

МОДУЛЬ №2

СПЕЦИФІКАЦІЯ ТА ВІДСТЕЖЕННЯ ВИМОГ

8. ЛАБОРАТОРНЕ ЗАНЯТТЯ 2.1

АНАЛІЗ ВИМОГ ЗА ДОПОМОГОЮ UML

Мета: дослідити засоби уніфікованої мови моделювання UML для аналізу вимог та набути практичних навичок у застосуванні інструментальних засобів Enterprise Architect, IBM Rational Rose та Microsoft Visio при побудові UML-діаграм.

Завдання

1. Ознайомитися із особливостями виконання процесу аналізу вимог.
2. Побудувати діаграми прецедентів (варіантів використання) системи, що задана згідно з варіантом (додаток 1) .
3. Побудувати діаграму компонентів системи.
4. Побудувати діаграму розгортання системи.
5. Навести вербальний опис усіх варіантів використання системи в текстовому редакторі MS Word.

Основні теоретичні відомості

Аналіз вимог – це процес вивчення потреб і цілей користувачів, класифікація і перетворення їх на вимоги до системи, апаратного і програмного забезпечення, встановлення і вирішення конфліктів між вимогами, визначення пріоритетів, меж системи і принципів взаємодії із середовищем функціонування.

Результатом аналізу вимог є набір артефактів, що подається у вигляді текстових документів, моделей та прототипів програмного забезпечення.

Цілями процесу аналізу вимог є:

- 1) Досягнення однакового розуміння замовниками, користувачами та розробниками того, що повинна робити система;
- 2) Надання розробникам якнайкращого розуміння вимог до системи;
- 3) Визначення границь системи;
- 4) Визначення інтерфейсу користувача та системи;

Для аналізу вимог широко застосовуються засоби уніфікованої мови моделювання UML: діаграми варіантів використання, компонентів та розгортання.

УВАГА! UML діаграми мають відповідати опису функціональних вимог теми 4.

Діаграми варіантів використання або прецедентів (Use Case Diagram) застосовуються для відображення функціональності програмного забезпечення, границь та контексту предметної області, для якої призначено програмне забезпечення із зовнішнім середовищем.

Графічним компонентом діаграми варіантів використання є дійові особи (актори), варіанти використання та зв'язки між ними (рис. 2.1).

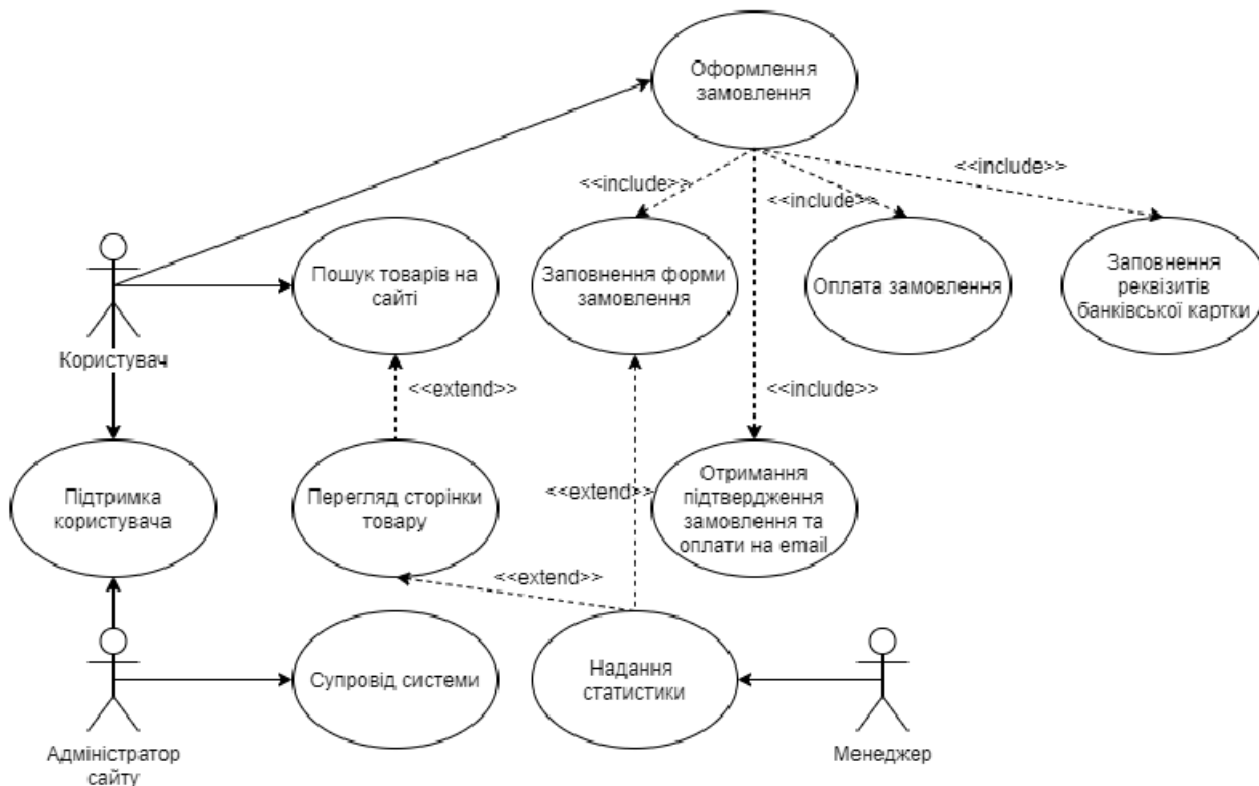


Рис. 2.1. Приклад діаграми прецедентів

Діаграми варіантів використання супроводжуються вербальними описами (специфікаціями). тобто кожний прецедент може бути описаний за допомогою фіксованого потоку подій.

Діаграма компонентів (Component diagram) описує особливості фізичного зображення системи (рис. 2.2) та забезпечує перехід від логічного зображення до реалізації проекту в формі програмного коду. Основними графічними складовими діаграми компонентів є компоненти та з'єднувачі.

