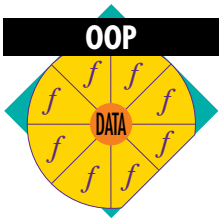
Choosing an Object-Oriented Analysis and Design Tool



By Michael J. Branson and Eric Herness

Successfully deploying object-oriented analysis and design tools requires a full understanding of the requirements and capabilities of the tools. This article provides a comprehensive list of these requirements. Scaling up to team environments is critical to successful deployment of this technology in large organizations. A rich and robust object-oriented analysis and design tool that works well in a team environment will help to ensure a consistent application of the object-oriented development process across the entire software development team.

Selecting the right tools for software development projects can be difficult. Every project has unique needs, and there are many different tools from which to choose. One category of tool selection is particularly challenging—choosing an object-oriented analysis and design tool—because there are so many issues to consider, such as methodology, scalability, and integration. This article details a set of requirements for object-oriented analysis and design tools and classifies those requirements into areas important in your software development projects.

Although each project is unique, all object-oriented development projects share some common set of needs. Figure 1 illustrates the different types of requirements that projects can have on object-oriented analysis and design tools. It also shows the relationship between the different types of tool requirements and certain project attributes.

The 'x' axis in Figure 1 depicts the degree to which projects require the analysis and design tool to pervade the object-oriented life cycle. Is the tool limited to covering iteration through analysis and design activities, or does it also cover creating and maintaining code? The 'y' axis represents project size since the importance of various analysis and design tool requirements depends upon the number of developers involved in the project.

The remainder of this article describes various requirements in each category, including detailed explanations for each requirement.

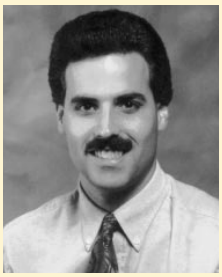
Methodology Requirements

Methodology requirements deal with support provided in the tool for the object-oriented paradigm. Developers use the tool to explore the software domain of the project and design a suitable implementation. An object-oriented analysis and design tool must allow developers to practice a software development method that leverages the features of object-oriented programming, such as abstraction, encapsulation, inheritance, and polymorphism.

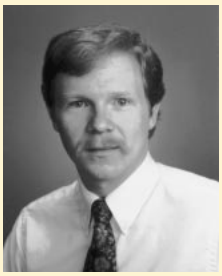
All projects, regardless of size, require a methodology. Our experience shows that the exact methodology chosen is not nearly as important as having the appropriate support for the chosen method. Tools, education, and processes must work together with the chosen method and be adapted to meet the needs of software developers in the most expedient way.[1] Therefore, methodology requirements are an issue for all projects when choosing an object-oriented analysis and design tool. The key methodology requirements are described in the following sections.

Object-oriented Method and Notation

An *object-oriented method* is a disciplined process for generating a set of models describing various



