CS 2451
Database Systems: Entity-Relationship (ER) Model Algebra

http://www.seas.gwu.edu/~bhagiweb/cs2541
Spring 2018
Instructor: Dr. Bhagi Narahari

Course Summary....

- Relational Data Model
- Formal query languages
  - Relational algebra and Relational Calculus
- SQL
  - DDL to define schema and constraints
  - Query component...basic SQL + non-RA operators (GroupBy etc.)
- Experience working with commercial DBMS and developing DB applications
  - MySQL, PHP
- Next - Database schema design: how to design a “good” schema, how to measure “good”?  
  - Normal Forms (3NF, BCNF)
- Detour (this class): Conceptual Level Database design
  - Entity-Relationship (ER) Model

How Does One Build a Database?

- Requirements Analysis: what data, apps, critical operations
  - Get from “client”
  - Typically expressed in some natural language
  - May require going back to the client for resolving questions
- Query and app development depends on client specifications

Building Database Applications: Steps

1. Start with a conceptual model
   - “On paper” using certain techniques
     - E-R Model
   - ignore low-level details – focus on logical representation
   - “step-wise refinement” of design with client input
2. Design & implement schema
   - Design and codify (in SQL) the relations/tables
   - Refine the schema – normalization
   - Do physical layout – indexes, etc.
3. Import the data
4. Write applications using DBMS and other tools
   - Many of the hard problems are taken care of by other people
     (DBMS, API writers, library authors, web server, etc.)
   - DBMS takes care of Query Optimization, Efficiency, etc.
Conceptual Model - Why?

- Convey database design and properties in simple but precise manner
  - Interpreted by any type of user
  - Does not need to know anything about CS
  - Capture the business rules of the application
- Picture is worth a thousand words

Conceptual Database Design

- **Conceptual database design** involves modeling the collected information at a high-level of abstraction without using a particular data model or DBMS.
- Since conceptual database design occurs independently from a particular DBMS or data model, we need high-level modeling languages to perform conceptual design.
- The entity-relationship (ER) model was originally proposed by Peter Chen in 1976 for conceptual design.
  - Can also perform ER modeling using Unified Modeling Language (UML) syntax.

An Example: “mini” banner

- Database containing information about
  - Students
  - Faculty
  - Courses
- Students take courses
- Faculty teach courses
- How to ‘define’ student/faculty/course?
  - What data is needed?

Entity-Relationship Modeling

- **Entity-relationship modeling** is a top-down approach to database design that models the data as entities, attributes, and relationships.
- The ER model refines entities and relationships by including properties of entities and relationships called attributes, and by defining constraints on entities, relationships, and attributes.
- The ER model conveys knowledge at a high-level (conceptual level) which is suitable for interaction with technical and non-technical users.
- Since the ER model is data model independent, it can later be converted into the desired logical model (e.g. relational model).
Entity Relationship Model
- Based on collection of real world objects or concept called entities; ex: employee, student
  - attribute represents properties of entity; s.s.num
- relationship represents interaction between entities
- overall logical structure represented by ER diagram representing entity sets, relationships, attributes
- Conceptual design:
  - What are the entities and relationships in the enterprise?
  - What information about these entities and relationships should we store in the database?
  - What are the integrity constraints or business rules that hold?
- Can map an ER diagram into a relational schema.

ER Model Definitions
- **Entity**: Real-world object distinguishable from other objects.
  - An entity is described (in DB) using a set of attributes.
- **Entity Set**: A collection of similar entities. E.g., all employees.
  - All entities in an entity set have the same set of attributes. (Until we consider ISA hierarchies, anyway!)
  - Each entity set has a key.
  - Each attribute has a domain.
- An **entity instance** is a particular example or occurrence of an entity type...eg: Employee John Doe
- Representation/Syntax:
  - Entity set represented by rectangle
  - Attribute represented by Oval (same as before)
  - Composite Attribute: when it has multiple fields (ex: address)

ER Model Basics (Contd.)
- **Relationship**: Association among two or more entities.
  - E.g., Dan takes Database Course; Attishoo works in Pharmacy department.
    - Relationship can also have attributes (that appear only for this relationship set)
- Representation/Syntax: a Diamond symbol
  - Attributes represented by Oval (same as before)
- **Relationship Set**: Collection of similar relationships.
  - An n-ary relationship set R relates n entity sets E1 ... En; each relationship in R involves entities e1 ∈ E1, ..., en ∈ En
  - Same entity set could participate in different relationship sets, or in different "roles" in same set.