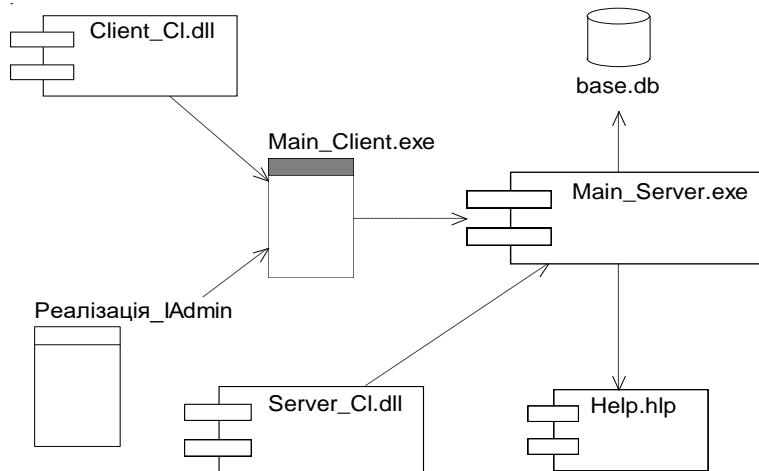


Рис. 2.2. Приклад діаграми компонентів

Компонент є частиною фізичної реалізації системи (наприклад, програмний модуль або інтерфейс користувача), який інкапсулює певний набір функціональних можливостей. З'єднувачі здійснюють взаємодію між користувачами.

Діаграма розгортання (Deployment diagram) відображає загальну конфігурацію і топологію системи, фізичний взаємозв'язок між програмними та апаратними компонентами (рис. 2.3).

Основними графічними компонентами є вузли та зв'язки між ними.

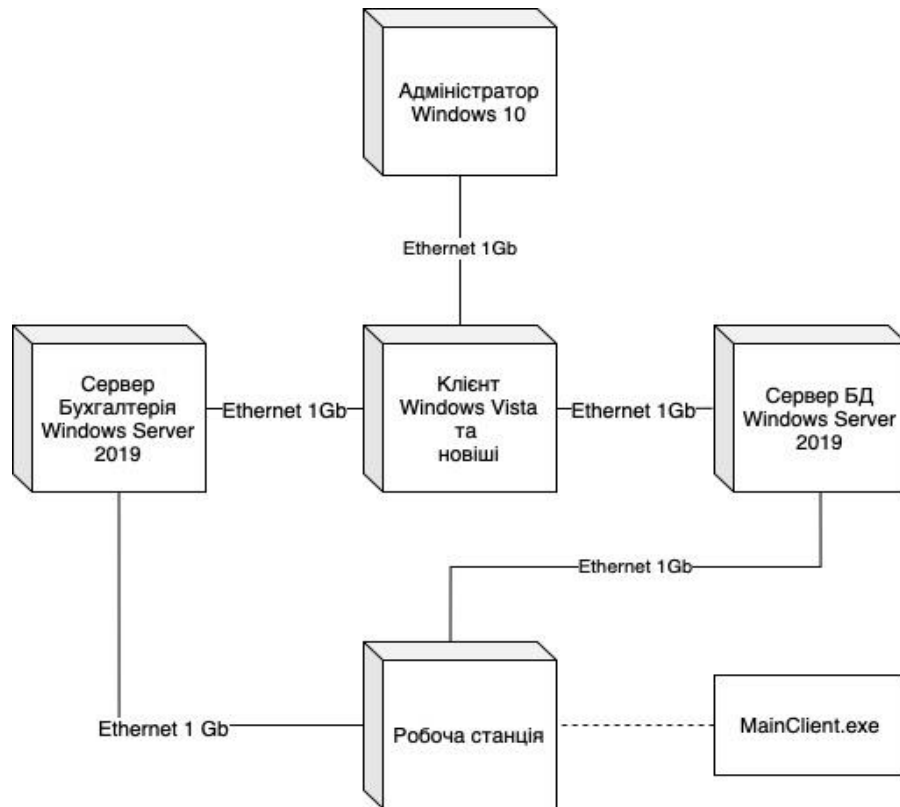


Рис. 2.3. Приклад діаграми розгортання

Вузол - це фізичний об'єкт. Він може бути обчислювальним ресурсом (який має пам'ять, процесор), ресурсом механічної обробки тужність, надійність та пропускну здатність.

Зв'язки між вузлами показують канали, за допомогою яких відбуваються комунікаційні з'єднання.

Контрольні питання

1. Надати визначення аналізу та пояснити його призначення.
2. Назвіть методи та засоби аналізу. Розкрийте особливості застосування методів аналізу.
3. Які існують методи збору та виявлення вимог? Розкрийте їх сутність.
4. Яке призначення діаграми прецедентів?
5. Опишіть основні компоненти діаграми прецедентів, та їх призначення.
6. Які інструментальні засоби використовуються для побудови діаграм прецедентів, компонентів та розгортання?
7. Опишіть основні складові діаграм компонентів та розміщення, їх призначення.

Література: [1, 7, 8]

9. ЛАБОРАТОРНЕ ЗАНЯТТЯ 2.2

АНАЛІЗ ВИМОГ ЗА ДОПОМОГОЮ СТРУКТУРНОГО ПІДХОДУ

Мета: дослідити процес аналізу вимог за допомогою діаграм потоків даних та набутти практичних навичок у використанні інструментальних засобів Enterprise Architect і Microsoft Visio при їх побудові

Завдання

1. На основі технічного завдання з лабораторної роботи № 2.1 виконати аналіз функціональних та експлуатаційних вимог до програмного продукту;
2. Визначити основні технічні рішення (вибір мови програмування, структура програмного продукту, склад функцій ПП, режими функціонування) і занести результати в документ, який називається «Ескізний проект»;
3. Визначити та побудувати діаграми потоків даних для розв'язуваної завдання;
4. Визначити та побудувати діаграми «сутність-зв'язок», якщо програмний продукт містить базу даних;
5. Визначити та побудувати функціональні діаграми;
6. Визначити та побудувати діаграми переходів станів;
7. Визначити та побудувати специфікації процесів;
8. Додати словник термінів;
9. Оформити результати, використовуючи MS Office або MS Visio в вигляді ескізного проекту.

