

The COR Model for Analyzing Information Systems Change

Donald J. Berndt and Alan R. Hevner

Department of Information Systems and Decision Sciences

College of Business Administration

University of South Florida

Email: berndt@bsn.usf.edu and ahevner@bsn01.usf.edu

Abstract

Today's world is one of rapid and constant change. Computer-based information systems, as integral components of our businesses and personal lives, must change and adapt to maintain their effectiveness. IS professionals face the challenges of analyzing, enabling, and controlling information systems changes. In this paper, we present a formal model for viewing and analyzing information systems change. Drawing upon the biological model of punctuated equilibrium, we identify equilibriums in the IS life cycle at which system metrics can be collected and compared with previous equilibriums. Based upon one or more system metrics, a set of COR measures are developed to study system core, obsolescence, and recency. We demonstrate the application of the COR model to two interesting cases of information systems change. Conclusions and future research directions complete the paper.

1. Information Systems Change

Change is the only constant in today's world. For information systems to survive, they must continually adapt to changing conditions and new business environments. At the same time, new and improved information systems instigate changes in the environment (i.e., act as change agents). IS professionals who are responsible for developing and maintaining systems face the significant challenges of analyzing, enabling, and controlling information systems changes.

Information systems change and evolve over time. The complete IS life cycle begins with a conceptual phase, moves through initial system development, continues its life in operation with ongoing maintenance and evolution, and, eventually, reaches obsolescence. There are many opportunities

for system change to occur during this complete life cycle. In Section 2, we present an original way of viewing the IS life cycle model based upon the biological theory of *punctuated equilibrium* [Gould 1985]. In essence, information systems evolve over time from one stable phase (i.e., equilibrium) to the next. Changes are introduced into the system at specific points in time (i.e., punctuations) that delineate equilibriums. System change can be measured and analyzed between two or more equilibriums in the life cycle. This new view provides IS professionals with added insights for managing and enabling system change.

Based on the concept of a punctuated system life cycle, we develop a mathematical model for analyzing change, termed the COR model. Between any two equilibriums in the life cycle we can select appropriate system metrics and analyze system *core*, *obsolescence*, and *recency*. The COR vector of values can then be extended into a matrix based upon the definition of a second dimension that categorizes the change, such as the level of automation.

The following section describes the punctuated life cycle model. The COR model is described in detail in Section 3. In Section 4, we demonstrate two interesting applications of the COR model. These realistic case studies show the range and power of the COR model for studying and managing IS change. The paper concludes with our future research directions.

2. The Punctuated IS Life Cycle

Just as the evolution of biological life forms is no longer viewed as a smooth, continuous process, so too should we view the development and evolution of information systems as a punctuated, and not continuous, process. Figure 1 presents a high-level view of a typical information system life cycle process.

