Systems Engineering Experience with UML on a Complex System

Laurence Doyle
ITT Industries
100 Kingsland Rd. Clifton, NJ
ldoyle1@stevens.edu

Michael Pennotti, Ph.D.
Stevens Institute of Technology
Castle Point Station, Hoboken, NJ
mpennott@stevens.edu

Abstract

Experience using UML for systems engineering on a complex system is described. The system was an experimental situation awareness system for small military units such as a hostage rescue mission. This experience provides insight into the use of process-oriented and object-oriented models. Real projects that experiment with methodology are an important source of information that cannot be duplicated on a contrived problem.

Most of the functional requirements in this system involved software but there was also unique hardware. Use case specifications were derived directly from a very high level specification from the customer without any intervening artifact. Subsystem interaction diagrams were developed from the use case specifications. The use of UML as a systems design tool had the advantage that it greatly facilitated the transition into software development.

While this approach was largely successful, there were some cases when the object-oriented model did not fit the problem. In addition to their role in task automation, models provide a mental representation that helps us solve engineering problems. Object-oriented and process-oriented models provide two different representations. Because this project attempted to apply an object-oriented model so broadly, it was a de facto experiment in the use of such models. An examination of the cases where a process-oriented model fit better than an object-oriented model provides insight into the differences between these two views. The experience on this project supports idea that each view is a better cognitive fit to a different set of problems.

Introduction

The project described in this paper employed an object-oriented UML model for a large part of the systems engineering as well as the software development. This project began in 1999 and predated much of the recent work to integrate systems and software engineering models. The experience on this project is none the less relevant to the question of how to best use object-oriented and process-oriented models in systems engineering.

Software and systems engineering have been pursuing divergent methodologies for modelling and design. While object-oriented models are now the dominant approach to software engineering, systems engineering has used process-oriented, sometimes called functional or structured, models. Process-oriented and object-oriented models have much in common. They both develop requirements from use cases. When augmented by sufficient behavioural modelling, both can lead to executable models of the system. Both provide representations used when architecting a system.

It is increasingly common for much, if not most, of the functional requirements of a

system to be implemented in software. The different models cause significant problems such as inefficient communication, inability to share data, difficulty in tracing requirements, and duplicate work. In addition to their use in automated tools, these models provide mental representations or understanding a system, solving problems and creating new systems.

Systems engineering has historically used models with a process-oriented view such as functional flow and IDEF0 diagrams. More recently, (Douglas 00), (Lykins00), and (Cantor 01) have described adaptations of object-oriented methods to systems engineering. However, the use of object-oriented models for systems engineering is a recent development and there is relatively little practical experience.

Because the project described here attempted to apply object-oriented beyond the traditional boundaries of software engineering, the project was a de facto experiment in the use of such models. Real projects that experiment with methodology are an important source of information than cannot be duplicated on a contrived problem. In this project, an integrated UML model included most of the functional requirements of the system. This was largely successful. However, there were some key exceptions that provide some insight into the nature of models. In these cases, the object-oriented representation of the system did not seem to fit the problem.

Previous Work

Improved integration of systems and software engineering is subject of much ongoing work. The Software Productivity Consortium has developed the Integrated Systems and Software Engineering Process (ISSEP), "a process model that provides a high-level abstraction of the complex process for engineering software-intensive systems". Based on ISSEP, (Lykins 02) describes a project called Object-Oriented System Engineering Process (OOSEP) to extend UML for systems engineering. INCOSE and OMG are currently developing and extension of UML for systems engineering called SysML.