Conceptual Design Process
- What are the entities being represented?
- What are the relationships?
- What info (attributes) do we store about each?
- What keys & integrity constraints do we have?
Connectivity in the E-R Diagram?

- Attributes can *only* be connected to entities or relationships
- Entities can *only* be connected via relationships
- As for the edges, let's consider kinds of relationships and integrity constraints…

Entity-Relationship Diagram for the Example

*Underlined attributes are keys*

(Warning: different ER implementations have slightly different notation here!)

Example of a composite attribute

Figure 3.4
A hierarchy of composite attributes.
Roles: Labeled Edges

Sometimes a relationship connects the same entity, and the entity has more than one role:

```
Student
    TA
      course

Student
    TA
      id
      name
```

(Nadala, Roxana)

This often indicates the need for recursive queries

Roles vs. Separate Entities

What is the difference between these two representations?

```
Student
    TA
      id
      name

Person
    TA
      id
      name
```

Weak Entity Sets

- A weak entity can be identified uniquely only by considering the primary key of another (owner) entity.
  - Owner entity set and weak entity set must participate in a one-to-many relationship set (one owner, many weak entities).
  - Weak entity set must have total participation in this identifying relationship set.
  - If Student is deleted, then we MUST delete the Parent
- Syntax: Bold face rectangles, Double lined rectangles,…

NOTATION for ER diagrams
Alternate “syntax” for ER Model: UML Notation

- If you are familiar with UML, then ER database design can be expressed using Unified Modeling Language (UML) diagrams.

UML class diagrams

- Represent classes (similar to entity types) as large rounded boxes with three sections:
  - Top section includes entity type (class) name
  - Second section includes attributes
  - Third section includes class operations (operations are not in basic ER model)
- Relationships (called associations) represented as lines connecting the classes
- Other UML terminology also differs from ER terminology
- Used in database design and object-oriented software design
- UML has many other types of diagrams for software design

ER Model Example in UML notation

Defining Constraints in ER Model

- Contraints capture properties of the relationship and entities
- Every entity set has a key attribute..similar to Rel. Model
  - No two elements can have the same value on this attribute
    Example: Student ID
- How many elements in entity set are associated with another entity in the relationship ?
  - Can a student take more than one course ?
- Does every element in the entity set appear/participate in the relationship ?
  - Must every student take a course ?
- Define constraints based on properties of the mapping/relation between entity sets
Properties of relations

- Binary relationships can be classified as one-to-one, many-to-one, one-to-many, many-to-many
- What is the type of mapping/relation

Example: the Teaches relationship

- Want to model the info that each course is taught by one faculty.
  - Type of mapping ???
  - 1-to-1
    - Note: This is a Mapping and not a function!
- A student can take more than one course
  - 1 to Many
- Every course must have an instructor
  - Each element in the Course entity set must participate/appear in the Teaches relationship
- A faculty may teach zero or more courses

Example: the Takes Course Relationship

- Student can be enrolled in many courses and each Course can have many students
  - Type of mapping:
    - Many to Many
- Want to model the condition that every student must take at least one course
  - Each student must appear in Takes relationship
- How many courses can a student take?
  - Do we want to specify a limit?
- How many students must be enrolled in a course?
  - Is there a minimum size for a class?

Mapping Cardinality, Participation Constraints, Structural constraints

- Type of mapping (cardinality)
  - 1-1, 1-many, many-many, many-1
  - Provides some information on relationship sets
- Participation constraints
  - Total vs Partial
    - Total: Every student sid must appear in Takes relationship
    - Partial: All faculty need not appear in Teaches relationship
- Structural constraints:
  - Minimum and maximum times they can appear in relationship
  - Syntax ??
The (min,max) notation for relationship constraints

Read the min,max numbers next to the entity type and looking away from the entity type

COMPANY ER Schema Diagram using (min, max) notation

Conceptual Design Using the ER Model

- **Design choices:**
  - Should a concept be modeled as an entity or an attribute?
  - Should a concept be modeled as an entity or a relationship?
  - Identifying relationships:  constraints, type, participation

- **Constraints in the ER Model:**
  - A lot of data semantics can (and should) be captured.
  - But some constraints cannot be captured in ER diagrams.