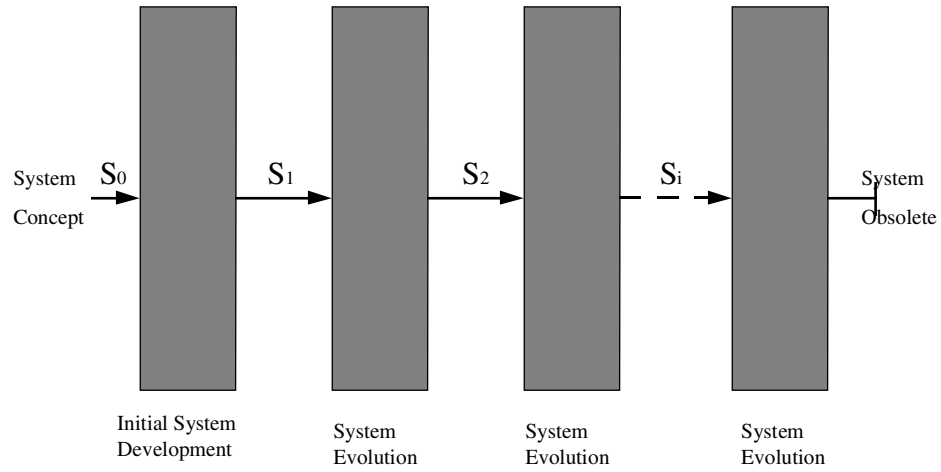


Figure 1: Information Systems Punctuated Life Cycle

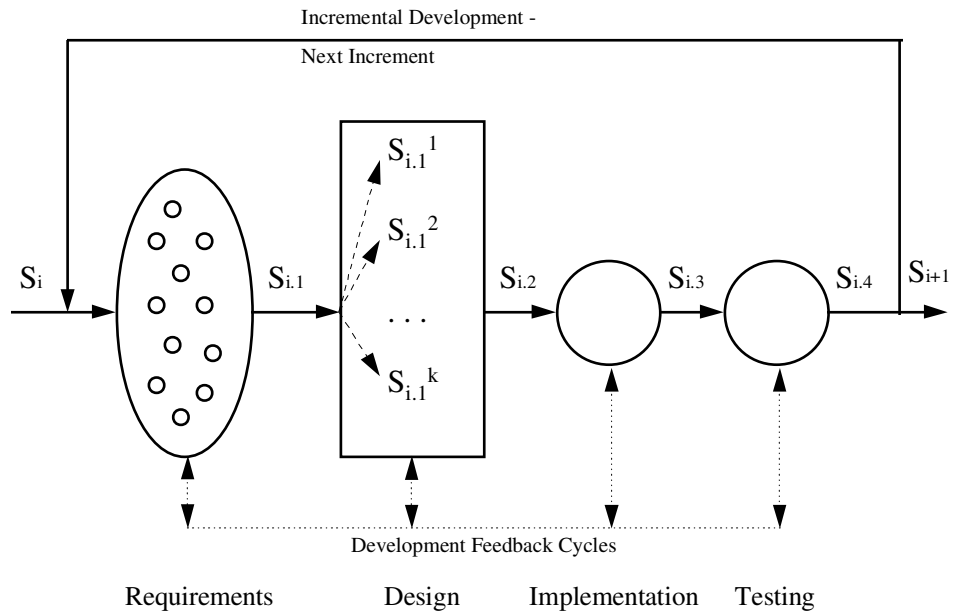


Figure 2: Punctuated Development Process

At the macro-level of the process, the system, S_i , represents the i^{th} equilibrium (e.g., operational release). The punctuations, represented by development and evolution subprocesses, are the points of system change.

At the next lower level of process detail, we can discover equilibrium points within the system development and evolution subprocesses¹ (the punctuation points of Figure 1). The development process, as shown in Figure 2, is made up of four development phases:

- Requirements - The description of system objectives and intentions, typically in structured English or some other semantically-rich representation.
- Design - A creative description of how the requirements are to be satisfied via a combination of hardware, software, and human behavior. Design representations include data, procedures, coordination, and human-computer interfaces.
- Implementation - The design is implemented as an operational computer-based information system. Hardware, software, and human behavior components are integrated into a functional whole.
- Testing - The integrated system undergoes a series of tests to ensure its compliance with the requirements specification. Defects found in testing may result in iterations (i.e., feedback loops) of requirements, design, and implementation activities to resolve.

There are a number of system development process models in use [Trammell et al. 1996]. All combine some variation of sequential and iterative progress through the above four activities. For example, the *incremental development process* builds systems in a succession of cumulative subsets of user function. Each system increment is built on top of the previous increment in another pass through the four development phases. The important principle is to select and customize the appropriate process model to the individual information system and goals of development (e.g., high-quality, time-to-market, etc.).

Although development activities seem to change the system continuously, we can identify several

¹ Evolution is differentiated from development in that an existing system is available upon which changes are made. Our subsequent discussion of the system development process will encompass both development and evolution.

equilibrium points in the development process. At the completion of each development phase, a stable version of the system, in terms of its artifacts (e.g., requirements, design, code, documentation), is produced. These system equilibriums are represented as $S_{i,j}$ in Figure 2.

The punctuated process model can also provide an effective way to evaluate design alternatives, as shown in Figure 2. Disciplined creativity in system development is contained in the design phase. The evaluation and selection of design alternatives is an important activity. An analysis of change should be an integral part of this decision. For example, an acceptable design with less change than a competing design may have significant advantages for implementation and testing, as well as operational maintenance. We designate design alternatives in Figure 2 with a superscript, $S_{i,j}^k$. Before the development process continues to the next implementation phase, a decision must be made to select one of the design alternatives.

Now, based on the identification of equilibriums, both macro (Figure 1) and micro (Figure 2), in the punctuated system life cycle, we present a mathematical model for analyzing system change.

