Module Guide for Plutos

Team #10, Plutos
Payton Chan
Eric Chen
Fondson Lu
Jason Tan
Angela Wang

January 14, 2025

1 Revision History

Date	**Version**	**Notes**
Date 1	1.0	Section 2-8, 10-12
Date 2	1.1	Notes

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	**description**
AC	Anticipated Change
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
Plutos	Explanation of program name
UC	Unlikely Change
OCR	Optical Character Recognition
JWT	JSON Web Token
[etc. —SS]	[... —SS]

Contents

List of Tables

List of Figures

3 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (Parnas et al., 1984). We advocate a decomposition based on the principle of information hiding (Parnas, 1972). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules layed out by Parnas et al. (1984), as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (Parnas et al., 1984). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility, and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 9 describes the use relation between modules.

4 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 4.1, and unlikely changes are listed in Section 4.2.

4.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: Addition or removal of input file formats.

AC2: Changes to the default predefined categories provided to the users.

AC3: The list of languages and/or currencies accepted by the system.

AC4: Replacement or retraining of the model used for data extraction and categorization.

AC5: Changes to the user interface.

AC6: Modifications to the heuristics of data parsing and categorization accuracy.

AC7: Adoption of new testing methodologies.

4.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: Input/Output devices.

UC2: Supported Platforms (iOS and Android).

UC3: Use of ML and OCR for data extraction.

UC4: Deviation of application purpose to other fields (investments, contracts, etc.).

5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 1. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Hardware-Hiding Module

M2: Behavior-Hiding Module

- **M2.1:** Input Format Module
- **M2.2:** Output Generation Module

M3: Software Decision Module

- **M3.1:** OCR Processing Module
- **M3.2:** Machine Learning Module
- **M3.3:** Budget Calculation Module

M4: User Interaction Module

- **M4.1:** Upload Interface Module
- **M4.2:** Results Display Module

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Input Format Module Output Generation Module
Software Decision Module	OCR Processing Module Machine Learning Module Budget Calculation Module
User Interaction Module	Upload Interface Module Results Display Module

Table 1: Module Hierarchy

6 Connection Between Requirements and Design

To satisfy the various requirements, several key design decisions have been made. Each requirement from the SRS is addressed by one or more specific modules, and their corresponding rationales can be found here:

User Account Management (FR1 - FR6): The system requires secure user account management functionality, including account creation, login, logout, and password reset. To meet these requirements, the User Interaction Module is designed to handle user authentication (R3, R4), while the Hardware-Hiding Module and Behavior-Hiding Module ensure that the user interface can interact seamlessly with the backend system. Modules like the Input Format Module handle user inputs during account creation and login processes (R1, R2).

Receipt Scanning Input (FR7 - FR11): The system must handle receipt input via camera capture or file upload. The Input Format Module is responsible for processing images taken by the user or uploaded from their device (R7, R8). The OCR Processing Module is designed to extract relevant data from these images to ensure accurate recognition of receipt details (R14, R15). The Output Generation Module is also involved in providing a preview and confirming receipt data (R10).

Manual Receipt Input (FR12 - FR13): The system supports manual entry of receipt details, validated by the system to ensure completeness. The User Interaction Module facilitates the input from the user, while the Input Format Module and Output Generation Module handle the data and interface to ensure accurate input and validation (R12, R13).

Database Management (FR14 - FR15): The system is designed to securely store user data, including account information and receipt images. This is addressed through a combination of modules such as the Hardware-Hiding Module, which separates the database interactions, and the Software Decision Module, which ensures secure data handling (R14, R15).

Item Recognition and Categorization (FR16 - FR18): For the identification and categorization of items in receipts, the OCR Processing Module and Machine Learning Module are used. These modules work together to extract relevant data from receipt images and categorize the items accordingly (R16, R17, R18). Modifications to item attributes can be handled by the User Interaction Module (R18).

Financial Tracking (FR19 - FR24): The system is designed to track user spending, generate budgets, and send notifications when certain thresholds are met. Modules like the Budget Calculation Module, Machine Learning Module, and OCR Processing Module are responsible for financial analysis and tracking (R19 - R24). The User Interaction Module plays a significant role in displaying this information in a user-friendly format (R19, R20, R21).

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas et al. (1984). The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. *Plutos* means the module will be implemented by the Plutos software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented.

7.1 Hardware Hiding Modules (M1)

Secrets: The data structure and algorithm used to implement the virtual hardware.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software. So, the system can use it to display outputs or to accept inputs.

