Chapter 3
Traditional Data Oriented versus Process Oriented Reengineering of Legacy Systems

Raul Valverde
Concordia University, Canada

Malleswara Talla
Concordia University, Canada

ABSTRACT

The chapter presents data oriented and process oriented models of legacy systems. It discusses the details of systems development and evolution models mainly aiming at an ongoing reengineering of legacy systems. It proposes few strategies for reengineering of both data oriented model and process oriented models. The legacy systems often miss automatic interfaces to external systems, so the chapter presents a strategy focusing on automatic update of data of the system. Likewise, the chapter also presents a strategy for process reengineering in order to integrate external systems. Finally, a legacy system is envisioned as a comprehensive mix of both data and process oriented, while proposing a gradual ongoing reengineering of both data structures and process methods.

DOI: 10.4018/978-1-4666-0155-0.ch003

1. INTRODUCTION

A legacy system is an application that uses an outdated hardware or software platform. Due to the fact that a legacy system is very old, it is hard to find the skill set for maintaining the software or replacing hardware parts if a software bug is encountered or a hardware failure occurs. Recent trend is the shortening life period of systems to adapt to the new systems quickly; several systems are becoming outdated too soon and joining the group of legacy systems. The huge investment in a legacy system often compels reengineering and reuse of the system for evolution, maintenance and adaptation to component based software models and newer hardware platforms. Therefore, legacy systems range from traditional systems to recent component-based systems. The same modeling techniques that were used during legacy systems development also serve as building blocks during reengineering phase. This chapter presents both traditional and component-based modeling techniques and proposes few reengineering approaches that would prolong the life span of a legacy system that uses either of the modeling techniques.

The systems documentation includes models that describe the requirements collected during system analysis stage of software development process (Jacobson, I., Christerson, M., Jonsson, P. & Overgaard, G., 1993). These models simplify the system complexity and help software maintenance team to quickly understand what was done and aid in software maintenance or enhancement (Satzinger, J.W., Jackson R.B., and Burd S.D., 2002). These requirement models for a system are often remodelled during system architecture based on the methodology chosen. This chapter focuses on traditional data and process models and attempts to propose reengineering of such systems.

The traditional data oriented reengineering focuses on migrating databases of legacy systems to contemporary relational databases, enterprise data standardization, integration of disparate information systems, data quality assurance, etc. (Alice H. Muntz, and Christian T. Ramiller, 1994). The data oriented reengineering can also be conducted in a simplified manner by focusing on improving aspects of data one by one in a more practical manner as presented in this chapter. This approach eliminates the possibility of system failures, or minimizes the impact on overall system while reengineering the system as needed. On the other hand, the process oriented reengineering takes into account a sequence of work activities and remodels them in an attempt to revamp a part of the legacy system. The effort could be as simple as extending the legacy system to an external process as proposed this chapter.