3. The COR Model

In this section, we develop a formal model for analyzing information systems change.² Our goal is to define a general model that is independent of the life cycle stage and associated metrics. That is, the model should be applicable at the macro level of system releases (see Figure 1), as well as at the micro level within a single pass of development that may include the comparison of design alternatives (see Figure 2). We would like to measure change between any two equilibrium points in the system life cycle. This requires us to instantiate the model by defining a countable metric over the entities of interest (e.g., requirements or design artifacts, such as data flow diagrams). It is important that the metric be a meaningful measure of system change. There are a number of useful metrics that can be used with the COR model [Grady 1992, Moller and Paulish 1993]. For example, the following metrics are appropriate for use in the model.

² The COR model is an extension of the model proposed in [Berndt 1993] and subsequently applied in [Lucas et al. 1996].

- Requirements—Number of requirements
- Design—Data flow diagram processes, Objects, Object methods
- Implementation—Lines of code, Function points
- Testing—Test cases, Open defects

The selection of one or more metrics for use in the COR model will be determined by the objectives of the change analysis.

3.1 The COR Vector

We will develop the model in several stages, beginning with a set-theoretic approach to the basic framework. The model rests on the comparison of an appropriate metric based upon system artifacts at two equilibrium points in the system life cycle. For instance, in the Merrill Lynch case discussed later, we use processes described in data flow diagrams (DFDs) as the unit of analysis within the design phase. The COR model begins by identifying a common ‘core’ between system descriptions. In the case of DFDs, this might be accomplished by developing diagrams in tandem, with system overlap supported by simple labels or encoding by the drawing tool. The three major categories identified in the model are the common *core* entities, *obsolete* entities, and *recent* entities. For example, suppose we are comparing an artifact at equilibrium point S_i with the previous version of the artifact from equilibrium point S_{i-1} . If we imagine these two artifacts as sets of entities E_i , such as DFD processes or requirements, we can define the sets corresponding to our three main categories.

1. *Core* entities are the entities that are common to both equilibrium points, the set intersection $E_i \cap E_{i-1}$, where $c_i = |E_i \cap E_{i-1}|$.
2. *Obsolete* entities are the entities in the old system that are not in the new system, the set difference $E_{i-1} - E_i$, where $o_i = |E_{i-1} - E_i|$.
3. *Recent* entities are the entities in the new system that are not in the old system, the set difference $E_i - E_{i-1}$, where $r_i = |E_i - E_{i-1}|$.

This artifact-level analysis requires that we build a mapping between entities in the two artifacts, putting paired entities in the core set and unpaired entities in the obsolete or recent sets. The analysis yields the vector of three values, defined from the perspective of the i^{th} system. Once the mapping is constructed, we can calculate the following summary measures.

- $core\ commonality = \left(\frac{c_i}{c_i + o_i + r_i} \right)$
- $obsolescence = \left(\frac{o_i}{c_i + o_i + r_i} \right)$
- $recency = \left(\frac{r_i}{c_i + o_i + r_i} \right)$

Core commonality gives us a measure of system overlap; the stable entities that exist in both artifacts. Obsolescence analyzes the number of retired entities existing only in the old system. Recency measures the novel entities introduced in the last punctuative interval.

We now move to a finer level of detail, noting that the common core can be partitioned into two new sets. Some entities will be *altered* (i.e., modified), while others will be *unaltered* (i.e., unchanged between artifacts). This new dichotomy leads to a revised vector of measures as depicted in Table 1.

Table 1: COR Model Values at Equilibrium S_i

Entity Set	Variable
Core	c_i
• Altered Core	a_i
• Unaltered Core	u_i
Obsolete	o_i
Recent	r_i

This new level of detail leads to new measures of change. Stability measures the system overlap that has remained unchanged, while enhancement analyzes the modifications to core entities.

- $stability = \left(\frac{u_i}{c_i + o_i + r_i} \right)$
- $enhancement = \left(\frac{a_i}{c_i + o_i + r_i} \right)$

Another measure to consider here is an aggregate measure of change, which includes entities that have been modified in the core, as well as obsolete and recent entities. This can be characterized as system-wide change as defined below.