Основні теоретичні відомості

Теоретична частина. Розробка специфікацій

Розробка програмного забезпечення починається з аналізу вимог до нього. В результаті аналізу отримують специфікації програмного забезпечення, що, будують загальну модель його взаємодії з користувачем або іншими програмами і конкретизують його основні функції.

При структурному підході до програмування на етапі аналізу і визначення специфікацій розробляють три типи моделей:

- моделі функцій, моделі даних;
- моделі потоків даних.

Оскільки різні моделі описують проектоване програмне забезпечення з різних сторін, рекомендується використовувати відразу кілька моделей, що розробляються у вигляді діаграм, і пояснювати їх текстовими описами, словниками і т. п.

Структурний аналіз передбачає використання наступних видів моделей:

- діаграм потоків даних (DFD - Data Flow Diagrams), описують взаємодію джерел і споживачів інформації через процеси, які повинні бути реалізовані в системі;
- діаграм «сутність-зв'язок» (ERD - Entity-Relationship Diagrams), що описують бази даних розробляється системи;
- діаграм переходів станів (STD - State Transition Diagrams), що характеризують поведінку системи під часу;
- функціональних діаграм (методика SADT); • специфікацій процесів; • словника термінів.

Специфікації процесів Специфікації процесів зазвичай представляють у вигляді короткого текстового опису, схем алгоритмів, Псевдокод, Flow-форм або діаграм Насс – Шнейдермана.

Словник термінів

Словник термінів являє собою короткий опис основних понять, що використовуються при складанні специфікацій.

Він повинен включати визначення основних понять предметної області, опис структур елементів даних, їх типів і форматів, а також всіх скорочень і умовних позначень.

Діаграми переходів станів

За допомогою *діаграм переходів станів* можна моделювати подальше функціонування системи на основі її попереднього і поточного функціонування. Модельована система в будь-який заданий момент часу знаходиться точно в одному з кінцевого безлічі станів. З плином часу вона може змінити свій стан, при цьому переходи між станами повинні бути точно визначені.

Функціональні діаграми

Функціональні діаграми відображають взаємозв'язки функцій розроблюваного програмного забезпечення. Вони створюються на ранніх етапах проектування систем, для того щоб допомогти проектувальнику виявити основні функції і складові частини проектованої системи і, по можливості, виявити і усунути істотні помилки. Для створення функціональних діаграм пропонується використовувати методологію SADT.

Діаграми потоків даних Для опису потоків інформації в системі застосовуються діаграми потоків даних (DFD - Data flow diagrams). DFD дозволяє описати необхідну поведінку системи у вигляді сукупності процесів, що взаємодіють за допомогою пов'язують їх потоків даних. DFD показує, як кожен з процесів перетворює свої вхідні потоки даних у вихідні потоки даних і як процеси взаємодіють між собою.

Діаграми «сутність-зв'язок» Діаграма сутність-зв'язок - інструмент розробки моделей даних, що забезпечує стандартний спосіб визначення даних і відносин між ними. Вона включає суті і взаємозв'язку, що відображають основні бізнес-правила предметної області. Така діаграма не дуже деталізована, в неї включаються основні сутності і зв'язку між ними, які задовольняють вимогам, що пред'являються до ІС.

Структурний аналіз — метод дослідження системи, який починається з її загального огляду і потім деталізується, набуваючи ієрархічної структури з дедалі більшою кількістю рівнів. Він є один із формалізованих методів аналізу вимог до програмного забезпечення, за яким програмний продукт розглядається як перетворювач інформаційного потоку даних. Основний елемент структурного аналізу — діаграма потоків даних.

Діаграми потоків даних (Data Flow Diagrams) є основним засобом моделювання функціональних вимог. За їх допомогою вимоги розбиваються на функціональні компоненти (процеси) і зображуються у вигляді мережі, пов'язаної потоками даних. Головна мета таких засобів — продемонструвати, як кожний процес перетворює свої вхідні дані у вихідні, а також виявити відношення між цими процесами.

Основні компоненти діаграм потоків даних такі:

- зовнішні сутності — матеріальні об'єкти або фізичні особи, що є джерелом або приймачем інформації;
- процеси — діяльність із перетворення вхідних потоків даних у вихідні відповідно до визначеного алгоритму;
- сховище даних (накопичувач) — абстрактний пристрій, призначений для зберігання інформації;
- потоки даних — механізми для моделювання передачі інформації від джерела до приймача.

Для зображення діаграм потоків даних традиційно використовуються дві різні нотації (табл. 2.1): Йордана-ДеМарко і Гейна-Сарсона. У лабораторній роботі акцент робиться на застосуванні нотації Йордана-ДеМарко.

Декомпозиція діаграм потоків даних здійснюється на основі процесів: кожний процес розкривається за допомогою діаграм потоків даних нижнього рівня. Діаграма вищого (нульового) рівня діаграм потоків даних — контекстна діаграма, яка відображає взаємодію розроблюваної системи із зовнішніми системами і встановлює границі аналізованої системи.

Таблиця 2.1.

Компоненти діаграм даних за нотаціями Йордана-ДеМарко та Гейна-Сарсона

Компоненти	Нотація Йордана-ДеМарко	Нотація Гейна-Сарсона
------------	-------------------------	-----------------------

Потік даних	<div> <div>Назва</div> <div>→</div> </div>	<div> <div>Назва</div> <div>→</div> </div>
Процес	<div> <div>Назва</div> <div>номер</div> </div>	<div> <div>Назва</div> <div>Номер</div> </div>
Сховище даних (накопичувач)	<div> <div>Назва</div> </div>	<div> <div>Назва</div> </div>
Зовнішня сутність	<div> <div>Назва</div> </div>	<div> <div>Назва</div> </div>

Діаграма має зіркоподібну топологію, в центрі якої перебуває головний процес, з'єднаний із приймачами і джерелами інформації, за допомогою яких із системою взаємодіють користувачі та інші системи (рис. 2.4).



Рис. 2.4. Приклад діаграми потоків даних

Після опису основного процесу виконується декомпозиція, тобто визначаються процеси, із яких складається основний процес. Далі процеси діляться на підпроцеси і так до досягнення необхідного рівня деталізації. Так будується ієрархія діаграм потоків даних.

