Payoff

This article compares structured development and object-oriented methodologies and their impact on software maintainability, reusability, and productivity. Although object orientation is, in concept, a superior methodology, IS managers must evaluate costs, benefits, and organizational factors before adopting it. Practices for migrating to an object-oriented development environment are discussed.

Introduction

With the ever-increasing complexity of IS requirements in business, the concepts and techniques underlying structured system methodologies are limited in their ability to produce high-quality products at a low cost within a limited time. The object-oriented (OO) methodology is considered to be a way to build quality products and deliver them quickly to users. It is thus important that IS managers understand why OO development is a better way of modeling complex realities, what factors they must take into consideration when making a decision to adopt this new methodology, and how to successfully move to object orientation. These are the issues this article addresses.

Basics of Structured Versus Oo Methodologies

There are essentially two main software development methodologies: structured and object-oriented. Their characteristics vis-a-vis the systems development process are summarized in Exhibit 1.
<table>
<thead>
<tr>
<th></th>
<th>Object-Oriented (OO)</th>
<th>Structured</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Concepts</strong></td>
<td>Object decomposition</td>
<td>Functional decomposition</td>
</tr>
<tr>
<td></td>
<td>Data abstraction</td>
<td>Procedure abstraction</td>
</tr>
<tr>
<td></td>
<td>Message passing</td>
<td>Functional call</td>
</tr>
<tr>
<td></td>
<td>Encapsulation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Inheritance</td>
<td></td>
</tr>
<tr>
<td><strong>Modeling Tools</strong></td>
<td>Class and object diagram</td>
<td>DFD (process model) and ER diagram</td>
</tr>
<tr>
<td></td>
<td>STD, timing diagram</td>
<td>STD</td>
</tr>
<tr>
<td></td>
<td>Service chart</td>
<td>Process specification</td>
</tr>
<tr>
<td></td>
<td>Object dictionary</td>
<td>Data dictionary</td>
</tr>
<tr>
<td><strong>Systems Life</strong></td>
<td>Bottom-up design</td>
<td>Top-down design</td>
</tr>
<tr>
<td><strong>Cycle Features</strong></td>
<td>Class library</td>
<td>Procedure library</td>
</tr>
<tr>
<td></td>
<td>Overlapping phases</td>
<td>Distinct phases</td>
</tr>
<tr>
<td></td>
<td>More incremental</td>
<td>Less incremental</td>
</tr>
<tr>
<td></td>
<td>More iterative</td>
<td>Less iterative</td>
</tr>
<tr>
<td></td>
<td>Smooth transition</td>
<td>Discrete transition</td>
</tr>
<tr>
<td><strong>Examples of Programming Languages</strong></td>
<td>C++, Smalltalk, CLOS</td>
<td>COBOL, C, FORTRAN</td>
</tr>
<tr>
<td></td>
<td>OO data base</td>
<td>Relational data base</td>
</tr>
</tbody>
</table>

Both structured and object-oriented methodologies have evolved from computer programming, which can be viewed from a functional paradigm or an object-oriented paradigm. In the functional paradigm, computer programs consist of interrelated modules that perform functional requirements. Thus, function (or procedure) is a primary focus and data is shared by functions or processes. Structured methodology is based on the functional paradigm. In the object-oriented paradigm, computer programs consist of interrelated classes of real objects; data and procedures are encapsulated within the object.

Structured analysis typically employs the data flow diagram (DFD) as a process modeling tool and the entity-relationship (ER) diagram as a data modeling tool. Data Flow Diagram are transformed into structure charts that show specific function modules and their hierarchical relationships. The Entity Relationship diagram as a logical data base model can be normalized and translated into a relational data base. The entire design is implemented using a traditional language such as Common Business Oriented Language or C or some 4GL in conjunction with (typically) a relational DBMS.

In object-oriented development, an object model is built to represent a problem domain and its solution domain. There are four major elements of an object model: abstraction, encapsulation, modularity, and hierarchy. Key abstractions are the classes and objects. Encapsulation refers to the bundling of data and methods together within an object so that data can be accessed only through the object's own methods. Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules. Hierarchy can be either aggregation or a generalization/specialization relationship.