Table 2: Automation Matrix in COR Model

Entity Set	Human	Human-Computer	Computer
Core	c_i^h	c_i^{hc}	c_i^c
• Altered Core	$(a_{old}^h, a_{new}^h)_i$	$(a_{old}^{hc}, a_{new}^{hc})_i$	$(a_{old}^c, a_{new}^c)_i$
• Unaltered Core	u_i^h	u_i^{hc}	u_i^c
Obsolete	o_i^h	o_i^{hc}	o_i^c
Recent	r_i^h	r_i^{hc}	r_i^c

- $change = \left(\frac{a_i + o_i + r_i}{c_i + o_i + r_i} \right)$

We can also calculate the current system size in total number of entities, $t_i = c_i + r_i$. A second measure of interest is the number of modified entities, $m_i = a_i + o_i + r_i$. These sums can be used to produce different forms of the above measures. For instance, one might define stability with respect to current system size, or just the core (c_i), rather than the total number of entities.

3.2 Characterizing Change: The COR Matrix

The above analysis starts with three simple categories and introduces two subcategories that reflect important aspects of change. However, we do not define the type of change that is being measured. In this section, we extend the simple vector of measures to form a COR matrix. In order to form the matrix, we add a second dimension which characterizes the type of change being analyzed.

For instance, in the Merrill Lynch case we consider the degree of entity automation. Three coarse levels of automation are identified: *human* processes, *human-computer* (i.e., computer-aided) processes, and fully *computerized* processes. Any group of change categories can be used to define this second dimension. Other potential change dimensions include: cost estimates, time estimates, and level of complexity. Table 2 shows a COR matrix constructed with three levels of system automation and the entity sets derived above. The set of altered entities is further subdivided to reflect their appropriate categorization in the old and new systems. That is, a before/after dichotomy is used to preserve the information regarding how the altered entities evolved between the equilibrium points. This allows us to measure shifts among the change type categories, comparing the two equilibrium points. The altered entities within the core are the only entities which may have shifted categories between the old and

new systems (requiring two entries in the matrix). The other entity classes will have only one categorization from the perspective of the new (i^{th}) system.

By grouping the human-computer and computer automation categories as automated entities, we define the level of automation in the current system, S_i , and the previous system, S_{i-1} , as follows.

- $automation_i = \left(\frac{(c_i^{hc} + c_i^c) + (r_i^{hc} + r_i^c)}{c_i + r_i} \right)^3$

- $automation_{i-1} = \left(\frac{(a_{old}^{hc} + a_{old}^c) + (u_i^{hc} + u_i^c) + (o_i^{hc} + o_i^c)}{c_i + o_i} \right)$

These two values can be compared to determine the change in automation between the two equilibrium points. A possible goal is for the level of automation to increase as you go from S_{i-1} to S_i .

The COR model can be used effectively in many different environments and at several levels of abstraction. The next section presents three case studies that apply the model in very different contexts.

4. Applications of the COR Model

The challenge of this research is in the application of the COR model to interesting information systems contexts. The case studies presented in this section demonstrate important examples of change analyses.

³ Note that c_i reflects the current system S_i ; that is, c_i is the sum of the unaltered entities and the new altered entities. Therefore, the term $(c_i^{hc} + c_i^c)$ is equivalent to $(a_{new}^{hc} + a_{new}^c) + (u_i^{hc} + u_i^c)$.

4.1 Reengineering at Merrill Lynch

Merrill Lynch is one of the largest financial services firms in the country, operating more than 500 branch offices. One important service is the handling of physical certificates, such as stock certificates and any associated legal documents. Merrill Lynch is responsible for receiving customer certificates at branch offices, storing the certificates for safekeeping, and posting the holdings to customer accounts. Handling certificates without loss or mistakes is both good customer service and a policy enforced by the Securities and Exchange Commission (SEC). The high-level process consists of six steps as presented below. For a more detailed discussion of the process see [Lucas et al. 1996].

1. The customer brings certificates to the branch office.
2. The branch office provides preliminary processing.
3. The certificates are sent to a securities processing center.
4. The center verifies and checks the certificates.
5. The center processes certificates, routing them to a final destination.
6. The center posts data to the customer's account.

Merrill Lynch receives approximately 3,500 certificates each day, satisfying a variety of customer needs. A customer may simply want Merrill Lynch to hold certificates for safekeeping or the customer may be surrendering certificates after selling stocks. Other reasons for handling certificates include the inheritance of certificates, bond calls, or company reorganizations (with new shares being issued). Whatever the reason, Merrill Lynch depends upon secure certificate handling and the timely posting of customer information.

