**🧠 System Example: “Online Exam Grading System”**

A computer-based system where students take online exams, answers are evaluated, and scores are stored in a database.

This system has a **clear transform flow**:

* Input: Student answers
* Transformation: Evaluate → Calculate score → Assign grade
* Output: Graded report

**⚙️ 1. System Overview**

The **Online Exam Grading System** accepts student submissions, evaluates each answer against a solution key, computes total marks, assigns a grade, and stores the result for reporting.

**📊 2. Data Flow Diagram (DFD – Level 1)**

@startuml

title DFD - Online Exam Grading System

actor "Student" as Student

actor "Admin" as Admin

rectangle "Online Exam Grading System" {

process "1. Collect Student Answers" as P1

process "2. Evaluate Answers" as P2

process "3. Compute Total Score" as P3

process "4. Assign Grade" as P4

process "5. Store Results" as P5

}

database "Question Bank" as QB

database "Result Database" as RD

Student --> P1 : Submit Answers

P1 --> P2 : Student Responses

P2 --> P3 : Evaluated Answers

P3 --> P4 : Total Marks

P4 --> P5 : Grade + Score

P2 --> QB : Fetch Answer Key

P5 --> RD : Save Record

Admin --> RD : Retrieve Results

P5 --> Student : Display Report

@enduml

✅ **Flow Boundaries:**

* **Input boundary:** From *Student* to *Collect Student Answers*
* **Output boundary:** From *Store Results* to *Student* / *Admin*
* **Transform center:** Steps 2–4 (Evaluate → Compute → Grade)

**🧩 3. Processing Narrative (Step-by-Step Flow)**

1. **Collect Student Answers:**  
   The system receives the student’s completed exam through the web interface and formats it for evaluation.
2. **Evaluate Answers:**  
   Each answer is compared to the correct answer stored in the *Question Bank*. Partial credit may be given for certain question types.
3. **Compute Total Score:**  
   The system sums up all awarded marks and calculates the total score for that student.
4. **Assign Grade:**  
   Based on predefined grading rules (e.g., A ≥ 85, B ≥ 70, etc.), the system determines the student’s grade.
5. **Store Results:**  
   The final report (student ID, marks, grade, time) is stored in the *Result Database*. The student and admin can view or download it.

**🏗️ 4. Mapping DFD to Software Architecture (Transform Mapping)**

According to **Pressman Section 9.6.1**, we:

1. **Isolate the Transform Center:**  
   → Evaluate Answers → Compute Total Score → Assign Grade  
   (This is the core transformation.)
2. **Identify Input Flow and Output Flow:**
   * **Input flow:** Collect Student Answers
   * **Output flow:** Store Results and Display Report
3. **Perform First-Level Factoring:**
   * Top-level module: OnlineExamSystem
   * Submodules follow the logical transform flow

**🧱 5. Software Architecture (Transform Flow Structure)**

@startuml

title Transform Flow Architecture - Online Exam Grading System

rectangle "OnlineExamSystem" {

rectangle "Input Flow" {

component "CollectStudentAnswers()"

}

rectangle "Transform Center" {

component "EvaluateAnswers()"

component "ComputeTotalScore()"

component "AssignGrade()"

}

rectangle "Output Flow" {

component "StoreResults()"

component "DisplayReport()"

}

}

CollectStudentAnswers() --> EvaluateAnswers()

EvaluateAnswers() --> ComputeTotalScore()

ComputeTotalScore() --> AssignGrade()

AssignGrade() --> StoreResults()

StoreResults() --> DisplayReport()

@enduml

**🧩 6. Summary of Transform Mapping Steps**

| **Step** | **Description** |
| --- | --- |
| **1. Identify Input/Output Boundaries** | Student input and Report output |
| **2. Isolate Transform Center** | Evaluate → Compute → Grade |
| **3. Perform Factoring** | Separate input, transform, and output modules |
| **4. Create Top-Level Control Module** | OnlineExamSystem manages overall coordination |
| **5. Define Interfaces** | Each submodule communicates through well-defined data structures |

**✅ Final Notes (According to Pressman)**

* The **transform flow** divides the system into **input**, **transformation**, and **output** segments.
* Each segment is implemented as a **layer of the architecture**, resulting in a **top-down modular structure**.
* This promotes **modularity**, **maintainability**, and **easy testing** of each logical segment.

Would you like me to now apply your **💎 SAJEEL’s Palette** theme (the blue-violet aesthetic) to this same DFD + architecture UML so it looks professional for submission?

