**Engineering method**

***First step: Identificate the problem***

**SOFTWARE ENGINEERING PROBLEM SPECIFICATION TABLE, identifying the following elements**

| **CLIENT** | **Discrete Structures** |
| --- | --- |
| **USER** | **Users** |
| **FUNCTIONAL REQUIREMENTS** | R1- Add tasks or reminder  R2- Modify task or reminder  R3- Delete task or reminder  R4- Display a list of tasks  R5- Management of Priority Task  R6- Management of Non-Priority Task  R7- Undo user action |
| **CONTEXT OF THE PROBLEM** | A company wishes to develop a system that allows the management of tasks and reminders. This system should allow users to coordinate their activities through functions to organize, add, manage, modify and delete their tasks. The tasks and reminders of each user should be stored with all the important information and their priority specified. The program must be able to show to the user a list of all the tasks and reminders stored, that has to be organized by the limit date or priority. The priority tasks will be managed first and the non-priority tasks will be managed according to the order of arrival. Also, the software must have a function to undo some actions of a user. |
| **NON-FUNCTIONAL REQUIREMENTS** | RN1 – Implement a graphic interface  RN2 –Create a user-friendly visualization of the app |
|  |  |

**Functional Requirements Analysis Table (Note: One table for each functional requirement)**

| Name or identifier | R1 - Add tasks or reminder | | |
| --- | --- | --- | --- |
| Summary | When a user wants to add a new task, it is necessary to give the basic information of the task, such as: id, title, description, limit date and category of priority. | | |
| Inputs | Input name | Data type | Selection or repetition condition |
| Id | String | - |
| Title | String | “Task Title” |
| Description | String | - |
| Limit date | Calendar | - |
| Priority | int | 1. Priority 2. Non-Priority |
| Result or postcondition | These new tasks are stored in a slot of a hash table depending on the identifier, and contains an object of type Task. These tasks contain the whole information that the user specified previously. If the operation is successful, the program shows a message informing the user the state of his requirement. | | |
| Outputs | Output name | Data type | Selection or repetition condition |
| msg | String | “The operation is succesfull” |

| Name or identifier | R2 - Modify task or reminder | | |
| --- | --- | --- | --- |
| Summary | The software must modify the tasks that the user previously added. The user needs to select the task to modify and provide the new information of the task or reminder. | | |
| Inputs | Input name | Data type | Selection or repetition condition |
| id | string | - |
| newTitle | String | - |
| newDescription | String | - |
| newDate | Calendar | - |
|  | newPriority | int | 1. Priority  2. non-priority |
| Result or postcondition | The program searches the task selected by the user, in the hash table depending on the id, to modify and replace the old information by the new one. If the operation is successful, the program will show a message informing the state of the process. | | |
| Outputs | Output name | Data type | Selection or repetition condition |
|  | msg | String | State of the operation |

| Name or identifier | R3 - Delete task or reminder | | |
| --- | --- | --- | --- |
| Summary | The program must delete a task of the hash table selected by the user. The user needs to provide the id of the task. | | |
| Inputs | Input name | Data type | Selection or repetition condition |
| id | String | ID of the task |
| Result or postcondition | The program searches the task selected in the hash table and deletes the information. If the operation is successful, the program will show a message informing the state of the process | | |
| Outputs | Output name | Data type | Selection or repetition condition |
| msg | String | State of the operation |

| Name or identifier | R4 - Display a list of tasks | | |
| --- | --- | --- | --- |
| Summary | The software must allow the user to visualize a list of their tasks and reminders previously added. The list has to be ordered depending on the limit date or the priority | | |
| Inputs | Input name | Data type | Selection or repetition condition |
|  |  |  |
| Result or postcondition | The program should show to the user the list of each task stored in every slot of the hash table organized depending on the priority. | | |
| Outputs | Output name | Data type | Selection or repetition condition |
| list | string | List of every task of the user |

| Name or identifier | R5- Management of Priority Task | | |
| --- | --- | --- | --- |
| Summary | The software must use a structure of priorities for the most essential tasks. Depending on the importance the task is added in the tail of the structure, this allows to ensure that the most relevant tasks are managed first. | | |
| Inputs | Input name | Data type | Selection or repetition condition |
|  |  |  |
| Result or postcondition | The program organizes the tasks in the Queue depending on the level of importance. | | |
| Outputs |  |  |  |
|  |  |  |

