DATABASE DEVELOPMENT LIFE CYCLE

Pranshu Gupta[1]
Ramon A. Mata-Toledo[2]
Morgan D. Monger[3]

Abstract

A software development life cycle model (SDLC) consists of a set of processes (planning, requirements, design, development, testing, installation and maintenance) defined to accomplish the task of developing a software application that is functionally correct and satisfies the user's needs. These set of processes, when arranged in different orders, characterize different types of life cycles. When developing a database, the order of these tasks is very important to efficiently and correctly transform the user's requirements into an operational database. These SDLCs are generally defined very broadly and are not specific for a particular type of application. In this paper the authors emphasize that there should be a SDLC that is specific to database applications. Database applications do not have the same characteristics as other software applications and thus a specific database development life cycle (DBDLC) is needed. A DBDLC should accommodate properties like scope restriction, progressive enhancement, incremental planning and pre-defined structure.

Keywords: Software Development, Database, DBMS, lifecycle model, traditional lifecycles

Introduction

Database management systems are generally categorized as transaction processing systems, decision support systems and/or knowledge-based systems. During their development each of these types of DBMS introduces different problems and challenges. Traditionally, SDLC models designed for developing DBMS followed the design-first-implement-later approach because of the DBMS were mainly of the transaction processing type [Weitzel and Kerschberg, 1989]. The authors believe, as we will explain later, that the design-first-implement-later approach does not work for the databases underlying data mining or knowledge-base systems or for that matter for any system where the requirements change very frequently.

Some of the traditional SDLCs models used for software development are: waterfall, prototypes, spiral and rapid application development (RAD). These life cycles models are defined broadly in terms of what each individual phase accomplish, the input and output documents it produces or requires, and the processes that are necessary in completing each phase. In general, the output deliverables from the previous phase serve as an input to the next phase. However, in these models it can be observed also that usually there is no interaction between two consecutive phases; therefore, no feedback between these

1 Computing and Information Sciences, Kansas State University, Manhattan, KS 66502
2 Department of Computer Science, James Madison University, Harrisonburg, VA 22801
3 Lead Developer/Designer, Datatel Inc., Fairfax, VA 22033

phases exists. When creating a database system the feedback between some of the life cycle phases is very critical and necessary to produce a functionally complete database management system [Mata-Toledo, Adams and Norton, 2007].

When choosing or defining a lifecycle model for database systems we need to take into account properties such as scope restriction, progressive enhancement, incremental planning and pre-defined structure [Weitzel and Kerschberg, 1989]. In addition, it is essential that the requirements and goals should be documented using a requirements traceability matrix (RTM) that will help in limiting the project to its envisioned scope. The database development life cycle should allow the incorporation of new user's requirements at a later phase due to the interactive nature that should exist between the user and the developers. This would make the enhancement of a product easier and would not increase the cost significantly. For this reason incremental planning is important for database system development. Apart from the initial planning phase, individual planning is required for the design and the requirements revision phases as they highly influence the overall implementation and the evaluation of the entire system. A life cycle model lacking any of aforementioned properties (scope restriction, progressive enhancement, incremental planning and pre-defined structure) would increase the cost, time and effort to develop a DBMS.

Traditional Lifecycle Models

This section discusses the traditional lifecycle models and shows that, at least one of the properties required for database system development (scope restriction, progressive enhancement, incremental planning and pre-defined structure), is missing from each of these lifecycles. For this reason, these life cycle models are not completely suitable for developing database systems. In the remaining of this section we briefly describe some of the most popular software models and point out their deficiencies for developing DBMSs.
Waterfall model: This is the most common of all software models [Pressman, 2007]. The phases in the waterfall cycle are: project planning, requirements definition, design, development, testing, and installation and acceptance (See Figure 1). Each of these phases receives an input and produces an output (that serves as the input for next phase) in the form of deliverables.

The waterfall model accommodates the scope restriction and the pre-defined structure properties of the lifecycle. The requirements definition phase deals with scope restriction based on the discussions with the end user. The pre-defined structure establishes a set of standard guidelines to carry out the activities required of each phase as well as the documentation that needs to be produced. Therefore, the waterfall model, by taking into account the pre-defined structure property, helps the designers, developers, and other project participants to work in a familiar environment with fewer miscommunications while allowing completion of the project in a timely manner [Shell Method™ Process Repository, 2005].

