**IDEA GROUP PUBLISHING**
1331 E. Chocolate Avenue, Hershey PA 17033-1117, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

# Leveraging Objects for Mission-Critical Applications

**Mahesh S. Raisinghani, University of Dallas, USA**
**Bruce Adams, Electronic Data Systems, USA**

*Mahesh S. Raisinghani is an Assistant Professor of Information Systems at the Graduate School of Management, University of Dallas. Professor Raisinghani's research interests are in object-oriented database management systems, group dynamics/ knowledge management in virtual teams, strategic impact of electronic commerce on organizations, and global information systems.*

*Bruce Adams has been with Electronic Data Systems since 1986. He is a System Architect specializing in Object-Oriented systems development and is currently working in the Financial Industry Group. In the past he has worked as a consultant at AT&T, AT&T Bell Labs, Western Union, and GMAC. He is the co-author of EDS' Compass development process.*

*Conceptual integrity is central to product quality.*
—Fred Brooks, The Mythical Man-Month

## EXECUTIVE SUMMARY

This case study is based on an enterprise-wide consulting project for a financial services firm in a major metropolitan area of the southwest United States. This case addresses the underlying principles (i.e., techniques and processes) and real-world practical application of object orientation (O-O). The objectives of this case study are to reinforce the student's foundation in fundamental O-O concepts, to provide an in-depth example of the application of O-O analysis and design techniques and formalisms and to enable the student to transfer this knowledge to the student's actual work.

The merit of the development process described here, Compass, is that it presents a repeatable process for delivery of client server architectures, object oriented systems, and distributed objects and components. It helps manage three interdependent variables common to most projects, i.e., deliverables, resources, and time, in a cost effective and efficient manner. The concept behind Compass is to integrate the best-proven solutions currently available, drawing upon several best-of-breed approaches used within the Information Services industry.

### *Background*

EDS is implementing a mission-critical business system, which is the cornerstone for the customer's strategic vision. For EDS/customer confidentiality reasons the company will not be named.

The system is being developed for a company that wants to grow its customer base by a focused targeting of niche markets. In order achieve this goal it was deemed necessary to expand the technological capabilities provided by the current mainframe based system. These capabilities would allow the business to reach potential customers in new ways such as the Internet / Intranet, kiosks, and "portable" business offices using laptop computers. The new system was also required to provide flexibility in areas such as new product creation.

The motivation for using the O-O paradigm for software development is illustrated by a closer look at two primary approaches to software development. These are the function-oriented paradigm, where functions are defined first and data is defined later, and the data-oriented paradigm, where data are defined first and functions are defined later. Both these approaches have their drawbacks. The drawbacks of the function-oriented paradigm are as follows:

- The difficulty in maintaining the consistency between the data flow diagram (DFD) and the data dictionary.
- The difficulty in eliminating or minimizing the "ripple-effect".
- The difficulty in software reuse due to tight coupling between the functions, and treating data and functions separately.
- The difficulty in following data flows in structured analysis and building hierarchies of tasks in structured design.

The drawbacks of the data-oriented paradigm are as follows:

- The difficulty in maintaining the consistency between the process model and the snapshot model.
- The difficulty in comprehending the effect of the interaction between the functions which are not explicitly modeled in the process or transaction specification language.
- The difficulty in software reuse since data and functions are treated separately.
- The difficulty in converting an analysis model into a good software design.

The object-oriented (O-O) approach attempts to give a balanced treatment of functions and data. An object is a state and a set of methods that explicitly embodies an abstraction characterized by the behavior of relevant requests and an entity that has the state and functionality. In the O-O context, an object is "an instance of a class" (www.cyberdyne-object-sys.com/oofaq). Thus the class is the fundamental building block of OO software. A class determines everything about an object (Elmasri and Navathe, 1993; Brown, 1997). A class defines a data type where a type consists of both a set of states and a set of operations which transition between those states. A class provides a set of (usually public) operations, and a set of (usually non-public) data attributes representing the abstract values associated with instances of the type. Object-oriented concepts such as inheritance, polymorphism, encapsulation, and data abstraction that lead to compatibility, flexibility, reuse, extensibility, and easier maintenance are the primary reasons for using the O-O paradigm in this case.

Distributed objects are extended objects that are not restricted to a single program and can exist as independent entities and be remotely accessible by other objects. Rather than producing monolithic applications, distributed object systems tend to consist of a number of small units or components. Components are the smallest, self-managing, and independent parts of a distributed-object model that work in heterogeneous environments (Lewandowski, 1998). The key benefits of distributed objects are that they allow the development of scaleable client/server systems by virtue of modularized software that features interchangeable parts and an option to add components for custom solutions in advanced architectures.

