1 Database System Development Life-cycle (DSDLC)
- Planning and Requirement Analysis
- Database Design and Data Modelling
- Implementation, Testing, and Maintenance
Development of inf. systems has to be controlled. (SW development is a quite complex & complicated process)

Software development life-cycle (SDLC) describes a process for planning, creating, testing, and deploying an information system.

The SDLC process consists of several phases. (based on best-practices is SW development)

1. planning ⇒ a goal, objectives, and criteria
2. analysis ⇒ a requirement specification
3. design ⇒ a detailed design model
4. implementation ⇒ the system implemented
5. maintenance ⇒ the system in production

(adopted from “Systems development life cycle. Wikipedia.”)
Database System Development Life-cycle (DSDLC)

- Planning and Requirement Analysis
- Database Design and Data Modelling
- Implementation, Testing, and Maintenance

(adapted from “Liisa Auer: Designing Databases. Oulu University of Applied Sciences.”)
## DSDLC Phases

1. **Database planning**  
   (how to realize the life-cycle stages most efficiently and effectively)

2. **System definition**  
   (the scope of the db. system in terms of users, user views and transactions)

3. **Req. collection and analysis**  
   (analysis of the requirements for the new database system)

4. **Database design**  
   + DBMS selection (optional)  
   (conceptual, logical, and physical design; selecting a DBMS before the phy. design)

5. **Application design**  
   (db. client application programs)

6. **Prototyping (optional)**  
   (prototypes demonstrating how the final system will look and function)

7. **Implementation**  
   (creating the physical database definitions and the application programs)

8. **Data conversion and loading**  
   (loading data from the old system to the new system, converting)

9. **Testing**  
   (validation against the requirements specified by the users)

10. **Operational maintenance**  
    (continuously monitored and maintained)
Database System Development Life-cycle (DSDLC)

(adopted from “Liisa Auer: Designing Databases. Oulu University of Applied Sciences.”)
Database Planning

**Definition (Database planning)**

Db. planning are management activities that allow the stages of a db. system development life-cycle to be realized as efficiently and effectively as possible.

- The **mission statement** defines the major aims of the db. system. (sets a goal to clarify the purpose of the database system)

- The **mission objectives** identify particular tasks that the db. system must support to fulfil the mission statement. (determines the system boundaries that will be analysed in the next phase)

- The resulting document describes how data will be collected, how the format should be specified, what necessary documentation will be needed, and how design and implementation should proceed, etc.
Database System Development Life-cycle (DSDLC)

(adopted from “Liisa Auer: Designing Databases. Oulu University of Applied Sciences.”)
System Definition

Definition (Database system definition)

System definition describes the scope and boundaries of the database application and the major user views.

- **By user views** defines what is required of a db. system from the perspective of particular users or enterprise application area. (e.g., managers or a marketing dept. require to access particular data)

- Each user view defines what is required of the db. system in terms of
  - the data to be held
  - the transactions to be performed on the data

- Multiple user views may have overlapping or distinct requirements. (the db. system has to provide all the user views, i.e. address all the requirements)
Database System Development Life-cycle (DSDLC)

Database planning

System definition

Requirements collection and analysis

DBMS selection (optional)

Prototyping (optional)

Database design

Application design

Implementation

Data conversion and loading

Testing

Operational maintenance

(adopted from “Liisa Auer: Designing Databases. Oulu University of Applied Sciences.”)
Requirements Collection and Analysis

Definition (Requirements collection and analysis, RCA)

RCA is a process of collecting and analysing information about the part of the organization that is to be supported by the database system, and using this information to identify the requirements for the new system.

- **Fact-finding techniques** are used to gather the information for RAC.
- **Requirements specification techniques** to analyse the information. (integrating the user views or ER models based on the user views)
- RAC results into the **requirements specification** for the new db. system. (a collection of specification documents, including diagrams, models, charts, etc.)
Fact-finding and Req. Specification Techniques

Examples of the fact-finding techniques:
- examining documentation
- interviewing
- observing the enterprise in operation
- research
- questionnaires

Examples of the requirements specification techniques:
- Structured Analysis and Design (SAD)
- Data Flow Diagrams (DFDs)
- Hierarchical Input Process Output (HIPO) charts
- usage of Computer-Aided Software Engineering (CASE) tools
- Unified Modelling Language (UML) diagrams
Centralized Integration of Multiple User Views in RAC

View Integration of Multiple User Views in RAC

Database System Development Life-cycle (DSDLC)

- Planning and Requirement Analysis
- Database Design and Data Modelling
- Implementation, Testing, and Maintenance

(adopted from “Liisa Auer: Designing Databases. Oulu University of Applied Sciences.”)
Database Design

Definition (Database Design)

The process of creating a design that will support the enterprise’s mission statement and mission objectives for the required db. system.

