implementation view (that would show cardinatilies and connote messages being passed between the two objects), customers do not interact with new accounts. They interact with the branch manager (typically). This is an instance of how design and analysis views are cluttered in OO approach. In this context, it is of interest to quote Jacobson (1994) who claims: "We think it is bizarre to apply the way of thinking that governs computer systems to business processes (p.36)."

For Parsons and Wand (1997) the starting point is that object-oriented design methods use ontologies as domain models for specifying software systems. In doing so, object-oriented analysis may interfere with understanding the domain and drawing attention to implementation concerns. For analysis, representation-based foundations are more suitable than implementation-based approaches. In so arguing the weakness of the implementation driven approach of object-orientation, Parsons and Wand (1997) have argued that representation-based foundations are more suitable than implementation-based approaches. Another instance of such arguments is that the current object-oriented paradigm is driven by implementation considerations rather than conceptual aspects (Artale, 1996). The object-oriented approach emerged as an implementation paradigm, motivated by the objective of building better software more efficiently (Parsons and Wand, 1997).

The implication of such arguments is that conceptualising the whole system in terms of "objects" is relatively far more difficult than to experience coding improvements as a result of object-orientation. For instance, there is no standard equivalent of a context level DFD in traditional OO methodologies. The advantage of DFDs is that they support the notion of emergence and hierarchy. Decomposition that is easy from the end users' perspective is often not supported. In most OO frameworks structural and behavioural decomposition are defined as separate concepts with no or few combinations or relations to each other. Concepts for structural decomposition are e.g. aggregation and composition. Concepts for behavioural decomposition are e.g. composite states for state machine specified behaviour and procedures as part of transitions. While aggregation and composition are specified by special associations between classes, state machines are specified for individual classes. Implications of a structural decomposition on the behaviour and of behavioural decomposition on the structure are often defined rather vaguely.

Similar sentiments have been expressed as expert comments. For instance, Glass (1995) reports that in the context of scientific and engineering realm too "users don't think in terms of objects, they think in algorithms and tasks (p.1)." The direct implication is that object-oriented methodologies do not necessarily lend themselves well to understanding and analysing problems and situations that are inherently process-oriented or have temporal linkages. Consequently, many practitioners depend on "traditional" process-oriented models to analyse business situations and then "translate" those specifications into object-models for

implementation. This is made even more important by the fact that well accepted SRS guidelines and formats continue to follow and adhere to the IEEE standards (IEEE, 1998).

## 5.1 Perspective matters

It makes a difference when we analyse modelling approaches from a specific perspective. Stated differently, a user's view of an information system is very different from the developer's view. It has been stated that OO is natural. The question arises, natural for whom? The OO paradigm sprang from language (Simula and Smalltalk in 1960's, and C++, Eiffel later), matured into design and finally (in the mid- to late 80's) moved into analysis. So it is "natural" for programmers. The OO approach was developed by developers for developers to make interaction with users easier and more meaningful.

Often, the OO approach is presented as a simpler and more efficient alternative to the PO modelling approach. A major problem is that there are many OO approaches. The unified modelling language (UML) has attempted to unify diverse OO methods into a unified standard. However, the UML lacks a system-theoretical ontological foundation (Soffer et al., 2001).One of the main problems of UML is model multiplicity. As Dori (2002) observes, "even with superb CASE tools, keeping the diagrams synchronized and preventing the introduction of contradictions and mismatches in the overall system (which, in UML, exists only in the modeller's mind) become daunting tasks beyond anyone's cognitive ability (p. 83)." Dori goes on to observe that "many software developers struggle with UML's sheer complexities and inconsistencies, especially regarding the modelling of system dynamics in OO settings. *Disliking it, they use it mainly because their organizations follow the standard (italics mine)*. "

Two things emerge from the preceding discussion. First, software developers use the UML (in spite of its limitations) because it can be, at some level, used to generate software systems. So developers do see some value. The second point has to do with the end user community. If the UML presents so many difficulties to the developer community, it can certainly not be a "natural" approach for end users.

## 5.2 The design-analysis relationship

For both OO and PO, there are problems when it comes to the analysis-design boundary or interface. For both PO --- we know there are weaknesses (for instance how do you relate DFDs to structure charts) and OO (but what about the fuzzy boundary between OOA and OOD; or for that matter the belief that OOA is not required, or rather, if OOA is done properly, we can skip OOD because the OOA deliverable serves as the ideal input to OOP).