Merrill Lynch originally handled certificates in two regional securities processing centers, which processed and forwarded certificates to a home office center and other destinations. Problems were often discovered late in the process, and were resolved by telephone negotiation with branch office representatives. At times, this meant contacting the customers and having them bring additional documentation during subsequent visits to the branch office. A second characteristic was the removal of the physical certificates from the vault and transfer to other departments for review. At each stage of the transfer, a microfilm record was kept as an audit trail for tracking any lost certificates (as per SEC regulations).

In order to better meet certificate handling demands, Merrill Lynch reengineered the certificate handling process. The new certificate processing system relies on digital image processing, optical character recognition, and an extensive reorganization of the workforce. A legal expert system and a document collection system help reduce errors in branch offices. Electronic tracking of the certificates provides a check on physical processing. Lastly, once the certificates are digitized, the images can be sent anywhere for review without removing the physical certificates from secure facilities. A case study of the certificate handling process included the development of data flow diagrams for both the old and new systems [Lucas et al. 1996]. In this section, we present the results of reexamining these data flow diagrams using the COR model. In the context of the punctuated life cycle, we are comparing two design phase artifacts (i.e., data flow diagrams).

Table 3 is the detailed COR matrix that results from analyzing the certificate processing DFDs with respect to automation categories. Table 4 presents the change measures for the reengineered certificate handling process.

Table 3: Automation Matrix for Merrill Lynch Case Study

Entity Set	Human	Human-Computer	Computer
Core	3	7	5
• Altered Core	(7, 0)	(0, 5)	(3, 5)
• Unaltered Core	3	2	0
Obsolete	3	0	0
Recent	0	0	4

Table 4: COR Change Measurements

Change Measure	Value
Total Entities	19
Modified Entities	17
Core Commonality	0.68
Obsolescence	0.14
Recency	0.18
Stability	0.23
Enhancement	0.45
Change	0.77
Current Level of Automation	0.84
Previous Level of Automation	0.28

How can we interpret these measures? The COR matrix in Table 3 shows us that a fairly large set of entities are in the core set, existing in both the old and new systems. However, the altered/unaltered dichotomy shows that many of these processes are modified with respect to their automation category. The old and new categories within the altered set show a definite migration from manual processes to computerized processes. It is also interesting to note that all obsolete processes are manual and the recently developed processes are fully computerized.

The measures in Table 4 serve to reinforce these conclusions. Commonality (0.68) indicates a large core set. However, stability (0.23) tell us that only a small portion of entities remain untouched. The system-wide change (0.77) highlights the significant difference between equilibrium points—consistent with the idea of a reengineering effort. Lastly, the current automation (0.84), as compared with the previous automation (0.28), shows a large gain with regard to the extent of automation in this human-computer information system.

4.2 Controlling Requirements Changes

In the development and evolution of software intensive information systems, the management and control of requirements is an extremely difficult activity. Many postmortem reports on failed system development projects cite *requirements control* as the number one cause of project failure [Glass 1992, Air Force 1994]. A key symptom of loss of control in requirements is termed ‘requirements creep.’ Requirements creep occurs when the set of system requirements never becomes stable. Requirements are continually being added, deleted, or changed by the customer and the development team.

The system designers are always trying to hit a moving target and rarely succeed.

An ideal system development scenario would freeze (i.e., stabilize) the set of requirements at the beginning of the project. However, this is unrealistic. Humphrey [1995] states a *Requirements Uncertainty Principle*:

For a new software system, the requirements will not be completely known until after the users have used it.

Therefore, since requirements cannot be completely specified at the start of the project, they must evolve over the course of system development and the complete system life cycle. Requirements definition is a never-ending activity. A means for controlling and tracking requirements changes is essential. System development projects must establish an initial requirements baseline as a basis for change negotiations. This baseline becomes a signed agreement between the customer and the development team. All changes to the requirements set are evaluated as to their feasibility and impact on project schedule, budget, and staffing. [Humphrey 1989]