Three approaches to database design:

- **bottom-up** – by normalization of super-entities storing all the data
  1. list all possible attributes (e.g., by integration of user views)
  2. separate the attributes into several related entities by normalization
     (ensure that the entities respect, or modify them to respect, normal forms)

- **top-down** – by refining a few high-level entities and relationships
  1. describe a few high-level (major) entities and relationships
  2. apply successive top-down refinements to identify
     lower-level entities, relationships, and the associated attributes

- **inside-out/mixed** – the most common approach
  (switch between to-down, i.e., identification, and bottom-up, i.e., normalization)
Data Modelling in Database Design

Modelling of db. schema at particular level of detail with respect to:

- Structural validity
  (consistency with the way the enterprise defines and organizes information)

- Simplicity
  (ease of understanding by IS professionals and non-technical users)

- Expressibility
  (to distinguish between different data, their relationships, and constraints)

- Non-redundancy
  (the representation of any one piece of information exactly once in the model)

- Share-ability
  (without dependency on any particular application or technology)

- Extensibility
  (to evolve to support new requirements with minimal effect on existing users)

- Integrity
  (consistency with the way the enterprise uses and manages information)

- Diagrammatic representation
  (representation of a model using an easily understood diagrammatic notation)
Three stages of data modelling:

1. conceptual design
   (a data model independent of all physical considerations)

2. logical design
   (a specific data model, but independent of a particular DBMS)

3. physical design
   (a description of the implementation by a particular DBMS)

During the data modelling/database design is recommended to:

- communicate interactively with the users as much as possible
  (to verify that the modelled db. schema meets requirements, the user views)

- use diagrams to represent as much of the data models as possible
  (CASE tools to create diagrams & generate corresponding DDL statements)

- follow a structured methodology via the data modelling process
Conceptual Data Model

1. Identify entity types (will be future tables)
2. Identify relationship types between the entity types
   (including their names and the multiplicity constraints of relationship types)
3. Identify and associate attributes with entity or relationship types
   (simple/composite attributes, single/multi-valued attributes, derived attributes)
4. Determine attribute domains (data-types)
5. Determine candidate, primary, and alternate key attributes
6. Consider use of enhanced modelling concepts (optional step)
   (the concepts such as specialization/generalization, aggregation, and composition)
7. Check model for redundancy and remove redundant attrs. & rels.
8. Validate conceptual model against user transactions
   (to ensure that the conceptual model supports the required transactions)
9. Review conceptual data model with user
Logical Data Model

1. Derive relations for logical (internal) data model
   (switch from entities of the conceptual model to relations of the logical model)

2. Validate relations using normalization
   (organize the relations to minimize data redundancy; see the next lecture)

3. Validate relations against user transactions

4. Check integrity constraints
   (required data, domain constraints, multiplicity, entity and referential integrity, . . . )

5. Review logical data model with user

6. Merge logical data models into global model (optional step)
   (in the case of view integration of multiple user views, as described on page 15)

7. Check for future growth
   (make the model ready to possible significant changes in the foreseeable future)
Physical Data Model

1. Translate logical data model for target DBMS
   (switch from the relations of the logical model to tables of the physical model)
   - Design base relations (i.e., tables, not views)
   - Design representation of derived data (computed attributes)
   - Design general constraints (the constraints outside the tables)

2. Design file organizations and indexes
   - Analyze transactions (frequency, criticality, peak load times, etc.)
   - Choose file organizations (different storage formats, clusters)
   - Choose indexes (unique/non-unique, different index types)
   - Estimate disk space requirements

3. Design user views

4. Design security mechanisms

5. Consider the introduction of controlled redundancy
   (the strictly controlled redundancy in data to increase performance)

6. Monitor and tune the operational system
   (number of file-descriptors/TCP connections, firewall configuration, etc.)
Database System Development Life-cycle (DSDLC)

- Planning and Requirement Analysis
- Database Design and Data Modelling
- Implementation, Testing, and Maintenance

(adopted from “Liisa Auer: Designing Databases. Oulu University of Applied Sciences.”)
DBMS Selection

- DBMS (or its type) has to be selected before logical modelling. (DBMS features determine availability and properties of components utilized in the logical model; e.g., data-types, constraints, type of views, concurrency control, etc.)

- The main steps to selecting a DBMS are
  1. define terms of reference of study
     (set objective and scope, criteria/requirements, preliminary candidates, constraints and time-scale)
  2. short-list two or three products
     (check budget/price, level of vendor support, compatibility with SW and HW, benchmarks/performance, functionality/features)
  3. evaluate products
     (evaluate capabilities in: logical and physical data definition, query language, transaction handling, utilities, development, and other features; ranking, weighting and assigning the score of each capability for each RDBMS)
  4. select and recommend the best RDBMS and produce a report
Database System Development Life-cycle (DSDLC)

(adopted from “Liisa Auer: Designing Databases. Oulu University of Applied Sciences.”)
A prototype is just a working model/db. schema/system.
(does not normally have all the required features or functionality of the final)

To determine/check the requirements of a proposed db. system.