Summary of Conceptual Design

- **Conceptual design follows requirements analysis,**
  - Yields a high-level description of data to be stored
  - Visual language – the diagram is the syntax!

- **ER model popular for conceptual design**
  - Constructs are expressive, close to the way people think about their applications.
  - There are additional constructs in a “real” ER model based tools.

- **Can automate mapping of ER model to relational tables!**
A detailed example: The Company Database

- **COMPANY** database keeps track of Employees and Departments
  - Employees identified by SSN, Name, Location
  - Department specified by Department ID (did), Name, Budget
- Each department has a unique manager
  - Database must keep track of starting date
- Each employee works in a department
  - Database must keep track of starting date

Initial Conceptual Design of Entity Types for the **COMPANY** Database Schema

- Based on the requirements, we can identify four initial entity types in the **COMPANY** database:
  - **DEPARTMENT**
  - **PROJECT**
  - **EMPLOYEE**
  - **DEPENDENT**
- Their initial conceptual design is shown on the following slide
- The initial attributes shown are derived from the requirements description

Refining the initial design by introducing relationships

- The initial design is typically not complete
- Some aspects in the requirements will be represented as relationships
- ER model has three main concepts:
  - Entities (and their entity types and entity sets)
  - Attributes (simple, composite, multivalued)
  - Relationships (and their relationship types and relationship sets)
- We introduce relationship concepts next
Relationships and Relationship Types (1)

- A relationship relates two or more distinct entities with a specific meaning.
  - For example, EMPLOYEE John Smith works on the ProductX PROJECT, or EMPLOYEE Franklin Wong manages the Research DEPARTMENT.
- Relationships of the same type are grouped or typed into a relationship type.
  - For example, the WORKS_ON relationship type in which EMPLOYEES and PROJECTs participate, or the MANAGES relationship type in which EMPLOYEES and DEPARTMENTs participate.
- The degree of a relationship type is the number of participating entity types.
  - Both MANAGES and WORKS_ON are binary relationships.

Relationship type vs. relationship set

- Relationship Type:
  - Is the schema description of a relationship
  - Identifies the relationship name and the participating entity types
  - Also identifies certain relationship constraints
- Relationship Set:
  - The current set of relationship instances represented in the database
  - The current state of a relationship type

Relationship instances of the WORKS_FOR N:1 relationship between EMPLOYEE and DEPARTMENT

![Diagram of WORKS_FOR N:1 relationship between EMPLOYEE and DEPARTMENT](image)

Relationship instances of the M:N WORKS_ON relationship between EMPLOYEE and PROJECT

![Diagram of M:N WORKS_ON relationship between EMPLOYEE and PROJECT](image)
Relationship type vs. relationship set (2)

- Previous figures displayed the relationship sets
- Each instance in the set relates individual participating entities – one from each participating entity type
- In ER diagrams, we represent the relationship type as follows:
  - Diamond-shaped box is used to display a relationship type
  - Connected to the participating entity types via straight lines
  - Note that the relationship type is not shown with an arrow. The name should be typically be readable from left to right and top to bottom.

Refining the COMPANY database schema by introducing relationships

- By examining the requirements, six relationship types are identified
- All are binary relationships (degree 2)
- Listed below with their participating entity types:
  - WORKS_FOR (between EMPLOYEE, DEPARTMENT)
  - MANAGES (also between EMPLOYEE, DEPARTMENT)
  - CONTROLS (between DEPARTMENT, PROJECT)
  - WORKS_ON (between EMPLOYEE, PROJECT)
  - SUPERVISION (between EMPLOYEE (as subordinate), EMPLOYEE (as supervisor))
  - DEPENDENTS_OF (between EMPLOYEE, DEPENDENT)

Discussion on Relationship Types

- In the refined design, some attributes from the initial entity types are refined into relationships:
  - Manager of DEPARTMENT -> MANAGES
  - Works_on of EMPLOYEE -> WORKS_ON
  - Department of EMPLOYEE -> WORKS_FOR
  etc
- In general, more than one relationship type can exist between the same participating entity types
  - MANAGES and WORKS_FOR are distinct relationship types between EMPLOYEE and DEPARTMENT
  - Different meanings and different relationship instances.
**Constraints on Relationships**

