**Лабораторна робота №2  
Дослідження процесу розробки програмного забезпечення.**

**Специфікування вимог**

Мета – отримати навички специфікування вимог до програмного забезпечення.

Завдання

1. Сформувати робочу групу (3-5 чоловік), розподілити ролі.
2. За узгодженням з викладачем обрати варіант завдання (табл..1) для виконання лабораторних робіт.
3. Провести попередній аналіз предметної області, визначити функції ПЗ, що проектується.
4. Побудувати діаграму прецедентів на основі проведеного попереднього аналізу.
5. Специфікувати вимоги.
6. Обрати та обґрунтувати обрану модель життєвого циклу.
7. Підготувати захист у вигляді мітингу із замовником.

|  |  |  |
| --- | --- | --- |
| 6 | Система  обліку  робочого  часу | дозволяє керівникам видавати завдання й відслідковувати хід  їхнього виконання, а виконавцям - вести облік робочого часу,  витраченого на виконання кожного завдання. |

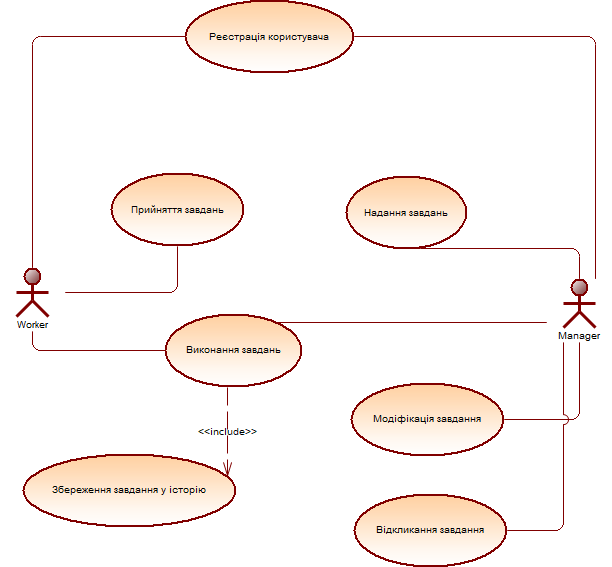
**Аналіз предметної області**

**Система:** Система обліку робочого часу.  
 **Опис системи:** дозволяє керівникам видавати завдання й відслідковувати хід їхнього виконання, а виконавцям - вести облік робочого часу, витраченого на виконання кожного завдання.

Опис предметної області

Предметна область: Система обліку часу  
 Загальна область: система обліку часу (весь процес виконання завдання та слідкування за ним)  
 Основні предметно-значимі актори: Виконавець, Керівник  
 Основні предметно-значимі атрибути:   
 Виконавець - ПІБ, контактні дані, прийняті завдання  
 Керівник - ПІБ, контактні дані, видані завдання  
  
 Програмне забезпечення проектується як система обліку часу. Ця система у даному випадку - це служба, яка надає можливість видавати, приймати та слідкувати за ходом виконання завдання. Перша дія, что може виконуватися - це видача завдання керівником, це і є початком програми. Кінцевих результатів цієї дії багато, але у самої програми якогось кінця нема. Основна частина процесу - це обмін інформацією та взаємодія через цю систему між виконавцем та керівником. Їх взаємодія починається з отриманням виконавцем завдання. Після цього виконавець та керівник взаємодіють при виконанні завдання. Необхідними для цього є наступні дані - контактна інформація керівника та виконавця, час затрачений на виконання завдання виконавцем та сам текст завдання.

**Діаграма та аналіз прецедентів**

****

|  |  |
| --- | --- |
|  | **Прийняття завдань** |
| **Description** | **Прийняття завдання виконавцем** |
| **Actor** | **Виконавець** |
| **Pre-Conditions** | **1. Виконавець зареєстрований в систему**  **2. Є вільні завдання** |
| **Main Flow** | **1. Виконавець переглядає завдання**  **2. Виконавець вибирає вільне завдання**  **3. Виконавець приймає завдання**  **4. Виконавець надає особисті дані** |
| **Alternative Flow** | **1. Виконавець переглядає завдання**  **2. 2а. Вильних завдань немає  2б. Виконавець виходить із системи 3. Виконавець не вибирає завдання** |
| **Post-Condition** | **1. Виконавець починає робити завдання**  **2. Завдання стає зайнятим** |