### *Setting the Stage*

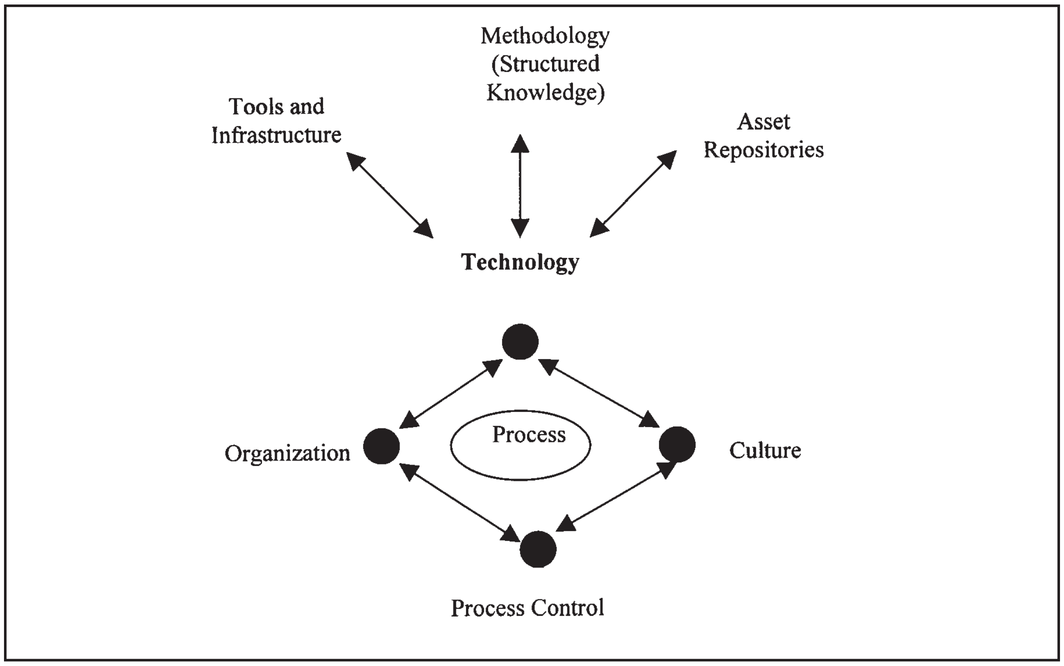
EDS and members of the existing information systems (IS) staff defined a process to gather defined project metrics for quantification of the results and a formal testing process in which deliverables must pass through a series of "quality-gates" (Q-Gate).

Compass, which was developed by EDS, is being used to provide a formal OO development process and OO methodology support. Like any of the popular O-O analysis and design approaches, Compass should not be seen as a definitive and comprehensive source of guidance for O-O software development. It should be treated as a set of pragmatic guidelines and instructional tools where the focus is on sensible use of formalism and process in laying the foundation for reliable O-O software development

## WHY IS ANOTHER PARADIGM NEEDED?

Current research indicates that practitioners are not necessarily using the methodologies as

***Figure 1: The Diamond of Change***



defined by their originators, but rather are customizing these methodologies to suit their application development needs (Hardy et al, 1995, Vlasborn et al, 1995, Vessey and Glass, 1998). The development process described in this case uses the best-practice approach in borrowing from several different methodologies both to overcome the deficiencies of any one methodology and to customize the approach to meet the needs of the project and organization.

More specifically, it addresses the two categories of deficiencies identified by Arnold et al. (1991). These deficiencies have to do with notations used to capture the system under development and/or the process used to drive the development from requirements definition to implementation. Business processes change with almost every project. The diamond of change represented in Figure 1 must be effectively managed in a balanced manner in order to respond to the needs of the business (AlBanna and Osterhaus, 1998). Shorter time to market and software replacement timeframes provide competitive advantage to the organization.

With the value of the U.S. software industry estimated at $92.5 billion and the U.S. currently providing almost 60% of the global software market (Albanna and Osterhaus, 1998), the diamond of change represents the forces shaping the software development environment. Although traditional organization theory assumes that the organization has a static and definable culture, the reality is that it is dynamic and IS managers of responsive, learning organizations need to make increasingly difficult decisions in the related dimensions of process, organization, and culture. The next section reviews the Compass process that addresses these dimensions of the diamond and creates a dynamic balance between them.

## *The Compass Process*

Compass is designed to support iterative OO development activities and to work in conjunction with existing EDS development processes for project management. This allows managers familiar with the existing project management approaches to adapt them to projects using *Compass.*

Compass supports two different perspectives on the project: the manager's perspective (the macro view) and the developer's perspective (the micro view). The manager needs to understand from a scheduling perspective how the project is organized and maps the different phases to a project schedule. The developer is also concerned with the phases but is also concerned with the daily development tasks, such as identifying abstractions, establishing the attendant relationships, identifying the semantics, and creating the various artifacts associated with each phase.

Compass was designed to be a developer's guide focusing on the critical areas in developing OO systems. Examples, templates, and detailed documentation explain the steps in the development process. This documentation includes a detailed view summarizing each step of the process, the inputs, the artifacts produced, and whether or not there is a quality gate inspection required. A "participation matrix" is also provided.

The participation matrix specifies, for each step, a listing of all the necessary project roles and their level of participation in the event. For example, producing an Analysis Behavior model the business analysis may have the primary responsibility, the technical lead the secondary responsibility and the architects might want to be notified of the results.

An important point to bear in mind is that the Compass process is derived from a combination of several popular O-O methodologies and does not represent an entirely new one. It is designed to synchronize and integrate with other existing processes, i.e. for project management and testing, to provide the support necessary for system development.