**You said:**

make another system in a same pattern and manner

**ChatGPT said:**

Alright 👍 here’s **another complete system** written in the **same pattern and manner** as before (with a clear **Processing Narrative**, **Data Flow Diagram (PlantUML code)**, and **Mapping to Software Architecture using the Transform Flow technique from Section 9.6.1**).

**🌐 System: Online Course Enrollment and Management System (OCEMS)**

**1. System Description**

The **Online Course Enrollment and Management System (OCEMS)** allows students to browse available courses, register for them, and manage their enrollment online. Instructors can upload course materials and grades, while the admin manages user accounts and course details.

**2. Processing Narrative**

**Input**

* Student inputs login credentials.
* Student selects a course for enrollment.
* Instructor uploads course material or student grades.
* Admin adds or updates course and user details.

**Transform Flow Characteristics**

This system exhibits **transform flow** because data flows inward from external entities (students, instructors, admin), gets **validated and transformed** into structured internal data (such as enrollment records or course updates), and then flows outward as **user-specific outputs** (confirmation messages, updated dashboards, or reports).

**Processing Steps**

1. **Input Data Acquisition:** User inputs (login credentials, course selections, or updates).
2. **Validation:** The system validates user credentials and input details.
3. **Transformation:** The system processes input — enrolls student, updates materials, or manages course data.
4. **Output Generation:** Confirmation, updated course list, or reports are displayed.

**3. Data Flow Diagram (DFD) — Transform Flow**

Here’s the **PlantUML DFD code** for this system 👇

@startuml

title Online Course Enrollment & Management System (OCEMS) - DFD

!define RECTANGLE class

skinparam rectangle {

BackgroundColor #fdfdfd

BorderColor #1a237e

RoundCorner 20

Shadowing false

}

actor "Student" as S

actor "Instructor" as I

actor "Admin" as A

rectangle "Input Acquisition" as Input {

S --> (Submit Login & Course Selection)

I --> (Upload Materials/Grades)

A --> (Add/Update Course Data)

}

rectangle "Processing & Validation" as Process {

(Submit Login & Course Selection) --> (Validate User Input)

(Upload Materials/Grades) --> (Validate User Input)

(Add/Update Course Data) --> (Validate User Input)

(Validate User Input) --> (Transform Data)

}

rectangle "Output & Reporting" as Output {

(Transform Data) --> (Generate Confirmation/Reports)

(Generate Confirmation/Reports) --> S

(Generate Confirmation/Reports) --> I

(Generate Confirmation/Reports) --> A

}

@enduml

**4. Mapping DFD to Software Architecture (Transform Flow → Layered Architecture)**

Following **Section 9.6.1 technique**, the DFD is mapped into a **Layered Architecture**:

**a. Input Layer (Acquisition Boundary)**

Handles user interactions and collects inputs:

* LoginInterface
* CourseSelectionInterface
* InstructorUploadInterface
* AdminDashboardInterface

**b. Transformation Layer (Processing Core)**

Performs all business logic and validations:

* AuthenticationManager
* EnrollmentManager
* CourseManager
* ValidationService

**c. Output Layer (Entity & Reporting Boundary)**

Generates user-visible results:

* ConfirmationService
* ReportGenerator
* NotificationManager

**5. Software Architecture (PlantUML)**

@startuml

title Online Course Enrollment & Management System (OCEMS) - Software Architecture

package "Input Layer" {

[LoginInterface]

[CourseSelectionInterface]

[InstructorUploadInterface]

[AdminDashboardInterface]

}

package "Transformation Layer" {

[AuthenticationManager]

[EnrollmentManager]

[CourseManager]

[ValidationService]

}

package "Output Layer" {

[ConfirmationService]

[ReportGenerator]

[NotificationManager]

}

[LoginInterface] --> [AuthenticationManager]

[CourseSelectionInterface] --> [EnrollmentManager]

[InstructorUploadInterface] --> [CourseManager]

[AdminDashboardInterface] --> [CourseManager]

[AuthenticationManager] --> [ValidationService]

[EnrollmentManager] --> [ValidationService]

[CourseManager] --> [ValidationService]

[ValidationService] --> [ConfirmationService]

[ValidationService] --> [ReportGenerator]

[ValidationService] --> [NotificationManager]

@enduml

**✅ Summary**

| **Element** | **Description** |
| --- | --- |
| **System Type** | Online Course Enrollment & Management |
| **Flow Type** | Transform Flow |
| **Architecture Style** | Layered |
| **Flow Mapping** | Input → Validation → Transformation → Output |
| **DFD Characteristics** | Clearly separates input, processing, and output stages |