On the other hand, the waterfall model lacks the progressive enhancement and incremental planning property. In this model, the requirements are finalized early in the cycle. In consequence, it is difficult to introduce new requirements or features at later phases of the development process [Shell Method™ Process Repository, 2005]. This waterfall model, which was derived from the "hardware world", views the software development from a manufacturing perception where items are produced once and reproduced many times [Pfleeger and Atlee, 2010]. A software development process does not work this way because the software evolves as the details of the problem are understood and discussed with the end user.

The waterfall model has a documentation driven approach which, from the user's point of view, is considered one of its main weaknesses. The system specifications, which are finalized early in the lifecycle, may be written in a non-familiar style or in a formal language that may be difficult for the end user to understand [Schach, 2008]. Generally, the end user agrees to these specifications without having a clear understanding of what the final product will be like. This leads to misunderstood or missing requirements in the software requirements specifications (SRS). For this reason, in general, the user has to wait until the installation phase is complete to see the overall functionality of the system. It should be obvious then that the lack of incremental planning in this model makes it difficult to use when developing a database system particularly when the latter supports, for instance, a data mining or data warehouse operations where the "impromptu" demands imposed on the system vary frequently or cannot be easily anticipated.

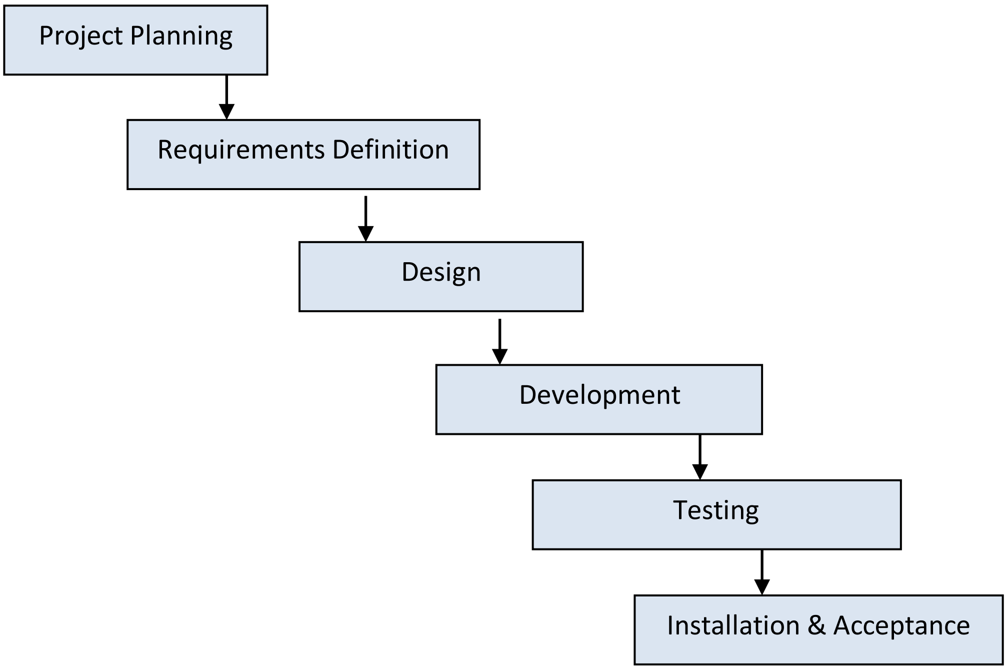


Figure.1. Waterfall model [Pressman, 2007]

Prototype model: In this life cycle model, the developers create a prototype of the application based on a limited version of the user requirements [Pfleeger and Atlee, 2010]. The prototype consists mainly of a "hallow graphics" which shows some basic and simple functionality. However, this may create a problem because the user may view the prototype as it were the final product overlooking some of the requirements specified in the SRS which may not be met fully by this "final product" [Pfleeger and Atlee, 2010].

The prototype model limits the pre-defined structure property of a lifecycle. When a prototype is designed, the developer uses minimal code to show some requirements. During this process no integration with other tools is shown. This leads to uncertainty about the final product. The prototype may have to be re-designed in order to provide a finalized product and thus it may not look the same as the one shown to the user initially.