Головна мета побудови ієрархії діаграм потоків даних полягає в тому, щоб зробити вимоги зрозумілими й чіткими на кожному рівні деталізації.

У процесі побудови ієрархії діаграм потоків даних необхідно дотримуватися таких правил:

а) балансування — при деталізації процесу дочірня діаграма міститиме лише ті компоненти інформаційних потоків, із якими має інформаційний зв'язок відповідний процес на батьківській діаграмі;

б) нумерації — при деталізації процесів слід підтримувати їх ієрархічну нумерацію;

в) правило семи — для того, щоб діаграма була «читабельною», на діаграмі не повинно бути більше семи процесів.

Звіт з лабораторної роботи повинен складатися з:

1. Постановки завдання.
2. Документа «Ескізний проект», що містить:
 - вибір методу рішення і мови програмування;
 - специфікації процесів;
 - всі отримані діаграми;
 - словник термінів.

Контрольні питання

1. Які особливості структурного аналізу як методу аналізу вимог?
2. Наведіть перелік основних компонентів діаграм потоків даних та розкрийте їх призначення.
3. Які існують нотації для представлення діаграм потоків даних?
4. Що представляє собою ієрархія діаграм потоків даних? Які правила її побудови існують?
5. Які інструментальні засоби існують для побудови діаграм потоків даних?
6. Назвіть етапи розробки програмного забезпечення.
7. Що таке життєвий цикл програмного забезпечення?
8. У чому полягає постановка задачі і передпроектні дослідження?
9. Назвіть функціональні і експлуатаційні вимоги до програмному продукту.
10. Перелічіть складові ескізного проекту.
11. Охарактеризуйте специфікації і моделі.

Література: [2, 3, 10].

10. ЛАБОРАТОРНЕ ЗАНЯТТЯ 2.3

СТРУКТУРНИЙ ПІДХІД ДО ПРОГРАМУВАННЯ. СТАДІЯ «ТЕХНІЧНИЙ ПРОЕКТ».

АНАЛІЗ ВИМОГ ЗА ДОПОМОГОЮ ТЕХНОЛОГІЙ IDEFX

Мета: дослідити процес аналізу вимог за допомогою технологій IDEFX та набути практичних навичок в застосуванні інструментальних засобів Microsoft Visio та BPWin при побудові функціональної моделі.

Завдання

1. Вивчити особливості застосування технологій IDEF0 та IDEF3.
2. Обрати варіант використання системи, що аналізується, і провести аналіз бізнес-процесів, використовуючи технологію IDEF0.
3. Побудувати контекстну діаграму та дочірню діаграму 1-го рівня в нотації IDEF0, здійснити їх текстовий опис.
4. Обрати один із функціональних блоків діаграми IDEF0 та побудувати діаграму декомпозиції цього блоку в нотації IDEF3, навести її текстовий опис.

Основні теоретичні відомості

IDEF0. Основні поняття IDEF0

IDEF0 (Integrated Definition Function Modeling) – методологія функціонального моделювання. В основі IDEF0 методології лежить поняття блоку, який відображає деяку бізнес.

Технологія IDEF0 — технологія аналізу системи в цілому як набору зв'язаних між собою дій або функцій. Вона призначена для створення функціональної моделі, яка відображає структуру і функції системи, а також потоки інформації і матеріальних об'єктів, що зв'язують ці функції.

IDEF0 поєднує в собі невелику за об'ємом графічну нотацію із суворими і чітко визначеними рекомендаціями, призначеними для побудови якісної і зрозумілої моделі системи.

Згідно з технологією IDEF0, процес, що аналізується, зображується у вигляді сукупності множини взаємопов'язаних дій (робіт), які взаємодіють між собою на основі визначених правил, з урахуванням інформаційних, людських і виробничих ресурсів, що споживаються, які мають чітко визначений вхід і вихід.

Основні елементи нотації IDEF0 такі (рис. 2.5):

- функціональні блоки;
- стрілки.

Функціональні блоки відображають роботу (процес, діяльність, функцію або задачу), яка має фіксовану мету і приводить до деякого кінцевого результату. Графічно функціональні блоки зображуються у вигляді прямокутників.



Рис. 2.5. Типи стрілок функціонального блока

Всередині кожного блока поміщається його назва і номер. Назва повинна відображати певну дію. Номери блоків використовуються для їх ідентифікації на діаграмі і у відповідному тексті.

Взаємодія робіт між собою і зовнішнім світом описується у вигляді стрілок: входу, виходу, управління і механізму.

Стрілка входу відображає вхідні документи, матеріальні та інформаційні ресурси, необхідні для виконання процесу.

Стрілка виходу відображає вихідні документи, матеріальні та інформаційні ресурси, що є результатом виконання процесу.

Стрілка управління відображає правила, обмеження та інші управляючі впливи. Стрілка механізму виконання відображає ресурси, необхідні для виконання функції (людина, обладнання, автоматизована система).

Існує п'ять основних видів комбінованих стрілок: вихід-вхід, вихід-управління, вихід-механізм, вихід-зворотний зв'язок на управління, вихід-зворотний зв'язок на вхід.

Модель IDEF0 завжди починається з зображення системи як єдиного цілого — з контекстної діаграми. Ця діаграма становить собою загальний опис системи і її взаємодію з зовнішнім середовищем. Контекстна діаграма складається із одного блока, що описує функцію верхнього рівня, її входи, виходи, управління і механізми. В кожній моделі може бути лише одна контекстна діаграма.

Принципи моделювання в IDEF0

В IDEF0 реалізовані три базових принципа моделювання процесів:

- принцип функціональної декомпозиції;
- принцип обмеження складності;
- принцип контекста.

Принцип функціональної декомпозиції представляє собою спосіб моделювання типової ситуації, коли будь-яка дія, операція, функція можуть бути розбиті на більш простіші дії, операції, функції.

Принцип обмеження складності. При роботі з IDEF0 діаграмами суттєвою є умова їх зрозумілості та зручності у читанні. Суть принципу – кількість боків не менша двох і не більше шести.

Принцип контекстної діаграми. Моделювання ділового процесу починається з побудови контекстної діаграми. На ній зображають один блок – головна бізнес-функція, яка моделюється системою.