However, there is hardly any agreement on which aspects of the object-oriented development process belong to analysis and what parts to design, because the boundary between OO analysis and OO design is distorted (Jalote 1997). The primary difference between them is that OO analysis models uses terminology in the problem domain, to help the analysts (doing the translation task of understanding users' views and translating it into the OO view) to understand and specify the problem, while OO design focuses on and models the solution to the problem (so that the developer can get an OO specification). Therefore, the methodologies and their representations employed in OO analysis and OO design look quite similar, OO analysis dealing with the problem domain and OO design with the solution domain. Although there is no clear cut between analysis and design, some authors like

| | User (Phase I) | Analyst (Phase II) | Developer (Phase III) |
|---|---|---|---|
| Expectations | Meeting functional requirements and using the software system effectively and productively | Getting the user to think through all possible scenarios and requirements and validating those. | Getting a stable set of requirements and specifications so that the software can be engineered. |
| Concerns | Users think in terms of their workflow, how they relate to people and things around them | Sees both sides and is a conduit. May buffer or filter information and, in the process, help or hinder. | Sees specifications and produces software. Is quite distanced from the actual deployment of the software. |
| Language employed | Uses natural language that is typical to the context and to the domain in which users operate. | Has to be bilingual to translate requirements into specifications help negotiate when there is disagreement | Uses formalisms and programming languages and is not concerned about the context and semantics. |

Figure 1: Suggested framework for the coexistence of OO and PO modelling approaches

Jalote (1997) consider it to be one of the strong points of the object-oriented approach, in which the transition from analysis to design is "seamless". The problem is that the desirable characteristics of de-linking of design and analysis (separating the "what to build" from the "how to build") are often lost. OO enthusiasts often claim that this fusion of analysis and design is a productivity enhancer. However, in OO analysis, the objects focus on the problem domain and represent (generally) things or concepts with meaning in the problem domain and, in OO design, the objects are called semantic objects as they have meaning in the problem domain (Monarchi and Puhr 1992). In addition to analysis, the process-oriented design concentrates not only on the static structure of the problem or solution domain, but also on the dynamic behaviour of the system.

We are proposing a more encompassing ontological framework that leverages the strengths of every stakeholder and participant in the system development value chain. The basis for this is shown in Figure 1.

We divide the system development into requirements modelling, design and development. It is clear that in most applications, Phase I is dependent on the client's or user's ontology. It is legitimate to ask whether users, in general, have an ontology. One problem of developing a specific ontology is that of infinite regress based on the extreme case of every individual has his/her own ontology. In general, the PO view offers far more utility that the OO view. End users are more comfortable with narrating what (changes) happen to people and things, when these changes take place, the temporal order in which such changes affect people and things, the outcomes of certain acts or events, decisions and bottlenecks in a sequence of events, whether events or activities are parallel or serialized, and so on. While it is entirely legitimate to argue that when users identify people and things, they are, for all intents and purposes, identifying objects (and maybe, classes). However, we have argued in this paper that users should not be persuaded to think in terms of objects only (as is required by the OO approach) because (a) that is not a user's natural world view and (b) it undermines the user's narrative process by limiting a user's vocabulary.

We also propose Phase 2 to be a formal analysis and design stage where analysts play a pivotal role. A primary role of systems analysts is one where they help domain experts (or the clients) communicate effectively with the programming and development community. Systems analysts assume primacy because they enable the coexistence of both ontologies. In so doing, the programmer or developer would get what they want – an OO specification and a user would be able to able to employ a more friendly and natural language to communicate requirements.

A critical implication of the approach that is being suggested is that it helps avoid becoming methodology-centric. An end-to-end methodology leaves little room for flexibility and assumes that since the grammar on which the methodology is premised is mathematically closed, the end product will be acceptable. However, the methodology is as good as the users. Our proposal removes the assumption that OO is "natural." This frees users from thinking in terms of objects. This approach also re-establishes the importance of adhering to a well-defined process. It is important to distinguish between a process and a methodology. To start with a process could subsume one or more methodologies. However, the converse is not true. A process can be considered a generic set of steps to provide an output when some inputs are provided. Take the "process" of requirements elicitation. This process, when defined for an organization, could list out alternate methodologies that can be used to operationally this process. These techniques could include use cases, scenarios (Holbrook, 1990) narratives, prototyping (Jordan et al., 1989). Essentially, we would have allowed the inclusion of multiple perspectives and users (Easterbrook, 1991). The process model has been reflected upon by others also (Richards, 2000). The advantage of adopting a process-centric view for system development is that the logic of process discipline takes over. The logic of process discipline is that the quality of the conversion process (from user specified requirements – in whatever form – to programmer preferred object models) is now subject to the same continuous improvement pressures that all other deliverables are in the software development lifecycle.