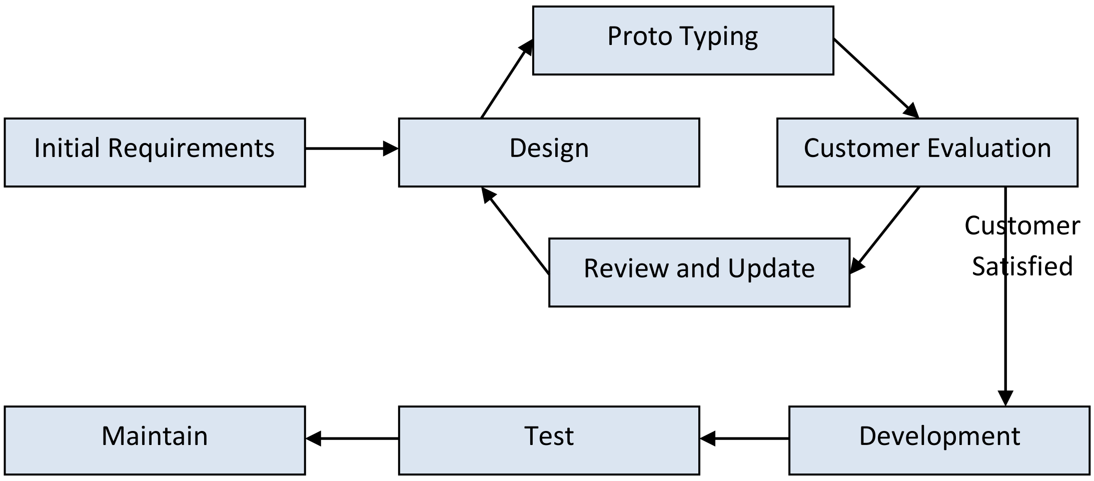


Figure.2. Prototype model [Pfleeger and Atlee, 2010]

This lifecycle model does support the progressive enhancement property. However, since the user is only shown a prototype there may be features that the user would like to incorporate but which may too costly or time consuming to incorporate later in the project. [Shell Method™ Process Repository, 2005].

In the prototype model, the requirements are finalized early in lifecycle as shown in Figure 2. The iterations are focused on design, prototyping, customer evaluation and review phases. This model lacks the incremental planning property as there is no planning after the initial planning phase.

Spiral model: This model is a combination of the prototyping and waterfall model [Pfleeger and Atlee, 2010]. Starting with the requirements and a development plan, the system prototypes and the risks involved in their developments are analyzed through an iterative process. During each iteration alternative prototypes are considered based upon the documented constraints and risks of the previous iteration [Pfleeger and Atlee, 2010]. With each subsequent prototype the risks or constraints are minimized or eliminated. After an operational prototype has been finalized (with minimal or no risks), the detailed design document is created (See Figure 3).
The spiral model supports the scope restriction property of a lifecycle. The requirements are designed in a hierarchical pattern; any additional requirements are build on the first set of requirements implemented [Shell Method™ Process Repository, 2005]. In this model, the problem to be solved is well defined from the start. In consequence, the scope of the project is also restricted.

To control risk, the spiral model combines the development activities with a risk management process [Pfleeger and Atlee, 2010]. This latter process requires expertise in the area of risk evaluation which makes the activities that need to be carried out very complex and difficult. The risk evaluation process imposes the consideration of constraints such as cost, time and effort for the entire project. The pre-defined structure property for this lifecycle model, in terms of the number of activities, is so complex that it raises the problem of controllability and efficiency during development of the system.