| Name or identifier | R6- Management of Non-Priority Task | | |
| --- | --- | --- | --- |
| Summary | The software must use a Queue for the non-priority tasks and manage them in order of arrival | | |
| Inputs | Input name | Data type | Selection or repetition condition |
|  |  |  |
| Result or postcondition | Non-priority tasks should be handled in the order in which they were added, so that the FIFO principle is followed | | |
| Outputs |  |  |  |
|  |  |  |

| Name or identifier | R7- Undo user action | | |
| --- | --- | --- | --- |
| Summary | The system must implement an "undo" function that allows users to revert the last action performed, whether it is adding, modifying or deleting a task. This function should use a stack (LIFO) to keep track of user actions. | | |
| Inputs | Input name | Data type | Selection or repetition condition |
|  |  |  |
| Result or postcondition | FALTA ARREGLAR  When a user uses the undo function, the last action performed should be reverted, restoring the affected task or reminder information to its state prior to the action. | | |
| Outputs |  |  |  |
|  |  |  |

**Definition of the problem:** Create a task and reminder management system that allows users to organize and manage their to-dos, with priority options and the ability to undo actions.

***Step 2: Information Gathering***

**Definitions:**

* **Structure:** A structure is an organized way of storing data. They are very important for making different operations on some stored collections efficiently. For example, we can find some structures in computing such as : arrays, linked lists, trees, stacks, queues and many others. Each of these structures has its own characteristics, it depends on the context of the problem to use their functionalities and advantages to develop a solution.
* **Hash table:** A hash table is a structure for storing information efficiently. It is designed for fast data access by associating a unique key with a value. For storing data in the slot it is necessary to use a hash function to convert the key given in an index of the hash table. This allows us to retrieve the index with the value to store. Hash tables are very efficient for data retrieval because they have constant-time (O(1)) in the state of complexity and (O(n)) in the worst case.
* **Queue:** A Queue is a data structure that follows the FIFO principle (First in, first out). The first element added to the queue is the first to be removed.Queues are used when the tasks must be processed in the order of arrival. Also,it is a linear dynamic structure, this means that it has a specific order and the size can increase or decrease. The basic operations are enqueue (add an element to the tail), dequeue (remove and return an element to the tail), front, empty and full.
* **Stack:** A stack is a data structure that follows the LIFO principle (Last in first out).the last item placed on the stack is the first one to be removed. Is an object where we can insert and take elements according to the LIFO. It is limited because we can only insert and delete to the top of the stack. The basic operations are push (add at the top), pop (delete and get the top element), peek (take the top element) and empty(see if the stack is empty).
* **Heap**: Is a data structure for organizing and managing elements efficiently. They are useful for finding the maximum element or minimum element. This structure has a specific property. In a maximum heap each parent has a value larger than the value of his childs. In a minimum heap each parent node is smaller than his children.

* **Heapify:** It is an ordering algorithm that allows us to maintain the properties of the structure of a heap. This is achieved in the case of a maximum heap by exchanging nodes in subtree until the largest node is the root and in the case of a minimum heap by exchanging until the smallest node is the root, that is, until each subtree is a valid heap. This tool is fundamental for the operation of other algorithms such as heapsort.
* **HeapSort**: It is a very efficient sorting algorithm. Its function is to implement a heap to sort a structure in an order. It can also be said that it serves to order a structure in a maximum heap. It achieves this by removing and reintegrating the root node continuously until it manages to convert a simple structure to a maximum heap.

***Step 3: Search creative solutions***

In today's world we know many technologies, it is for this reason that to solve the problem posed there are thousands of possibilities. In this document we will present some of them.

**//NO SE SI ESTA SIRVE**

**Alternative 1: Implementation of a data graph.** Using a graph to model the problem poses a possible solution to the problem because we could implement it in the task management system instead of a hashtable. Its use represents a facility for the user, since it allows him to customize his tasks more thoroughly. In addition, the network provides more flexibility in expressing priorities, making it a very useful tool for the system.

**Alternative 2: Develop a program.**

***Step 4: Preliminary designs***

***Step 5: Selection of the best solution***

***Step 6: Preparation of Reports and Specifications***

***Step 7: Implementation***