When UML was initially considering for systems engineering, several issues were raised. While issues specifically related to UML have been addressed in UML 2.0 and SysML, some issues suggested cognitive problems with the object-oriented models in the systems engineering context:

- (Skipper 02) states that object-oriented diagrams are hard to understand by non-software engineers
- (Steiner 02) states there is no standard mechanism for requirements analysis, allocation, and traceability.
- (Cocks 99) and (Steiner02) state that there is no standard approach to modelling the problem domain separate from solution domain.
- (Oliver 02) explains how the concept of inheritance violates physical reality

These issues seem to be deeper than just the constructs of UML. While UML 2.0 and SysML now provide the functional models, we are still faced with the question of whether to use an object-oriented or functional model in any given situation. A major consideration in this is cognitive fit. Cognitive fit exists when the problem representation and the problem itself match. Research by (Vessey 91), (Sinha 92) and others supports the idea that problem solving is enhanced when there is good cognitive fit between the problem and the representation of the problem.

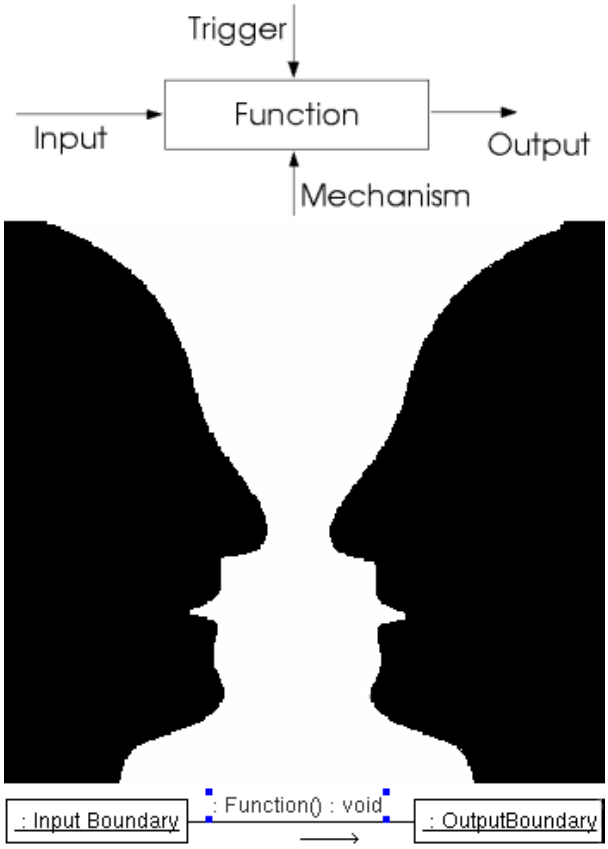
Some researchers have performed controlled experiments to evaluate the object-oriented and process-oriented views of software. (Argawal 99) performed an experiment to evaluate cognitive fit in requirements modelling in both process and object oriented methodologies. Cognitive fit exists when the problem representation and the problem itself match. "As cognitive-fit theory predicted, superior performance was observed when the process-oriented tool was applied to the process-oriented task. For the object-oriented task, however, the performance effects of cognitive fit require

further investigation since there was no difference in subject performance across the two tools." (Morris 96) evaluated both object-oriented and process-oriented methods as perceived by both experts and novices. They did not find that object-oriented methods improved either subjective mental workload or time to completion. But they did find that novices were more satisfied with their work. (Davey 94) evaluated the impressions of moderately experienced procedural programmers who were learning object-oriented programming. They found that almost all the test subjects preferred the object-oriented view. (Argawal 96) performed an empirical study of people's comprehension of both object-oriented and process-oriented (i.e. structured) models of a system. They found that for most complex questions, the process-oriented model was easier to understand. But they also found that, when addressing a particular question, the model that had the better cognitive fit to the question was better. (Corritore 00) found that object-oriented programmers looked at fewer files in order to modify a program.

The researchers cited above studied teams of students performing tasks such as requirements analysis and design using both object-oriented and process-oriented methods. None of the studies provided unequivocal support for either view over the other. Most of the researchers noted difficulties with their experiments and expressed the need for the controlled studies to be augmented with experience from actual projects. The experience recounted here is intended to serve that purpose.

(Doyle 04) investigated cognitive fit in object-oriented and process-oriented systems engineering models by closely examining the low level structures. The two models reverse what is in the foreground and background as shown in the models of a function at the top and bottom of Figure 1.