При призначенні головної бізнес-функції необхідно завжди мати ціль моделювання і точку зору на модель. Одне й те ж підприємство може бути описане по-різному в залежності з якої точки зору його розглядають: директор підприємства і податковий інспектор бачать по-різному.

Після опису основної функції виконується функціональна декомпозиція, тобто визначаються функції, із яких складається основна (рис. 2.6). При цьому кожний функціональний блок може бути декомпозований, тобто його можна зобразити у вигляді сукупності інших взаємопов'язаних функціональних блоків, які детально описують вихідний блок. Кожна діаграма зазвичай містить 3-5 блоків, розміщених за «ступінчатою» схемою відповідно до їх домінування, яке розуміється як вплив, що здійснюється одним блоком на інший.

Технологія IDEF3 — технологія моделювання і стандарт документування процесів, які відбуваються у системі. IDEF3 показує у причинно-наслідкові зв'язки між подіями, функціями, використовуючи структурний метод передачі знань про те, як функціонує система чи процес. Ця технологія доповнює технологію IDEF0.

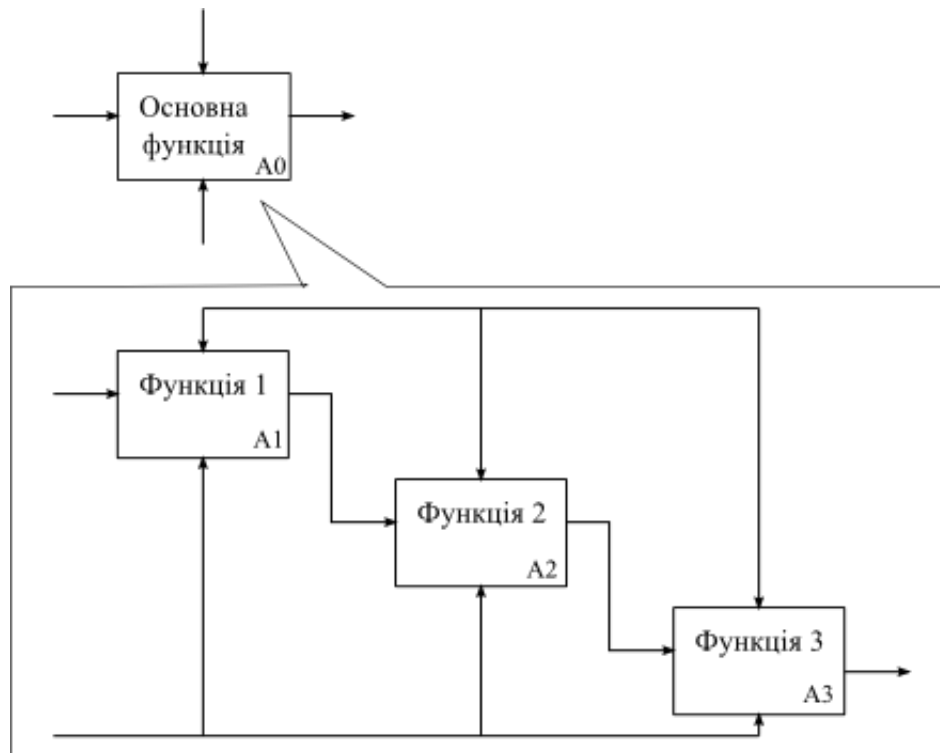


Рис. 2.6. Принцип декомпозиції

За допомогою графічної нотації IDEF3 описується логіка виконання робіт, черговість їх запуску і завершення. Нотація IDEF3 частіше застосовується для побудови процесів нижнього рівня, крім того її можна використовувати при декомпозиції блоків процесу IDEF0. Основні елементи нотації такі (табл. 1.2):

- одиниці роботи (функції);
- об'єкти посилань;
- перехрестя;
- зв'язки.

Одиниця роботи є центральним компонентом моделі, яка використовується для опису роботи (функції).

Об'єкт посилання — компонент моделі, що використовується для опису посилань на інші діаграми моделі, циклічні переходи в рамках однієї моделі, різні коментарі до функцій.

Перехрестя — компоненти моделі, що використовуються для відображення логіки взаємодії стрілок при злитті і розгалуженні або для відображення множини подій, які можуть або повинні бути завершені перед початком наступної роботи. Існують такі типи перехрестя: «І», «АБО», «АБО, що виключає», які визначають зв'язки між роботами (функціями) в рамках процесу.

Зв'язки компоненти, що відображають взаємовідношення між роботами (функціями), В IDEF3 існує три типи стрілок, що зображують зв'язки:

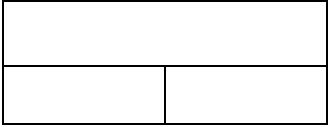

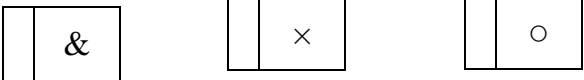
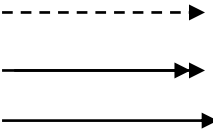
1) старша (стрілка передування) — суцільна лінія, що пов'язує одиниці робіт (функції);

2) відношення — пунктирна лінія, яка використовується для зображення зв'язків між одиницями робіт (функціями), а також між одиницями робіт і об'єктами-коментарями;

3) потоки об'єктів — стрілка з двома наконечниками, що показує потік об'єктів від однієї одиниці роботи (функції) до іншої.

Таблиця 2.2.

Зображення елементів нотації IDEF3

Назва	Графічне зображення
Одиниця роботи (функція)	
Об'єкт посилення	
Перехрестя	
Зв'язки	

Контрольні питання

- 1) Опишіть особливості аналіз вимог за допомогою технології IDEF0.
- 2) Які основні елементи нотації IDEF0?
- 3) Аналіз вимог за допомогою технології IDEF3.
- 4) Які особливості графічної нотації технології IDEF3?
- 5) Яка різниця між технологіями IDEF0 та IDEF3?
- 6) Які переваги та недоліки застосування технологій IDEF0 та IDEF3?

Література: [11, 12].

11. ЛАБОРАТОРНА РОБОТА 2.4

СПЕЦИФІКАЦІЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Мета – дослідити процес створення специфікації функціональних вимог до програмного забезпечення та набути практичних навичок у виділенні та документуванні вимог.