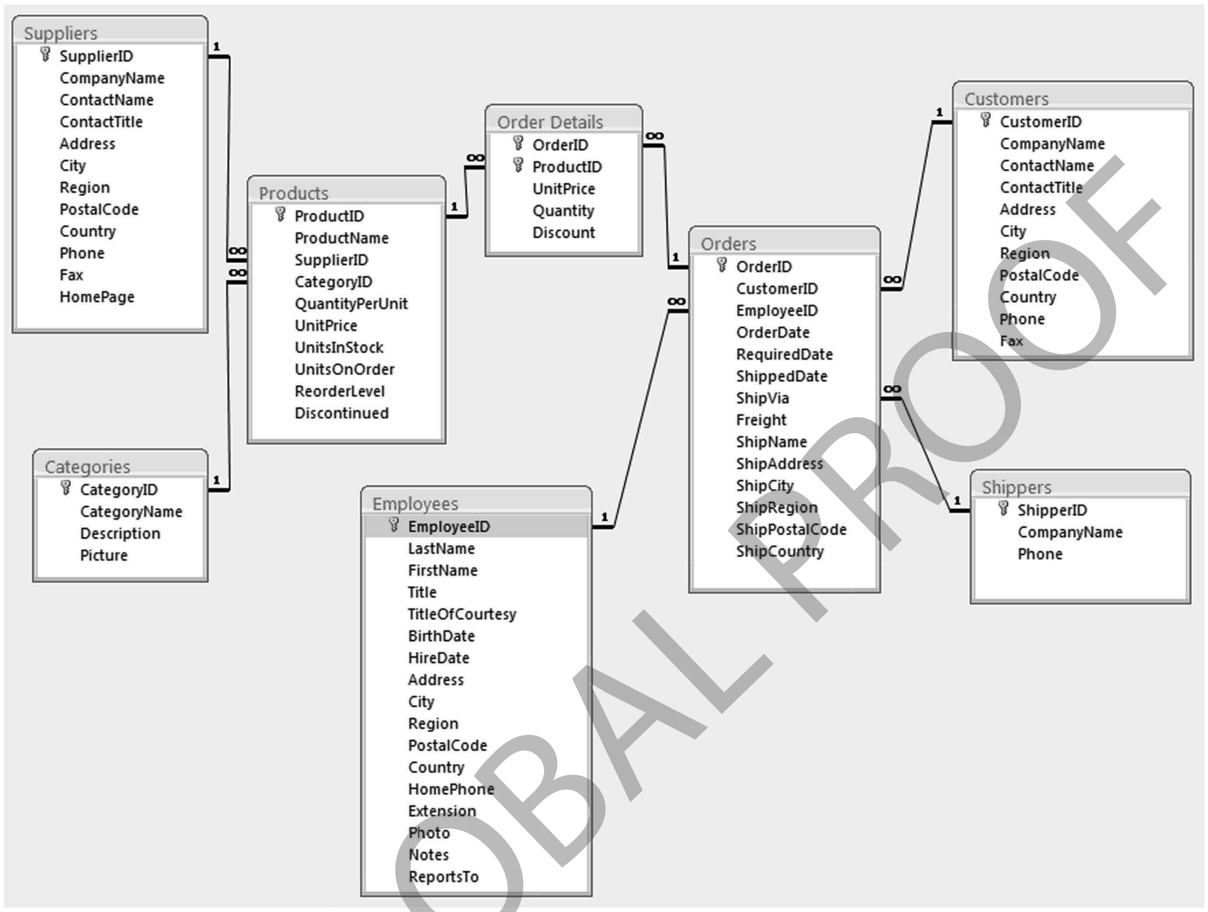
2. TRADITIONAL DATA MODELS AND REENGINEERING TECHNIQUES

The traditional methodologies include *data* models of organization's data, a technique for organizing and documenting a system's data (Whitten, J. L., Bentley D. L. and Dittman K.V., 2000). Data modeling is often called *database modeling* because a data model is often implemented as a database (Satzinger & Jackson & Burd 2002), which is depicted in an *Entity Relationship Diagram (ERD)* for presenting the data in terms of the entities and their relationships.

An *entity* is a class of persons, places, objects, events, or concepts about which data is captured and stored.

An entity instance is a single occurrence of an entity. An *attribute* is a descriptive property or characteristic of an entity. A compound attribute could include multiple attributes, in other words, a group of attributes or a data structure. A *relationship* is a natural business association that exists between one or more entities. The relationship may represent an event that links the entities or merely a logical affinity that exists between entities. *Cardinality* defines a minimum and a maximum number of occurrences of one

Figure 1. Sample ERD diagram of a typical order processing system



entity that may be related to a single occurrence of the other entity. Because all relationships are bi-directional, cardinality must be defined in both directions for every relationship. Every entity must have an identifier or *key*. A key is an attribute, or a group of attributes, which assumes a unique value for each entity instance. It is sometimes called an identifier. A *primary key* is that candidate key which will most commonly be used to uniquely identify a single entity instance.