- **Constraints on Relationship Types**
  - (Also known as ratio constraints)
  - Cardinality Ratio (specifies maximum participation)
    - One-to-one (1:1)
    - One-to-many (1:N) or Many-to-one (N:1)
    - Many-to-many (M:N)
  - Existence Dependency Constraint (specifies minimum participation) (also called participation constraint)
    - zero (optional participation, not existence-dependent)
    - one or more (mandatory participation, existence-dependent)

---

**Many-to-one (N:1) Relationship**

- A relationship type between the same participating entity type in **distinct roles**
- Also called a **self-referencing** relationship type.
- Example: the SUPERVISION relationship
- EMPLOYEE participates twice in two distinct roles:
  - supervisor (or boss) role
  - supervisee (or subordinate) role
- Each relationship instance relates two distinct EMPLOYEE entities:
  - One employee in supervisor role
  - One employee in supervisee role

---

**Many-to-many (M:N) Relationship**

- A relationship type between different participating entity types
- Also called a **self-referencing** relationship type.
- Example: the SUPERVISION relationship
- EMPLOYEE participates in two roles:
  - supervisor (or boss) role
  - supervisee (or subordinate) role
- Each relationship instance relates two distinct EMPLOYEE entities:
  - One employee in supervisor role
  - One employee in supervisee role

---

**Recursive Relationship Type**

- A relationship type between the same participating entity type in **distinct roles**
- Also called a **self-referencing** relationship type.
- Example: the SUPERVISION relationship
- EMPLOYEE participates twice in two distinct roles:
  - supervisor (or boss) role
  - supervisee (or subordinate) role
- Each relationship instance relates two distinct EMPLOYEE entities:
  - One employee in supervisor role
  - One employee in supervisee role
Displaying a recursive relationship

- In a recursive relationship type.
  - Both participations are same entity type in different roles.
  - For example, SUPERVISION relationships between EMPLOYEE (in role of supervisor or boss) and (another) EMPLOYEE (in role of subordinate or worker).
- In following figure, first role participation labeled with 1 and second role participation labeled with 2.
- In ER diagram, need to display role names to distinguish participations.

Recursive Relationship Type is: SUPERVISION (participation role names are shown)

Weak Entity Types

- An entity that does not have a key attribute and that is identification-dependent on another entity type.
- A weak entity must participate in an identifying relationship type with an owner or identifying entity type
- Entities are identified by the combination of:
  - A partial key of the weak entity type
  - The particular entity they are related to in the identifying relationship type
- Example:
  - A DEPENDENT entity is identified by the dependent’s first name, and the specific EMPLOYEE with whom the dependent is related
  - Name of DEPENDENT is the partial key
  - DEPENDENT is a weak entity type
  - EMPLOYEE is its identifying entity type via the identifying relationship type DEPENDENT_OF
Attributes of Relationship types

- A relationship type can have attributes:
  - For example, HoursPerWeek of WORKS_ON
  - Its value for each relationship instance describes the number of hours per week that an EMPLOYEE works on a PROJECT. A value of HoursPerWeek depends on a particular (employee, project) combination
  - Most relationship attributes are used with M:N relationships
    - In 1:N relationships, they can be transferred to the entity type on the N-side of the relationship

Example Attribute of a Relationship Type: Hours of WORKS_ON

Notation for Constraints on Relationships

- Cardinality ratio (of a binary relationship): 1:1, 1:N, N:1, or M:N
  - Shown by placing appropriate numbers on the relationship edges.
- Participation constraint (on each participating entity type):
  - total (called existence dependency) or partial.
    - Total shown by double line, partial by single line.
  - NOTE: These are easy to specify for Binary Relationship Types.

Alternative (min, max) notation for relationship structural constraints:

- Specified on each participation of an entity type E in a relationship type R
- Specifies that each entity e in E participates in at least min and at most max relationship instances in R
- Default (no constraint): min=0, max=n (signifying no limit)
- Must have min≤max, min≥0, max≥1
- Derived from the knowledge of mini-world constraints
- Examples:
  - A department has exactly one manager and an employee can manage at most one department.
    - Specify (0,1) for participation of EMPLOYEE in MANAGES
    - Specify (1,1) for participation of DEPARTMENT in MANAGES
  - An employee can work for exactly one department but a department can have any number of employees.
    - Specify (1,1) for participation of EMPLOYEE in WORKS_FOR
    - Specify (0,n) for participation of DEPARTMENT in WORKS_FOR
The (min,max) notation for relationship constraints

- **Employee**
  - **MANAGES** (0, 1) - Department
  - **WORKS FOR** (1, N) - Department

Read the min,max numbers next to the entity type and looking **away from** the entity type.