Завдання

1. Ознайомитися з особливостями створення специфікації вимог до програмного забезпечення.
2. Ознайомитися із базовим змістом стандарту IEEE 830.
3. Дослідити предметну галузь, задану згідно з варіантом та виділити функціональні та нефункціональні вимоги до програмного забезпечення.
4. Створити специфікацію функціональних вимог до програмного забезпечення на основі шаблону, що рекомендований стандартом IEEE 830.

Основні теоретичні відомості

Специфікація вимог до програмного забезпечення (Software Requirements Specification) — процес формалізованого опису функціональних і нефункціональних вимог, вимог до характеристик якості, які відпрацьовуватимуться на процесах життєвого циклу (ЖЦ) програмного забезпечення.

У специфікації вимог відображається структура програмного забезпечення, вимоги до функцій, якості і документації, а також задається архітектура системи і програмного забезпечення, алгоритми, логіка керування і структура даних.

Специфікуються також системні вимоги, нефункціональні вимоги і вимоги до взаємодії з іншими компонентами і платформами (БД, СКБД маршаллінг даних, мережа тощо).

Під час виконання процесу створюється документ («специфікація вимог до програмного забезпечення»), в якому чітко описуються всі особливості й можливості продукту, а також необхідні обмеження.

Специфікацію складають як представники розробника, так і представники замовника або представники обох сторін (що рекомендується).

Специфікація вимог до програмного забезпечення необхідна замовникам, менеджерам проекту, команді розробників програмного забезпечення, тестувальникам, спеціалістам із обслуговування і підтримки, укладачам документації та ін.

Створення специфікації вимог до програмного забезпечення регламентується чинним стандартом IEEE 830-1998 «Recommended Practice for Software Requirements Specification».

Шаблон специфікації вимог до програмного забезпечення, відповідно до зазначеного стандарту, складається із таких розділів: вступу, загального опису, зображення функціональних та нефункціональних вимог, додатків.

1. Вступ. У розділі описується структура і принцип використання специфікації вимог до програмного забезпечення.

1.1. Призначення. Визначається продукт або система, вимоги для якого/якої зазначаються в документі, в тому числі редакція або номер випуску.

1.2. Угоди, прийняті в документах. Описуються всі стандарти, прийняті в документі: типографічні стандарти, що характеризують стилі тексту, особливості виділення або зауваження.

1.3. Передбачувана аудиторія і рекомендації з читання. Зазначаються користувачі, для яких призначена дана специфікація, та рекомендації для кожного класу читачів із приводу послідовності читання документа; описується зміст документа і його структура.

1.4. Границі проекту. Коротко описується програмне забезпечення та його призначення. Зазначається зв'язок майбутнього продукту з користувачами або бізнес-цілями і стратегіями підприємства замовника.

1.5. Посилання. Наводиться перелік усіх документів або ресурсів, на які здійснюється посилання в специфікації.

2. Загальний опис. У розділі розглядається продукт та середовище, в якому він застосовуватиметься, аудиторія користувачів, а також відомі обмеження, припущення і залежності.

2.1. Загальне бачення продукту. Здійснюється опис змісту і походження продукту.

2.2. Особливості (функціональність) продукту. Наводиться перелік головних функцій та особливостей продукту.

2.3. Класи і характеристики користувачів. Наводяться класи користувачів, які працюватимуть із продуктом, визначаються привілейовані користувачі і здійснюється опис їх відповідних характеристик.

2.4. Середовище функціонування продукту (операційне середовище). Здійснюється опис робочого середовища програмного забезпечення, включаючи апаратні засоби, операційні системи і їх версії, а також географічне місцезнаходження користувачів, серверів та баз даних. Крім того, зазначається перелік усіх компонентів програмного забезпечення або застосувань, із якими система повинна бути сумісна.

2.5. Обмеження, правила та стандарти. Зазначаються обмеження можливостей, доступних розробникам (технології, засоби, мови програмування і бази даних, які варто використовувати або уникати; типи і версії встановлених веб-браузерів; зворотна сумісність із продуктами, випущеними раніше, обмеження, пов'язані з обладнанням.

2.6. Документація для користувачів. Зазначаються всі компоненти користувацької документації, що постачаються разом із програмним забезпеченням (інструкції для користувачів, онлайн-довідки і навчальні програми).

2.7. Припущення і залежності. Наводиться опис залежностей проекту від зовнішніх факторів, як то дата випуску наступної версії операційної системи або випуск промислового стандарту, своєчасність поставки компонента.

3. Функціональність системи. У розділі описуються всі вимоги до програмного забезпечення на тому рівні деталізації, що є достатнім, щоб дати розробникам можливість створювати продукт (систему), який/яка задовольнить цим вимогам.

3.1. Функціональний блок X (таких блоків може бути декілька).

3.1.1. Опис та пріоритет (короткий опис функції із зазначенням її пріоритету: високий, середній, низький).

3.1.2. Причинно-наслідкові зв'язки, алгоритми (опис послідовностей впливів, що чиняться на систему і відгуки системи, що визначають реакцію конкретної функції).

3.1.3 Функціональні вимоги (деталізація функції «X», тобто перелічення деталізованих функціональних вимог, включаючи реакцію на очікувані помилки й неправильні дії).

4. *Вимоги до зовнішніх інтерфейсів.* Опис обладнання, програмного забезпечення або елементів бази даних, із якими система (компонент) повинна взаємодіяти.

4.1. *Інтерфейси користувачів.* Опис логічних характеристик кожного користувацького інтерфейсу: посилання на стандарти графічного інтерфейсу користувачів, стандарти шрифтів, значків, назв кнопок, зображень, кольорових схем, послідовність полів вкладок, часто використовуваних елементів управління; конфігурація екранів; стандарти відображення повідомлення; спеціальні можливості для користувачів із вадами зору.

4.2. *Апаратні інтерфейси.* Опис характеристик кожного інтерфейсу між компонентами програмного забезпечення і обладнанням системи (типи пристроїв, що підтримуються, взаємодія даних і елементів управління між

програмним забезпеченням і обладнанням, а також протоколи взаємодії, які використовуватимуться).