*Project Foundation*

The Compass process begins with a project startup phase called Project Foundation. Project Foundation's primary purposes are:

- Establishing the scope of the project
- Creating an architectural vision for the system to be developed
- Planning the organization of the project into Releases
- Assessing and managing risks identified within the project
- Creating the Project Plan
- Producing initial requirements for the project. This will include identification and preliminary specification of the Use Cases for the project. These specifications will be elaborated during the iterations.

Each Release of the overall project has a specific goal designed to stage the eventual delivery of the entire project. Each Release is broken up into an appropriate number of Iterations. Each Iteration functions within a Release in the same way that each Release functions within the overall project. Each Iteration may be considered a miniature project within its Release.

The core of the system will be developed initially and additional functionality will be added in layers. A useful metaphor for describing this "Core/Layer" approach to building systems is the onion. An onion is made up of a central core. The core looks, smells, and tastes like an onion, but a relatively small onion. The core of a system is like the core of the onion. To this core, each successive iteration and release adds layers of functionality onto the growing "onion" system. In this manner, capabilities are added to the core, which grows toward the finished product.

A significant byproduct of this process is that portions of the system are available very early in the lifecycle of the project. This allows the customer to provide early feedback and, if appropriate, portions of the system may be deployed to production creating a "staged" deployment.

*Project Release*

The Foundation phase is followed by one or more Releases. A Release produces a completed project or a shippable version of the software. Each Release is comprised of one or many iterations. Iterations are driven by Use Cases that are assigned to teams. Each release of the overall project has a specific goal designed to stage the eventual delivery of the entire project.

*Iteration Phases*

Each Iteration is made up of four phases, which correspond, to the traditional "waterfall" approaches: Conceptualization, Analysis, Design, and Evolution. Rather than being a sequential process, there may and should be considerable overlap among the phases.

For each new iteration there is a point at which the vision for the subsequent work is established.

Booch (1994, 1996) refers to this as "Conceptualization". The purpose of this step is to establish the "core requirements" and architectural vision. Establishing success criteria and performing risk management activities are also crucial artifacts of Conceptualization.

The purpose of analysis is to provide a description of the problem. The modeling during this phase will focus on abstractions that form the vocabulary of the problem domain.

The objective is to build a model of what the system does (its behavior) rather than how it does it (its form).

In practice, it is absolutely imperative that Analysis be kept separate from Design. Many developers tend to prefer the more technical design and coding phases. Unless there is a distinct Analysis phase, the tendency is for developers to go directly to design and coding. If the underlying Analysis models and the supporting requirements are weak, this in turn weakens the underlying semantic model. As stated earlier, this semantic model provides one of the major benefits of an OO approach.

The Design phase is begun as soon as there is a reasonably complete set of Analysis models for an iteration. The Design phase can be considered a process of expanding or extending the work that was begun in the analysis phase. The models developed in analysis are extended to include design heuristics and notations. Rather than switching from one type of modeling to another, during design the analysis models are transformed into a design.

Evolution is the process of taking the design artifacts and producing working components of the system. Included in this process are the necessary testing steps to ensure accuracy and conformance to established success criteria. Also during evolution the deployment plan that will be used to place the system into production is developed

Finally, the tested release package is deployed according to the deployment plan and turned over to production support.

## *Compass Architecture*

Compass revolves around the 4+1 View of Software Architecture illustrated in Figure 1. Each view addresses a specific area of interest to stakeholders. The four views of the architecture are organized around key use cases, which constitute a fifth view. Each view is described by appropriate notation that represents the view's key aspects.

The 4+1 Views can be characterized as follows:

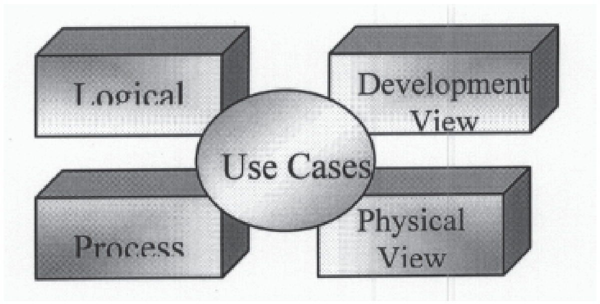
**Logical View** — describes how the software design supports the requirements of the system. This is captured as various O-O artifacts such as domain class models, dynamic models, and state transition diagrams.

**Process View** — captures the runtime aspects of the system, such as threads of control, process / node deployment, and the resource requirements of the systems.

**Physical View** — encompasses network topology, hardware, system software, transport mechanisms, data links, and DBMS.

***Figure 2: The 4+1 View of Software Architecture***

See Appendix A for further details on the Compass methodology



**Development View** — describes the development environment, tactical policy, and the architecture itself.

**Use Cases**—captures requirements and both demonstrates and validates the convergence of the four views.

## *Project Description*

The project is being developed using a CORBA based architecture running on UNIX servers. Users have PCs running NT clients running Java applets. The system also provides access to key mainframe legacy systems.

The system provides sales support and customer servicing for financial products that represent the core business of the company. The system will be deployed in over one thousand branches and supporting between five- and nine thousand users.

The system is expected to be relatively long lived, supporting the business over the next decade and possibly longer. An object-oriented approach was selected since much of the benefit from OO occurs in the maintenance phase of the lifecycle. This is due to several factors including the ability to support abstractions that provide encapsulation of data and behavior in objects.