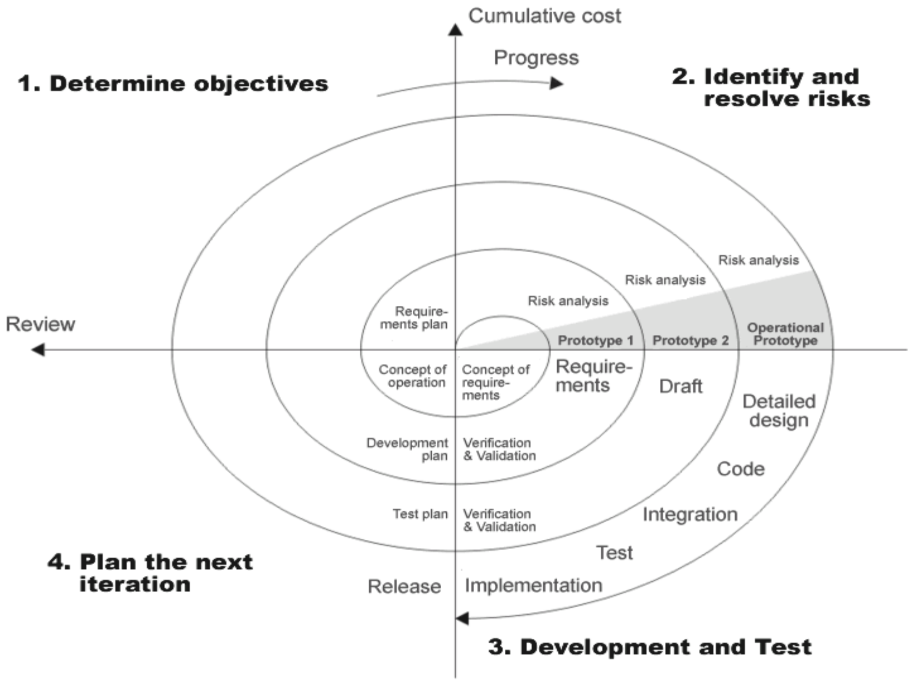


Figure.3. Spiral model [Schach, 2008]

The progressive enhancement property is not accommodated in this lifecycle model because, even though, the system is evolving with each phase, no new features can be added to the SRS due to the fact that the requirements have been finalized in an earlier phase.

Figure 3 shows the activities and phases of the spiral model and its iterative nature. However, notice that the incremental planning property is still missing from this lifecycle. The initial iterations are focused on alternatives and risks involved in the prototype selected. However, none of these iterations focus on updating the SRS by discussing it with the end user. As a result of this the requirements may not be updated; this may lead to having missing or misunderstood requirements. Due to its iterative nature this model may work well for developing requirements that are well understood from the beginning of the project. However, it is not a good model for developing database systems where new requirements may arise during the later phases of the project. The spiral model also assumes that software is developed in discrete phases; for this reason it does not satisfy the property of incremental planning [Schach, 2008].

Rapid application development model (RAD): The basic approach of this model is to let the user try the application before it is finally delivered. The users provide feedback based upon their hands-on experience with the system.

The foremost problem with this model is that it is very easy to get caught in an unending and uncontrollable cycle of enhancements. This will lead to violations of the progressive enhancement and scope restriction property.

As the name of this model implies a prototype is created and installed as soon as possible at the user's site for their review. This model lacks the predefined structure because, in general, the rapid prototype phase is completed without strictly adhering to the guideline documents and the processes already defined to complete this phase [Schach, 2008].

As Figure 4 shows the incremental planning property of a lifecycle is missing in this model too. After the prototype is completed and evaluated by the end user the requirements may or may not change. If there are no changes in the requirements, then development of the system will continue as initially envisioned. However, if significant requirement changes are necessary, then it is imperative that a timeline for the remaining of the project be established but this is not generally done [Schach, 2008].

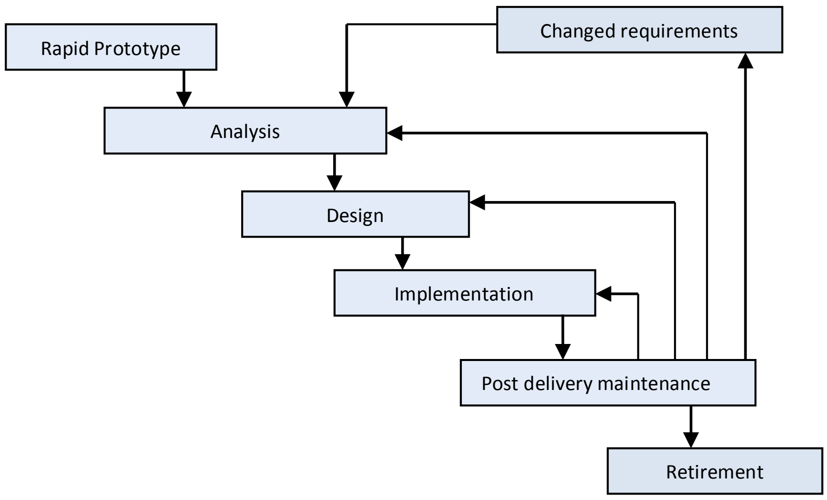


Figure.4. Rapid prototyping model [Schach, 2008]