Object modeling starts with the identification of classes and objects in a problem domain. Class diagrams shows classes and their relationships, and object diagrams show objects and messages between objects. Analysis stage merges into design stage to identify lower-level class objects and their attributes and methods. The reusable design components or classes are identified from class libraries. New subclasses or new superclasses are iteratively identified and hierarchical inheritance relationships are established. Design is implemented using Object-Oriented Programming Language such as Smalltalk and C++ in conjunction with some object-oriented DBMS or extended relational DBMS.
Comparison of Systems Life Cycles

Because structured and object-oriented methodologies are based on different modeling philosophies and techniques, they reveal different features of the systems life cycle. Whereas structured systems development relies on top-down functional decomposition, object-oriented systems development has attributes of both top-down analysis (identifying right classes and objects) and bottom-up design (reuse of codes or classes from the existing class libraries).

The process of object-oriented systems development is also more incremental, concurrent, and iterative than structured development. The use of class libraries facilitates rapid prototyping at the analysis and design stages, and the systems requirements specification is not fixed at an early stage of the life cycle. Analysts, designers, and programmers all have the same focus on objects and can work with the object model concurrently.

Whereas object-oriented development is characterized by a direct mapping between analysis and design phases, structured development has no such direct mapping. The transformation from a Data Flow Diagram to a structure chart is, for example, a significant perspective change. On the other hand, object-oriented development deals with the same object concepts for the problem-domain representation and the solution-domain representation, which implies more seamless transition from analysis to design to coding (see Exhibit 2). Hence, it is easier to trace the development process and validate whether the requirements are satisfied, which result in better quality products.

OO Versus Structured Methodology Life Cycle

Object-oriented development is characterized by the addition of reusable classes to the class libraries. One of the beneficial results of Object-Oriented Design is well-defined, validated classes that can be stored in the class libraries. The functional modules resulting from structured development are not as reusable as the object modules.

Software Quality and Productivity Issues

Maintainability

Maintainability is defined as the ease with which a software system can be corrected when errors occur and expanded to satisfy new requirements. The object-oriented methodology is believed to have a higher potential for producing more maintainable software systems than structured methodology. Maintainability is achieved by effectively using such OO properties as information hiding, polymorphism, and late binding.

Because each object has its own data and methods, any necessary changes are localized within the object. It is easier to add new objects without changing the existing program structure because object modules are highly independent. Polymorphism means that the response to the same message sent to different objects can be different depending on the characteristics of the objects. An object-oriented program using polymorphism eliminates the need for repeated “case” statements that frequently appear in a program written in procedural language. The result is less code that is more maintainable. The process of selecting the appropriate method on the basis of an object's type is called binding. Late binding is a useful feature because developers need not declare an object's type before
execution. Structured design does not effectively address information hiding and encapsulation, so changes to one part of a software system often cause a problem in a relatively unrelated program area.

**Reusability**

Software reuse includes employing existing code or design directly and modifying existing code or design to adapt to new application requirements. The object-oriented methodology has a greater potential of providing reusability than structured methodology at both the design and code level.

Encapsulation and message passing make Object-Oriented Design highly modular, resulting in more reusable design and code. Although structured methodology also encourages a high degree of modularity, coupling and coherence are often difficult to apply in practice in the structured design. Furthermore, whereas object-oriented methodology and such features as inheritance of common methods and attributes from superclasses and reuse of objects from class libraries, structured development fails to offer these software reuse mechanisms.

**Productivity**

Software productivity may be defined as the number of software development tasks performed per unit of some meaningful labor input. The main source of productivity for adopters of an object-oriented methodology comes from the proper use of class libraries developed from previous applications. Because greater reusability makes it possible to reduce the efforts for maintenance of existing applications, more resources become available for new applications development, thereby potentially reducing the applications backlog that exists in most organizations.

Apart from reusability, Object-Oriented Programming requires less code than structured programming with procedural languages because of its features of inheritance and polymorphism. In addition, because object-oriented concepts are thought to be more natural than structured specifications, Object-Oriented Analysis and design specifications may be more understandable to developers and users. This can result in improved communications between developers and users and easier validation of specifications, which in turn contributes significantly to software productivity.

**Analyzing Costs and Benefits of Object-Oriented Development**

It has been argued thus far that object-oriented is conceptually a superior methodology in terms of its impact of software quality and productivity. However, whenever IS shops adopt any new technology, there are costs involved. Management must weigh the potential costs and benefits when moving to object-oriented systems development.

Costs and benefits of using an object-oriented methodology can be tangible or intangible. Tangible costs include quantifiable items related to:

- The purchase of products, such as:
  - Languages and programming tools.
  - Developers' toolkits.
Object-oriented data bases.
- Object-oriented Computer-Aided Software Engineering tools.
- Class libraries.
- Hardware platforms.
- Personnel and training related costs, such as:
  - Hiring new personnel.
  - Education and training programs.
  - Fees for outside consultants.

