

13: Database Development Process

A core aspect of software engineering is the subdivision of the development process into a series of phases, or steps, each of which focuses on one aspect of the development. The collection of these steps is sometimes referred to as the *software development life cycle (SDLC)*. The software product moves through this life cycle (sometimes repeatedly as it is refined or redeveloped) until it is finally retired from use. Ideally, each phase in the life cycle can be checked for correctness before moving on to the next phase.

Software Development Life Cycle – Waterfall

Let us start with an overview of the *waterfall model* such as you will find in most software engineering textbooks. This waterfall figure, seen in Figure 13.1, illustrates a general waterfall model that could apply to any computer system development. It shows the process as a strict sequence of steps where the output of one step is the input to the next and all of one step has to be completed before moving onto the next.

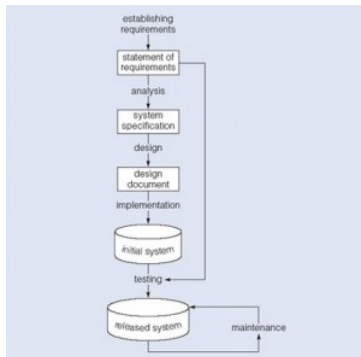


Figure 13.1. Waterfall model.

We can use the *waterfall process* as a means of identifying the tasks that are required, together with the input and output for each activity. What is important is the scope of the activities, which can be summarized as follows:

- *Establishing requirements* involves consultation with, and agreement among, stakeholders about what they want from a system, expressed as a statement of requirements.
- *Analysis* starts by considering the statement of requirements and finishes by producing a system specification. The specification is a formal representation of what a system should do, expressed in terms that are independent of how it may be realized.
- *Design* begins with a system specification, produces design documents and provides a detailed description of how a system should be constructed.
- *Implementation* is the construction of a computer system according to a given design document and taking into account the environment in which the system will be operating (e.g., specific hardware or software available for the development). Implementation may be staged, usually with an initial system that can be validated and tested before a final system is released for use.
- *Testing* compares the implemented system against the design documents and requirements specification and produces an acceptance report or, more usually, a list of errors and bugs that require a review of the analysis, design and implementation processes to correct (testing is usually the task that leads to the waterfall model iterating through the life cycle).
- *Maintenance* involves dealing with changes in the requirements or the implementation environment, bug fixing or porting of the system to new environments (e.g., migrating a system from a standalone PC to a UNIX workstation or a networked environment). Since maintenance involves the analysis of the changes required, design of a solution, implementation and testing of that solution over the lifetime of a maintained software system, the waterfall life cycle will be repeatedly revisited.

Database Life Cycle

We can use the waterfall cycle as the basis for a model of database development that incorporates three assumptions:

1. We can separate the development of a database – that is, specification and creation of a schema to define data in a database – from the user processes that make use of the database.
2. We can use the three-schema architecture as a basis for distinguishing the activities associated with a schema.
3. We can represent the constraints to enforce the semantics of the data once within a database, rather than within every user process that uses the data.

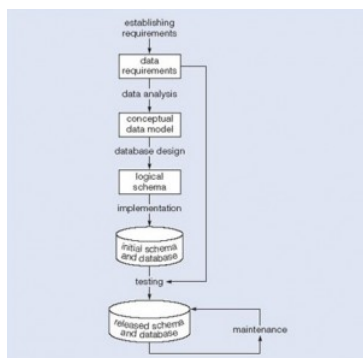


Figure 13.2. A waterfall model of the activities and their outputs for database development.

Using these assumptions and Figure 13.2, we can see that this diagram represents a model of the activities and their outputs for database development. It is applicable to any class of DBMS, not just a relational approach.

Database application development is the process of obtaining real-world requirements, analyzing requirements, designing the data and functions of the system, and then implementing the operations in the system.

Requirements Gathering

The first step is *requirements gathering*. During this step, the database designers have to interview the customers (database users) to understand the proposed system and obtain and document the data and functional requirements. The result of this step is a document that includes the detailed requirements provided by the users.

Establishing requirements involves consultation with, and agreement among, all the users as to what persistent data they want to store along with an agreement as to the meaning and interpretation of the data elements. The data administrator plays a key role in this process as they overview the business, legal and ethical issues within the organization that impact on the data requirements.

The *data requirements document* is used to confirm the understanding of requirements with users. To make sure that it is easily understood, it should not be overly formal or highly encoded. The document should give a concise summary of all users' requirements – not just a collection of individuals' requirements – as the intention is to develop a single shared database.

The requirements should not describe how the data is to be processed, but rather what the data items are, what attributes they have, what constraints apply and the relationships that hold between the data items.

Analysis

Data analysis begins with the statement of data requirements and then produces a conceptual data model. The aim of analysis is to obtain a detailed description of the data that will suit user requirements so that both high and low level properties of data and their use are dealt with. These include properties such as the possible range of values that can be permitted for attributes (e.g., in the school database example, the student course code, course title and credit points).