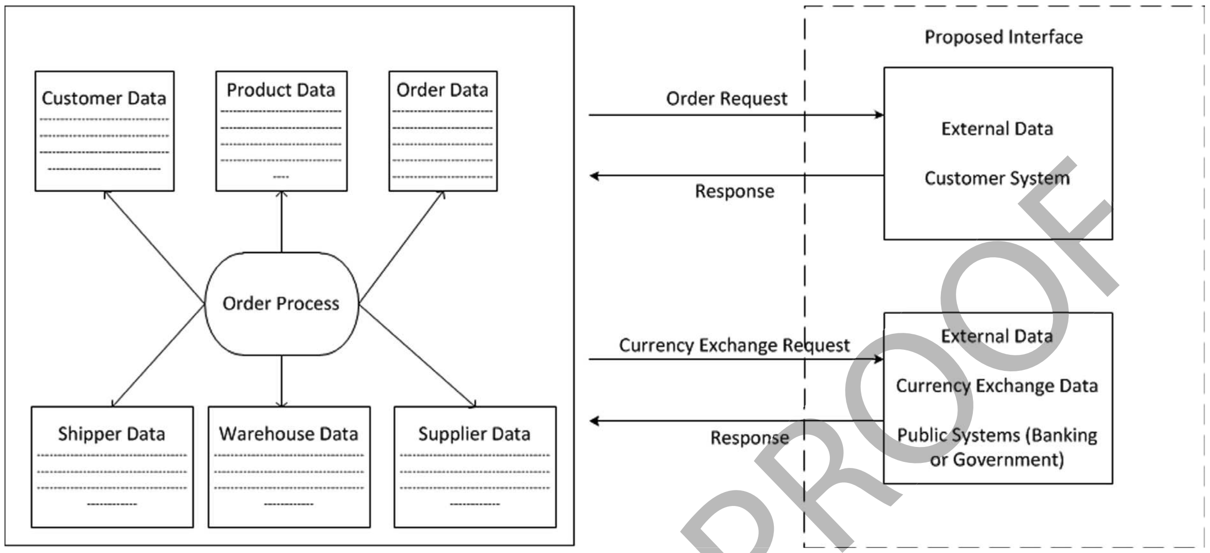
A relationship implies that instances of one entity are related to instances of another entity. To be able to identify those instances for any given entity, the primary key of one entity must be migrated into the other entity as a *foreign key*. A foreign key is a primary key of one entity that is contributed to (duplicated in) another entity for the purpose of identifying instances of a relationship. A foreign key (always in a child entity) always matches the primary key (in a parent entity).

Reengineering Techniques for Data Models

The business logic and data are almost always interwoven that makes reengineering harder to accomplish. It is important to classify and focus on the importance of data in the context of application as follows:

- Volume of data: huge number of instances of an entity,

Figure 2. Legacy systems reengineering to use external data sources

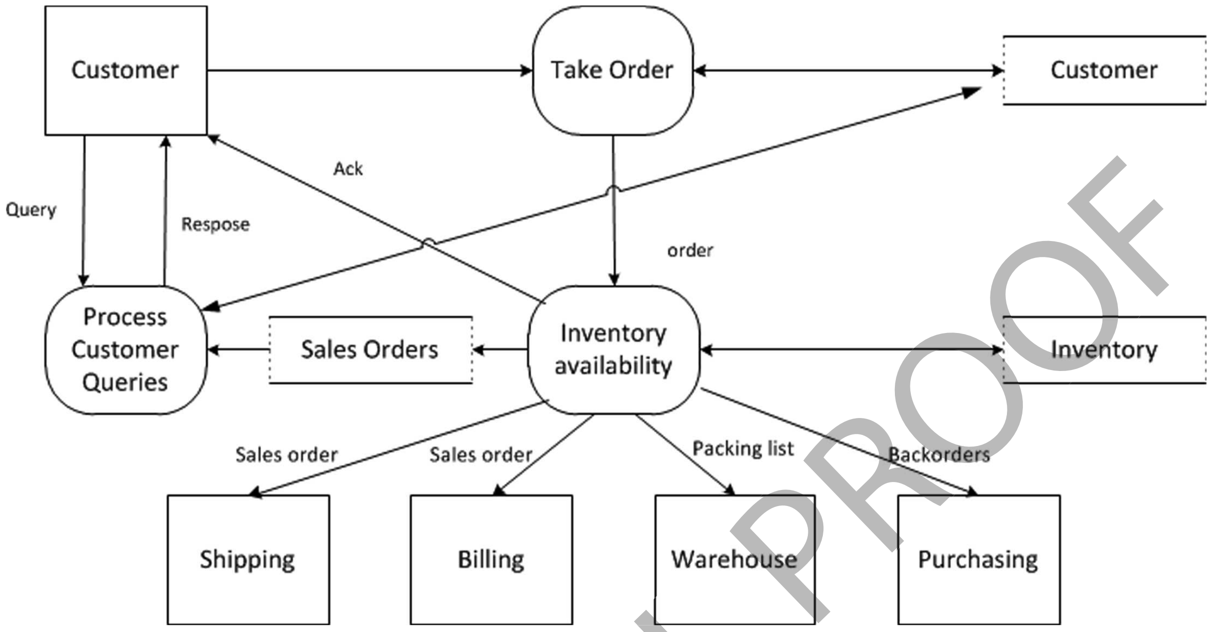


- Security: business critical data,
- Mining: data that relates to a user search *keyword*,
- Complexity: connected to several other entities that makes it harder to isolate,
- Availability: easily available within the local database,
- Redundancy: duplicate data within system for simplifying systems development, or data which is available on a different system.
- Heterogeneity: different in nature, etc.

It is optimistic to accomplish a total reengineering of an application at once, but often it proves to be a wrong approach as the probability of failure in reengineering increases. However, if the reengineering is carried out in a pragmatic manner focussing one aspect at a time, the reengineering becomes easier and successful. Here, we propose an approach for reengineering that focuses on reducing data duplication, which is classified above as redundancy of data. Traditional legacy systems focus on centralized processing whereas the contemporary systems use distributed data processing architectures for parallel, specialized, and secure processing that increases the speed of response or results. Traditional legacy systems duplicate a lot of data into its local data base; such data requires periodic updates and introduces a margin of error as the data does not reflect upon real time data. The source of such data into the database may be often in the form of spread sheets or flat files. The legacy system can be reengineered by changing the sourcing of such data to use the original data source over the network, instead of a lengthy spread sheets or data entry.

The proposed reengineering technique can use the existing network applications that provide data in real time. The upside of this approach is that it improves the systems performance and reduces quantization error, introduced due to periodic updates of data. The downside occurs when any of the source systems become unavailable, but such drawback is same with any distributed processing application. In such case, local data replica of the source system will serve the purpose in the interim period. In Figure 2, two interfaces to different systems are proposed: a customer interface via which a customer can directly update an order processing part of a legacy system; an external system interface where the currency rate fluc-

Figure 3. A sample data flow diagram



tuations can be automatically incorporated into a legacy system which eliminates any manual data entry of currency exchange rates. Likewise, system improvements can be carried out one by one till the entire legacy system is reengineered.

3. TRADITIONAL PROCESS MODELS AND REENGINEERING TECHNIQUES

The traditional approach for information systems software development describes activities as processes carried out within and across organizations. Typically, a business process represents a set of workflows in an organization, and a software application is designed to reflect upon those workflows as interactions among entities and data stores, to eventually complete a task or deliver a service. The flow of data through processes, logic, policies, and procedures to be implemented are modeled in a *data flow diagram* (DFD) that serves as a tool. It depicts the flow of data through a system and the work or processing performed by that system. In a process, work is performed in response to incoming requests or data flows or conditions (Whitten & Bentley & Dittman 2000).

Unlike traditional data models, the process models present the concurrency of interacting processes as a set of flows of data among entities and data-stores. The Figure 3 reflects upon an order processing process that interacts with entities such as customers, shipping, billing, warehouse, and purchasing. The repository of data are presented in Figure 3, as data stores such as customers, inventory, and sales orders each of which maintains all necessary information and their status. The figure 3 serves for the processes for taking orders, checking the inventory availability, and customer queries.

The system exchanges inputs and outputs within its environment.

An *external agent* or an *external entity* defines a person, an organization unit, or a system internal or external to organization that lies outside the software project (system) scope, but interacts with the system being studied (Whitten & Bentley & Dittman 2000). External agents provide net inputs into the system, and receive net outputs from the system (Satzinger & Jackson & Burd 2002).

An external agent is represented by a square shape on the data flow diagram. A data store is an inventory of data that is frequently implemented as a file or database (Whitten & Bentley & Dittman 2000). It is also a "data at rest" compared to a data flow which is "data in motion." Data stores depicted on a DFD store all instances of data entities (depicted on an ERD). A data flow represents an input of data to a process, or the output of data from a process. It may also be used to represent the creation, reading, deletion, or updating of data in a file or database. A control flow (dotted arrow) represents a condition or non-data event that triggers a process.