Phase 2 is important because it allows participants to operationalize the requirements negotiation concept (Boehm et al., 1995). The requirements negotiation concept accepts the fact that part of the understanding that emerges among stakeholders of a system includes pruning some parts of the taxonomy and elaborating others (Grünbacher and Briggs, 2001). The resulting taxonomy (ontology as we have defined it) becomes an organizing framework for emergent win conditions. The win conditions include an expanded vocabulary and *hence* a common language.

Decoupling the user's view from the developers view and linking them by human (analyst) intermediation is not new or radical. By doing that we will be formalizing that which is already explicitly practiced (that there are layers and layers

of interpretations between users' requirements and the developers' views of the design based on specification). By formalizing such intermediation we will allow participants in the system development process to retain and deal with their metaphors (Kendall and Kendall, 1993). Additionally, we will preserve the variety that is inherent in any functionally rich system.

We believe that instead of a monolithic methodology (e.g., one based on UML), there should be a place for multiple methodologies. In order for proper and continuous natural selection to take place in the world of methodologies, all three classes of participants – users, analysts and developers – need to participate vigorously in the larger discourse. So far, the developer community has been the most active and dominant in showing concern for this issue. Therefore, unless the community of end users (or researchers representing them) start playing a more active role in the process of system development and critically reflecting on how to conceptualise and think about information systems, we will keep on ending up with analytical and modelling approaches that are more responsive to the needs of the developer community.

Our proposal is consistent with other proposals (Kosaka, 1997) that posit that a shift of ontological assumptions in systems analysis from the realist world to the socially constructing world. This would facilitate the smooth transition of OO modelling from a static view to a dynamic one.

Apart from abstract issues like temporal dynamics, there are other major implications of our proposed approach for both research and practice. Once modelling is understood as a social process as much as a technical process, the importance accorded to the ontology of different stakeholders will be formalized. From a research standpoint, this will mean the development of meta-models for retaining multiple ontologies while attempting to aspire to a closed and integrated framework to seamlessly accommodate all these ontologies. In this context metrics in connection to the use of specific modelling languages for different modelling tasks should also be developed based on existing work (Krogstie, 1998).

From a pedagogical standpoint, classroom instructors will be better able to sustain a discourse on the appropriateness of modelling approaches for different constituencies and stages in a lifecycle. Specifically, the uncomfortable relationship that exists between business school IS curriculum and the engineering school CS curriculum will be mitigated. Our framework provides a framework to see exactly where the complementarities lie.

# 6 CONCLUSION

Taking a balanced look at both approaches (the process-oriented and object-oriented) and decoupling them helps avoid falling into the trap of accepting (in error) the primacy of one worldview over another. We have shown in this paper the both the process-oriented and object-oriented approaches are desirable and useful when applied appropriately.

# REFERENCES

Alspaugh, T. A. and Ant´on, A. I. (2001). Object-Orientation in Requirements, Specifications and Models, TR-2001-12, Department of Computer Science, College of Engineering, North Carolina State University, Raleigh, NC 27695-7534 USA, http://www.isr.uci.edu/~alspaugh/pubs/alspaugh-tr12-oo-2001.pdf.

Boehm, B.; Bose, P.; Horowitz, E. and Lee, M. J. (1995). Software Requirements Negotiation and Renegotiation Aids: A Theory-W Based Spiral Approach, Proceedings of ICSE 95.

Booch, G., Rumbaugh, J., and Jacobson, I. (1996). The Unified Modeling Language for Object-Oriented Development, Version 0.9, Rational Software Corporation, July 1996.

Coad, P. and Yourdon, E. (1991). Object Oriented Analysis, Prentice-Hall Englewood Cliffs NJ.

de Champeaux, D.; Constantine, L.; Jacobson, I.; Mellor, S.; Ward , P; and Yourdon, E. (1990a). Panel session: Structured analysis and object-oriented analysis, Proceedings of the European Conference on object-oriented programming systems, languages, and applications, October 21-25, Ottawa, Canada. (Addendum to proceedings), 15-17.

de Champeaux, D.; Constantine, L.; Jacobson, I.; Mellor, S.; Ward , P; and Yourdon, E. (1990b). Structured analysis and object oriented analysis, Proceedings of the European Conference on object-oriented programming systems, languages, and applications, October 21-25, Ottawa, Canada, 135-139 .

Dori, D. and Reinhartz-Berger, I. (2003). An OPM-Based Metamodel of System Development Process, Proceedings of the 22nd International Conference on Conceptual Modeling (ER 2003), Chicago Illinois, October 13-16.