4.3 Програмні інтерфейси. Опис з'єднань продукту та інших компонентів програмного забезпечення, в тому числі бази даних, операційної системи, бібліотеки; опис служб, необхідних зовнішнім компонентам програмного забезпечення, і природи взаємодії між компонентами.

4.4 Інтерфейси передачі інформації. Опис характеристик різноманітних інтерфейсів зв'язку: електронна пошта, веб-браузери, протоколи мережевого з'єднання та особливості шифрування, частота передачі даних.

5. Інші нефункціональні вимоги

5.1 Вимоги до продуктивності (робочі характеристики). Опис статичних (кількість одночасно підтримуваних користувачів та терміналів, обсяг і тип інформації, що обробляється) та динамічних вимог (кількість транзакцій і завдань, обсяг даних, що обробляються в певний період часу в умовах як нормального, так і пікового навантаження), що висуваються в цілому до програмного забезпечення або до взаємодії користувача з програмою.

5.2 Проектні обмеження. Опис обмежень, що накладаються стандартами, апаратним забезпеченням.

5.3 Атрибути системи:

а) *надійність* — чинники, необхідні для установки необхідного рівня надійності програмного забезпечення;

б) *доступність* — чинники, що гарантують визначений рівень доступності системи, такі як контрольні точки, відновлення і пере- запуск;

в) *безпека* — чинники, що захищають програмне забезпечення від випадкового і зловмисного доступу використання, модифікації, руйнування та розголошення;

г) *супроводжуваність* — атрибути, пов'язані з легкістю підтримки програмного забезпечення;

д) *перенесення* — наводяться атрибути програмного забезпечення, пов'язані з легкістю перенесення програмного забезпечення на інші комп'ютери і/або операційні системи.

ДОДАТОК А. Словник термінів. Роз'яснення термінів, які необхідно знати користувачеві для правильного розуміння специфікації.

ДОДАТОК Б. Моделі аналізу. Наводиться графічне зображення діаграм: потоків даних, класів, діаграм «сутність-зв'язок», IDEF0, IDEF3 тощо.

Функціональні вимоги — вимоги, що описують поведінку системи й сервіси (функції), які вона виконує, та залежать від типу системи, що розробляється, та потреб користувачів. Тобто функціональні вимоги визначають фундаментальні операції, які повинні виконуватися програмним забезпеченням при прийнятті і обробці вхідних даних, обробці й генерації вихідних даних. Вони зазвичай починаються зі слів «Система повинна...».

Функціональні вимоги включають:

- а) перевірку достовірності по входах;
- б) точну послідовність операцій;
- в) відгуки на нестандартні, помилкові ситуації,
- г) зв'язок виходів із входами, включаючи послідовності введення/виведення;
- д) формули для перетворення введення-виведення.

Нефункціональні вимоги — вимоги, що відображають обмеження, пов'язані з функціонуванням системи. Серед нефункціональних вимог виділяють вимоги до зовнішніх інтерфейсів та продуктивності, проектні обмеження та атрибути.

Відповідно до стандарту IEEE 830, нефункціональні вимоги подаються у четвертому та п'ятому розділі шаблону специфікації вимог до програмного забезпечення.

Контрольні запитання та завдання

- 1) Які існують способи зображення вимог?
 - 2) Яка роль функціональних вимог? Наведіть приклади функціональних вимог.
 - 3) Що таке специфікація вимог до програмного забезпечення? Розкрийте особливості її створення.
 - 4) Які державні та закордонні стандарти документування вимог існують?
 - 5) Яке значення має специфікація вимог до програмного забезпечення для процесу розробки, його учасників і зацікавлених сторін?
 - 6) Опишіть базовий зміст специфікації вимог.
 - 7) Розкрийте особливості зображення функціональних вимог у специфікації вимог до програмного забезпечення.
-
- 1) Які існують характеристики правильно складеної специфікації вимог?
 - 2) Які існують критерії повноти специфікації вимог?
 - 3) Що таке нефункціональна вимога?

- 4) Яке призначення нефункціональних вимог? Наведіть приклади.
- 5) Як ранжируються вимоги?
- 6) Як забезпечується модифікація вимог?

Література: [1], [2], [6].

12. ЛАБОРАТОРНА РОБОТА 2.5

ВСТАНОВЛЕННЯ МЕТРИК, ОРІЄНТОВАНИХ НА ВИМОГИ

Мета: визначити метрики, що орієнтовані на вимоги, та набути практичні навички з підрахунку їх значень.

Завдання

1. Встановити метрики, орієнтовані на вимоги.
2. Дослідити особливості підрахунку значень метрик, орієнтованих на вимоги.
3. Визначити значення метрик, орієнтованих на вимоги, за наведеними формулами.
4. Побудувати матрицю відповідності функціональних вимог до програмного продукту і підготованих тестових сценаріїв.

Основні теоретичні відомості

Метрика, орієнтована на вимоги — це міра, що дозволяє отримати числове значення деяких властивостей вимог.

Метрики, що орієнтовані на вимоги, дають можливість контролювати специфікації вимог, зміни вимог, а також степінь їх задоволення. Серед основних метрик, орієнтованих на вимоги, виділяють такі:

- 1) стабільність вимог (*requirement stability*),
- 2) рух вимог (*requirements creep*);
- 3) відповідність вимогам (*requirement conformance*).

Стабільність вимог — метрика, що відображає коефіцієнт стабільності вимог за той періоду часу, що пройшов від моменту взаємного погодження вимог та затвердження базової версії специфікації вимог і до даного моменту. Вона вимірюється індексом стабільності вимог *requirement stability index (RSI)*], що є показником, який використовується для відслідковування внесених змін (додавання, редагування, видалення) в початково встановлені вимоги.

Для визначення індексу стабільності вимог (RSI) використовуються формули (2.1) та (2.2).

$$RSI = \frac{(N - |C|)}{N},$$

(2.1)

де N — кількість визначених і затверджених вимог до проекту; C — кількість змін, що пропонується внести до вимог.