Database Development Lifecycle

As we have shown in the previous paragraphs, each of the traditional lifecycle models is missing at least one of the four properties required for database system development. In this section the authors propose a new lifecycle model that is adapted from the traditional lifecycles and which is enhanced for database system development (See Figure 7). This new model satisfies properties such as scope restriction, progressive enhancement, incremental planning and pre-defined structure.

In most traditional life cycles, the first phase is the project planning phase. Although it is a good idea to plan the project from its inception it is also true that, unless the problem, its requirements, and its constraints are well understood it is very difficult to lay out a realistic timeline for the entire project. For this reason, we propose that this initial phase be limited to planning, not about the entire project, but about the collection of requirements definition and information about the organization. In other words, we need a plan on how we are going to proceed to identify the problem as a whole, its scope, constraints, and overall functionality. The resulting document is generally the project plan document.

The next phase of this model, the requirement definition and organizational data collection phase, should have as its ultimate goal to provide a complete set of requirements, from the user point of view, for the database system under consideration. This phase, by its very nature, requires a high degree of interaction with people at all levels of the organization, from top management to the entry level clerical workers. Essential activities of this phase are: direct examination of the organizational documents as well as their dataflow through the organization and the overall operation of the latter. Additional information can be collected by means of interviews, questionnaires, and in situ inspection of personnel activities at all organizational levels. This phase should also produce a preliminary document of the present needs and future expansion as currently perceived by all users. Figure 5 shows the deliverables for this phase, namely, the software requirement specification (SRS) and the requirements traceability matrix (RTM). These deliverables serve as the input to the next phase, the requirement analysis phase.

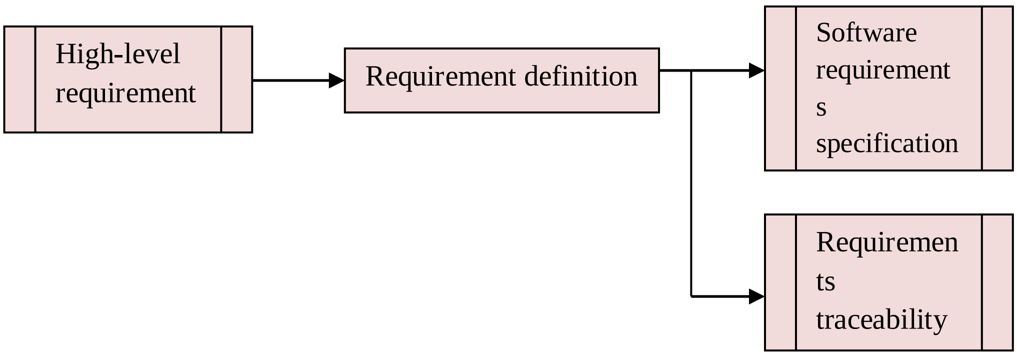
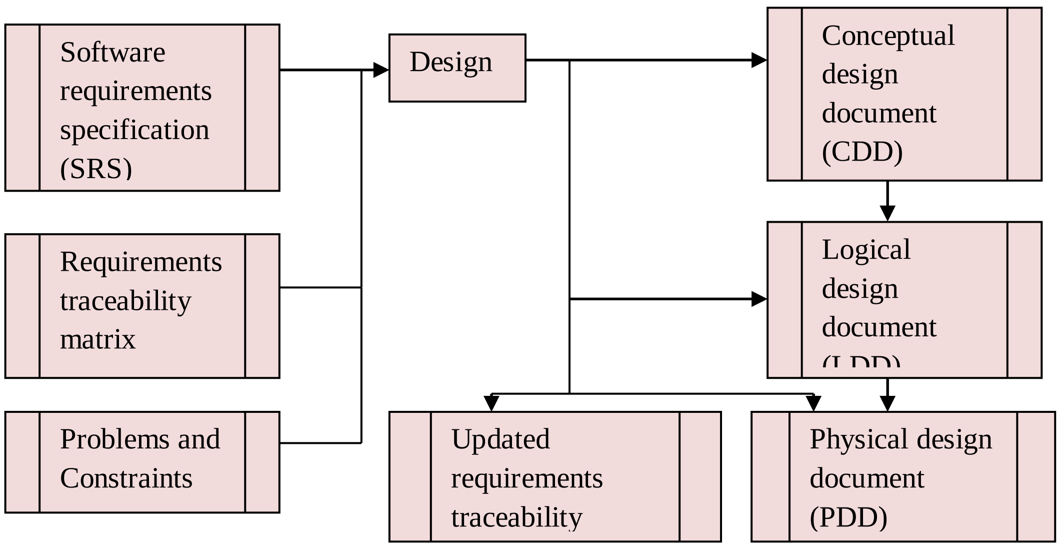