An OO approach also supports a rigorous modeling effort that enables the establishment of a semantically accurate representation of the business. In fact, some early object oriented systems were developed to support real world simulation applications. Today, practitioners use OO approaches to develop systems that simulate the real world business that the system is supporting. This allows key workflows, products, relevant parties, and the accounting model to be supported by software that closely parallels the behavior of the business.

Such a semantically rich system is more adaptive to change than the current legacy system. It will also be more conducive to new and more flexible distributed object (Lewandowski, 1998) and network approaches, such as Intranets and Extranets. This will greatly enhance the range of employees that have access to the system and support a large amount of information needed for the financial transactions, distribution of objects across the network, and high performance of the system with servers of thousands of pages, frequently changing content and references.

Utilizing a web front end and a standard browser gives users greater access to key business systems. These systems are currently accessible only through character mode terminals on a local area network. This will reduce overhead costs for current and new sites requiring system access to support the business.

This approach will also enable new sales capabilities such as allowing direct dialup access to a mobile sales force equipped with laptops and a modem. This allows access to core business systems from a wide range of locations: customers' homes, kiosks at a special event, essentially anywhere with a telephone line.

These and other additional business objectives in developing the system can be summarized as follows:

- To provide a simplified and more intuitive interface that would reduce training costs and allow new employees to become productive more quickly.
- To provide a standardized architecture that would allow integration of the current product lines.
- To increase productivity with a modern system that would allow for greater flexibility in sales support.
- To provide a system that would allow for greater market penetration by utilizing technologies, portable computers for sales people in the field.
- To provide a system that would allow for faster response to market trends providing a competitive advantage in the market place.
- To establish an architectural approach providing a highly distributed network with a web-enabled thin client, fat server processing scheme supporting an enterprise wide Intranet.

### *Current Status*

The first phase of software has been released in a pilot deployment. Based on the continued success of the pilot, the system will be rolled out in a phased release to the existing field sites. The remaining application that make up the overall system will be completed and deployed as pilots and then rolled out in a phased manner.

### *Successes and Failures*

- The organization has developed a common software framework that can be used to support the evolution of the core business support applications.
- The first phase of the business applications has been delivered on schedule and has been deployed in pilot. It is scheduled to be phased-in to production in the near future.
- A solid semantic model of the business has been developed which can be reused in the development of additional applications in the financial domain.
- The customer is gaining a system that supports the technology necessary to the corporate strategic vision.

In addition to the EDS OO developers, the customer wanted IS staff to help with the development of the system. This knowledge transfer was viewed as critical for the long-term success of the project. The customer's IS staff members assigned to the project were a mix of experience levels: interns and new hires along with developers having years of experience on the legacy system.

Also the OO mentors have varying degrees of ability in the knowledge transfer. The skills that make a good OO developer are not the same skills that make a good trainer or mentor. The project has had mixed results with mentoring. It was found that some of the developers are better suited to strictly technical work. This did not present a problem once the resources were properly aligned.

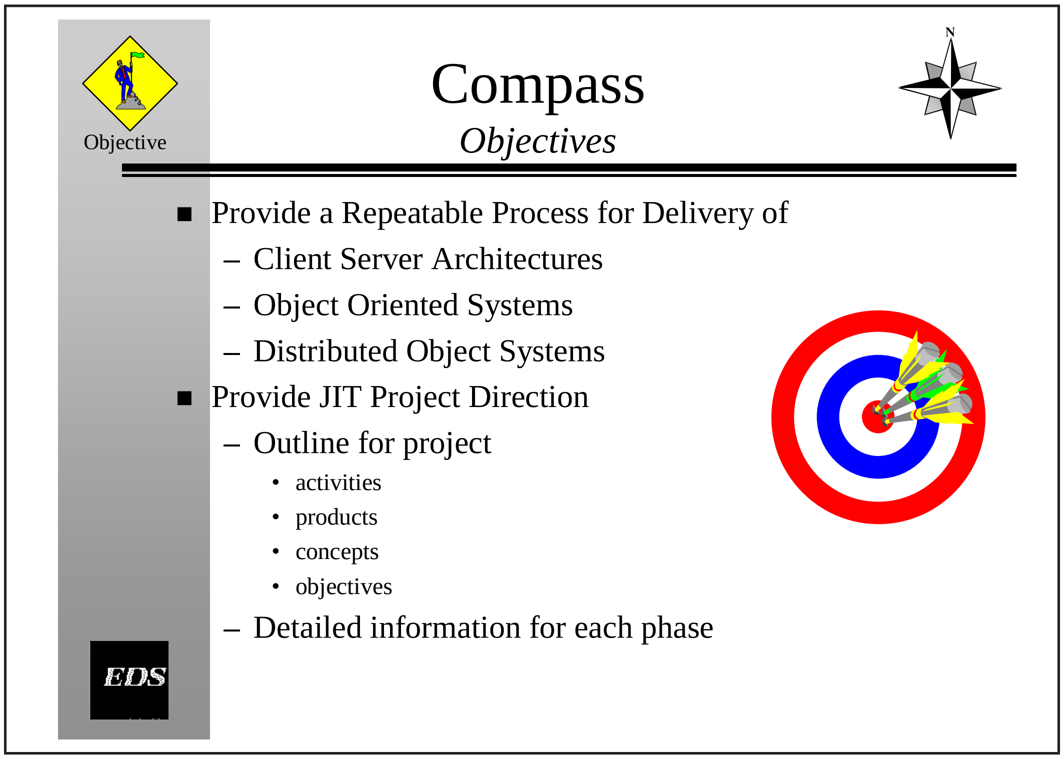
Finally, factors contributing to the lack of organized corporate O-O strategies include poor reuse statistics, lack of shrink-wrap business components, and an overall disenchantment with in-house development. The team members involved in this project placed a strong emphasis on assuring the stakeholders that each of these factors had been addressed in this solution. For instance, new techniques for estimating project size and tracking development (i.e., a Q-gate process) were developed to manage the iterative process of O-O software development.