The following strategy for process modeling is proposed in (Whitten & Bentley & Dittman 2000):

1. Draw a context DFD to establish initial project scope. A *context diagram* defines the scope and boundary for the system and project. Because the scope of a software project frequently changes during the requirements and design stages of software development, the context diagram is also subject to changes to reflect upon the same
2. Draw a *functional decomposition diagram* to partition the system into subsystems. A decomposition diagram shows the top-down functional decomposition or structure of a system. It provides the beginnings of an outline for drawing data flow diagrams.
3. Create an *event-response* or *use-case* list for the system to define events for which the system must have a response. The purpose of this step is to determine what business events the system must respond to, and what responses are appropriate. Some of the inputs on the context diagram are associated with events.
4. Draw an event DFD (or event handler) for each event. For each event, illustrate any data stores from which records must be 'read' should be added to the event diagram. Data flows should be added and named to reflect what data is read by the process. Any data stores in which records must be 'created', 'deleted', or 'updated' should be included in the event diagram. Data flows to the data stores should be named to reflect the nature of the update.
6. Merge event DFDs into a system diagram (or, for larger systems, subsystem diagrams). The system diagram is said to be 'exploded' from the single process that was created on the context diagram. The system diagram(s) shows either: all of the events for the system on a single diagram, or all of the events for a single subsystem on a single diagram.
7. Draw detailed, primitive DFDs for the more complex event handlers. Each event process on the system diagram(s) must be exploded into either: a procedural description or a primitive data flow diagram For event processes that are not very complex – in other words, they are both an event and an elementary process, they should be described in one page (usually much less) of Structured English. Event processes with more complex event diagrams should be exploded into a more detailed, primitive data flow diagram. Theses data flow diagrams show all the elementary process, data stores and data flows for single events
8. Document the logic of each elementary process using structured English.
9. Document data flows and processes in the data dictionary

Reengineering Techniques for Process Models

The same legacy software systems are *in-use* although an organization radically changed or reorganized its processes. An organization employs different ways of adapting the changing process by means of appropriate data entry into and data extraction out of a legacy system to continue using it. This additional work of data management often becomes cumbersome, instead, it is profitable to gradually reengineer the legacy system.

A consolidated methodology for reengineering is presented in (Subramanian Muthu, Larry Whitman, et al., 1999) mainly focusing on business process, as a series of following steps:

- Prepare for reengineering,
- Map and analyze *As-Is* process,
- Design *To-Be* processes
- Implement reengineered processes one by one, and
- Improve continuously.

While planning a legacy system reengineering, it is essential to orient the targeted reengineering towards improving a production or service delivery process to accomplish few goals such as (a) work balancing, (b) work-flow redesign, (c) eliminating non-value-added activities, (d) revising administrative controls as presented in (Victor Raj, 2008).

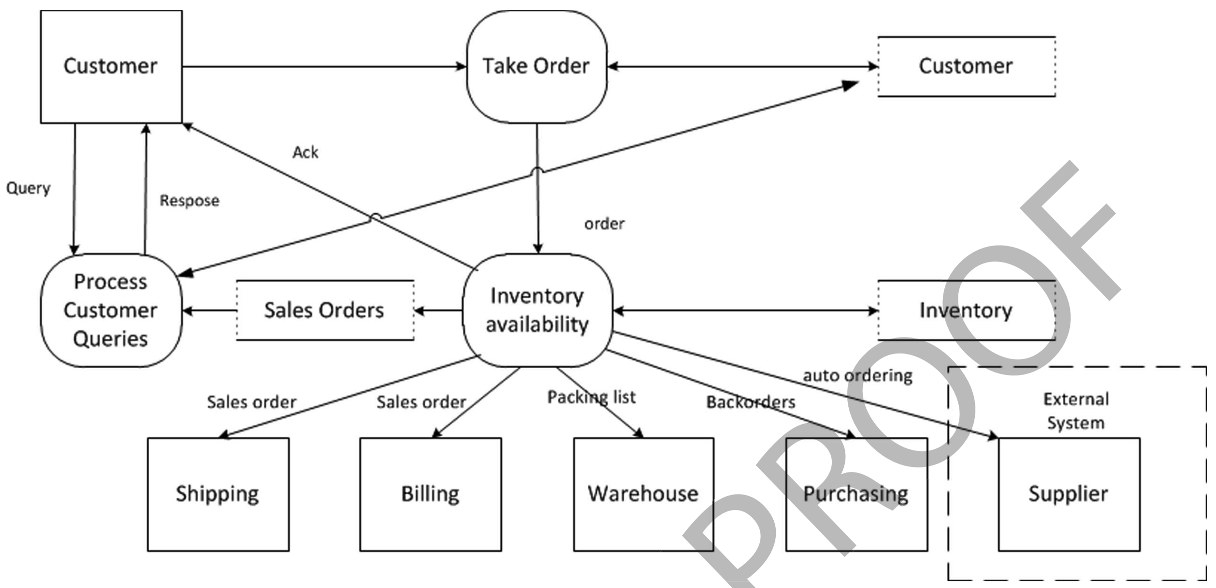
In an attempt to reengineer a legacy system, one can visualize a process as a set of sub-processes for classifying and focusing on the importance of process areas in the context of application as follows:

- Distributed processing to improve response time,
- Data redundancy across processes,
- Increased automation to integrate and minimize manual processes,
- Simplifying cross functional process interactions,
- Eliminating redundant sub-processes,
- New business process creation and integration,
- Integrating contemporary applications, e.g. Customer Relationship Management (CRM), Supply Chain Management, (SCM), Enterprise Resources Planning (ERP), etc.

It is optimistic to accomplish a total reengineering of an application at once, but often it proves to be a wrong approach as the probability of failure increases in such a reengineering strategy. However, if the reengineering is carried out in a pragmatic manner focussing one feature or aspect at a time, the reengineering becomes easier and successful. Here, we propose an approach for reengineering that focuses on integrating a new process, e.g. supplier process via web interface as presented in Figure 4. Traditional legacy systems focus on centralized processing whereas the contemporary systems use distributed data processing architectures for parallel, specialized, and secure processing that increases the speed of processing. It is possible to reengineer few parts of a legacy system by providing user interfaces to integrate new processes; for example, automatic inventory replenishment can be achieved by extending the system to integrate a supplier process as follows in Figure 4.

This approach in Figure 4 integrates the external supplier processes and eliminates order transaction processing delays. The legacy system can apply existing business rules and automatically generate new orders while respecting the lead time to suppliers. It also eliminates or simplifies the purchasing process. Although legacy systems use older software development methodologies and older hardware platforms, they still employ modular design, and it is possible to reengineer parts of a business process at a time, selectively.

Figure 4. Legacy systems reengineering to integrate external supplier process



4. TRADITIONAL DATA MODELS VS. PROCESS MODELS

Although the data models and process models are often envisioned as different approaches, they are in fact interwoven sine every process has a set of inputs and a set of outputs that are basically data, and processing involves manipulation of data. In this context, process reengineering involves changing the business rules or the way the business is conducted, whereas data reengineering involves how to organize, control and manage the data. The legacy systems often interface external systems via a set of manual processes, unlike contemporary systems that often extend itself to external processes via Intranets and Extranets using the Internet technologies. The data model is often described using ERD whereas the process model is described using DFD. The data model reengineering while impacting the data model, it also involves the process that requires some changes; however the focus would be mainly data-oriented. On the other hand, the process model reengineering sometimes may require a different input and may produce a different type of output, however the focus would be mainly the process-oriented changes such as implementing new business rules.

The Figure 5 presents the processes and data models distinctly, while identifying the internal and external processes and entities as well. While developing data-oriented reengineered model, both original and reengineered versions of data are kept and used interchangeably, in order to allow smoother and gradual reengineering.