Implemented By: OS

7.2 Behaviour-Hiding Module (M2)

Secrets: The contents of the required behaviours.

Services: Includes programs that provide externally visible behaviour of the system as specified in the software requirements specification (SRS) documents. This module serves as a communication layer between the hardware-hiding module and the software decision module. The programs in this module will need to change if there are changes in the SRS.

Implemented By: –

Type of Module: Abstract Data Type

7.2.1 Input Format Module (M2.1)

Secrets: The format and structure of the input data.

Services: Converts the input data into the data structure used by the input parameters module.

Implemented By: Plutos

Type of Module: Abstract Data Type

7.2.2 Output Generation Module (M2.2)

Secrets: The output format and its internal representation (e.g., report structure for budget plans, exportable formats like CSV or PDF).

Services: Converts the processed data into a user-facing format for screen display or file export.

Implemented By: Plutos

Type of Module: Abstract Object

7.3 Software Decision Module (M3)

Secrets: The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are *not* described in the SRS.

Services: Includes data structure and algorithms used in the system that do not provide direct interaction with the user.

Implemented By: –

7.3.1 OCR Processing Module (M3.1)

Secrets: The specific OCR algorithms or libraries used for extracting text from images.

Services: Processes input image files to extract text and convert it into structured data.

Implemented By: Plutos

7.3.2 Machine Learning Module (M3.2)

Secrets: The model architecture, training datasets, and feature engineering methods used for categorizing expenses.

Services: Predicts spending categories based on parsed data and predefined labels.

Implemented By: Plutos

7.3.3 Budget Calculation Module (M3.3)

Secrets: The algorithms used for generating budget plans and financial suggestions based on input data.

Services: Translates categorized expenses into meaningful budgets, applying user-specific or general budgeting rules.

Implemented By: Plutos

7.4 User Interaction Module (M4)

Secrets: The design and structure of the UI components used to display categorized data and budgets.

Services: Shows categorized expenses, financial suggestions, and budget plans in a user-friendly format.

Implemented By: Plutos

7.4.1 Upload Interface Module (M4.1)

Secrets: How the interface handles file uploads or image capture from a camera.

Services: Accepts files or captures images, then forwards them to the input module for preprocessing.

Implemented By: Plutos

7.4.2 Results Display Module (M4.2)

Secrets: The design and structure of the UI components used to display categorized data and budgets.

Services: Displays categorized expenses, financial suggestions, and budget plans in a user-friendly format.

Implemented By: Plutos

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	**Modules**
R1	M1, M2.1, M2.2, M4
R2	M2.1, M4
R3	M4
R4	M2.2, M4
R5	M2.2, M4
R6	M2.2, M4
R7	M2.2, M4
R8	M2.2, M4
R9	M2.2
R10	M2.2, M4
R11	M2.2, M4
R12	M4.1
R13	M4.1
R14	M3.1, M3.2
R15	M3.1, M3.2
R16	M3.1, M3.2, M3.3
R17	M3.1, M3.2, M3.3
R18	M3.1, M3.2, M3.3
R19	M2.2, M3.1, M3.2, M3.3
R20	M2.2, M3.1, M3.2, M3.3
R21	M2.2, M3.1, M3.2, M3.3
R22	M2.2, M3.1, M3.2, M3.3
R23	M2.2, M3.1, M3.2, M3.3
R24	M2.2, M3.1, M3.2, M3.3

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M2.1
AC2	M3.2, M3.3
AC3	M4
AC4	M3.1, M3.2
AC5	M4.2
AC6	M3.2, M3.3
AC7	M3.1, M3.2, M3.3

Table 3: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas (1978) said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

[The uses relation is not a data flow diagram. In the code there will often be an import statement in module A when it directly uses module B. Module B provides the services that module A needs. The code for module A needs to be able to see these services (hence the import statement). Since the uses relation is transitive, there is a use relation without an import, but the arrows in the diagram typically correspond to the presence of import statement. —SS]

[If module A uses module B, the arrow is directed from A to B. —SS]