The use of the COR model is a natural and effective way to measure and control requirements change throughout system development and evolution. We present two scenarios of how the COR model can be effectively used to measure requirements change. We use a simple count of requirements as a metric in the model. Each requirement entity in the system is given a unique identifier for traceability throughout the system life cycle. The details of requirements numbering schemes and traceability matrices are well-covered in references such as [Davis 1993, Gilb 1988]. The following two scenarios illustrate how tracking the number of requirements via the COR model provides important insights for requirements control.⁴

Scenario 1 - Requirements Changes during Design

A time of significant project risk occurs during the initial design activity, when the development team makes a first pass at interpreting customer requirements. Glass [1992] cites preliminary data from an IBM development project that showed an ‘explosion’ of requirements during the high-level design phase. Each customer requirement was exploded into from 10 to 100

⁴ The numeric values used in the scenarios do not come from an actual project; however, they are representative of real software development projects on which the authors have worked.

new system requirements by the design team. Regardless of whether the new requirements were true system requirements or design features, it is critical for the project to monitor changes in the number of requirements during the development process and identify situations where requirements explosions occur.

From Figure 2, we identify the initial set of customer requirements at equilibrium point $S_{0,1}$ and the new set of requirements after design at equilibrium point $S_{0,2}$. After the requirements phase, the total number of requirements is $t_{0,1} = 650$. Upon completion of the first incremental design phase, the COR values are found in Table 5 and the change analysis measurements are contained in Table 6:

Table 5: COR Measures for Scenario 1

Entity Set	Value
Core	$c_{0,2} = 610$
• Altered Core	$a_{0,2} = 210$
• Unaltered Core	$u_{0,2} = 400$
Obsolete	$o_{0,2} = 40$
Recent	$r_{0,2} = 180$

These measurements indicate an above average amount of requirements volatility during the initial design phase. (Although the values are well below the requirements explosion cited by Glass [1992].) This significant amount of change should alert the project manager to the risk of requirements instability. The introduction of requirements inspections with participation from requirements analysts, system designers, testers, and customers would be a strong recommendation for the project [Wheeler et al. 1996].

Table 6: COR Change Measures for Scenario 1

Change Measure	Value
Total Requirements	790
Modified Requirements	430
Core Commonality	0.73
Obsolescence	0.05
Recency	0.22
Stability	0.48
Enhancement	0.25
Change	0.52

Scenario 2 - Requirements Change Between Releases

The evolution of software-based products is typically controlled via a release schedule. Each new release contains new features, changes in existing features, and defect fixes. Changes between successive releases can be effectively monitored by using the COR model on number of requirements in the new release. A major risk to providing timely, high-quality product releases is the tendency to ‘bite off more than you can chew’ on a given release. A high percentage of requirements change between release equilibrium points would indicate this risk.

In Figure 1, we denote successive releases in the system life cycle as S_{i-1} and S_i . In system release S_{i-1} , there are a total of $t_{i-1} = 830$ requirements satisfied. The evolution of the system to release S_i results in the following COR values in Table 7 and the change measurements in Table 8.

Table 7: COR Measures for Scenario 2

Entity Set	Count
Core	$c_i = 815$
• Altered Core	$a_i = 35$
• Unaltered Core	$u_i = 780$
Obsolete	$o_i = 15$
Recent	$r_i = 50$

Table 8: COR Change Measures for Scenario 2

Change Measure	Value
Total Requirements	865
Modified Requirements	100
Core Commonality	0.93
Obsolescence	0.02
Recency	0.06
Stability	0.89
Enhancement	0.04
Change	0.11

These measurements appear typical for a release schedule that is under control. Approximately 11% of the system requirements have been modified in the new release. Maintaining a historical baseline of system change metrics over all releases would provide invaluable support for estimating the schedule, budget, and staffing for future releases.

5. Conclusions and Future Research

We have presented a formal model for analyzing and controlling information systems change. There are two principal contributions of this paper:

1. The adaptation of the biological concept of punctuated equilibrium to form a punctuated life cycle for information systems. This provides a more realistic perspective on the uneven development path and life cycle of most information systems.
2. The development of the COR model for measuring information systems change. This model complements the punctuated life cycle, measuring change between equilibrium points at various levels of abstraction. The COR vector compares a meaningful metric across two development artifacts at two different system equilibrium points. Change measures are recorded for core entities (altered and unaltered), obsolete entities, and recent entities. The COR vector is expanded into a COR matrix by adding a change dimension, supporting more detailed analysis.