In the process-oriented diagram at the top of Figure 1, the function, like the vase, is the foreground. In the object-oriented diagram at the bottom of Figure 1, the function, like the space between the faces, is where the two foreground objects interact. In reversing foreground and background, these two views



also reverse what is persistent and transient, and what is concrete and abstract.

Figure 1. Object and Process Oriented models reverse foreground and background.

Cognitive science research cited by (Doyle 04) has established a close connection between analogies as mental representations and our effectiveness at problem solving. Mental representations are based on a particular metaphor. Metaphors provide good cognitive fit when three conditions are present:

- The analogy provides implied constraints that are true for the actual problem.

- There is a consistent mapping between the metaphor and the problem
- The metaphor provides a way to chunk the problem into segments compatible with our short term memory.

(Lewis 94) performed research supporting the first two conditions. (Simon 73) studied the relationship of short-term memory and problem solving by studying chess players. He found that expert players look at the board in "chunks" of five to seven possibilities. For each chunk, they would carve out five to seven additional chunks. This mental process has become known as "chunking". (Wiedenbeck 93) reached similar conclusions in a study of how expert and novice programmers understand programs.

(Doyle 04) noticed that metaphors used by various authors are different depending upon whether they were writing about object-oriented or process-oriented models. When writing about process-oriented models, authors tended to use artifact metaphors. When writing about object-oriented models, authors tended to use natural kind metaphors. (Pinker 97), (Gelman 84) and others found that different metaphors provide a different set of implied constraints. By examining how these implied constraints relate to systems and software engineering problems, (Doyle 04) concludes that object oriented methods are a better cognitive fit for the problems of robustness to change, reuse, and the comprehensibility of components. Process-oriented methods provide a better cognitive fit for the problems of comprehensibility of the overall system, requirements flow down, and an implementation independent view of the system.

System Description

The system addressed in this paper was an experimental situation awareness and communications system for small military units such as a hostage rescue mission. For demonstration purposes, approximately fifty man-pack units were built. Each unit included multi-sensor geo-location, push-to-talk voice and a heads-up display showing friendly and unfriendly positions superimposed on aerial photographs. A self-organizing peer-to-peer radio network provided communications for both voice and data. The radio also performed ranging to support indoor navigation where GPS is not available. This system was developed over a three-year period and culminated with a successful field demonstration. The physical architecture of an individual unit is shown in Figure 2.

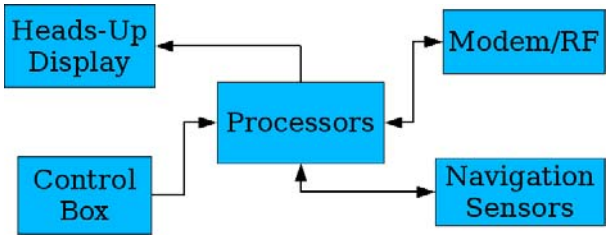


Figure 2. Physical Architecture of an individual unit.

A simplified UML use case diagram for the software is shown in Figure 2. Although the actual system included many additional use cases, those shown are sufficient for this discussion.

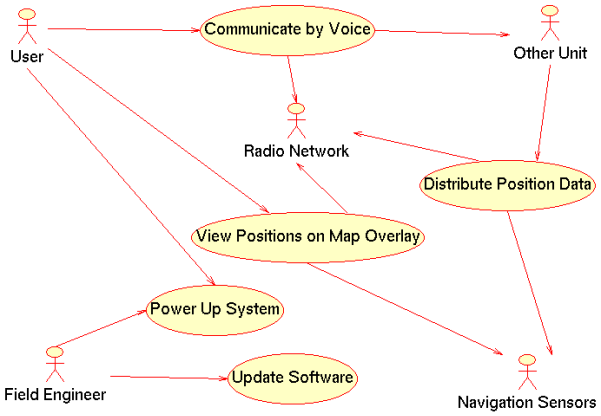


Figure 3. Use case diagram

Note that the navigation sensors and radio links are considered to be outside the system in this view. Although these are actually part of the system in the larger sense, no attempt was made to represent these components within the UML model. Since these subsystems had their own specifications, they were considered to be outside the system for the purposes of the UML model. Most, but not all, of the functional requirements were

implemented in software. Non-functional requirements and specifications for the hardware were treated separately.