Easterbrook, S. (1991) Elicitation of Requirements from Multiple Perspectives PhD Thesis, Department of Computing, Imperial College of Science, Technology and Medicine, University of London, London SW7 2BZ.

Gane, C. P. and Sarson, T. (1979). Structured system analysis: Tools and techniques, Prentice-Hall International, Englewood Cliffs, NJ.

Grünbacher, P. and Briggs, R. O. (2001). Surfacing Tacit Knowledge in Requirements Negotiation: Experiences using EasyWinWin, Proceedings of the 34th Hawaii International Conference on System Sciences.

Gruninger, M. and Lee, J. (2002). Ontology: Applications and design, Communications of the ACM, 45(2), 39-47.

Holbrook, C. H. (1990). A Scenario-Based Methodology for Conducting Requirement Elicitation, ACM SIGSOFT Software Engineering Notes, 15 (1), 95-104.

Hughes, J. and Wood-Harper, T. (1999). An empirical model of the information system development process: A case study of an automotive manufacturer, Proceedings of the 10th Australian Conference on Information Systems, 1181-1192.

IEEE (1998). 830-1998 IEEE Recommended Practice for Software Requirements Specifications.

Jacobson, I. (1992). Object-Oriented Software Engineering: A Use-Case Driven Approach, Addison-Wesley: Reading, Massachusetts.

Jacobson, I. (1994). The Object Advantage, Addison-Wesley, Workingham, England.

Jacobson, I., Christersson, M., Jonsson, P., and Overgaard, G.G. (1992). Object-Oriented Software Engineering, Addison-Wesley, Reading MA.

Jordan, P. W.; Keller, K. S.; Tucker, R. W. and Vogel, D. (1989). Software Storming: Combining Rapid Prototyping and Knowledge Engineering, IEEE Computer, 39-48.

Kendall, J.E. and Kendall, K.E. "Metaphors and Methodologies: Living Beyond the Systems Machine," MIS Quarterly, Vol. 17, No.2, 1993, pp. 37-47.

Kendall, J.E. and Kendall, K.E. Metaphors and their Meaning for Information Systems Development," European Journal of Information Systems, Vol. 3, No. 1, 1994, pp. 37-47.

Kosaka, T. (1997). Task analysis makes is methodologies object oriented, Working Paper, Department of Management, Aichi-Gakuin University, (http://www.ms.kuki.sut.ac.jp/KMSLab/kosaka/papers/bcs9609.pdf), Last accessed October 30, 2003.

Kvavik, K. H.; Karimi, S.; Cypher, A. and Mayhew, D. J. (1994). User-centered processes and evaluation in product development, Interaction, 1(3), 65-71.

Lee, J. and Wyner, G. M. (2003). Defining specialization for dataflow diagrams, Information Systems, 28(6), 651-671.

Parsons, J. and Wand, Y. (1997). Choosing classes in conceptual modeling. Communications of the ACM, 40(6), 63–69.

Richards, D. (2000). A process model for requirements elicitation, Chan, Taizan: Ng, Celeste See Pui(Ed/s) in Proceedings of the 11th Australasian Conference on Information System (ACIS 2000), Information Systems Management Research Centre, Queensland University, Brisbane, Australia.

Rickman, D. M. (2000).A Process for Combining Object Oriented and Structured Analysis and Design, 3rd Annual Systems Engineering & Supportability Conference, 23-26 October.

Ross, D. (1977). Structured analysis (SA): A language for communicating ideas, IEEE Transactions on Software Engineering, 3(1), 16-34.

Rumbaugh, J.; Blaha, M.; Premerlarni, W.; Eddy, F. and Lorenson, W. (1991). Object-Oriented Modeling and Design, Prentice-Hall Englewood Cliffs NJ.

Shlaer, S., and Mellor, S. (1992). Object LifeCycles: Modeling the World in States, Prentice-Hall Englewood Cliffs NJ.

Soffer, P.; Golany, B.; Dori, D. and Wand, Y. (2001). Modeling off-the-shelf information system requirements: An ontological approach, Requirements Engineering, 6(3), 183-199.

Steels, L. and Kaplan, F. (1999). Situated Grounded Word Semantics, IJCAI-99, the Sixteenth International Joint Conference on Artificial Intelligence, July 31-Aug 2, Stockholm, Sweden.

Krogstie, J. (1998). Integrating the Understanding of Quality in Requirements Specification and Conceptual Modelling, ACM SIGSOFT Software Engineering Notes, 3(1), 86-91.