High-level requirement
Requirement definition
Software requirement s specification
Requiremen ts traceability

Figure.5. Requirements definition phase

After the previous phase has been completed it is necessary to analyze the data to consider issues of extreme importance such as feasibility, cost, scope and boundaries, performance issues, security issues, portability requirements, maintenance and the design model of the expected system. This analysis of the requirements and organizational data helps to identify potential problems and constraints that could arise during development phases.
Once the aforementioned requirements and issues have been thoroughly analyzed it is necessary to envision a timeline for future work. During this timeline planning phase it is necessary to update the project plan document initially created and thus addressing the issue of incremental planning. As was indicated early incremental planning is missing in some of the traditional lifecycle models. It is the opinion of the authors that incremental planning is an essential property which needs to be satisfied throughout the entire lifecycle as indicated in Figure 7.


Software requirements specification (SRS)
Requirements traceability matrix
Problems and Constraints
Design
Conceptual design document (CDD)
Logical design document (LDD)
Updated requirements traceability
Physical design document (PDD)
Figure.6. Deliverable for the design phase

The next two phases of this proposed model comprise the database design phase and the application design phase. The former of these two phases consists of the creating a conceptual design, selecting a database model, and producing a logical and physical design of the system as shown in Figure 7. The database design phase requires understanding of both the operational and business requirements of the organization. The purpose of the conceptual design step of the design phase is to create a high-level overview of the database using, for example, an entity-relationship model [Vanslyke, 2009]. The next step is to choose a database model suitable for the system in consideration [Rob and Coronel, 1997]. The conceptual design then needs to be converted into a logical design. To achieve this conversion the logical design uses as its input the conceptual design document (CDD) as shown in Figure 6. The logical design serves as a communication tool that describes the logical functioning and the system structure to the users [Dave, 2010]. The logical design provides a more detailed view of the database than that of the conceptual design. The last step in the database design phase is to convert the logical design into a physical design. The deliverable resulting from this last conversion is the physical design document (PDD) as shown in Figure 6. The physical design emphasizes the internal aspects of the database, e.g. the operations and processes to carry out the necessary tasks [Dave, 2010]. Figure 6 shows the deliverables for the database design phase, namely, conceptual design document (CDD), logical design document (LDD), physical design document (PDD) and the updated RTM. The physical design documents are late used in the database implementation and loading phase.
During the design phase it is important to interact with the users. As result of this process the requirements may change. It is imperative then that any change to the requirements be reflected in the RTM and any other relevant document. In doing so, we address the issue of progressive enhancement. We need to mindful that this interaction process is crucial but we also need to be aware not to fall into an unending cycle of changes that may alter the initial scope of the system.

While the database is being designed, the application design phase is carried out in parallel. The application design documents should be discussed with the user and changes should be made to the RTM if needed. The design phase is followed by the database implementation and loading phase. The database is implemented using the physical design documents developed earlier during the design phase. The database implementation and loading phase includes steps such as the follows: creating database tables, populating the tables, building constraints and querying the data.