Object-Oriented Systems Engineering

At the beginning of the project, some of the systems engineers, in addition to the software team, received training in Rational's object-oriented process. This process encompasses the complete development process including requirements specification, requirements analysis, systems design, software design, coding and test. This paper focuses on the first three of these activities that are generally considered to fall under the purview of systems engineering.

The requirements were specified in the form of use case specifications that were derived directly from a very high level specification from the customer. This differs from the more conventional approach described by Buede (2000) where use cases are a means of discovering requirements. Here, the use case specifications are the requirements.

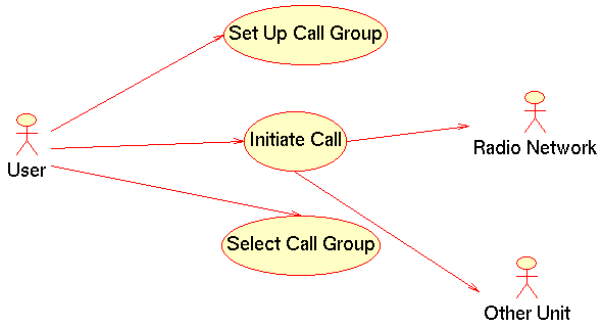


Figure 4. Detailed Use Cases for "Communicate by Voice"

The top level use cases were broken down into simpler uses cases such as those shown in Figure 4. It is important to note that this is not a functional decomposition. The top level use cases are merely packages of related use cases that help organize requirements. The use case specifications contained preconditions, main flow, alternate flows and post conditions as described by (Booch 99). Ultimately, over one hundred use cases were specified.

These use case specifications were then used to develop system level UML interaction diagrams that showed the interaction of inter-connected subsystems as shown in Figure 5.

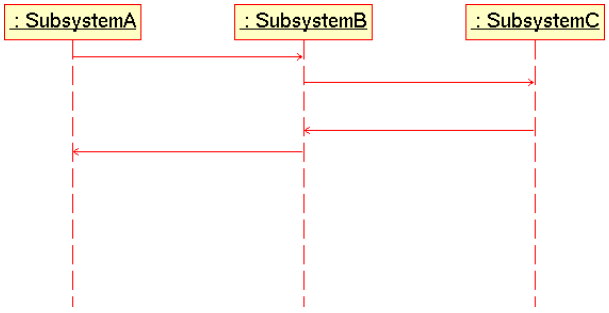


Figure 5. Subsystem Interactions

From these interaction diagrams, lower level sets of use cases were derived for each subsystem. Use case diagrams were then developed for each subsystem where other subsystems are shown as actors as shown in Figure 6.

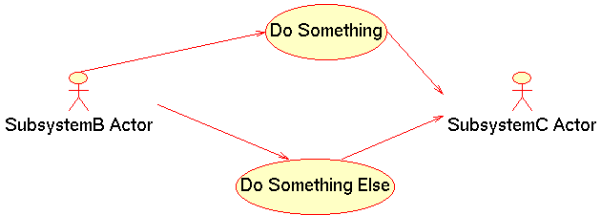


Figure 6. Subsystem use cases

Although this process was developed independently, it closely follows the "System of subordinate systems" architectural pattern described in (Ericsson 01).

The key advantage to this approach is that the work products form this system design flowed seamlessly into the software development process. The benefits of this are described in (Douglas 00) and (Cantor 01).

While this approach was largely successful, there were some times when the object-oriented methodology associated with UML did not work well. The following three examples will be described in detail:

1. **Getting Started.** The object-oriented view proved difficult when initially organizing the project and developing an overall concept for how the system works.