It is sometimes required to switch from reengineered data to original data models during reengineered system development. It makes it easier if both original data model and reengineered data model coexist and the software has the ability to choose the data model. The “ifdef … endif, and ifndef ….. endif” pre-processor statements can be used during reengineered software development stage.

```
#ifdef  ORIGINAL_DATA
   …. software with original data
#endif
#ifndef  ORIGINAL_DATA
```

Figure 5. Data oriented vision versus process oriented vision of a legacy system

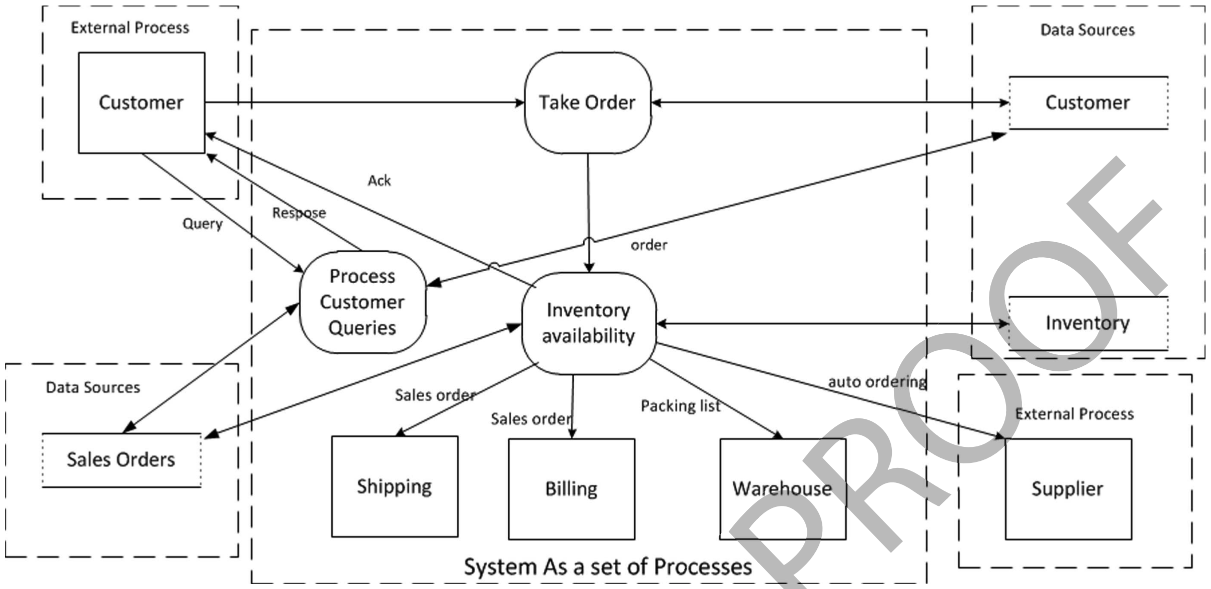
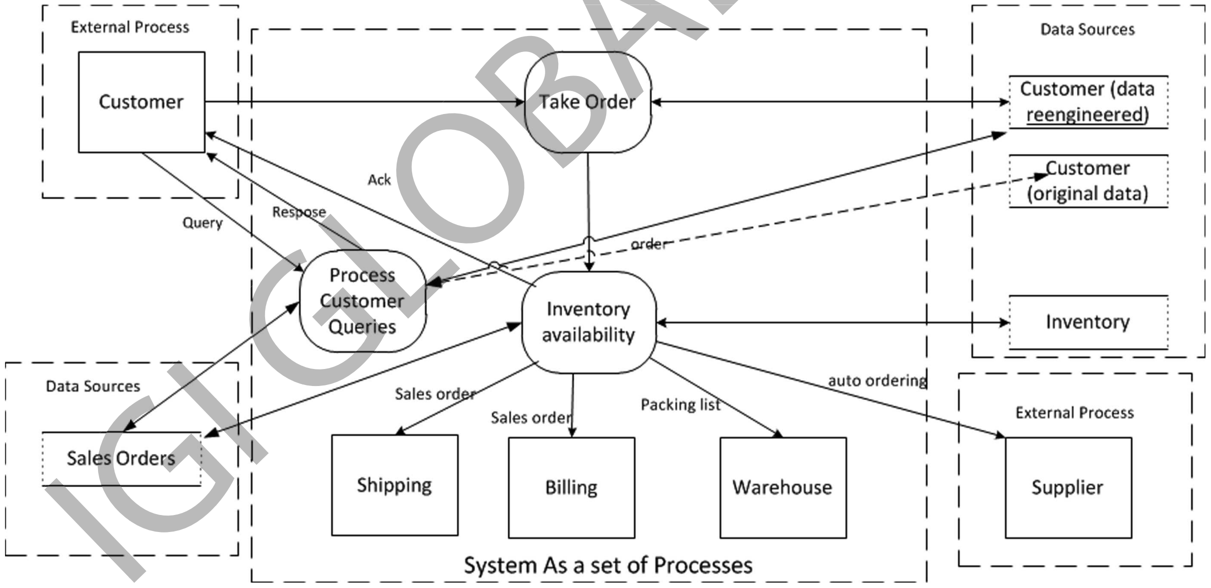