Tangible benefits occur from faster systems development and-lower-cost maintenance. An object-oriented pilot project is one way of identifying quantifiable benefits.
Intangible costs are associated with efforts to overcome organizational resistance to the new technology implementation. Software personnel with accumulated experience with traditional methodologies may resist the new methodology, emphasizing the disadvantages of using it. A decline of morale among developers may delay projects. Difficulties may also arise when organizational rules and procedures have to be changed to accommodate the new methodology.
Intangible benefits include opportunities to employ multimedia and better Graphical User Interface, and to have easier communication between developers and users and increased end-user satisfaction because of a more correct requirement determination.

Factors Favoring Object Orientation

Migrating to object-oriented systems development may be beneficial to a company only if the benefits of adopting the new methodology exceed the costs. Certain salient situational factors and project characteristics have to be studied to determine if the ratio of expected benefits to costs is favorable.

Degree of Change in Systems Environments.

The more dynamic the IS environment, the more attractive the object-oriented methodology is likely to be. These environments demand high-quality systems, faster systems development, satisfying frequent changes in user needs, and adaptability to changing business environment. These requirements are better satisfied by object-oriented rather than traditional structured methodologies.

Number of Projects.

The larger the number of software projects in a company, the greater the relative advantage of adopting an object-oriented methodology. Because different projects may contain common components of classes or objects that can be reused from the class libraries, the object-oriented methodology can result in faster and less expensive software development.
Complexity of Data.

Object-oriented data bases can handle applications using complex data (e.g., voice, video, and graphics as well as text and numbers) more efficiently than relational data bases can.\(^ {72} \) The object-oriented data base is more compatible with Object-Oriented Analysis and design than structured analysis and design. CASE, Computer-Aided Design/Computer-Aided Manufacturing, office systems, and other science and engineering applications have already been developed using object-oriented technology. As multimedia applications become popular in the areas of business Transaction Processing systems, expert system, and executive information systems, the benefits of using object-oriented technology may well outweigh its costs.

Complexity of Problem Domain.

The more hierarchical the nature of the problem domain, the more complex is the systems development projects. It is difficult to decompose a complex system into lower-level manageable components and allocate scarce resources to each component. The ability of OO modeling to deal with hierarchies through aggregation and generalization can mitigate these problems. The lack of modeling semantics in the structured methodology makes it difficult to partition complex systems properly.

Dynamics of Problem Domain.

The features of real-time systems include simultaneous processing of tasks, extensive communications between tasks, assignment of different priorities to different tasks, and interruption of one processing task before it is finished. A set of notations to express concurrency is an important requirement for modeling real-time systems.\(^ {73} \) Although the extensions of structured analysis include modeling tools (e.g., Data Flow Diagram) showing control process and control flow, they may not be an adequate means of dealing with concurrency.

Degree of Networking.

Increased use of distributed data base and client/server computing will likely increase the adoption and use of OO methodology.\(^ {74} \) The Object Management Group (OMG) is preparing a reference model for an Object Management Architecture to facilitate interoperability of software components in a distributed heterogenous environment. Such standards are expected to contribute to the diffusion of object-oriented technologies.

An Analysis Tool

The following analysis, which is based on the previous discussions, is intended to help IS management decide whether it is beneficial to migrate from a structured methodology to an object-oriented development environment. The analysis framework (shown in Exhibits 3 and 4) depends on quantifiable costs of adopting and using a methodology, as well as the

\(^ {73} \) G. Booch, Object Oriented Design with Applications (New York: Benjamin/Cummings Co., 1991).
intensity of the situational factors. For the sake of simplicity, cost functions are assumed to be linear. Total costs are divided into fixed costs and variable costs for each methodology. Fixed costs are constant across the intensity levels of the situational factor, whereas variable costs are proportional to the situational factor.

Maintenance Volume Example

Exhibit 3 shows how the volume of maintenance influences a firm that is currently using structured methodology to consider adopting OO methodology. Volume of maintenance is a major component of the degree of change in systems environments. The fixed cost of structured development does not affect the methodology selection decision, because it represents expenses spent in the past and does not vary with the frequency and size of maintenance. The slope of total costs (fixed plus variable) of using a methodology represents the increase in costs due to an increase in the volume of maintenance. Because object orientation has higher reusability, higher maintainability, and better modeling power, there are increased fixed costs but a decrease in variable costs.

