2. Activities

This section will briefly cover the steps for building conceptual, logical, and physical data models, as well as maintaining and reviewing data models. Both forward engineering and reverse engineering will be discussed.

2.1 Plan for Data Modeling

A plan for data modeling contains tasks such as evaluating organizational requirements, creating standards, and determining data model storage.

The deliverables of the data modeling process include:

- **Diagram**: A data model contains one or more diagrams. The diagram is the visual that captures the requirements in a precise form. It depicts a level of detail (e.g., conceptual, logical, or physical), a scheme (relational, dimensional, object-oriented, fact-based, time-based, or NoSQL), and a notation within that scheme (e.g., information engineering, unified modeling language, object-role modeling).

- **Definitions**: Definitions for entities, attributes, and relationships are essential to maintaining the precision on a data model.

- **Issues and outstanding questions**: Frequently the data modeling process raises issues and questions that may not be addressed during the data modeling phase. In addition, often the people or groups responsible for resolving these issues or answering these questions reside outside of the group building the data model. Therefore, often a document is delivered that contains the current set of issues and outstanding questions. An example of an outstanding issue for the student model might be, "If a **Student** leaves and then returns, are they assigned a different **Student Number** or do they keep their original **Student Number**?"

- **Lineage**: For physical and sometimes logical data models, it is important to know the data lineage, that is, where the data comes from. Often lineage takes the form of a source/target mapping, where one can capture the source system attributes and how they populate the target system attributes. Lineage can also trace the data modeling components from conceptual to logical to physical within the same modeling effort. There are two reasons why lineage is important to capture during the data modeling. First, the data modeler will obtain a very strong understanding of the data requirements and therefore is in the best position to determine the source attributes. Second, determining the source attributes can be an effective tool to validate the accuracy of the model and the mapping (i.e., a reality check).

2.2 Build the Data Model

To build the models, modelers often rely heavily on previous analysis and modeling work. They may study existing data models and databases, refer to published standards, and incorporate any data requirements. After studying these inputs, they start building the model. Modeling is a very iterative process (Figure 53). Modelers draft the model, and then return to business professionals and business analysts to clarify terms and business rules. They then update the model and ask more questions (Hoberman, 2014).

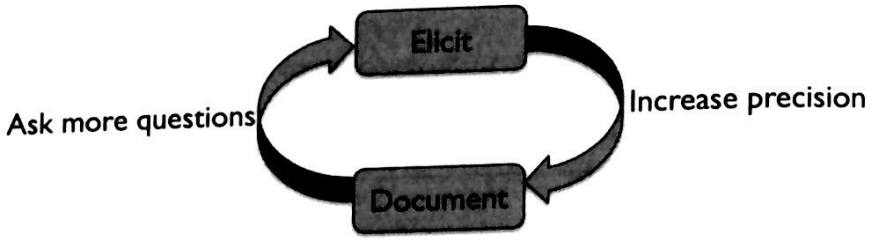

Figure 53 Modeling is Iterative

2.2.1 Forward Engineering

Forward engineering is the process of building a new application beginning with the requirements. The CDM is completed first to understand the scope of the initiative and the key terminology within that scope. Then the LDM is completed to document the business solution, followed by the PDM to document the technical solution.

2.2.1.1 Conceptual Data Modeling

Creating the CDM involves the following steps:

- **Select Scheme**: Decide whether the data model should be built following a relational, dimensional, fact-based, or NoSQL scheme. Refer to the earlier discussion on scheme and when to choose each scheme (see Section 1.3.4).
- **Select Notation**: Once the scheme is selected, choose the appropriate notation, such as information engineering or object role modeling. Choosing a notation depends on standards within an organization and the familiarity of users of a particular model with a particular notation.
- **Complete Initial CDM**: The initial CDM should capture the viewpoint of a user group. It should not complicate the process by trying to figure out how their viewpoint fits with other departments or with the organization as a whole.
    - Collect the highest-level concepts (nouns) that exist for the organization. Common concepts are **Time**, **Geography**, **Customer/Member/Client**, **Product/Service**, and **Transaction**.
    - Then collect the activities (verbs) that connect these concepts. Relationships can go both ways, or involve more than two concepts. Examples are: **Customers** have multiple **Geographic Locations** (home, work, etc.), **Geographic Locations** have many **Customers**. **Transactions** occur at a **Time**, at a **Facility**, for a **Customer**, selling a **Product**.