Michael J. Branson



Eric Herness

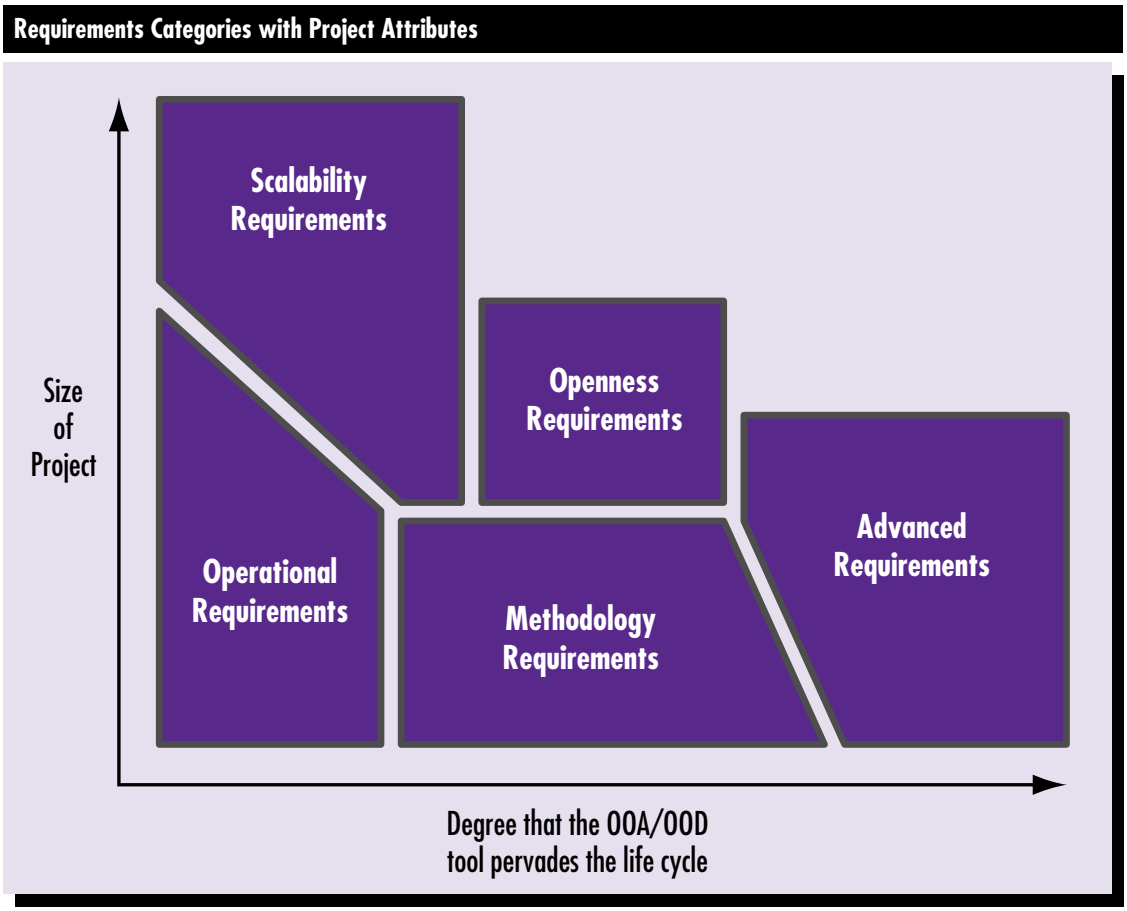


Figure 1. Positioning of requirements categories with project attributes

aspects of the software system under development, using some well-defined notation.[2]

Selecting the method to use for your project is an important decision. The tool used for analysis and design should support the chosen method. It is not effective to have developers thinking within the context of a particular object-oriented method, then translating those thoughts into models that are not part of that method and notation. The tool must map to the intellectual concepts with which developers have chosen to work.

Many object-oriented methods and notations exist.[2, 3, 4, 5] Choosing a well-known method and notation allows software developers and project managers to seek support services including literature, education, and consultation from many sources. Those practicing software development using a popular method can leverage a larger marketplace of ideas that exists in the method's user community.

Examples of AIX-based tools that support a particular method include Rational Rose® from Rational® Software Corporation that supports the Booch method, and Objectory® SE from Objectory Corporation that supports the Jacobson use case-driven approach to object-oriented software development. Also, a class of meta-tools that are not tied to a particular method allow you to define the notation that you want supported. Although these tools provide greater flexibility, they require customization to meet all the requirements met by method-specific products.

Frequently, tools only support part of the chosen method. Be sure that when a tool is advertised to support a particular notation, it supports the notation fully—or at least to the extent that your project needs to leverage it.

Legal Uses of the Notation

Legal uses of the notation must be enforced. If an analysis and design tool allows misuse of a notation, then it is simply a drawing tool. Using the tool, you should be able to construct models that are consistent with the notation and prohibit illegal use of this notation. The tool should

enforce the syntactic correctness of models produced. Proper use of the notation allows projects to experience the benefits offered by the method's rigor.

Static Model of Classes

The methodology should support a static model of classes and their relationships. All object-oriented methods have some notion of a static model of the software system. Booch, for example, uses class diagrams.[2] The exact content of this static model varies from method to method.

The static model depicts entities (usually classes of objects) and relationships between them. Relationships minimally include association, aggregation, and inheritance. Whatever the notation and method, the static model is essential to the object-oriented analysis and design methodology.

Class Attributes and Behavior

Static entities require more specification than merely naming. The key attributes and behavior of the entity should be documented as part of analysis; implementation language-specific characteristics of a class should be documented during the design activity. Some tools may not allow for such specifications; others may include specifications, but may be limited to only language-independent characteristics. The detail allowed by the specifications determines how long the developers can use the tool before they must move to other forms of specifications, such as the implementation language itself. This is often the gating factor that determines whether the tool can generate reasonable code.