Effect of Maintenance Volume on Methodology

For example, the low volume of maintenance of company A in Exhibit 3 indicates that the firm may continue to use structured methodology; the high volume of maintenance for company C signals that the adoption of OO methodology may be the better choice. For the indifference point B (where total costs of the methodologies are equal), OO development may be recommended if maintenance requirements are likely to increase as the company grows.

An overall analysis to determine whether OO methodology is a desirable option for a firm can be based on an aggregated situational factor that is a weighted sum of all situational factors and project characteristics. As shown in Exhibit 4, if the overall situational factor falls on the left-hand side of the indifference point X, the firm need not adopt the new software development methodology. On the other hand, if it falls on the right-hand side of the indifference point, then the adoption of OO methodology is recommended.

Effect of Overall Situational Factor on Methodology

One limitation of this analytical model is that it incorporates only quantifiable items. Thus, intangible items that are not easily quantifiable also must be taken into consideration when making the adoption decision. In addition, integrated Computer-Aided Software Engineering (ICASE) tool (e.g., Texas Instrument's IEF and KnowledgeWare's ADW) support for structured methodology upgrades the methodology considerably to model systems with a complex problem domain and frequent changes in systems environment. As shown in Exhibit 5, the situation on the left-hand side of X1 indicates the continued use of structured methodology, the situation between X1 and X2 suggests the use of structured methodology supported by Integrated CASE tool, and the situation on the right-hand side of X2 signals the adoption of OO methodology.
Effect of Overall Situational Factor on Methodology Selection With ICASE Tool Use

Assessing Organizational Resources

Although the analytical model incorporates major factors influencing the adoption decision—namely, the relative economic merits (i.e., tangible costs) of the two methodologies as a function of software project characteristics—other important organizational and technological factors must also be assessed before moving to object-oriented development.

In those industries in which information systems are essentially customer-driven and strategically important, object-oriented technology is likely to be more attractive. Financial and telecommunications companies are currently showing a greater tendency to adopt and use object-oriented technology mainly because of the competitive nature of these industries. Without sufficient human and financial resources, it would be impossible to implement object-oriented software projects even if the implementation is desirable. Availability of staff members who understand OO technology and have experiences with object-oriented projects will make it easier to move to OO environments. It is essential to provide continued financial support for training staff, developing reusable software components, and experimenting with object-oriented methodology. Opportunities to learn object-oriented techniques by doing or with the help of outside consultants cannot be gained without organizational slack.

Assessing the State of the Technology

Potential object-oriented methodology adopters are expected to understand the current status of object-oriented technology, so they can judge the timing of an adoption decision. Even though object orientation is conceptually a superior methodology in terms of software quality and productivity, there are practical factors that may make it a less attractive choice to organizations. Object-oriented methodology is still evolving and is not as mature as structured methodologies. Its supporting tools are yet to be fully developed and the principles and guidelines for development are not as well understood or well defined as those for structured methodology.

Other technical factors act as hindrances to organizational adoption of object-oriented systems, namely:

- The overhead of processing messages is high, because dynamic binding in message processing can be more time-consuming than straight function calls in other languages.

- Systems developers need time to learn very large class libraries.

Early adopters of object-oriented methodologies have to assume the costs of experimenting with available OO techniques, but they stand to solve their problems with software engineering earlier and adapt their information systems faster to meet customer demands. Late adopters may lose the opportunities for accumulating know-how of object orientation, but they will be able to easily obtain validated class libraries from vendors and acquire object-oriented software personnel at a lower cost.
Hybrid Approaches to Migration

Pure object-oriented methodology is characterized by a systems life cycle of Object-Oriented Analysis, Object-Oriented Design, and object-oriented implementation. For many organizations, though, a direct migration to OO methodology may mean abandoning enormous investments in the traditional methodology. One way of mitigating this problem is to follow a hybrid (or evolutionary) approach to migrating toward object-oriented methodology.

Hybrid Systems Life Cycle
The traditional functional systems life cycle can be mixed with object-oriented systems life cycle. There are two possible ways of accomplishing this.75

Functional Analysis, OO Design, OO Implementation
It is not easy to identify classes and objects directly from the real world. One way of mitigating this problem is to find objects from Data Flow Diagram and Entity Relationship diagrams. Entities, data stores, and terminators may be objects, and processes may be services of objects. The documentation of structured techniques continues to be used at the systems analysis phase. In this approach, there remains the difficulty of moving from analysis to design, when compared to pure object-oriented life cycle.