- **Incorporate Enterprise Terminology**: Once the data modeler has captured the users' view in the boxes and lines, the data modeler next captures the enterprise perspective by ensuring consistency with enterprise terminology and rules. For example, there would be some reconciliation work involved if the audience conceptual data model had an entity called **Client**, and the enterprise perspective called this same concept **Customer**.

- **Obtain Sign-off**: After the initial model is complete, make sure the model is reviewed for data modeling best practices as well as its ability to meet the requirements. Usually email verification that the model looks accurate will suffice.

2.2.1.2 Logical Data Modeling

A logical data model (LDM) captures the detailed data requirements within the scope of a CDM.

2.2.1.2.1 Analyze Information Requirements

To identify information requirements, one must first identify business information needs, in the context of one or more business processes. As their input, business processes require information products that are themselves the output from other business processes. The names of these information products often identify an essential business vocabulary that serves as the basis for data modeling. Regardless of whether processes or data are modeled sequentially (in either order), or concurrently, effective analysis and design should ensure a relatively balanced view of data (nouns) and processes (verbs), with equal emphasis on both process and data modeling.

Requirements analysis includes the elicitation, organization, documentation, review, refinement, approval, and change control of business requirements. Some of these requirements identify business needs for data and information. Express requirement specifications in both words and diagrams.

Logical data modeling is an important means of expressing business data requirements. For many people, as the old saying goes, 'a picture is worth a thousand words'. However, some people do not relate easily to pictures; they relate better to reports and tables created by data modeling tools.

Many organizations have formal requirements. Management may guide drafting and refining formal requirement statements, such as "The system shall..." Written data requirement specification documents may be maintained using requirements management tools. The specifications gathered through the contents of any such documentation should carefully synchronize with the requirements captured with data models to facilitate impact analysis so one can answer questions like "Which parts of my data models represent or implement Requirement X?" or "Why is this entity here?"

2.2.1.2.2 Analyze Existing Documentation

It can often be a great jump-start to use pre-existing data artifacts, including already built data models and databases. Even if the data models are out-of-date, parts can be useful to start a new model. Make sure however, that any work done based on existing artifacts is validated by the SMEs for accuracy and currency. Companies

often use packaged applications, such as Enterprise Resource Planning (ERP) systems, that have their own data models. Creation of the LDM should take into account these data models and either use them, where applicable, or map them to the new enterprise data model. In addition, there could be useful data modeling patterns, such as a standard way of modeling the Party Role concept. Numerous industry data models capture how a generic industry, such as retail or manufacturing, should be modeled. These patterns or industry data models can then be customized to work for the particular project or initiative.

2.2.1.2.3 Add Associative Entities

Associative entities are used to describe Many-to-Many (or Many-to-Many-to-Many, etc.) relationships. An associative entity takes the identifying attributes from the entities involved in the relationship, and puts them into a new entity that just describes the relationship between the entities. This allows the addition of attributes to describe that relationship, like in effective and expiration dates. Associative entities may have more than two parents. Associative entities may become nodes in graph databases. In dimensional modeling, associative entities usually become fact tables.

2.2.1.2.4 Add Attributes

Add attributes to the conceptual entities. An attribute in a logical data model should be atomic. It should contain one and only one piece of data (fact) that cannot be divided into smaller pieces. For example, a conceptual attribute called phone number divides into several logical attributes for phone type code (home, office, fax, mobile, etc.), country code, (1 for US and Canada), area code, prefix, base phone number, and extension.

2.2.1.2.5 Assign Domains

Domains, which were discussed in Section 1.3.3.4, allow for consistency in format and value sets within and across projects. **Student Tuition Amount** and **Instructor Salary Amount** can both be assigned the **Amount** domain, for example, which will be a standard currency domain.

2.2.1.2.6 Assign Keys

Attributes assigned to entities are either key or non-key attributes. A key attribute helps identify one unique entity instance from all others, either fully (by itself) or partially (in combination with other key elements). Non-key attributes describe the entity instance but do not help uniquely identify it. Identify primary and alternate keys.

2.2.1.3 Physical Data Modeling

Logical data models require modifications and adaptations in order to have the resulting design perform well within storage applications. For example, changes required to accommodate Microsoft Access would be different from changes required to accommodate Teradata. Going forward, the term *table* will be used to refer to tables, files, and

schemas; the term *column* to refer to columns, fields, and elements; and the term *row* to refer to rows, records, or instances.

2.2.1.3.1 Resolve Logical Abstractions

Logical abstraction entities (supertypes and subtypes) become separate objects in the physical database design using one of two methods.

- **Subtype absorption**: The subtype entity attributes are included as nullable columns into a table representing the supertype entity.
- **Supertype partition**: The supertype entity's attributes are included in separate tables created for each subtype.