There are two prototyping strategies in common use today:

1. requirements prototyping
   (a one-time prototype just to determine the requirements, then discarded)

2. evolutionary prototyping
   (a simple prototype to determine the requirements which is repeatedly extended and improved towards a fully functional db. system)

Prototypes need to be discussed with users.
(the users verify that the prototypes meet the requirements on the db. system)
(adopted from “Liisa Auer: Designing Databases. Oulu University of Applied Sciences.”)
Application Design (and Implementation)

- The appl. design consists of transaction and user interface design.

- Transaction design describes trans. carried out by the appl. and it users. (db. transactions, i.e., sequences of SQL statements respecting ACID properties)

- For each transaction, the following should be described
  - type of the transaction (retrieval/update/mixed);
  - input data to be used by the transaction;
  - functional characteristics of the transaction (description);
  - an output of the transaction (results);
  - importance to the users (priority, criticality);
  - expected rate of usage (for each user, concurrently, peaks, etc.).

- User interface design deals with
  - forms for viewing and modifying data
    (to perform CRUD operations, i.e., to CReate, Update, Delete)
  - reports for printing the data (usually aggregated data)
Database System Development Life-cycle (DSDLC)

(adopted from “Liisa Auer: Designing Databases. Oulu University of Applied Sciences.”)
Implementation is achieved by Data Definition Language (DDL).

The result is an SQL script with DDL statements for db. schema. (alternatively, db. schema can be defined by a db. administration tool with GUI or in a CASE tool by transformation of a physical data model into SQL DDL statements)

The SQL script should be idempotent. (it should produce the same result even if it is executed several times)

The easiest way to an idempotent SQL script is to drop and (re)create every db. object and/or delete and (re)insert all data. (e.g., put \texttt{DROP TABLE mytable; before CREATE TABLE mytable ...;})

For large databases, it is recommended to use a CASE tool with synchronization between conceptual, logical, and physical models, and the resulting SQL script with DDL statements. (CASE tools can generate the SQL script from data models as well as analyse an existing SQL script/db. schema and create the corresponding data models)
Database System Development Life-cycle (DSDLC)

Database planning

System definition

Requirements collection and analysis

Database design

DBMS selection (optional)

Application design

Prototyping (optional)

Implementation

Data conversion and loading

Testing

Operational maintenance

(adopted from “Liisa Auer: Designing Databases. Oulu University of Applied Sciences.”)
When a new database system is replacing an old system, it is necessary to implement an utility that

1. reads data from the old system
   (difficult reading from an undocumented internal storage of a proprietary system)

2. converts data to the required format of the new system
   (split/merge entities to respect the new db. schema, convert data-types, etc.)

3. checks integrity constraints required by the new system
   (fix values breaking the constraints, merge additional ext. data to fix ref. integrity)

4. loads data into the new system

Moreover, it may be necessary to adapt existing legacy applications to be able to work with the new system (clients to access the database).
(e.g., by modification of the legacy apps., by employing an adapter between the apps. and new system, by implementing a backward-compatible API for the legacy apps.)
Database System Development Life-cycle (DSDLC)

Planning and Requirement Analysis
Database Design and Data Modelling
Implementation, Testing, and Maintenance

(adopted from “Liisa Auer: Designing Databases. Oulu University of Applied Sciences.”)
Before going live, the newly developed db. sys. should be tested.
- to uncover errors with the apps. and the db. structure
- to demonstrate that the db&apps. are working according to specif.
- to check compatibility of the system with legacy apps./data
- to measure performance and check that it is acceptable

In general, db. testing should also evaluate
- Learnability
  (how long does it take a new user to become productive with the system)
- Performance
  (how well does the system response match the user’s work practice)
- Robustness
  (how tolerant is the system of user error)
- Recoverability
  (how good is the system at recovering from user errors)
- Adapatability
  (how closely is the system tied to a single model of work)
Database System Development Life-cycle (DSDLC)

(adopted from “Liisa Auer: Designing Databases. Oulu University of Applied Sciences.”)
Operational Maintenance

- To keep functionality and performance within acceptable levels by
  - monitoring the performance of the db. system.
    (unacceptable performance $\Rightarrow$ tuning/reorganization of the db. required)
  - maintaining and upgrading the db. system.
    (repairing, backups, upgrading db. and apps. in recommended /security patches/ or required by re-design; may initiate a new iteration of the life-cycle)

- Can be performed according to well-established best-practiced.
  (providing/running the db. for client apps. is an IT service, it can be managed by IT Service Management practices, e.g., as described in IT Infrastructure Library, ITIL)

- **Data administration** is the management of the data resource.
  (the first part of db. sys. life-cycle, up to logical db. design)

- **Database administration** if the management of the RDBMS.
  (deals with the sys. physical realization, design and implementation, security and integrity controls, performance monitoring, and reorganizing the db., as necessary)
Db. system development life-cycle (DSDLC) consist of several stages.

Database design/modelling at different level of implementation details.

Data from old systems and legacy apps. have to be converted/adapted.

Finally, db. sys. in production needs to be monitored and maintained.

In the next lecture:

Entity-Relationship Modeling
(Entities, relationships and their types, object-oriented ERM and UML)
Thank you for your attention!

Marek Rychly
<mrychly@strathmore.edu>