Figure 6. Co-existing original and reengineered data models of a legacy system



```
   …. software with reengineered data
#endif
```

Once the reengineered software development and testing are completed, the software development team can remove the software that accesses original data model, or leave it in the code for near future testing and comparison of results.

The process reengineering enhances the organizations ability to conduct business differently so as to incorporate the latest *ways of working*. The business rules, external systems and external processes often change and there is a need for adaptation to the changing external processes and business rules. In order to avoid manual processes to deal with data related to external systems and external processes, the process reengineering should ne conducted in a timely manner. The data model reengineering focuses on improving the data structures, incorporating the data attributes and the associated implementation. Both data and process models eventually accomplish the need for adapting the business model to the most recent requirements in a cost effective manner. Once the systems reengineering is completed, before implementing the new model, all systems operations staff needs to be trained so that they understand the involved features, their impact and expected changes. The data model reengineering is relatively easier to accomplish than the process model engineering. However, well planned reengineering is often cost effective than new products that achieve the same goals. It is also possible to model a system in a hybrid fashion where some parts of the system can be data oriented whereas the other parts of implementation can reflect upon process model. In such case, reengineering can be of both data and process models. In an evolving system, the reengineering can be a gradual and ongoing activity that keeps the systems maintenance staff well occupied and proves the systems investment worthwhile.

4 CONCLUSION

In this chapter, the traditional data and process modelling of legacy systems and reengineering approaches were presented and discussed in detail. The reengineering approach focused on systems evolution, taking advantage of *in-house* experience of systems maintenance team which is capable of not only maintaining the software system but also evolution. A fully reengineered system reflects upon the existing business model that also includes forward looking feature set. It seems that traditional methodologies are not comprehensive enough to accurately model systems, and have sought to make them ever more complex. It is argued that the only way to meet business requirements in an enterprise setting is to create several models in detail across the width and depth of the business. The chapter recommends a gradual reengineering instead of radical one shot, big budget, complex, and expensive reengineering that increases the risk of failure. The chapter also suggested an organization to consider a hybrid reengineering approach where both data model for reorganizing the data and process model for implementing the required business logic in a pragmatic way. The chapter presented the approaches with examples. The reengineering approaches proposed in this chapter prolong the legacy systems life period in a cost effective manner.

REFERENCES

Jacobson, I., Christerson, M., Jonsson, P., & Overgaard, G. (1993). *Object-oriented software engineering: A use case driven approach*. Wokingham, UK: Addison-Wesley.

Muntz, A. H., & Ramiller, C. T. (1994). A requirement-based approach to data modeling and reengineering. *Proceedings of the 20th VLDB Conference,* Santiago, Chile, 1994, (pp. 643-646).

Raj, V. (2008). Process to product orientation – A re-engineering experience from a developing country. *Proceedings of the 2008 IAJC-IJME International Conference.* ISBN 978-1-60643-379-9

Satzinger, J. W., Jackson, R. B., & Burd, S. D. (2002). *Systems analysis and design in a changing world*. Boston, MA: Course Technology.

Subramanian, M., Whitman, L., et al. (1999). Business process reengineering: A consolidated methodology. *Proceedings of the 4th Annual International Conference on Industrial Engineering Theory, Applications, and Practice*, Nov 1999.

Whitten, J. L., Bentley, D. L., & Dittman, K. V. (2000). *Systems analysis and design methods*. New York, NY: McGraw-Hill.

KEY TERMS AND DEFINITIONS

Data Oriented Model: A model that focuses on organizing data in to data structures and databases.

Data Structure: A group of data that represents business activity or theme.

Legacy System: An application that uses an outdated hardware or software platform.

Process Oriented Model: A model that reflects upon the activities of a business process.

System Reengineering: Modifying a system that has been in-use for a long time to improve it and make it up-to-date in terms of business rules and system performance needs.