2. **Specifying and implementing the Power-On use case.** The implementation was closely tied to certain features of the operating system and hardware interfaces.
3. **Specifying and implementing the Update Software use case.** In this case, the extent to which functions should be automated was not clear.

In each of these cases, an object-oriented approach was first attempted. As such, these cases were real world experiments in the use of models. Looking at these cases points out why both object and process oriented views of a system are needed. The same people who were successful applying object-oriented analysis, design and programming for most of the system found a process-oriented approach more appropriate in these circumstances.

Getting Started

Using only the object-oriented representations in UML 1.2, we found it impossible to get started. Although the use case diagrams were good at showing what the system does, they provide no insight how it does it. People seemed to need a single view that explained approximately how the system worked in terms of a few comprehensible chunks. The functional decomposition in the IDEF0 diagram in Figure 7 provides this view.

The ability to organize a system into comprehensible chunks was also important for project management. The decomposition provided a basis for creating a team structure and managing the project. Because the decomposition is based on abstract functions, this provided a basis for organizing the project while many trade-offs were unresolved.

Because functions are named according to their purpose, there is no need to learn a new vocabulary when trying to understand the system in a process-oriented model. In object-oriented development, creating the architectural classes can be viewed as populating the model with the major phyla of pseudo-natural objects in an artificial world. Early in the development, we experienced the same phenomena observed by (Holmboe 04) where the team was linguistically challenged as it struggled to adjust to the new vocabulary. It was initially unnatural to begin talking about unfamiliar objects as though they were naturally part of the landscape. Although the team eventually got past this, it presented difficulty early in the system design.

Data flow diagrams for the main functions of the system were developed such as Figure 7. This view of the system shows both what the system does and, to some extent, how it does it. Hardware/software trade-offs were required for each of the functions shown in Figure 7. Because this view is independent of any particular implementation, it allows separate teams to perform tradeoffs to determine the boundaries of the system.

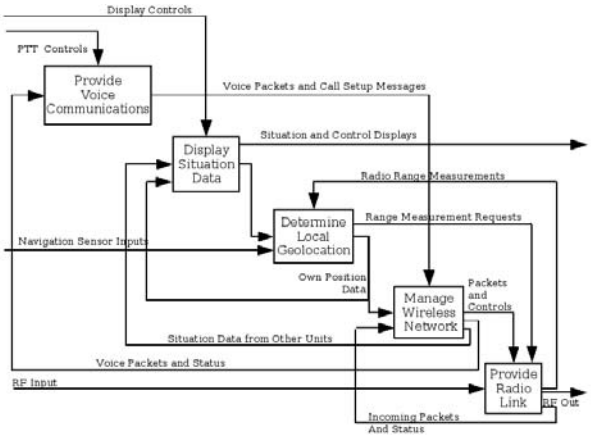


Figure 7. System Functional Decomposition

Although this type of diagram was not part of the UML, the development process was unable to get started until this view of the system had been developed.

Power-on Initialization

Most of the requirements for initializing the system after applying power are stated in terms of concrete outputs required in a particular sequence. For example, "The processor shall set the SDRAM row and column configuration bits." This use case was initially modelled with a UML activity diagram. But while an activity diagram is usually followed up by an interaction diagram,

this problem is so strictly procedural that introducing object-oriented notation simply added confusion.

In contrast, the view shown in Figure 8 clearly shows how a series of functions provides the required outputs.

