MIDTERM

Luca Novello | SYD466V1A | Professor Navid Mohaghegh

Question 1 - Part A (10%):

Explain DFDs. Ensure to cover:

- *Describe the purpose of DFDs and their importance in understanding the movement of data within and outside of a system.*
- *Explain why systems analysts use data-flow diagrams, focusing on their role in modelling processes, understanding information handling, and designing new information services.*

A Data Flow Diagram (DFD) is a visual tool that graphically illustrates movement of data between external entities and the processes and data stores within a system. It shows how data moves between external entities, processes, and data stores. Its purpose is to model how data is inputted, processed, stored, and output by a system. They help identify the movement of data both inside and outside the system, clarify system boundaries, and illustrate the transformation of data through various processes.

Systems analysts use DFDs to model processes, understand how data is handled, and design information services. DFDs are used to document the current system, plan the future system, and communicate system requirements clearly among stakeholders.

Question 1 - Part B (10%):

Explain Decomposition and Balancing in DFDs. Ensure to cover balanced DFDs and identify common indicators of imbalance DFDs.

Decomposition is the process of breaking down a high-level DFD into more detailed levels to show the internal processes and data flows. When decomposing a DFD, you must conserve inputs to and outputs from a process at the next level of decomposition

Indicators of Imbalanced DFDs:

- Missing or additional data flows in the lower-level DFD that were not present at the higher level.
- Inconsistent data store usage across levels.
- External entities appearing or disappearing between levels without explanation.
- Changes in the direction or type of data flow not justified by the process logic.

Balanced DFDs maintain integrity between levels, while imbalanced ones signal errors in analysis or design.

Question 1 - Part C (10%):

Explain the drawing rules and guidelines of DFDs. Make sure to summarize the rules for creating effective DFDs, emphasizing process inputs and outputs, data flow directions, and restrictions on data movement.

The drawing rules and guidelines of DFDs are as follows:

- **Basic Rules:**
  - Inputs to a process are always different than outputs
  - Objects always have a unique name
    - In order to keep the diagram uncluttered, you can repeat data stores and data flows on a diagram
- **Process Rules:**
  - No process can have only outputs (a miracle)
  - No process can have only inputs (black hole)
  - A process has a verb phrase label.
- **Data Store Rules:**
  - Data cannot be moved from one store to another
  - Data cannot move from an outside source to a data store
  - Data cannot move directly from a data store to a data sink
  - Data store has a noun phrase label
- **Source/Sink Rules:**
  - Data cannot move directly from a source to a sink
  - A source/sink has a noun phrase label
- **Data Flow Rules:**
  - A data flow has only one direction of flow between symbols
  - A fork means that exactly the same data go from a common location to two or more processes, data stores, or sources/sinks
  - A join means that exactly the same data come from any two or more different processes, data stores or sources/sinks to a common location
  - A data flow cannot go directly back to the same process it leaves
  - A data flow to a data store means update
  - A data flow from a data store means retrieve or use
  - A data flow has a noun phrase label

Question 2 – Part A (15%):

Analyze how an online cell phone app store works. List relevant data flows, data stores, processes, and external entities (sources/sinks) focusing on the Sale of Apps, Payments, and Sales Reporting. Clarify your assumptions and define the scope of your analysis.

Scope and Assumptions:

This DFD analysis focuses on how an online cell phone app store works related to the sale of apps, payment processing, and sales reporting. It assumes that users access the app store via mobile devices, purchase apps using online payment methods, and that the system provides reporting on sales data. Developer registration, app submission, and update processes are considered out of scope.

External Entities (Sources/Sinks):

- Customer – purchases apps.
- Payment Gateway – processes payments.
- Developer – receives reports and sales data.
- Bank – transfers payment to developer.
- Admin – views and monitors sales reports.

Processes:

1. Browse and Select App – customer searches and selects apps to purchase.
2. Process Payment – captures payment details.
3. Deliver App – grants access or download of purchased app to customer.
4. Generate Sales Report – compiles sales data for administrative use.
5. Transfer Payment to Developer – manages revenue sharing and sends payout to developer.