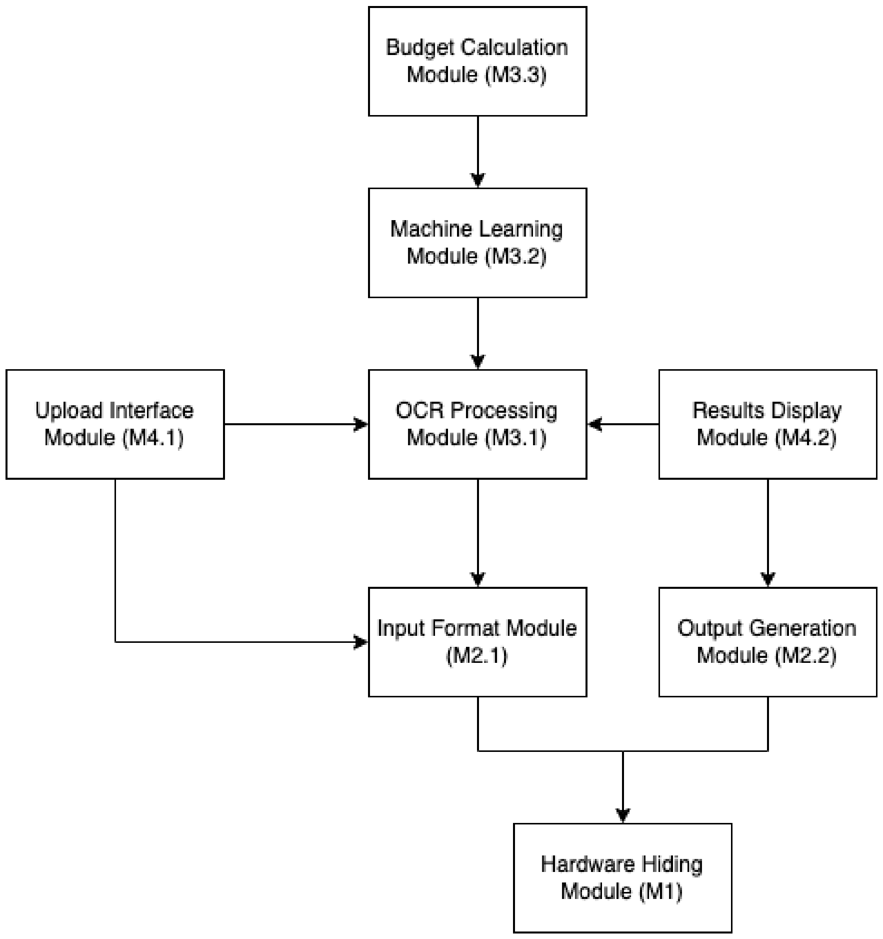
Budget Calculation Module (M3.3)
Machine Learning Module (M3.2)
Upload Interface Module (M4.1)
OCR Processing Module (M3.1)
Results Display Module (M4.2)
Input Format Module (M2.1)
Output Generation Module (M2.2)
Hardware Hiding Module (M1)

Figure 1: Use hierarchy among modules

10 User Interfaces

The user interfaces for *Plutos* are prototyped via Figma. All the wire frames are to be translated to front-end code using our programming language of choice (React Native + TypeScript). The corresponding UI pages can be found down below.

- User Account Management/Login.
- Add Income/Budget/Expense.

As *Plutos* is designed to be a fully-functioning app that lives on the cloud, there will be no dedicated hardware components. Hence there are no relevant physical user interfaces.

11 Design of Communication Protocols

1. **Overview**: The communication protocol facilitates data exchange between the mobile front end, back-end server, and third-party APIs (e.g., OCR, AI categorization, and external data sources). It ensures secure, reliable, and efficient interactions to enable receipt parsing, expense categorization, and budgeting functionalities.
2. **Communication Layers**:
   - **Application Layer**
     - **Purpose**: Handles user-facing operations such as sending images, fetching categorized expenses, and updating budgets.
     - **Data Format**: JSON
     - **Transport Mechanism**: HTTPS over RESTful API endpoints.
   - **Transport Layer**
     - **Protocol**: HTTPS (TLS 1.3 for encryption and secure data transmission).
     - **Port**: 443 (Default for HTTPS)
   - **Network Layer**
     - **Protocol**: IPv4/IPv6 for data routing and delivery.
     - **Fallback Handling**: Automatic retries for intermittent connectivity loss.
3. **Data Flow**
   - **Image Upload (Receipt Parsing)**
     - **Front End**:
       - Captures the receipt image using the device camera.
       - Encodes the image in Base64 or uploads as a binary file.

    - Sends the image along with metadata (e.g., user ID, timestamp) to the back end.
  - **Back End**:
    - Receives and validates the image payload.
    - Forwards the image to the OCR/AI service via an internal API call.
- **Expense Categorization**
  - **AI Service**:
    - Processes the image and extracts structured data (e.g., items, prices, dates).
    - Categorizes the extracted data using a machine learning model.
    - Returns the processed data to the back end.
  - **Back End**:
    - Formats the categorized data into a JSON response.
    - Sends it to the front end for user display.
- **Budget and Forecast Management**
  - **Front End**:
    - Sends requests for fetching/updating budgets or forecast predictions.
  - **Back End**:
    - Queries the database for budget/expense data.
    - Updates user budgets based on inputs.
    - Runs predictive models for forecasting.
  - **Response**:
    - Sends updated data or predictions to the front end.