## CONCLUSION

In conclusion, it is crucial to get the "buy-in" from the top management for successful implementation of an enterprise wide project. Since it is difficult to acquire systems engineering thinking and practice without continuous training and continuous learning from the lessons of both successful and failed projects, a systematic learning process that changes systems engineering thinking and improves the process is extremely important.

Although the Compass process is risk adaptive as opposed to risk aversive, there is intrinsically more risk in the evolutionary approach. Although the number of iterations necessary cannot be known in advance, it is advantageous to use a faster modeling technique such as the one described in this case study due to its inherent flexibility to mirror the rapidly changing real world.

## APPENDIX A



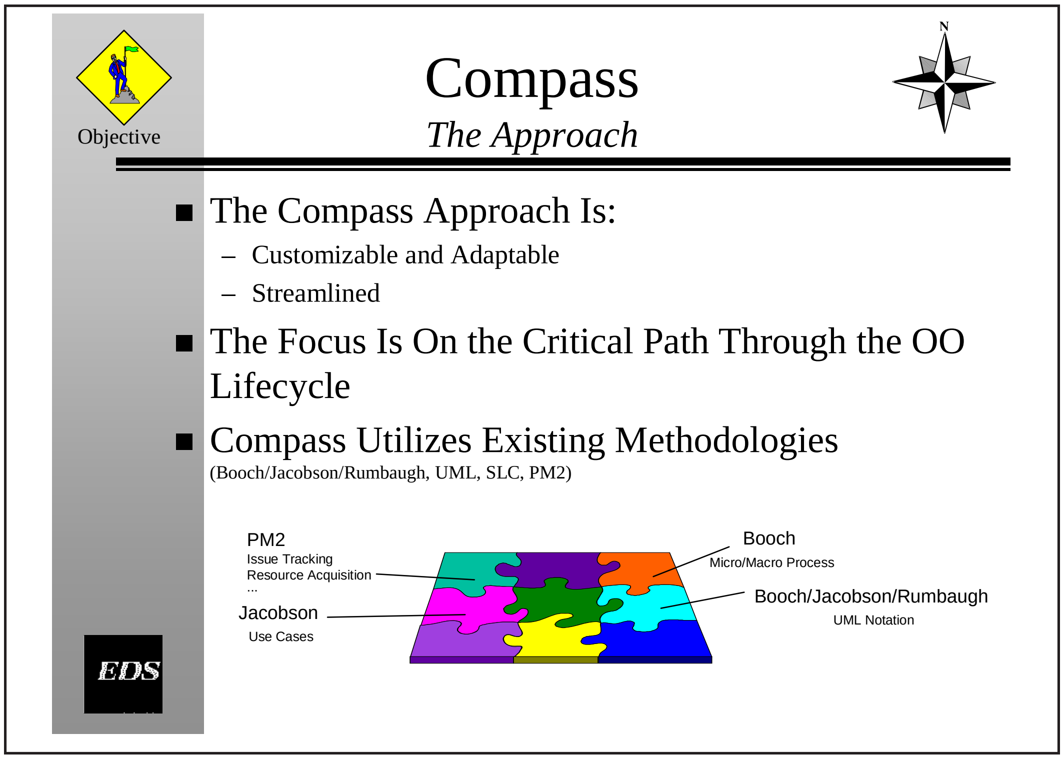
Objective

### Compass

*Objectives*

- Provide a Repeatable Process for Delivery of
  - Client Server Architectures
  - Object Oriented Systems
  - Distributed Object Systems
- Provide JIT Project Direction
  - Outline for project
    - activities
    - products
    - concepts
    - objectives
  - Detailed information for each phase

EDS



Objective

### Compass

*The Approach*

- The Compass Approach Is:
  - Customizable and Adaptable
  - Streamlined
- The Focus Is On the Critical Path Through the OO Lifecycle
- Compass Utilizes Existing Methodologies

(Booch/Jacobson/Rumbaugh, UML, SLC, PM2)