Data Flows:

- App Selection – from Customer to Browse and Select App.
- Payment Info – from Customer to Process Payment.
- Payment Confirmation – from Payment Gateway to Process Payment.
- App Access – from Deliver App to Customer.
- Sales Data – from Process Payment to Generate Sales Report.
- Sales Report – from Generate Sales Report to Admin.
- Revenue Report – from Generate Sales Report to Developer.
- Payout Request – from Generate Sales Report to Bank.
- Payment to Developer – from Bank to Developer.

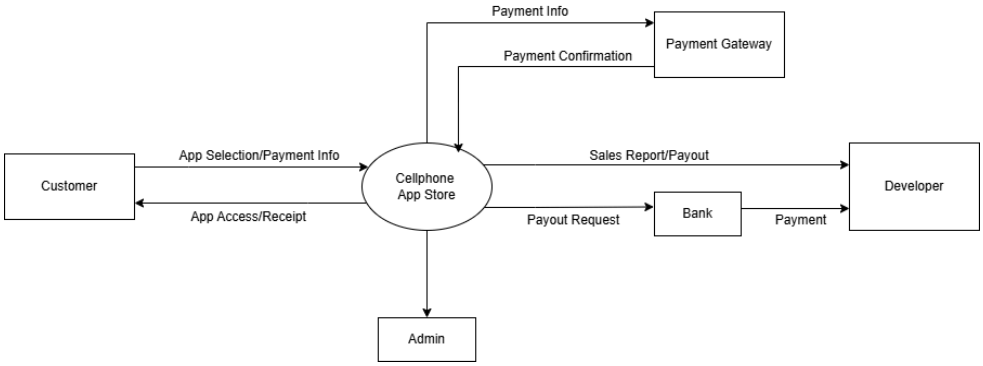
Data Stores:

- App Catalog.
- Customer Account Data.
- Sales Records.
- Developer Info.

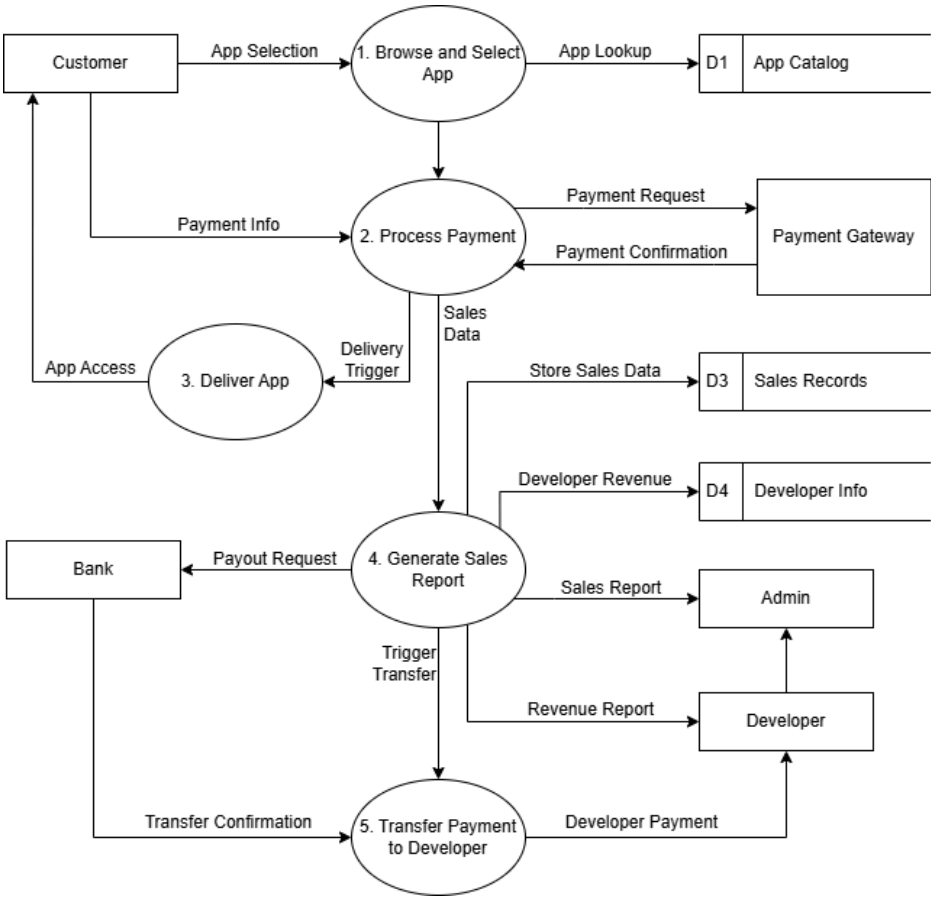
Question 2 – Part B (15%):

Draw a context diagram and a Level-0 DFD to represent the cellphone app store system you described in part A. Justify your choices for designating certain elements as processes versus sources/sinks.