The conceptual data model provides a shared, formal representation of what is being communicated between clients and developers during database development – it is focused on the data in a database, irrespective of the eventual use of that data in user processes or implementation of the data in specific computer environments. Therefore, a conceptual data model is concerned with the meaning and structure of data, but not with the details affecting how they are implemented.

The conceptual data model then is a formal representation of what data a database should contain and the constraints the data must satisfy. This should be expressed in terms that are independent of how the model may be implemented. As a result, analysis focuses on the questions, “What is required?” not “How is it achieved?”

Logical Design

Database design starts with a conceptual data model and produces a specification of a logical schema; this will determine the specific type of database system (network, relational, object-oriented) that is required. The relational representation is still independent of any specific DBMS; it is another conceptual data model.

We can use a relational representation of the conceptual data model as input to the logical design process. The output of this stage is a detailed relational specification, the logical schema, of all the tables and constraints needed to satisfy the description of the data in the conceptual data model. It is during this design activity that choices are made as to which tables are most appropriate for

representing the data in a database. These choices must take into account various design criteria including, for example, flexibility for change, control of duplication and how best to represent the constraints. It is the tables defined by the logical schema that determine what data are stored and how they may be manipulated in the database.

Database designers familiar with relational databases and SQL might be tempted to go directly to implementation after they have produced a conceptual data model. However, such a direct transformation of the relational representation to SQL tables does not necessarily result in a database that has all the desirable properties: completeness, integrity, flexibility, efficiency and usability. A good conceptual data model is an essential first step towards a database with these properties, but that does not mean that the direct transformation to SQL tables automatically produces a good database. This first step will accurately represent the tables and constraints needed to satisfy the conceptual data model description, and so will satisfy the completeness and integrity requirements, but it may be inflexible or offer poor usability. The first design is then flexed to improve the quality of the database design. *Flexing* is a term that is intended to capture the simultaneous ideas of bending something for a different purpose and weakening aspects of it as it is bent.

Figure 13.3 summarizes the iterative (repeated) steps involved in database design, based on the overview given. Its main purpose is to distinguish the general issue of what tables should be used from the detailed definition of the constituent parts of each table – these tables are considered one at a time, although they are not independent of each other. Each iteration that involves a revision of the tables would lead to a new design; collectively they are usually referred to as *second-cut designs*, even if the process iterates for more than a single loop.

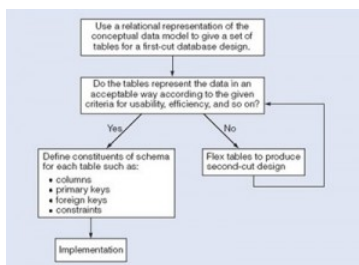


Figure 13.3. A summary of the iterative steps involved in database design.

First, for a given conceptual data model, it is not necessary that all the user requirements it represents be satisfied by a single database. There can be various reasons for the development of more than one database, such as the need for independent operation in different locations or departmental control over “their” data. However, if the collection of databases contains duplicated data and users need to access data in more than one database, then there are possible reasons that one database can satisfy multiple requirements, or issues related to data replication and distribution need to be examined.

Second, one of the assumptions about database development is that we can separate the development of a database from the development of user processes that make use of it. This is based on the expectation that, once a database has been implemented, all data required by currently identified user processes have been defined and can be accessed; but we also require flexibility to allow us to meet future requirements changes. In developing a database for some applications, it may be possible to predict the common requests that will be presented to the database and so we can optimize our design for the most common requests.

Third, at a detailed level, many aspects of database design and implementation depend on the particular DBMS being used. If the choice of DBMS is fixed or made prior to the design task, that choice can be used to determine design criteria rather than waiting until implementation. That is, it is possible to incorporate design decisions for a specific DBMS rather than produce a generic design and then tailor it to the DBMS during implementation.

It is not uncommon to find that a single design cannot simultaneously satisfy all the properties of a good database. So it is important that the designer has prioritized these properties (usually using information from the requirements specification); for example, to decide if integrity is more important than efficiency and whether usability is more important than flexibility in a given development.

At the end of our design stage, the logical schema will be specified by SQL data definition language (DDL) statements, which describe the database that needs to be implemented to meet the user requirements.

Implementation

Implementation involves the construction of a database according to the specification of a logical schema. This will include the specification of an appropriate storage schema, security enforcement, external schema and so on. Implementation is heavily

influenced by the choice of available DBMSs, database tools and operating environment. There are additional tasks beyond simply creating a database schema and implementing the constraints – data must be entered into the tables, issues relating to the users and user processes need to be addressed, and the management activities associated with wider aspects of corporate data management need to be supported. In keeping with the DBMS approach, we want as many of these concerns as possible to be addressed within the DBMS. We look at some of these concerns briefly now.

