# **Project Development Phase**

**Requirements Gathering and Analysis:**

* + **Objective:** Understand the client's needs, expectations, and the problem to be solved.
  + **Activities:** Conduct interviews, workshops, and surveys. Document and analyze gathered information to define project scope and objectives.

**Planning:**

* + **Objective:** Create a detailed project plan outlining tasks, timelines, resources, and dependencies.
  + **Activities:** Define project milestones, allocate resources, estimate budget, and set timelines. Develop a project schedule and allocate tasks to team members.

**Design:**

* + **Objective:** Create a blueprint or design of the solution based on gathered requirements.
  + **Activities:** Develop wireframes, mockups, and prototypes. Design user interfaces and architectural components. Determine the technology stack.

**Development:**

* + **Objective:** Write code and build the actual solution according to the design.
  + **Activities:** Write, test, and integrate code modules. Develop both front-end and back-end components. Implement databases and data storage solutions.

# **No. Of Functional Features Included in The Solution**

* **E-commerce Functionality:**
  + Product catalog with categories and detailed product listings.
  + Shopping cart for adding and managing selected items.
  + Secure checkout process with payment integration.
  + Order management and tracking.
* **Content Management:**
  + Blog platform for publishing fashion tips, trends, and style guides.
  + Ability to create, edit, and manage blog content.
* **Personal Styling Services (Optional):**
  + Appointment scheduling for personalized styling consultations.
  + Virtual styling options for online consultations.
* **Customer Loyalty Program:**
  + Loyalty program enrollment with points accumulation and rewards redemption.
  + Tracking and management of customer loyalty points.

# **Code-Layout, Readability and Reusability**

**Code Layout:**

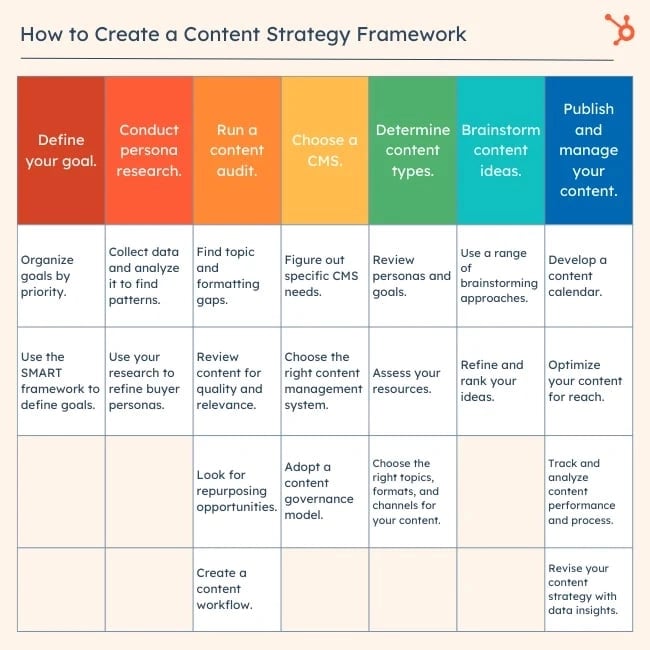
* **Consistent Indentation:** Use a consistent number of spaces or tabs for indentation to improve code readability.
* **Meaningful Variable and Function Names:** Use descriptive names that clearly indicate the purpose and functionality of variables and functions.

**Readability:**

* **Avoid Deep Nesting:** Limit the depth of nested structures (e.g., loops, conditional statements) to enhance code readability.
* **Keep Functions Short and Focused:** Functions should have a clear and specific purpose. Avoid overly long or complex functions.

**Reusability:**

* **Modularize Code:** Break down code into small, reusable modules or functions that can be used across different parts of the application.
* **Avoid Hard coding:** Parameterize values that may change over time or in different contexts, making the code more adaptable.
* **Design Patterns:** Implement design patterns that encourage code reuse and maintainability, such as Singleton, Factory, and Strategy patterns.