|  |  |
| --- | --- |
|  | **Виконання завдання** |
| **Description** | **Статус виконання завдання виконавцем** |
| **Actor** | **Виконавець, Керівник** |
| **Pre-Conditions** | **1. Виконавець має завдання** |
| **Main Flow** | **1. Виконавець виконує завдання 2. 2а. Керівник може переглянути час, який Виконавець витрачає на завдання  2б. Керівник може змінити текст завдання та строк його виконання 3. Виконавець виконав завдання 4. Виконавець надсилає завдання Керівнику 5. Керівник приймає виконанне завдання** |
| **Alternative Flow** | **1. 1а. Виконавець відмовляється від завдання  1б. Завдання становиться вільним 2. 2а. Керівник відкликає завдання  2б. Завдання потрапляє у історію завдань**  **3. Виконавець виконав завдання 4. Виконавець надсилає завдання Керівнику 5. 5а. Керівник не приймає виконане завдання  5б. Повертаємося до кроку 1** |
| **Post-Condition** | **1. Керівник отримує виконане завдання**  **2. Надане завдання переходить у історію** |

|  |  |
| --- | --- |
|  | **Надання завдання** |
| **Description** | **Надання завдання керівником** |
| **Actor** | **Керівник** |
| **Pre-Conditions** | **1. Керівник зареєстрований в систему** |
| **Main Flow** | **1. Керівник додає завдання**  **2. Керівник встановлює межі виконання завдання**  **3. Система розміщює завдання** |
| **Alternative Flow** | **1. Керівник додає завдання**  **1а. Система виводить помилку про некоректність введених даних 2. Керівник встановлює межі виконання завдання.  2а. Система виводить помилку про некоректність введених даних 3. Керівник вирішує не додавати завдання** |
| **Post-Condition** | **Вільне завдання з’являється для виконавців** |

|  |  |
| --- | --- |
|  | **Реєстрація користувача** |
| **Description** | **Користувач регіструється** |
| **Actor** | **Керівник, виконавець** |
| **Pre-Conditions** | **1. Користувач обрав опцію реєстрації у систему** |
| **Main Flow** | **1. Користувач заповнює свою інформацію**  **2. Користувач обирає у якості кого він буде використовувати систему.**  **3. Користувач відсилає дані.**  **4. Система реєструє користувача.** |
| **Alternative Flow** | **1. 1а Користувач заповнює свою інформацію**  **1б Користувач залишає деякі з полей пустими**  **1в Система повідомляє користувача та не опрацьовує запит.**  **2. Користувач обирає у якості кого він буде використовувати систему.**  **3 3а. Користувач відсилає дані.**  **3б Дані не надсилаються системі**  **3в Система не опрацьовує запит.**  **4. Система не опрацьовує.** |
| **Post-Condition** | **У системі зареєструвався новий користувач.** |

**Модель життєвого циклу**

**Ітераціна модель**

Ітераційна модель життєвого циклу, на відміну від каскадної, більш підходить під нашу програму, бо наша система буде підтримуватися та використовуватися протягом великого відрізку часу. Тому завдяки цій моделі ми матимемо можливість модифікувати його не тільки під час реалізації, а й у майбутньому, тоді як у каскадній системі ми маємо готовий продукт, чия архитектура не передбачає подальшу модифікацію. Це є головною причиною, чому ця система не підходить нам.

**Специфікація вимог** Функціональні

Ф1. Система обліку робочого часу повинна надавати керівникам можливість видавати завдання та відслідковувати хід їхнього виконання, отримуючи від нього текст завдання, контактні дані керівника (ПІБ, телефон і e-mail) та строк виконання завдання.  
 Ф2. Система обліку робочого часу повинна надавати виконавцям можливість переглядати та приймати завдання на виконання, отримуючи від нього його контактні дані (ПІБ, телефон і e-mail) та дату початку його виконання.  
 Ф3. Система обліку робочого часу повинна надавати виконавцям можливість виставляти завдання як виконане, отримуючи від нього дату закінчення завдання.  
 Ф4. Система обліку робочого часу повинна надавати керівникам можливість прийняти чи відхилити виконане завдання, відправивши його на перевиконання із анулюванням дати виконання.  
 Ф5. Система обліку робочого часу повинна надавати можливість керівникам змінювати завдання або строк його виконання.

Ф6. Система обліку робочого часу повинна надати можливість керівникам відкликати завдання.  
 Ф7. Система обліку робочого часу повинна зберігати історію виконаних та відкликаних завдань.

Ф8. Система обліку робочого часу повинна надати можливість виконавцям та керівникам переглядати історію виконаних та відкликаних завдань.  
 Ф10. Система обліку повинна підтримувати як і режим виконавця, так і режим керівника.

Ф12. Система обліку повинна зберігати свої дані стосовно завдань та користувачів у файлах.

Нефункціональні:  
 НФ1. Система обліку робочого часу повинна бути розроблена з використанням С++.  
 НФ3. Дотримання документації коду.  
 НФ4. Дотримання стилю коду згідно із конвенцією дійсною для використаної мови програмування.  
 НФ5. Система повинна мати архітектуру яка дозволяє розширювати функціонал.  
 НФ6. Система повинна запускатися на OC Windows 10.

**Висновок**

У даній лабораторній роботі ми ознайомились з базовими принципами мови UML, використавши їх практично та розробивши початковий етап програмного забезпечення.