**UML class diagrams**
- Represent classes (similar to entity types) as large rounded boxes with three sections:
  - Top section includes entity type (class) name
  - Second section includes attributes
  - Third section includes class operations (operations are not in basic ER model)
- Relationships (called associations) represented as lines connecting the classes
- Other UML terminology also differs from ER terminology
- Used in database design and object-oriented software design
- UML has many other types of diagrams for software design

**ER Model Example in UML notation**

**COMPANY ER Schema Diagram using (min, max) notation**
Database Design

- The ability to design databases and associated applications is critical to the success of the modern enterprise.

- Database design requires understanding both the operational and business requirements of an organization as well as the ability to model and realize those requirements in a database.

- Developing database and information systems is performed using a development lifecycle, which consists of a series of steps.

The Importance of Database Design

- Just as proper design is critical for developing large applications, success of database projects is determined by the effectiveness of database design.

- Some statistics on software projects:
  - 80 - 90% do not meet their performance goals
  - 80% delivered late and over budget
  - 40% fail or abandoned
  - 10 - 20% meet all their criteria for success
  - Have you been on a project that failed?  A) Yes    B) No

- The primary reasons for failure are improper requirements specifications, development methodologies, and design techniques.
Specification

- One of the primary reasons for a project to fail is that it is not clear what the project should accomplish.

- **Specification** involves detailed documentation of:
  - understanding how the project fits into the organization
  - project goals and success outcomes
  - project timelines and deliverables
  - measurement criteria for determining project success
  - information on users and user requirements

- Organizational planning is an early stage of specification where enterprise needs and goals are identified, new technology is evaluated, and new development projects are considered.

Specification: Mission Statements

- The *mission statement* of a database project is a specification of the major aims and objectives of the project.
  - Each mission objective should be stated clearly and preferably have defined metrics to evaluate if it is successfully completed.

- The IT manager should construct the mission statement and objectives after consulting with management and users.
  - The mission statement may also include:
    - the "Project Champion": manager or senior tech who takes ownership
    - the estimated costs and how these costs will be paid
    - standards for database/application development that must be followed
    - documentation of privacy and security concerns
    - legal issues including copyright when dealing with outside developers
    - information on how the project will interface with other systems

- A consultant is often asked to bid on a project given its mission statement and must carefully read any legal concerns.

Specification: Mission Statement Example

- NASA’s mission statement when going to the moon:
  - "I believe that this nation should commit itself to achieving the goal, before this decade is out, of landing a man on the moon and returning him safely to Earth."
  - (John F. Kennedy May 25, 1961)

- NASA fulfilled that goal on July 20, 1969, when Apollo 11’s lunar module *Eagle* touched down in the Sea of Tranquility, with Neil Armstrong and Buzz Aldrin aboard. A dozen men would walk on the moon before the Apollo program ended. The last of those men, Gene Cernan, left the desolate lunar surface with these words: "We leave as we came and, God willing, as we shall return, with peace and hope for all mankind."

Specification Requirements Gathering

- Requirements gathering collects details on organizational processes and user issues.
  - Often the organization itself does not know this information, and it can only be determined by collecting it from user interviews.
  - Through user interviews identify:
    - Who are the users? Group them into classes.
    - What do the users do now? (existing systems/processes)
    - What are the complaints and possibilities for improvement?
  - Determine the data used by the organization, identify data relationships, and determine how data is used and generated.
    - Identify unique fields (keys)
    - Determine data dependencies, relationships, and constraints (high-level)
    - Estimate the data sizes and their growth rates
Specifications

Requirements Gathering (2)

- The best way to determine user requirements is to:
  
  **ASK THEM, THEN LISTEN**

The biggest mistake that you can make is to walk into a company and say that:

"I am going to build you a C# application using Microsoft SQL Server. Give me the data that you have, and I will go away and build you a great system!"

Requirements gathering

- **RULE #1**: The user (and managers) are your ultimate consumers; listen and respect their opinions and needs.
- **RULE #1a**: Users (and managers) do not always know what they want or may want the wrong things. Sometimes you must convince them of other alternatives or provide solutions that are good compromises.