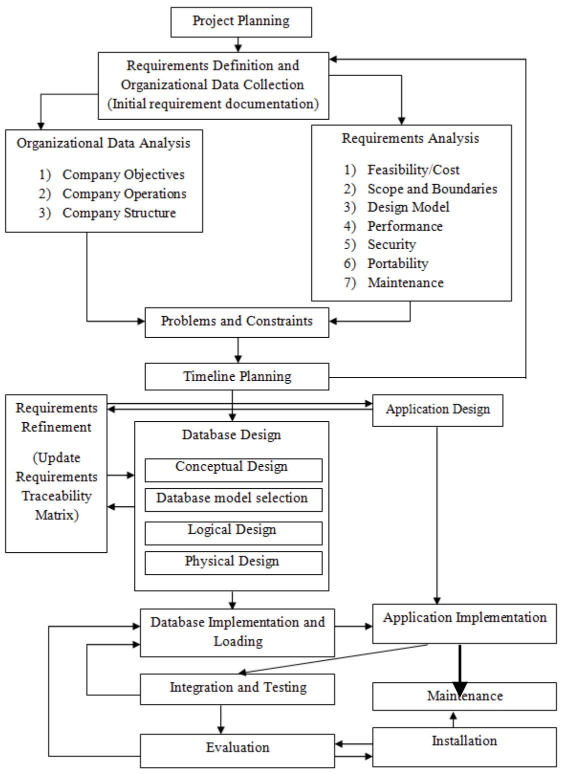


Figure.7. Database development life cycle

Next follows the application implementation phase. The application design documents from the application design phase serve as an input to this phase. The database is then integrated with the application(s) in the next phase i.e. the integration and testing phase. The integrated system is tested in this phase.

Finally we have the installation/evaluation phase. Here the use tries out the product and appraises its functionality and performance. After the system has been accepted by the user and it is operational, the maintenance phase begins. This maintenance phase will continue until the product has reached the end of its useful life. That is, until it no longer meets the new requirements of the user. At this point the whole process of developing a new system starts anew.

Conclusion

A complete and correct database system is difficult to create if the SDLC does not take into account the intrinsic characteristics of the system to be developed and the SDLC itself does not accommodate properties like scope restriction, progressive enhancement, incremental planning and pre-defined structure. As indicated before, traditional SDLCs lack at least one of the aforementioned properties making them not all suitable for the development of DBMSs, particularly, when the demands on the DBMS are unpredictable. One of main characteristics of this new proposed model is that it makes emphasis on activities that go back and forth between phases allowing either the incorporation of new requirements, if needed, or the correction of incomplete or misunderstood requirements. The idea is to allow for a system that is more flexible of the realities of developing a DBMS.

References

1. “The Software Development Life Cycle for small to medium database application.” *Shell Method™ Process Repository*. Digital Publications LLC.
2. Dave. “Database design steps / How to develop a database: Data Models & the Database Development Life Cycle” [Internet]. *Knol.,* Aug 15,2010, Available from: http://knol.google.com/k/dave/database-design-steps-how-to-develop-a/2pr18mcjtayt9/11.
3. Mata-Toledo R.A., Adams E., and Norton M., “Strategies for Determining a Design View: A Preamble to DBMS Modeling” Presented to the 16th *Annual Eastern Small College Computing Conference*. October 27-28, 2000, 29-39.
4. Pfleeger S. L., and Atlee J. M., *Software engineering,* NJ: Pearson Higher Education, 2010.
5. Pressman, R.S., *Software Engineering: A Practitioner's Approach,* NJ: Pearson Education, 2007.
6. Rob, P., and Coronel, C., *Database Systems: Design, Implementation, and Management*. Boston: Course Technology, 1997.
7. Schach S. R, *Object-oriented software engineering*, McGraw Hill, 2008.
8. Vanslyke, Craig., “Conceptual database design: Entity relationship modeling” [Internet]. *Knol.*, May 15, 2009 Available from: http://knol.google.com/k/craig-vanslyke/conceptual-database-design/1fwdlprfh17di/2.
9. Weitzel, John R., and Kerschberg, Larry. “Developing knowledge-based systems: reorganizing the system development life cycle.” *Commun. ACM* 32, 4, April 1989, 482-488.