4. **API Design**
   Endpoints

   - **POST/upload-receipt**
     - **Request**

```
{
 "userId": "12345",
 "image": "Base64String or Binary",
 "timestamp": "2025-01-13T10:00:00Z"
}
```

     - **Response**

```
{
 "status": "success",
 "data": {
  "parsedItems": [
   { "name": "Milk", "price": 3.5, "category": "Groceries" },
   { "name": "Bread", "price": 2.0, "category": "Groceries" }
  ]
 }
}
```

   - **GET/expenses**
     - **Request**: `?userId=12345`
     - **Response**:

```
{
 "status": "success",
 "data": {
  "totalExpenses": 150.75,
  "categories": {
   "Groceries": 75.5,
   "Entertainment": 30.0
  }
 }
}
```

   - **PUT/update-budget**
     - **Request**:

```
{
 "userId": "12345",
 "budget": {
  "Groceries": 200,
  "Entertainment": 100
 }
}
```

  - **Response**:

```
{
 "status": "success",
 "message": "Budget updated successfully."
}
```

5. **Error Handling**

   - **HTTP Status Codes**:
     - `200 OK`: Successful operation.
     - `400 Bad Request`: Invalid input or missing parameters.
     - `401 Unauthorized`: User authentication failed.
     - `500 Internal Server Error`: Server-side issues.
   - **Error Response Format**:

```
{
 "status": "error",
 "code": "XXX",
 "message": "Some error code lol idk"
}
```

6. **Security Measures**

   - **Authentication and Authorization Using JWT** - To ensure secure communication between the client (mobile app) and the backend server, the system utilizes JWT (JSON Web Token) for authentication and authorization.
     - **Token Generation**
       * Upon successful user login or registration, the backend server generates a JWT containing the user's unique identifier (user_id) and other claims (e.g., roles or permissions) as needed.
       * The token is signed using a strong secret key or a private key (in the case of asymmetric signing algorithms like RS256) to ensure authenticity.
     - **Token Structure**
       A JWT consists of three parts:

    - **Header**: Specifies the signing algorithm (e.g., HS256, RS256).
    - **Payload**: Contains claims, such as user information and token expiration (exp).
    - **Signature**: Ensures the token has not been tampered with.
- **Token Storage**
    - The client stores the JWT securely in memory or a secure storage mechanism (e.g., SecureStore in React Native or AsyncStorage with encryption).
    - Tokens must never be stored in plain text or local storage to avoid XSS vulnerabilities.
- **Token Validation**
    - The backend server validates incoming JWTs in the Authorization header of HTTP requests.
      Example
      `Authorization: Bearer <token>`
    - Validation includes:
        - Verifying the signature.
        - Checking the exp claim to ensure the token has not expired.
        - Ensuring the token was issued by a trusted source (iss claim).
- **Token Expiry and Refresh**
    - Tokens include a short expiration time (exp) for security.
    - A **refresh token** mechanism is implemented to issue new tokens without requiring reauthentication:
        - Refresh tokens are stored securely and exchanged only via secure channels (e.g., HTTPS).
        - When the access token expires, the client uses the refresh token to request a new JWT from the backend.
- **Securing JWT in Transit**
    - All API calls using JWT must be encrypted with HTTPS to prevent interception (man-in-the-middle attacks).
    - Implement strict **CORS** policies on the backend to prevent unauthorized access.
- **Revocation Mechanisms**
    - Implement a blacklist or token revocation strategy to invalidate compromised tokens.
    - Tokens can be associated with a unique identifier in the database (e.g., a session ID) and marked invalid upon logout or suspected compromise.

12 Timeline

A breakdown of the team's delegation of tasks and responsibilities can be found in the respective GitHub repository Issues tab. The delegation of tasks assumes an even distribution amongst team members as outlined in the Development Plan. However, if situation arises where a certain task requires additional member's assistance to complete, then the team will need to communicate it across.

All document related issues are to be completed prior to the respective document's Rev1 deadline. This is also the deadline date of the *Final Documentation* .

All feature related issues (functional and non-functional) are to be completed prior to the Rev1 final demonstration.

The link to the GitHub repository Issues tab for *Plutos* can be found here.

References

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

David L. Parnas. Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press. ISBN none.

D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.