Database Design

- The requirements gathering and specification provides you with a high-level understanding of the organization, its data, and the processes that you must model in the database.

- Database design involves constructing a suitable model of this information.

- Since the design process is complicated, especially for large databases, database design is divided into three phases:
  - Conceptual database design
  - Logical database design
  - Physical database design

Database Design

Conceptual Database Design

- **Conceptual database design** involves modeling the collected information at a high-level of abstraction without using a particular data model or DBMS.

- High-level modeling languages such as the Entity-Relationship (ER) model and UML are used.

- Conceptual database design is **top-down design** as you start by specifying entities (real-world objects) then build up the model by defining new entities, attributes, and relationships.
  
  We will also see a bottom-up design technique called **normalization** where you define the attributes first and then use dependency information to group them into relations.
Database Design

Logical Database Design
- **Logical database design** is the process of constructing a model of the information in the domain using a particular data model, but independent of the DBMS.
- The conceptual model is transformed into a logical model such as the relational model.
  - It may also be transformed to other logical models such as the object-oriented model, graphs, JSON, or XML.
- Since logical design selects a data model, it is now possible to model the information using the features of that model.
  - For example, modeling using the relational model allows you to specify keys, domains, and foreign key constraints.

Physical Database Design

Selecting a DBMS
- The process of selecting a DBMS is not always simple if the organization does not already have one or there are multiple DBMSs that can be selected.
- There are several factors to consider when selecting a DBMS:
  - **Features**: Does the DBMS have what you need? Security, constraints, indexes, triggers, SQL-compliance, JDBC/ODBC
  - **Compatibility**: Does the DBMS run on your hardware/software? Can the DBMS interface with existing systems?
  - **Performance**: Does the DBMS handle the data/query load? Scalability: Will the DBMS scale if the data/query load grows?
  - **Price**: Does the DBMS fit the project budget? Free is not necessarily cheap. How will the DBMS be supported?

Database Design

Physical Database Design
- **Physical database design** is the process of constructing a physical model of information on the secondary storage in a given data model for a particular DBMS.
- Physical database design involves the selection of a database system and determining how to represent the logical model on that DBMS. For the relational model this includes:
  - creating a set of relational tables using the logical model
  - increasing performance using indexes
  - implementing constraints and security restrictions
  - data partitioning and distribution
- Physical database design is *how*, and logical database design is the *what*.

Application Design
- The application can be designed *partially* in parallel with the database design.
- During the requirements gathering phase, you can collect information on how the data will be manipulated, the general layout of the screens, and requirements that the application will need from the DBMS.
- Two main areas of application design:
  - **Transactions** - What queries/updates will the application perform?
  - **User interface** - How will the data be displayed on the screen and how can the user initiate transactions?
Application Design Recommendations

- 1) Design the logical database schema first before seriously starting the application design.
   Changes to the logical schema have big impacts on your application and will reduce your development efficiency.

- 2) Prototype your application, but not your database design.
   It will reduce your development time if you spend the time getting your database design right before building your application.
   Flush out as many details as possible before finalizing your DB design and implementing on the DBMS. Then implement the logical design.
   Prototype your application in stages to get user feedback on the forms and reports as these will frequently change.

- 3) Isolate database access code into a few classes/methods.

Implementation

- The implementation stage builds the database and application designs.

- The database designs are implemented in the relational model using SQL DDL.

- The application is designed in a programming language such as HTML/JavaScript, PHP, C/C++, or Java.

Testing and Maintenance

- Testing and maintenance are the two final stages:
  - Testing is the process of executing the application programs with the intent of finding errors.
  - Operational maintenance is the process of monitoring and maintaining the system following installation.

- There is a significant cost associated with testing and maintaining existing systems. Good design should allow for a system to accommodate changes with minimal reprogramming.

- Maintenance also includes monitoring the DBMS for performance, security violations, and upgrades.

Database Design: Conclusion

- Database design is critical to the success of database development projects.

- Database design is divided into three phases:
  - Conceptual database design
  - Logical database design
  - Physical database design

- Your term project experience will take you through the design phases

- Effective requirements gathering is an important skill to master.
  - Term project does not fully capture this; requirements are given to you

- Projects should have a well-defined mission statement and project champion to increase their probability of success.