Dynamic Object Model

A dynamic model of objects and their interactions with one another is often necessary to accomplish a particular scenario. Most methods support one or more types of dynamic models. For example, Booch[2] supports object diagrams that show objects and their relationships in the logical design of a system. Booch[2], Rumbaugh[3], and Jacobson[4] support interaction diagrams that provide specific support for tracing the execution of a scenario in the system being modeled. Dynamic models are constructed to validate the static model of the system. Each scenario tests whether the evolving static model is complete and optimal for that portion of the system's dynamic behavior. Support for dynamic models is another essential element of an object-oriented analysis and design tool.

User-Oriented Scenario

End-user scenarios are handled differently in some methods. If your method relies upon user-oriented scenarios such as use cases[4], the analysis and design tool should support them.

Dynamic State of Object

The state information of some objects is so complex that it is often essential to use a tool to model the states and state transitions of the object. This is especially evident in real-time software objects and some user interface objects. An object-oriented analysis and design tool should provide the capability for state models. If your project will not include objects with complex state transitions, you may be able to use a tool that does not provide support for a state model.

When leveraging state models, look for a tool that supports nested states based upon the work of Harel[7]. Modeling with simple states often leads to very complex diagrams. Nested states make it easier to model complex state transitions.

Any analysis and design tool should contain certain operational requirements that make the tool practical or easy to use.

Architectural System Model

System architecture is important in any software undertaking. The system must be designed at the subsystem level as well as the object and class level. Although architectural diagrams will vary from method to method, support for them is very important. Look for a design tool that will enforce the architectural decisions set forth in much the same way that the notation might be enforced. The tool should ensure that the architected and exported interfaces to a subsystem are known. It should also ensure that classes internal to a subsystem are not used by objects outside that subsystem.

Physical Model

The tool should support a physical model (module view) of the software, the relationships between modules, and the allocation of modules to processors. So far, all the requirements have focused upon constructing a logical model of the software system. One step in every object-oriented development method is making a physical model from a logical model. This involves decisions about object and class distribution across modules, shared libraries, tasks, and threads. In simple situations such as a single-threaded software system that runs completely on a single machine, this mapping may be trivial and may not require modeling. But in distributed systems and multiprocessing

environments, good support for capturing the physical model is essential.

Operational Requirements

Besides supporting the development method, object-oriented analysis and design tools must provide a base set of functionality, not related to methodology. Any analysis and design tool should contain certain operational requirements that make the tool practical or easy to use. Software developers rely on these operational requirements to accomplish activities that are not necessarily defined by their object-oriented development method. Operational requirements are described below.

Item Dictionary

The tool should support an underlying dictionary of the items used in models. The dictionary, or repository, that holds the definitions of the items in the diagrams should be available for reference or usage on similar items in other diagrams. A reliable test of dictionary support is whether you ever need to type the same information twice. A dictionary also enables multiple views of the same underlying data. Multiple views are important when presenting material about designs to various participants in the development process.

The primary purpose of object-oriented analysis is to explore the software system domain that you are building.

Semantic Checking for Inconsistencies

The requirement for semantic checking between models or diagrams for inconsistencies relies upon the dictionary mentioned above. The tool should alert you to contradictions in a model. For example, if a class has been characterized as an abstract class in one diagram or specification, while in another diagram an object is being instantiated from it, you should be aware of this conflict.

Printing in Standard Format

Printing is an essential requirement. PostScript® printing support generally makes the tool usable in many environments. The ability to export data from the tool in a variety of formats is also desirable, with specifics being dependent upon the underlying operating system.

Generating Design Documents

The tool should allow you to generate design documents from collections of models and other design information. The primary purpose of object-oriented analysis is to explore the software system domain that you are building. The next step may be communicating what you have learned to others in the project; therefore, the tool's ability to communicate results of an analysis is an important feature. It is not enough to simply generate a good analysis; we need to share it with other people, and potentially with other tools as well. The results of design activity have the same requirements.

One inherent attribute of medium-to-large software development projects is the large amount of data to assimilate. Choosing a tool that packages and organizes the models created in analysis and design into a coherent document or set of documents is important. Without this support, these documents may need to be hand-crafted. They may be combined in a way that does not communicate well. It is imperative that an analysis and design tool make generating documents a push-button activity, which encourages more exploration and less document preparation.