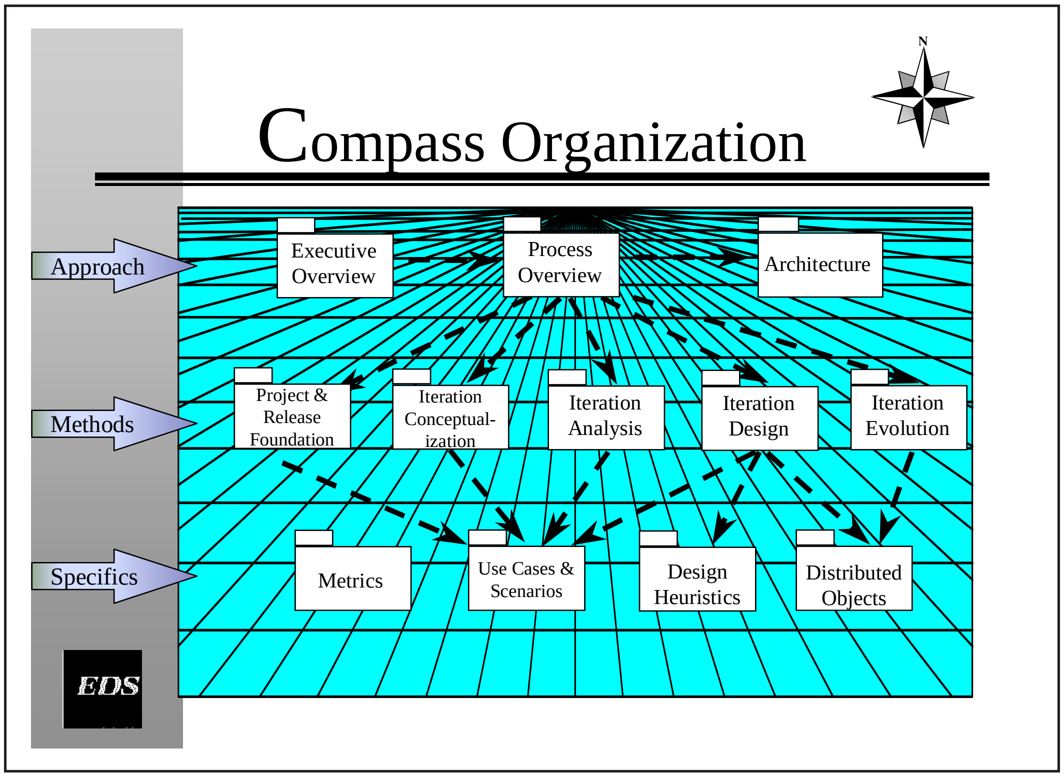
PM2
Issue Tracking
Resource Acquisition
...