In practice, implementation of the logical schema in a given DBMS requires a very detailed knowledge of the specific features and facilities that the DBMS has to offer. In an ideal world, and in keeping with good software engineering practice, the first stage of implementation would involve matching the design requirements with the best available implementing tools and then using those tools for the implementation. In database terms, this might involve choosing vendor products with DBMS and SQL variants most suited to the database we need to implement. However, we don't live in an ideal world and more often than not, hardware choice and decisions regarding the DBMS will have been made well in advance of consideration of the database design. Consequently, implementation can involve additional flexing of the design to overcome any software or hardware limitations.

Realizing the Design

After the logical design has been created, we need our database to be created according to the definitions we have produced. For an implementation with a relational DBMS, this will probably involve the use of SQL to create tables and constraints that satisfy the logical schema description and the choice of appropriate storage schema (if the DBMS permits that level of control).

One way to achieve this is to write the appropriate SQL DDL statements into a file that can be executed by a DBMS so that there is an independent record, a text file, of the SQL statements defining the database. Another method is to work interactively using a database tool like SQL Server Management Studio or Microsoft Access. Whatever mechanism is used to implement the logical schema, the result is that a database, with tables and constraints, is defined but will contain no data for the user processes.

Populating the Database

After a database has been created, there are two ways of populating the tables – either from existing data or through the use of the user applications developed for the database.

For some tables, there may be existing data from another database or data files. For example, in establishing a database for a hospital, you would expect that there are already some records of all the staff that have to be included in the database. Data might also be brought in from an outside agency (address lists are frequently brought in from external companies) or produced during a large data entry task (converting hard-copy manual records into computer files can be done by a data entry agency). In such situations, the simplest approach to populate the database is to use the import and export facilities found in the DBMS.

Facilities to import and export data in various standard formats are usually available (these functions are also known in some systems as loading and unloading data). Importing enables a file of data to be copied directly into a table. When data are held in a file format that is not appropriate for using the import function, then it is necessary to prepare an application program that reads in the old data, transforms them as necessary and then inserts them into the database using SQL code specifically produced for that purpose. The transfer of large quantities of existing data into a database is referred to as a *bulk load*. Bulk loading of data may involve very large quantities of data being loaded, one table at a time so you may find that there are DBMS facilities to postpone constraint checking until the end of the bulk loading.

Guidelines for Developing an ER Diagram

Note: These are general guidelines that will assist in developing a strong basis for the actual database design (the logical model).

1. Document all entities discovered during the information-gathering stage.
2. Document all attributes that belong to each entity. Select candidate and primary keys. Ensure that all non-key attributes for each entity are full-functionally dependent on the primary key.
3. Develop an initial ER diagram and review it with appropriate personnel. (Remember that this is an iterative process.)
4. Create new entities (tables) for multivalued attributes and repeating groups. Incorporate these new entities (tables) in the ER diagram. Review with appropriate personnel.
5. Verify ER modeling by normalizing tables.

Key Terms

analysis: starts by considering the statement of requirements and finishes by producing a system specification

bulk load: the transfer of large quantities of existing data into a database

data requirements document: used to confirm the understanding of requirements with the user

design: begins with a system specification, produces design documents and provides a detailed description of how a system should be constructed

establishing requirements: involves consultation with, and agreement among, stakeholders as to what they want from a system; expressed as a statement of requirements

flexing: a term that captures the simultaneous ideas of bending something for a different purpose and weakening aspects of it as it is bent

implementation: the construction of a computer system according to a given design document

maintenance: involves dealing with changes in the requirements or the implementation environment, bug fixing or porting of the system to new environments

requirements gathering: a process during which the database designer interviews the database user to understand the proposed system and obtain and document the data and functional requirements

second-cut designs: the collection of iterations that each involves a revision of the tables that lead to a new design

software development life cycle (SDLC): the series of steps involved in the database development process

testing: compares the implemented system against the design documents and requirements specification and produces an acceptance report

waterfall model: shows the database development process as a strict sequence of steps where the output of one step is the input to the next

waterfall process: a means of identifying the tasks required for database development, together with the input and output for each activity (see *waterfall model*)

Exercises

1. Describe the waterfall model. List the steps.
2. What does the acronym SDLC mean, and what does an SDLC portray?
3. What needs to be modified in the waterfall model to accommodate database design?
4. Provide the iterative steps involved in database design.

Attribution

This chapter of *Database Design* (including all images, except as otherwise noted) is a derivative copy of [The Database Development Life Cycle](#) by the Open University licensed under Creative Commons Attribution-NonCommercial-ShareAlike 3.0 License.

The following material was written by Adrienne Watt:

1. Key Terms
2. Exercises

13: Database Development Process is shared under a [CC BY](#) license and was authored, remixed, and/or curated by LibreTexts.

- **1.13: Database Development Process** by Adrienne Watt is licensed [CC BY 4.0](#). Original source: <https://opentextbc.ca/dbdesign01/>.