2.2.1.3.2 Add Attribute Details

Add details to the physical model, such as the technical name of each table and column (relational databases), or file and field (non-relational databases), or schema and element (XML databases).

Define the physical domain, physical data type, and length of each column or field. Add appropriate constraints (e.g., nullability and default values) for columns or fields, especially for NOT NULL constraints.

2.2.1.3.3 Add Reference Data Objects

Small Reference Data value sets in the logical data model can be implemented in a physical model in three common ways:

- **Create a matching separate code table**: Depending on the model, these can be unmanageably numerous.
- **Create a master shared code table**: For models with a large number of code tables, this can collapse them into one table; however, this means that a change to one reference list will change the entire table. Take care to avoid code value collisions as well.
- **Embed rules or valid codes into the appropriate object's definition**: Create a constraint in the object definition code that embeds the rule or list. For code lists that are only used as reference for one other object, this can be a good solution.

2.2.1.3.4 Assign Surrogate Keys

Assign unique key values that are not visible to the business and have no meaning or relationship with the data with which they are matched. This is an optional step and depends primarily on whether the natural key is large, composite, and whose attributes are assigned values that could change over time.

If a surrogate key is assigned to be the primary key of a table, make sure there is an alternate key on the original primary key. For example, if on the LDM the primary key for **Student** was **Student First Name**, **Student Last Name**, and **Student Birth Date** (i.e., a composite primary key), on the PDM the primary key for **Student** may be

the surrogate key **Student ID**. In this case, there should be an alternate key defined on the original primary key of **Student First Name**, **Student Last Name**, and **Student Birth Date**.

2.2.1.3.5 Denormalize for Performance

In some circumstances, denormalizing or adding redundancy can improve performance so much that it outweighs the cost of the duplicate storage and synchronization processing. Dimensional structures are the main means of denormalization.

2.2.1.3.6 Index for Performance

An index is an alternate path for accessing data in the database to optimize query (data retrieval) performance. Indexing can improve query performance in many cases. The database administrator or database developer must select and define appropriate indexes for database tables. Major RDBMS products support many types of indexes. Indexes can be unique or non-unique, clustered or non-clustered, partitioned or non-partitioned, single column or multi-column, b-tree or bitmap or hashed. Without an appropriate index, the DBMS will revert to reading every row in the table (table scan) to retrieve any data. On large tables, this is very costly. Try to build indexes on large tables to support the most frequently run queries, using the most frequently referenced columns, particularly keys (primary, alternate, and foreign).

2.2.1.3.7 Partition for Performance

Great consideration must be given to the partitioning strategy of the overall data model (dimensional) especially when facts contain many optional dimensional keys (sparse). Ideally, partitioning on a date key is recommended; when this is not possible, a study is required based on profiled results and workload analysis to propose and refine the subsequent partitioning model.

2.2.1.3.8 Create Views

Views can be used to control access to certain data elements, or to embed common join conditions or filters to standardize common objects or queries. Views themselves should be requirements-driven. In many cases, they will need to be developed via a process that mirrors the development of the LDM and PDM.

2.2.2 Reverse Engineering

Reverse engineering is the process of documenting an existing database. The PDM is completed first to understand the technical design of an existing system, followed by an LDM to document the business solution that the existing system meets, followed by the CDM to document the scope and key terminology within the existing system. Most data modeling tools support reverse engineering from a variety of databases; however, creating a readable layout of the model elements still requires a modeler. There are several common layouts (orthogonal, dimensional, and

hierarchical) which can be selected to get the process started, but contextual organization (grouping entities by subject area or function) is still largely a manual process.

2.3 Review the Data Models

As do other areas of IT, models require quality control. Continuous improvement practices should be employed. Techniques such as time-to-value, support costs, and data model quality validators such as the Data Model Scorecard® (Hoberman, 2009), can all be used to evaluate the model for correctness, completeness, and consistency. Once the CDM, LDM, and PDM are complete, they become very useful tools for any roles that need to understand the model, ranging from business analysts through developers.

2.4 Maintain the Data Models

Once the data models are built, they need to be kept current. Updates to the data model need to be made when requirements change and frequently when business processes change. Within a specific project, often when one model level needs to change, a corresponding higher level of model needs to change. For example, if a new column is added to a physical data model, that column frequently needs to be added as an attribute to the corresponding logical data model. A good practice at the end of each development iteration is to reverse engineer the latest physical data model and make sure it is still consistent with its corresponding logical data model. Many data modeling tools help automate this process of comparing physical with logical.