OO Analysis, OO Design, Functional Implementation
This approach implements object-oriented design in a procedural language that may support object-oriented concepts to a varying degree. There have been some efforts to give third-Generation Language features of object orientation (e.g., Object Common Business Oriented Language). If an organization has large investments in procedural programming such as COBOL and Ada, it may be hard to replace the procedural languages environment with a pure object-oriented language environment. Even though it is not possible to take full advantage of object orientation, a procedural program translated from Object-Oriented Design offers many of the advantages of Object-Oriented Programming.

Evolutionary OO Methodology
One evolutionary methodology that has been suggested is termed object-oriented structured design, which provides a standard design notation by supporting concepts of both structured and object-oriented design.76 By using this evolutionary approach, systems designers can shift smoothly from current design practices to a style that introduces object-oriented concepts.

Hybrid OO Languages
There are basically two types of languages supporting object-oriented concepts: pure languages (e.g., Smalltalk) and hybrid languages (C++, CLOS, Object-Pascal). Because a hybrid language is one that has added object-oriented features to procedural languages, it is often missing the features of pure languages and may be less powerful. Whereas it is easier for nonprogrammers to learn pure languages, experienced programmers who are already familiar with the base language may find it easier to learn a hybrid language.

Hybrid Object Data Base (Extended Relational Data Base)
An object-oriented data base stores objects (both data and procedures), whereas a relational data base stores tables (only data). When compared to a relational data base, a pure object-oriented data base (e.g., GemStone, Ontos) allows such relative advantages as fast access, flexibility, and storing binary large objects. An extended relational data base (e.g., UniSQL/X DBMS) is a hybrid approach that offers the advantages of the pure object-oriented data base without the need to abandon established relational technology. This hybrid approach is a recommended migration strategy to object orientation if the application is traditional transaction data processing that uses only simple data.

Because a hybrid approach allows the use of existing computing resources, its adoption costs are less than those for a pure object-oriented methodology. For example, existing code can be used by creating object wrappers around the code. For many organizations, the ultimate goal of using a pure object-oriented methodology may be reached through the use of a hybrid approach during a period of transition.

Management Practices for Implementation
To successfully move to object orientation, a supportive organizational environment for the change must be established. Some important managerial practices can be followed in this regard.

Observing Results of Pilot Projects
To successfully assimilate object-oriented methodology into an IS organization, pilot projects should be selected in areas where traditional methodology has failed to meet the expectation and object-oriented methodology is more appropriate. Pilot projects are a way of justifying object-oriented technology adoption when user organizations do not know what the results will be.

Many firms that have brought in languages such as C++ to experiment with in pilot projects have found that the object-oriented techniques cost more and take more time than traditional methods. One possible reason for this could be that these firms did not experiment full scale with pure object-oriented methodology (as a paradigm shift), focusing only on Object-Oriented Programming, which is only one aspect of object-oriented systems development. This narrow focus on programming rather than on the entire systems life cycle can be viewed as an idiosyncrasy of the early stage of object-oriented implementation. Furthermore, sometimes there are too few trained people to complete the project on time, so trained people have to spend some of their time training others, which in turn makes meeting project deadlines even less likely.

Management should be aware that in the early stages of OO implementation, lower productivity may be observed among analysts and programmers, but eventually productivity should increase as more projects are tried out. The increase in productivity would be achieved through cumulative learning of object-oriented techniques and the production and use of accumulated reusable components. An increase in productivity means a decrease in the slope of Total Costs of OO Methodology in Exhibit 4, thus making OO methodology more cost-effective.

Even if the risk of implementing an object-oriented methodology can be reduced by observing results of pilot projects, conversion of all the software projects to object orientation may cause conflicts between the existing systems and the new object-oriented systems. To minimize the conflicts, the scope of object-oriented applications should be enlarged only when marginal benefits of object orientation are greater than or equal to
marginal costs of object orientation. The less risky strategy is to progress toward object-oriented systems step by step.

**Training and Education**

One of the greatest barriers to the implementation process may be the lack of knowledge and skills for carrying out object-oriented projects within the organization. Designing and using existing software components is more important than simply understanding Object-Oriented Programming Language. It is hard to find trained object-oriented programmers, and training object-oriented software engineers can take up to a year. To succeed in introducing object-oriented methodology, budgets should be available for formal training, relevant journals and magazines, internal consulting help, or external help from consulting firms or vendors.