Context Diagram:



Level-0 DFD:

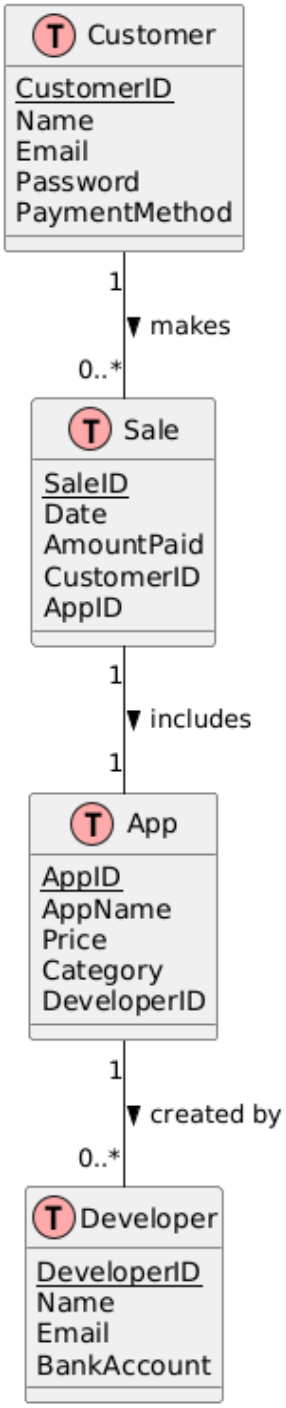


- The processes were chosen based on internal operations performed by the system: app selection, payment handling, app delivery, reporting, and fund transfer.
- External entities were designated as sources/sinks because they interact with the system but are not controlled by it.
- Data stores represent repositories that support persistence and internal operations, such as the Sales Records or App Catalog.

Question 3 – Part A (25%):

Provide an ERDs that showcases the sales and customer data in your online cell phone app store that you worked on Question 2.

ERD - Online Cell Phone App Store



Question 3 – Part B (15%):

Write the corresponding Java or C# classes for your ERD diagram in part A, ensuring accurate representation of the generalization and the attributes and operations of each class.

Customer Class:

```
public class Customer {
    private String customerId;
    private String name;
    private String email;
    private String password;
    private String paymentMethod;

    public Customer(String customerId, String name, String email, String password, String
paymentMethod) {
        this.customerId = customerId;
        this.name = name;
        this.email = email;
        this.password = password;
        this.paymentMethod = paymentMethod;
    }

    public void displayInfo() {
        System.out.println("Customer: " + name + ", Email: " + email);
    }
}
```

App Class:

```
public class App {
    private String appId;
    private String appName;
    private double price;
    private String category;
    private Developer developer;

    public App(String appId, String appName, double price, String category, Developer
developer) {
        this.appId = appId;
        this.appName = appName;
        this.price = price;
        this.category = category;
        this.developer = developer;
    }

    public void displayInfo() {
        System.out.println("App: " + appName + " ($" + price + ")");
    }
}
```

Sale Class:

```
import java.util.Date;

public class Sale {
    private String saleId;
    private Date date;
    private double amountPaid;
    private Customer customer;
    private App app;

    public Sale(String saleId, Date date, double amountPaid, Customer customer, App app) {
        this.saleId = saleId;
        this.date = date;
        this.amountPaid = amountPaid;
        this.customer = customer;
        this.app = app;
    }

    public void displayReceipt() {
        System.out.println("Sale: " + saleId + " - $" + amountPaid + " by " + customer);
    }
}
```

Developer Class:

```
public class Developer {
    private String developerId;
    private String name;
    private String email;
    private String bankAccount;

    public Developer(String developerId, String name, String email, String bankAccount) {
        this.developerId = developerId;
        this.name = name;
        this.email = email;
        this.bankAccount = bankAccount;
    }

    public void displayInfo() {
        System.out.println("Developer: " + name + ", Email: " + email);
    }
}
```