Platform Interoperability

In addition to generating a flexible set of documents, the analysis and design tool should integrate well with the underlying operating system platform. It should be easy to copy and paste portions of diagrams to other tools that operate on the platform. This enables you to quickly create ad hoc design packages and presentations. For example, if the word processor supports graphics image manipulation, then the tool should enable figures to be exported in the appropriate format. Hyperlinks and linking are also desirable characteristics because copying brings the risk of the material becoming outdated.

If there are tool integration facilities on your development platform, the analysis and design tool should integrate into the tools workbench. For example, in AIX Version 4, the capability to integrate the tool with desktops such as IBM's AIXwindows, which supports the Common Open Software Environment (COSE) consortium's Common Desktop Environment (CDE) specification, would provide interoperability at the user interface level. Desktop integration alone is not enough. The capability to integrate via underlying messaging mechanisms like ToolTalk® is necessary to gain true interoperability between the analysis and design tool and other tools in the environment.

The tool should also interoperate among platforms. If the tools run on various operating systems (AIX, OS/2, Windows™), design information should be easily exchanged among platforms. This is the first step toward supporting multiple and mobile users. This topic will be

explored in detail in the section entitled "Scalability Requirements."

Class Interfaces and Source Code

The tool should allow you to view and edit class interfaces and source code from models. In an iterative development activity, the ability to move quickly between the artifacts of analysis, design, implementation, and testing is important. When working within an analysis and design tool, the ability to select a class and see any source code that exists to implement the class is helpful.

Another related key feature includes the ability to view the source code using the developer's chosen editor. This implies a level of integration with other tools. Finding the source code from the design tool must be done correctly. It must use the same information to find source code that build or make tools would use to find source code during a compile.

Filters for Multiple Views

The tool should have the ability to apply filters to a particular model that allows multiple views (for example, inheritance only) of the same model. Another operational characteristic deals with the presentation of the information contained in the dictionary. First, the ability to get an inheritance-only or an aggregation-only view is important. Sometimes the density of a diagram is very high. While this is appealing to experienced designers, other consumers of design information might like a gentler introduction to system analysis and design.

Another filter example is one that deals with public versus private. Diagrams may contain various relationships and entities that are private. It is sometimes desirable to take a public-only view.

Model Management

A final operational characteristic is *model management*, the ability to evolve a system without requiring hours of tools housekeeping. For example, the movement of a class from one subsystem to another must be accomplished with ease. If there are objects instantiated from this class that is shown on many other diagrams, one simple command should move it from one system to another and update all other references to this class.

The same must happen at the subsystem level. Analysis and design often start with a few rudimentary subsystems, then proceed with capturing many classes and objects. Refinement of the architecture may require additional subsystems as well as modifications to the nesting of subsystems—a natural occurrence in most large projects. This type of iteration and evolution must be easily accomplished in an analysis and design tool.

Scalability Requirements

In order to support medium to large projects, tools must scale both vertically and horizontally. Vertical tools should be usable by all levels of the organization. Abstract views should be available for technical management and detailed views available for developers. Horizontal scaling of the tool refers to the size of the project and the number of people who will be using it.

Scalable to Large Projects

Tools must be scalable to support large projects. Each organization must determine the size of projects to be supported by a particular analysis and design tool. A good measure for this is the total number of classes expected in the systems to be constructed. It is important that the tool selected has been previously used in other projects of comparable scale.

Often tools differ in their definition of a project. Some tools are constructed with a project defined as work to be done by a single developer. Tools that isolate one developer's analysis and design work from work by other developers of the software system have very limited use. Analyzing and designing a software system requires a tool whose notion of project equates with the notion of an entire software system.

Analyzing and designing a software system requires a tool whose notion of project equates with the notion of an entire software system.

Multiple Users on Same Project

The tool should allow multiple users to work simultaneously on the same project, or allow for integration with the configuration management system in the development environment. Once again, the correct interpretation of the term "project" is important. If a tool scales well enough to allow the entire analysis and design of a software system to be done in one project, this presents the problem of how multiple developers work on the same project at the same time. Many analysis and design tools are single user. With a sufficient degree of openness, a single-user tool integrated with a configuration management system can behave like a multi-user tool.[6]

We believe that the *open* option is the right one. Analysis and design tool vendors are not generally focused on providing extensible and scalable configuration management support. A

better option for them is to provide the 'hooks' that allow configuration management tools to be the repository for design information. Multiple users can then check in and check out various design units. We have successfully used the CMVC/6000 product on AIX to support multiple users of a single-user analysis and design tool.