# **Utilization Of Algorithms, Dynamic Programming, Optimal Memory Utilization**

**Utilization of Algorithms:**

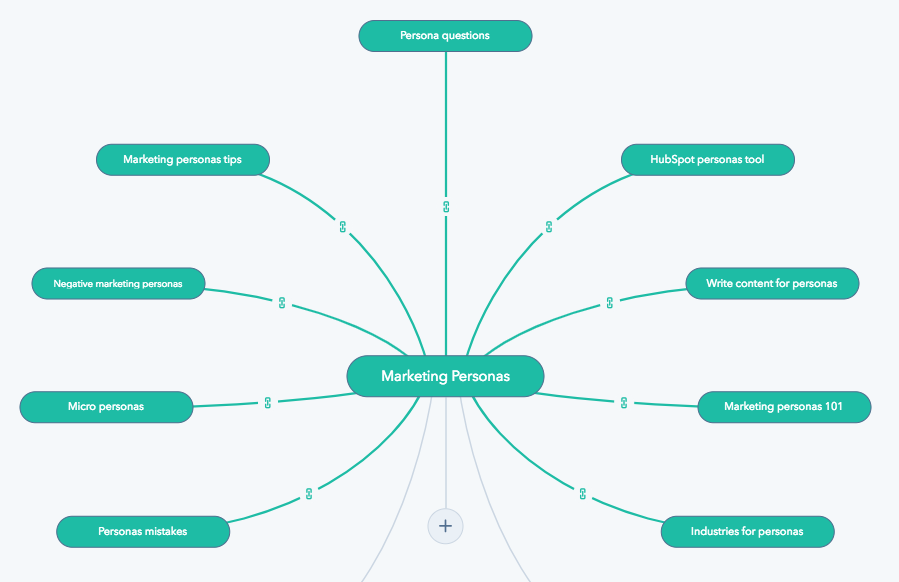
* **Selection of Appropriate Algorithms:**
  + Choose algorithms based on the specific problem to be solved. For example, sorting algorithms like Quick Sort or Merge Sort for sorting data, or graph algorithms like Dijkstra's or A\* for path finding.

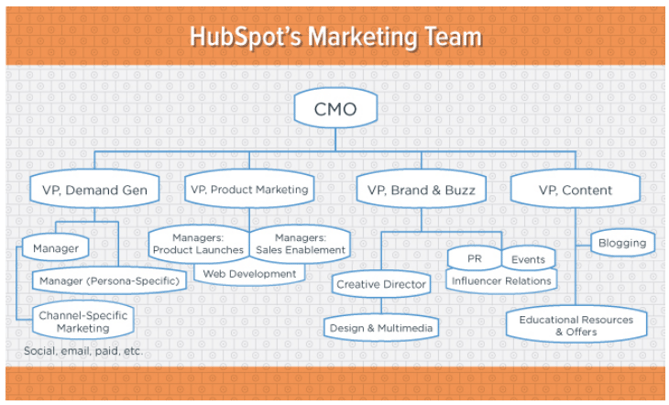
**Dynamic Programming:**

* **Identify Overlapping Sub problems:**
  + Break down complex problems into smaller sub problems and identify where solutions to these sub problems can be reused.

**Optimal Memory Utilization:**

* **Data Structures Selection:**
  + Choose appropriate data structures based on the requirements of the application. For example, use arrays for contiguous memory allocation or linked lists for dynamic data.





# **Debugging & Traceability**

**Debugging:**

* **Reproduce the Issue:**
  + Begin by attempting to reproduce the issue or bug. This helps in understanding the specific conditions under which it occurs.
* **Use Debugging Tools:**
  + Utilize integrated development environments (IDEs) or standalone debugging tools that provide features like breakpoints, watch variables, and step-through execution.

**Traceability:**

* **Version Control:**
  + Utilize version control systems (e.g., Get) to track changes in the codebase. This provides a history of code modifications and facilitates rollbacks if needed.
* **Use Descriptive Commit Messages:**
  + Provide clear and informative commit messages that describe the purpose and context of each change.

# **Exception Handling**

* **Identify Potential Exceptions:**
  + Analyze the code to identify areas where exceptions may occur. This could be during operations like database queries, file operations, network requests, or any other potentially risky operation.
* **Use Try-Catch Blocks:**
  + Wrap the risky code within a try block. If an exception occurs within this block, it will be caught by the corresponding catch block.
* Code:

try:

# Risky code here

except Exception Type as e:

# Handle the exception here

# e contains information about the exception

try:

# Risky code here

except FileNotFoundError as e:

# Handle file not found error

except DatabaseError as e:

# Handle database error

# Add more specific catch blocks as needed

try:

# Risky code here

except FileNotFoundError as e:

print("File not found. Please check the file path.")

except DatabaseError as e:

print("An error occurred while accessing the database. Please try again later.")

try:

# Risky code here

except FileNotFoundError as e:

print("File not found. Please check the file path.")

except DatabaseError as e:

print("An error occurred while accessing the database. Please try again later.")

finally:

# Clean up operations here (e.g., closing connections)

try:

# Risky code here

except FileNotFoundError as e:

print("Sorry, we couldn't find the file you requested. Please try again.")

except DatabaseError as e:

print("We're experiencing technical difficulties. Please try again later.")