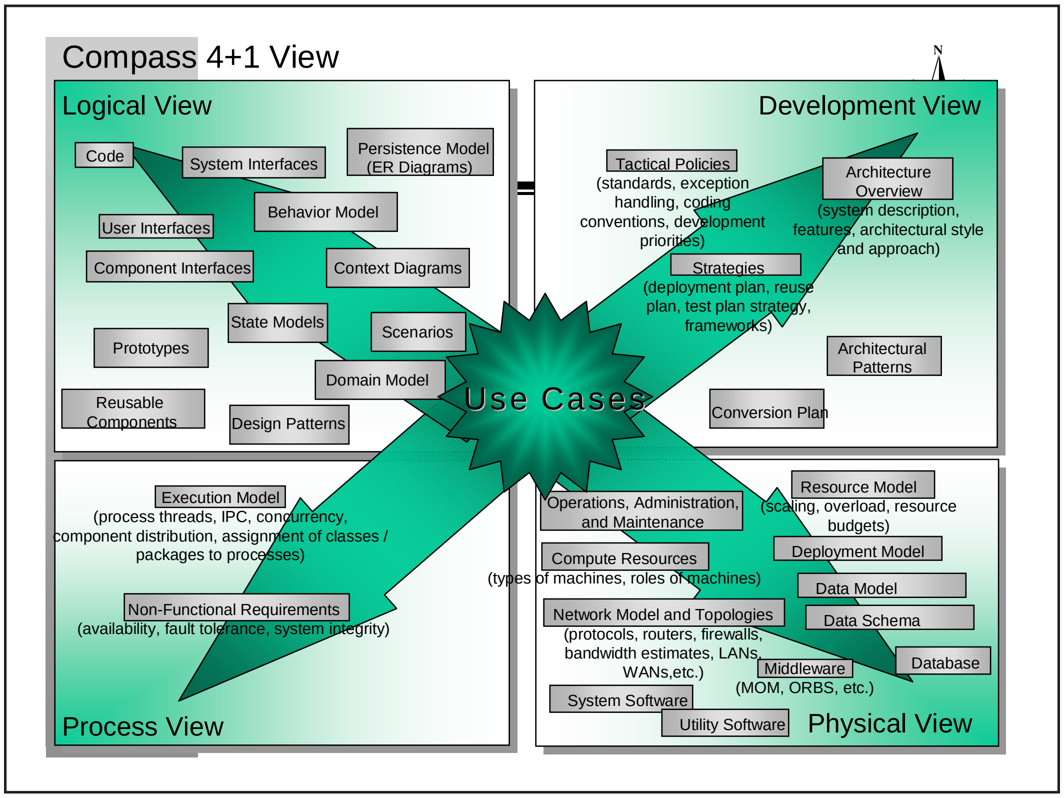
Jacobson
Use Cases

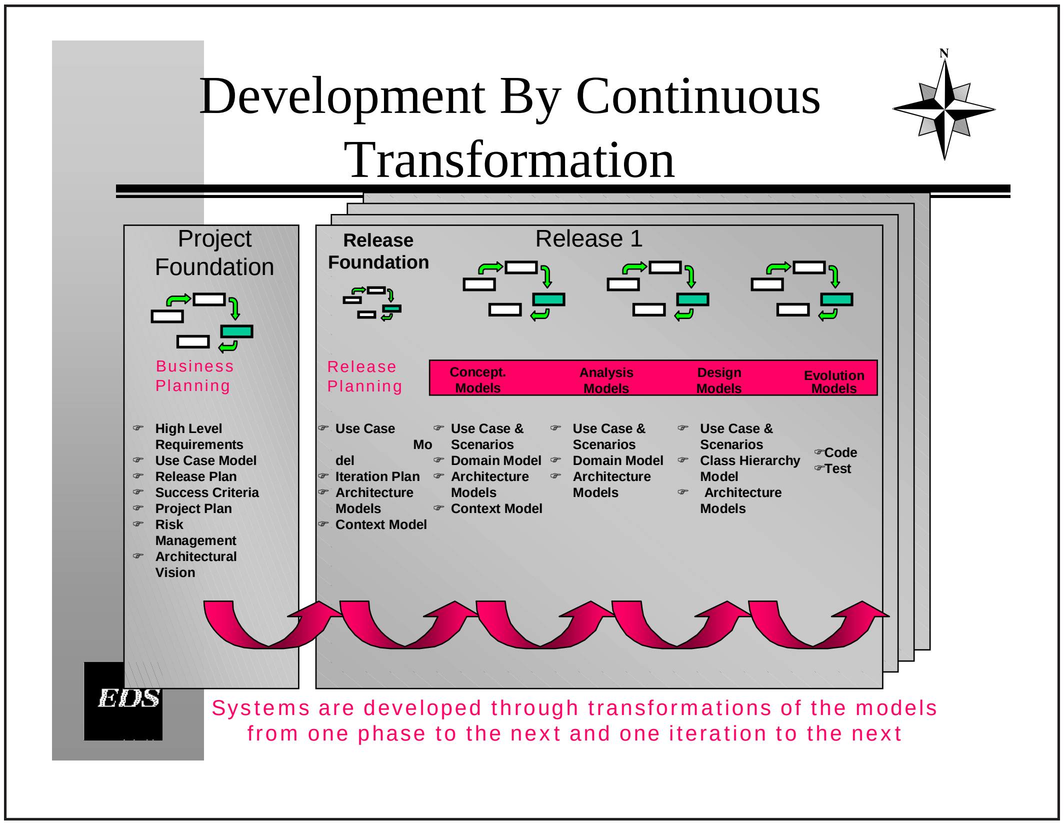
Booch
Micro/Macro Process

Booch/Jacobson/Rumbaugh
UML Notation

EDS


Compass Organization
N
Approach
Methods
Specifics
Executive Overview
Process Overview
Architecture
Project & Release Foundation
Iteration Conceptual-ization
Iteration Analysis
Iteration Design
Iteration Evolution
Metrics
Use Cases & Scenarios
Design Heuristics
Distributed Objects
EDS


Compass 4+1 View
N
Logical View
Code
System Interfaces
Persistence Model (ER Diagrams)
User Interfaces
Behavior Model
Component Interfaces
Context Diagrams
State Models
Scenarios
Prototypes
Domain Model
Reusable Components
Design Patterns
Development View
Tactical Policies
(standards, exception handling, coding conventions, development priorities)
Architecture Overview
(system description, features, architectural style and approach)
Strategies
(deployment plan, reuse plan, test plan strategy, frameworks)
Architectural Patterns
Conversion Plan
Use Cases
Process View
Execution Model
(process threads, IPC, concurrency, component distribution, assignment of classes / packages to processes)
Non-Functional Requirements
(availability, fault tolerance, system integrity)
Physical View
Operations, Administration, and Maintenance
Resource Model
(scaling, overload, resource budgets)
Deployment Model
Compute Resources
(types of machines, roles of machines)
Data Model
Network Model and Topologies
Data Schema
(protocols, routers, firewalls, bandwidth estimates, LANs, WANs,etc.)
Middleware
(MOM, ORBS, etc.)
Database
System Software
Utility Software



## Development By Continuous Transformation

**Project Foundation**

Business Planning

- High Level Requirements
- Use Case Model
- Release Plan
- Success Criteria
- Project Plan
- Risk Management
- Architectural Vision