The next issue is the number of units of granularity supported in the tool, which will be a key determinant for configuration management integration. Our experience indicates that the subsystem is a reasonable and sufficient level of granularity for most projects. This is based on using C++ and having subsystems that contain 5 to 50 classes. Your results may vary. Although the configuration management overhead associated with having class-level granularity is substantial, this is a reasonable requirement for domains where strict ownership and control are needed.

Since every software project is somewhat different, minor modifications to the analysis and design tool can be critical to ensuring reasonable adoption of the tool.

Maintain Different Versions of Model

A final scalabilty requirement deals with model versioning. The tool must be able to maintain different versions of a model or allow integration with the CM system to meet this need. Large projects will build systems in several increments.[8] Each increment will have an analysis and design model associated with it in addition to the code that is built. The iterative and incremental nature of the object-oriented development process dictates that multiple versions of the system will be analyzed and designed concurrently. One team may be designing increment *n*, while another is coding and testing increment *n-1*. Both require access to a 'version' of the analysis and design model that matches the code associated with the increment being worked. This requirement can be met by having a checkpointing or 'model freezing' facility in the tool or enabled by integrating the tool with the CM system used for the code.

Openness and Extensibility

In general, selected analysis and design tools should be extensible. Since every software project is somewhat different, minor modifications to the analysis and design tool can be critical to ensuring reasonable adoption of the tool. This section highlights two important 'openness and extensibility' items. It does not detail basic items such as add-on menus, callable Application Programming Interfaces (APIs) to the data dictionary, export formats, and user-interface preferences or resources.

Linking

Analysis and design models can be linked with other tools. Traceability is often required in object-oriented software development. In an object-oriented project, requirements can be linked to scenarios that can be linked to the dynamic model that illustrates them and the test cases that validate them. Items on models can also be linked to the source code that implements them. Many more possibilities exist for linking to support traceability.

Linking items in an analysis and design tool with items such as requirements, use cases, or code outside the tool requires openness. The tool should allow dictionary entries for an item to be extended. It should also allow the addition of functionality to the tool that uses this information to link to items from outside the tool or other tools from within the analysis and design tool.

Add-On Function Support

There may be more to be accomplished in a project's development process than what your chosen object-oriented development method identifies. A tool that supports the method may also need to support additional activities specific to your project's development process. To provide this support, the tool must offer features such as user-defined menus, and open its dictionary to the data used by these add-on functions.

It is not efficient to add menus to a tool if the data associated with these add-ons must be managed in a separate environment. For example, if a reuse ID was associated with some classes produced by your development organization, the ability to store that information with the rest of the data for that class would be helpful.

Advanced Requirements

Advanced requirements contains functions not found in most of today's analysis and design tools, or functions that are present in some, but still need revision. Although the lack of availability of some of these attributes should not cause you to reject a candidate tool, the presence of some of these capabilities might influence your decision-making. If the analysis and design tool supports these requirements, you can exploit the object-oriented paradigm more fully and use the tool later in the object-oriented life cycle.

Model Metrics

The design tool should be able to output certain metrics (such as number of classes, number of

changed classes, number of methods) about the models in a project. It might output these metrics or allow this type of information to be extracted from its dictionary. Metrics are an important project management tool. Today, some organizations pull object-oriented metrics from the source code. However, it is just as valid to measure these values in analysis and design models to make mid-project adjustments to the way the object model is being done.

Design Integration

The design tool should be able to distinguish between the design commitment or integration to assist metrics in distinguishing "real" design from early illustrations that have been put aside. The capability to mark or select classes that are system components should be part of checkpointing a design. Some tools will support identifying 'obsolete units' while others might have a reporting facility that will match classes defined on a static model against usages on dynamic models. Proper integration to the configuration management system can also help facilitate a clean 'build' of the design. The ability to populate a model from existing code and compare it to an existing model might also support finding the real 'design.'

Class Interfaces from Specifications

If the tool allows class information to be specified at a sufficient level of detail, it should be able to generate class interfaces (code and documentation) from specifications. The tool does not magically add any information to the interface when going through the generation step; however, it does generate code that reflects the specifications consistently. All class interfaces produced through generation may look and follow the same generation standard.