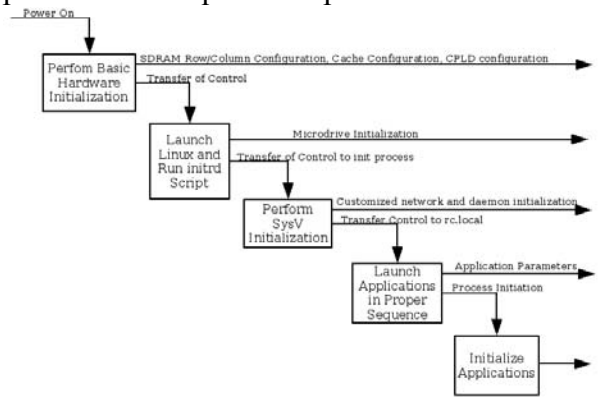


Figure 8. Functional Decomposition of Power On Initialization

In addition, the hardware and operating environment dictates the implementation. The functional decomposition shown in Figure 8 clearly shows how requirements have been allocated to the various phases of the system power-up sequence. Because the requirement of this process is to produce a series of outputs, the definition of objects provides no benefit. The representation in Figure 8 clearly attaches requirements to elements of the solution. Although Figure 8 shows functions that are entirely implemented in software, an object-oriented view seemed inappropriate.

Reprogramming

Because this was an experimental demonstration system, there was a requirement to perform rapid software updates in the field. A functional decomposition derived from the "Update Software" use case is shown in Figure 9.

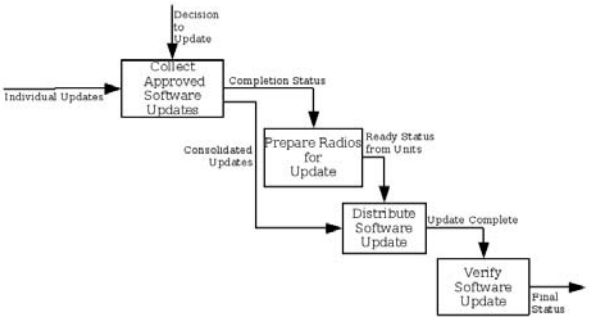


Figure 9. Functional Decomposition of Updating Software

This representation shows an implementation independent view of the problem. Each of the functions could be accomplished by many means. Software updates could be distributed over the air or by someone running around with a USB Zip drive. Since the success of this project depended on being able to update 50 units very rapidly, considerable ingenuity went into the ultimate solution. A representation that shows this as the interaction of objects imposes too many constraints on the solution. An implementation independent view promotes novelty. This functional view lead to a novel solution that involved software, physical infrastructure and operational procedures.

Conclusions

In retrospect, we were somewhat naïve to attempt to model the entire system in a single, unified model using only the elements available in UML 1.2, the version available at the time. While an object-oriented view was successful for most of the system, it was a poor fit in particular cases. Three such cases were presented here. Because UML 2.0 and SysML expand the choice of model elements, it may be possible to expand the scope of the model in the future. But we will still have to choose which view to use when. Initially considering the experience on this project supports the idea that no single view is the best cognitive fit to all problems.

References

Agarwal, R., De, P., Sinha, A. P., Comprehending Object and Process Models: An Empirical Study, *IEEE Transactions on Software Engineering*, July/August 1999, Vol. 25, No. 4, Pages 541-556

Agarwal, R., Sinha, A.P. and Tanniru, Mohan R., Cognitive Fit in Requirements Modeling: A Study of Object and Process Methodologies, Journal of Management Information Systems Vol. 13 No. 2, Fall 1996 pp. 137 – 162

Booch G., Jacobson I., Rumbaugh J., *The Unified Modeling Language User Guide,*, Addison-Weslry, Reading, Ma, 1999

Buede, D. M., *The Engineering Design of Systems: Models and Methods,* John Wiley and Sons, New York, NY, 2000

Cantor, M., RUP SE: The Rational Unified Process for Systems Engineering, *The Rational Edge*, November, 2001

Cantor, M., Applying UML to System Engineering: Some Lessons Learned, *INCOSE 2002 Panel: UML for Systems Engineering*, INCOSE, 2002

Cantor, M., "Thoughts on Functional Decomposition", *The Rational Edge*, April, 2003

Cocks, D., "The Suitability of Using Objects for Modeling at the Systems Level", *Proceedings of the Ninth Annual International Symposium of the International Council on Systems Engineering*, pages 1047-1054, INCOSE, 1999