We include two case studies to illustrate the theoretical models: a reexamination of the Merrill Lynch securities processing system, a design-level exercise, and a derived case study showing an effective use of the COR model to measure and control requirements change.

We are investigating two interesting directions to extend our research on information systems change. The first direction is the automation of the COR model. A key activity in the model is to identify a 'core' set of common entities between system descriptions. That is, an entity-level mapping must be constructed, with common entities paired. This is a labor-intensive activity in which the 'content' of entities must be analyzed. We plan to investigate automated tools to assist in the development and maintenance of system mappings. An entity-level database, with support for version control, would provide part of the required infrastructure. With regard to design tools, explicit support for the tandem development of alternative system descriptions would provide important support for easily recognizing a common core. For example, we can imagine maintaining a collection of data flow diagrams with common, obsolete, or recent entities displayed as appropriate.

The second active research direction is the application of the COR model to object-oriented (OO) system development. We plan to develop a case study

on the development of a real OO system. We will experiment by selecting various OO design artifacts and metrics to use in the COR model. Potential metrics would include: [Chidamber and Kemerer 1994, Lorenz and Kidd 1994]

- Number of objects
- Number of methods
- Coupling between objects (e.g., messages)
- Depth of inheritance trees

Booch [1996] states, "Stability is perhaps the best predictor of the health of an object-oriented system." His leading indicator of stability is the rate of change of architectural interfaces. Therefore, using interfaces as the design artifact, we will use the COR model to measure and analyze changes during the OO development phase and the complete punctuated life cycle of the OO system.

Acknowledgments

We wish to acknowledge the contributions of Hank Lucas and Greg Truman to the development of the original change model that appeared in [Lucas et al. 1996]. We also thank the referees of this paper for their thoughtful comments.

References

- [Air Force 1994] Department of the Air Force, *Guidelines for Successful Acquisition and Management of Software Intensive Systems*, Software Technology Support Center, 1994.
- [Berndt 1993] D. Berndt, "Techniques for the Comparative Analysis of Data Flow Diagrams," Working Paper IS93-006, Stern School of Business, New York University, 1993.
- [Booch 1996] G. Booch, *Object Solutions: Managing the Object-Oriented Project*, Addison-Wesley Publishing, Co., 1996.
- [Chidamber and Kemerer 1994] S. Chidamber and C. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering*, Vol. 20, No. 6, June 1994, pp. 476-493.
- [Davis 1993] A. Davis, *Software Requirements: Objects, Functions, and States*, Prentice-Hall, Inc., 1993.
- [Gilb 1988] T. Gilb, *Principles of Software Engineering Management*, Addison-Wesley, Inc., 1988.

- [Glass 1992] R. Glass, *Building Quality Software*, Prentice-Hall, Inc., 1992.
- [Gould 1985] S. Gould, *The Flamingo's Smile: Reflections in Natural History*, W.W. Norton and Co., 1985, p. 241.
- [Grady 1992] R. Grady, *Practical Software Metrics for Project Management and Process Improvement*, Prentice-Hall, Inc., 1992.
- [Humphrey 1989] W. Humphrey, *Managing the Software Development Process*, Addison-Wesley Publishing Co., 1989.
- [Humphrey 1995] W. Humphrey, *A Discipline for Software Engineering*, Addison-Wesley Publishing Co., 1995.
- [Lorenz and Kidd 1994] M. Lorenz and J. Kidd, *Object-Oriented Software Metrics*, PTR Prentice-Hall, Inc., 1994.
- [Lucas et al. 1996] H. Lucas, D. Berndt, and G. Truman, "A Reengineering Framework for Evaluating a Financial Imaging System," *Communications of the ACM*, Vol. 39, No. 5, May 1996, pp. 86 - 96.
- [Moller and Paulish 1993] K. Moller and D. Paulish, *Software Metrics: A Practitioner's Guide to Improved Product Development*, Chapman & Hall, Inc., 1993.
- [Trammell et al. 1996] C. Trammell, M. Pleszkoch, R. Linger, and A. Hevner, "The Incremental Development Process in Cleanroom Software Engineering," *Decision Support Systems*, Vol. 17, 1996, pp. 55 - 71.
- [Wheeler et al. 1996] D. Wheeler, B. Brykczynski, and R. Meeson, Jr., editors, *Software Inspection: An Industry Best Practice*, IEEE Computer Society, 1996.