Code Generation Parameters

If a tool generates code only in one fixed format, its generation feature is probably not worth using. If the tool requires hand-crafting the code after it is generated, there is little value in the code generation support.

The tool should allow you to customize code generation parameters. This will enable you to do simple tasks such as controlling the names of the files produced by the generator, controlling where documentation is placed in the interface, or controlling advanced attributes that are tied to the version of your implementation language and your project's chosen programming style. It is also important to ensure that the code generation support works with the compiler being used for the project.

Static Diagrams from Source Code

A valuable attribute of an analysis and design tool is the ability to generate diagrams statically from source code; that is, reverse engineer static diagrams from source code. This can help you understand a software system without a well-documented design. Another use may be to understand what was actually implemented in a project.

This capability can also produce models of reusable class libraries or frameworks. These models can then be brought into the design tool and used in projects to tie the analysis and design of the software system to used class libraries and frameworks.

Comparison of Generated and Manually Created Diagrams

Generating diagrams creates more data, which must be easily compared to the analysis and design model. The ability to compare the differences in a generated diagram and a manually created one needs additional support in most tools today. It is not sufficient to 'eyeball' a design diagram in one window and a generated diagram in another.

A valuable attribute of an analysis and design tool is the ability to generate diagrams statically from source code.

State Model

Another requirement is the ability to execute the state model. If a tool has allowed you to specify the complex state transitions of an object, it may also allow simulation of the state model. This capability may require that you specify the state information for all objects in the system, which may only be practical for real-time or other state-intensive system development.

Implementation Code from State Model

You should be able to generate implementation code from the state model. Relating to the previous requirement, implementation code generation can also be supported where detailed state information has been gathered. Unlike the execution or simulation requirement, code can be generated only for those objects that have state diagrams.

Conclusion

Many requirements exist for object-oriented analysis and design tools using the definitions of those tools prevalent today. As we move into the future, various tools will be integrated and distinctions between the types of tools will blur. As code generation technology emerges and object-oriented frameworks and domain class libraries become more pervasive, the tasks that developers expect analysis and design tools to accomplish will broaden and evolve. More time will be spent exploring (looking for potentially reusable units) and interrogating (testing a reusable component to evaluate its ability to work in a particular situation) to ensure a consistent application of the object-oriented development process across the entire software development team.

References

1. Branson, Michael and Herness, Eric. "The Object-Oriented Development Process." *Object Magazine* (November-December 1993): 66-70.
2. Booch, Grady. *Object-Oriented Analysis and Design with Applications*. Redwood City, California: Benjamin/Cummings Publishing Company, Inc., 1994.
3. Rumbaugh, James; Blaha, Michael; Premerlani, William; Eddy, Frederick; and Lorensen, William. *Object-Oriented Modeling and Design*. Englewood Cliffs, New Jersey: Prentice-Hall, 1991.
4. Jacobson, Ivar; Christerson, Magnus; Jonsson, Patrik; Overgaard, Gunnar. *Object-Oriented Software Engineering—A Use Case-Driven Approach*. Wokingham, England: Addison-Wesley Publishing Company, 1992.
5. Wirfs-Brock, Rebecca; Wilkerson, Brian; Wiener, Lauren. *Design Object-Oriented Software*. Englewood Cliffs, New Jersey: Prentice Hall, 1990.
6. Branson, Michael and Herness, Eric. "Building an OO Development Environment on AIX." *AIXpert* (February 1995): 35-44.
7. Harel, D. Statecharts. "A Visual Formalism for Complex Systems." *Science of Computer Programming*, Volume 8, 1987.
8. Branson, Michael and Herness, Eric. "Structure and Management of Object-Oriented Projects." *Object Magazine* (May 1994): 33-38.

Michael J. Branson, IBM Corporation, AS/400 Division, 3605 Highway 52 North, Rochester, MN 55901. Internet: mjb@chvmw3.vnet.ibm.com. Mr. Branson is an advisory programmer, tools strategist, and object-oriented consultant to internal projects in Rochester. He has a BS in Computer Science from Purdue University.

Eric Herness, IBM Corporation, Software Solutions Division, 3605 Highway 52 North, Rochester, MN 55901. Internet: herness@vnet.ibm.com. Mr. Herness is a senior programmer working in object-oriented systems development. He has a BS in Business Administration from the University of Wisconsin at Eau Claire and an MBA from the Carlson School of Management at the University of Minnesota.