Corritore, C., Wiedenbek, S., "Direction and Scope of Comprehension-Related Activities by Procedural and Object-Oriented Programmers: An Empirical Study", IEEE Workshop on Program Comprehension, 2000

Davey, B.; Tatnall, A.,"Introducing object environments: cognitive difficulties ", *Software Education Conference, 1994. Proceedings*. , 22-25 Nov 1994, Page(s): 128 -133

Douglass, B., "The UML For Systems Engineering", www.ilogix.com, 2000

Ericsson, M., Developing Large-Scale Systems with the Rational Unified Process, http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/2003/sis.pdf, *Rational White Paper*, 2000

Fowler, M., UML Distilled, Addison-Wesley, Reading, Ma, 1999

Gelman, S.A., "The development of induction within natural kind and artifact categories" *Cognitive Psychology,* Vol 20, pp 65-95, 1988

Hoffman, H., From function driven Systems Engineering to object oriented Software Engineering, I-Logix white paper, 2000

Keleman, D., Function, goals and intention: children's teleological reasoning about objects, *Tends in Cognitive Sciences,* Vol. 3, No. 12, Dec 1999

Lewis, M., "A method for selecting optimal analogies", IEEE International Conference on Systems, Man, and Cybernetics, 1994, Volume: 3 , 2-5 Oct. 1994, Pages:2743 - 2748 vol. 3

Lykins, H., Friedenthal, S., Meilich A., Adapting UML for an Object Oriented Systems Engineering, *INCOSE Cheasepeake Chapter 2001 Meeting*, October, 2001

Morris, M.G.; Speier, C.; Hoffer, J.A., "The impact of experience on individual performance and workload differences using object-oriented and process-oriented systems analysis techniques", *System Sciences, 1996., Proceedings of the Twenty-Ninth Hawaii International Conference on Man-machine interfaces*, Volume: 2 , 3-6 Jan 1996, Page(s): 232 - 241 vol.2

Pinker, S., How the Mind Works, W. W. Norton & Co. , New York, 1997

Sinha, A. P. and Vessy, I., Cognitive Fit: An Empirical Study of Recursion and Iteration, *IEEE Transactions on Software Engineering,* May 1992, pp 368-397 2001.

Skipper, J.F., Assessing the Suitability of UML for Capturing and Communicating Systems Engineering Design Models, www.vitechcorp.com

Vessey, I., Gallata, D., "Cognitive Fit: An Empirical Study of Information Acquisition", *Information Systems Research,* V2, N1, 1991, pp. 63-84

Wiedenbeck, S., Fix, V., and Scholtz, J., "Characteristics of the mental representations of novice and expert programmers: an empirical study", *International Journal of Man-machine Studies*, vol. 39, pp. 793-812, 1993.

Biography

Laurence Doyle is a staff scientist at ITT Industries. He holds an MS in computer science from Stevens Institute of Technology and a BS in computer science from Pratt Institute. He has been a lead programmer and software architect on aircraft, satellite, signal intelligence and military communications systems for 35 years. He has lead software development teams using both object-oriented and structured methods. He holds five patents and has published articles on satellite navigation, software methods, and real-time scheduling. He is currently a PhD candidate in systems engineering from Stevens Institute of Technology.

Dr. Michael Pennotti is Industry Professor of Systems Engineering and Director of the SDOE Program at Stevens Institute of Technology. A systems engineering leader for thirty-five years, he was Director of Advanced ASW Concepts at Bell Laboratories, Human Resources VP for Lucent Technologies' Enterprise Networks Group, and VP Quality at Avaya. He joined Stevens in 2001. He is a member of INCOSE, and a senior member of IEEE and the American Society for Quality. He holds PhD. and MS degrees in Electrical Engineering from the Polytechnic Institute of New York, a BEE from Manhattan College, and is a graduate of the AEA/Stanford Executive Institute for the management of high-technology companies.