Training and education should not be restricted to the professional IS staff alone. Users should also be included in this effort, especially in the early stages of implementation. Educating both IS staff and end users about object-oriented concepts and principles can eliminate communication problems between the two groups. Training and education not only promote learning object-oriented techniques, but also reduce the resistance of software staff who worry about losing their jobs. Training and education is a long-term investment that pays off when software productivity increases through the accumulation of specific object-oriented technology that could be the basis for a competitive advantage.

**Maintaining Realistic Expectations**

Top management support plays an important role in increasing the scope and rate of assimilation of the new technology within an organization. The allocation of a sufficient budget and close monitoring of the progress of implementation contribute to organizational learning of object-oriented methodology. Realistic management expectations are important, however, because unrealistic expectations can lead to a negative assessment of early efforts of object-oriented projects, which in turn may jeopardize further experimentation with the methodology. Management should understand that opinions of project leaders and team members who have experimented with object-oriented pilot projects can influence attitudes of other software staff.

**Reuse Planning**

Although reusability is the key to greater productivity of object-oriented methodology, programmers might prefer writing new code to using existing code unless a proper software reuse practice is established. Reusability may be fine in a small project group setting, but it may not be practical across an enterprise without organization. Some companies have created corporate support centers for software components to facilitate their internal software sharing. Performance reviews of analysts and programmers must include how they reuse software components from the class libraries and how they produce and add reusable software components to the libraries.

**Software Estimation Technique**

An accurate and reliable software estimation technique should be prepared to plan for object-oriented software development. One simple approach is estimation by analogy, in which an estimator who knows the effort required to develop similar software systems

---

78 Love, *Object Lessons*
estimates the effort required in the object-oriented software development. It is not clear yet what software metrics are appropriate for object-oriented software. Because object orientation is a paradigm shift, much more research is needed to provide guidelines for the practice of object-oriented systems development.

Conclusion

The decision of whether an IS organization should adopt an OO methodology can be made with the help of the analytical framework proposed in this article. This analysis is based on an accurate assessment of the costs and benefits—tangible and intangible—of structured versus object-oriented methodologies.

The framework is by no means the final authority for making a migration decision. Besides the economic factors, potential adopters of OO methodology must assess the relevant organizational resources and the characteristics of their development projects to determine the methodology’s fit with the organization. In addition, the current status of OO technology needs to be assessed. Among the technological issues to keep current with are emerging guidelines for analysis and design, modeling tools and OO Computer-Aided Software Engineering technology, OO technology standards, and class libraries.

Organizations that have difficulties moving directly to a pure OO methodology may consider implementing one of the hybrid migration approaches described in this article. Adopters are recommended to establish pilot projects, training and education programs, reuse planning, and project and personnel management practices fitting this new technology. Because few mainstream IS organizations have yet to adopt OO methodologies, there also needs to be continued research to collect empirical support (or lack thereof) for the arguments presented in this article to further guide the practice of object-oriented systems development.

Bibliography


Author Biographies

Bijoy Bordoloi

Bijoy Bordoloi is an associate professor of IS and management sciences at the University of Texas at Arlington. He received his PhD in MIS from Indiana University, Bloomington. His current research interests include adoption and diffusion of emerging information technologies, economics of IT, logical data base design, and software metrics.

Min-Hwa Lee

Min-Hwa Lee is a PhD candidate in IS at the University of Texas at Arlington. He received his MBA from Seoul National University, South Korea. His research interests include object-oriented methodology, outsourcing, and applications of artificial intelligence in business.
a. **Object-Oriented Life Cycle Phases Overlap and Each Phase Has the Same Focus on Objects.**

b. **A Life Cycle for Structured Methodology Has Distinct Phases and Each Phase Produces a Different Model.**
SFC  Fixed costs of structured Methodology
STC  Total costs of structured methodology (fixed plus variable)
OFC  Fixed costs of OO methodology
OTC  Total costs of OO methodology (fixed plus variable)

Note:
The slope of STC and OTC is the same as that of the variable costs.
SFC  Fixed costs of structured Methodology
STC  Total costs of structured methodology (fixed plus variable)
OFC  Fixed costs of OO methodology
OTC  Total costs of OO methodology (fixed plus variable)

Note:
The slope of STC and OTC is the same as that of the variable costs.
SFC  Fixed costs of structured Methodology
STC  Total costs of structured methodology (fixed plus variable)
OFC  Fixed costs of OO methodology
OTC  Total costs of OO methodology (fixed plus variable)

Note:
The slope of STC and OTC is the same as that of the variable costs.