**Release Foundation**

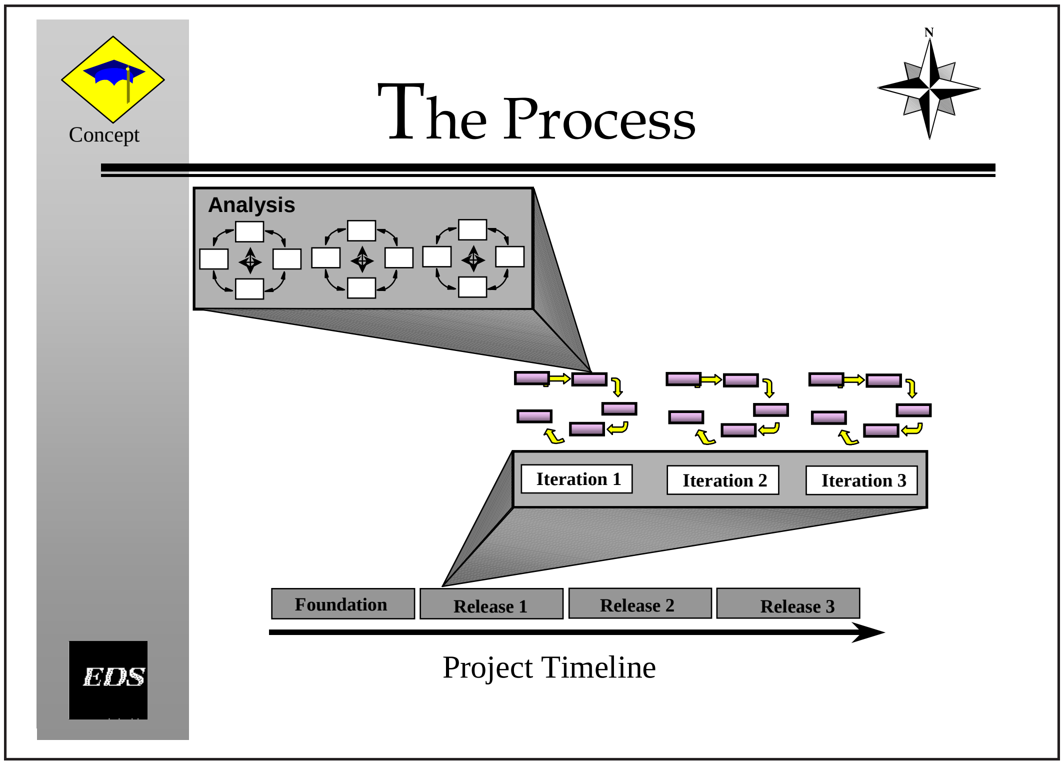
Release Planning

- Use Case Mo del
- Iteration Plan
- Architecture Models
- Context Model

**Release 1**

| Concept. Models | Analysis Models | Design Models | Evolution Models |
|---|---|---|---|
| Use Case & Scenarios<br>Domain Model<br>Architecture Models<br>Context Model | Use Case & Scenarios<br>Domain Model<br>Architecture Models | Use Case & Scenarios<br>Class Hierarchy Model<br>Architecture Models | Code<br>Test |

Systems are developed through transformations of the models from one phase to the next and one iteration to the next



## The Process

## ENDNOTE

[1]Details have been disguised to conceal the identity of the company and individuals involved in this case. The case is, however, an accurate depiction of object-oriented modeling using the Compass development process.

## REFERENCES

Arnold, P., Bodoff, D., Coleman, H., Gilchrist, H., and Hayes, F. (1991). Evaluation of five object oriented development methods. In *Journal of Object-Oriented Programming: Focus on Analysis and Design,* New York: SIGS Publication, Inc., 101-121.

AlBanna, S. J. and Osterhaus, J. (1998). Meeting the Software Challenge: A Model for IS Transformation. *Information Systems Management,* Winter, 15(1), 7-15.

Brown, D. (1997). *Object-Oriented Analysis-Objects in Plain English.* New York: John Wiley & Sons, Inc.

Booch, G. (1996). *Object Solutions: Managing the Object-Oriented Project.*Addison-Wesley.

Booch, G. (1994). *Object Oriented Analysis and Design,* 2nd Edition, Benjamin Cummings.

Coad, P. and Yourdon, E. (1991). *Object-Oriented Analysis.* Second edition. New Jersey: Prentice-Hall,. Inc.

Coad, P. and Yourdon, E. (1991). *Object-Oriented Design.* New Jersey: Prentice-Hall,. Inc.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., Jeremaes, P. (1994). *Object-Oriented Development-The Fusion Method.* New York: Prentice-Hall, Inc.

Elmasri, R. and Navathe, S. (1993). *Fundamentals of Database Systems*, II Edition, Benjamin/ Cummings.

Hardy, C. J. , Thompson, J. B. and Edwards, H. M. (1995). The use, limitations, and customization of structured systems development methods in the United Kingdom, *Information Software Technologies,* 37(9), September.

Kruchten, Philippe (1995). The 4+1 View Model of Architecture, *IEEE Software*, November

Jacobson, I., Christerson, M., Jonsson, P., and Overgaard, G. (1992). *Object oriented Software Engineering: A Use Case Driven Approach.* Reading, MA: ACM Press/Addison-Wesley.

Jacobson, Griss, and Jonsson (1997). S*oftware Reuse — Architecture, Process, and Organization for Business Success,* Addison-Wesley.

Lewandowski, S. M. (1998). Frameworks for Component-Based Client/Server Computing, *ACM Computing Surveys,* 30(1), 3-18.

Rumbaugh, J. (1994). Object-Oriented Modeling and Design. In *Proceedings, Object Expo 1994.* New York: SIGS Publications.

Rumbaugh, J., Blaha, M., Premerlani, F., and Lorenson, W. (1991). *Object-Oriented Modeling and Design.* Englewood Cliffs, NJ: Prentice Hall, Inc..

Vessey, I. and Glass, R. 1998. Strong Vs. Weak Approaches to Systems Development, *Communications of the ACM,* 41(4), April, 99-102.

Vlasborn, G., Rijsenbrij, D. and Glastra, M. (1995). Flexibilization of the methodology of system development. *Information Software Technology,* 37(11), November.