Значення індексу, що розраховується за формулою (2.1), має бути наближеним до одиниці.

$$RSI = \frac{(N_{пв} + N_{в} + N_{дв} + N_{вв})}{N_{пв}}, \quad (2.2)$$

де $N_{пв}$ — загальна кількість первинних вимог; $N_{в}$ — загальна кількість вимог, які потрібно змінити; $N_{дв}$ — загальна кількість доданих вимог; $N_{вв}$ — загальна кількість видалених вимог.

Рух вимог — метрика, що відображає коефіцієнт розгалуження (повзучості) вимог. Обчислюється за такою формулою:

$$RC = \frac{N_{дод}}{N} \cdot 100, \quad (2.3)$$

де $N_{дод}$ — загальна кількість доданих вимог; N — кількість визначених і початково затверджених вимог. Значення метрики має бути наближеним до нуля.

Відповідність вимогам — метрика, що дає можливість контролювати специфікації, зміни вимог, а також ступінь їх задоволення. Оцінка відповідності — діяльність, пов'язана з визначенням, прямо або опосередковано, того, що відповідні вимоги виконуються.

Типовими прикладами діяльності з оцінки відповідності є випробування та тестування програмного продукту. Тести відповідності призначені для надання користувачам продуктів деяких гарантій або упевненості в тому, що продукт працює так, як очікувалося, виконує функції відомим способом або має відомий інтерфейс або формат.

Для визначення ступеня відповідності вимогам будується **матриця відповідності**. Вона становить собою двомірну таблицю, яка містить відповідні функціональні вимоги до програмного продукту і підготовані тестові сценарії. У заголовках колонок таблиць розміщуються вимоги, а в заголовках рядків — тестові сценарії. На перехресті — позначка, яка говорить про те, що вимога поточної колонки покрита тестовим сценарієм поточного рядка.

Контрольні запитання та завдання

- 1) Дайте визначення поняття «метрика».
- 2) Які існують метрики, орієнтовані на вимоги?
- 3) Як визначити індекс стабільності вимог?
- 4) Як визначити метрику «відповідність вимогам»?

Література: [1], [2], [4], [8].

13. ЛАБОРАТОРНА РОБОТА 2.6

ЗАСОБИ КЕРУВАННЯ ТА ВІДСТЕЖЕННЯ ВИМОГ

Мета – дослідити особливості керування та відстеження вимог програмними засобами.

Завдання

1. Створити новий проект у програмному засобі керування та відстеження вимог (на вибір студента).
2. Створити нові типи вимог (функціональні, нефункціональні) та визначити для них атрибути.
3. Створити новий документ і додати до нього функціональні та нефункціональні вимоги.
4. Створити перегляд вимог.
5. Здійснити додавання нових, редагування та видалення існуючих вимог, налаштування їх атрибутів.
6. Провести фільтрацію вимог за пріоритетами та підрахувати кількість вимог у проекті, що мають високий та середній пріоритет.

Основні теоретичні відомості

Управління (керування) вимогами — процес, що включає ідентифікацію, виявлення, документацію, аналіз, відстеження, пріоритетність вимог, досягнення угод за вимогами, управління змінами та повідомлення зацікавлених осіб.

Для керування та відстеження вимог існують програмні засоби: IBM Rational RequisitePro, Telelogic DOORS, Borland Caliber RM, IBM Rational Requirements Composer, Requirements Hub.

Наприклад, створення проекту під назвою “Система документообігу” та у полі “опис проекту” вказано: “Система документообігу для юридичної компанії” у засобі Requirements Hub зображено на Рис.2.7.

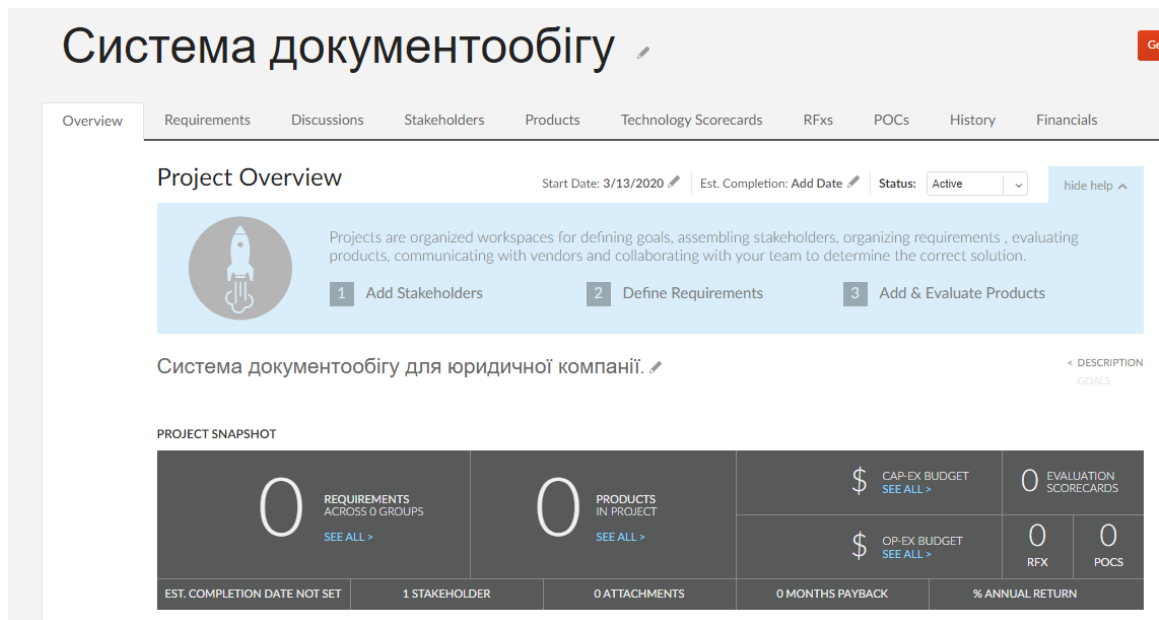


Рис.2.7. Вкладка “Overview” створеного проекту з розробки ПЗ

Можливість вибору вимог до категорії продукту або ж створення нової категорії представлена на Рис. 2.8. В свою чергу, для обраної категорії можна створювати групи категорій. Так, було створено нову категорію: “Автоматизована система” та нові групи: функціональні та нефункціональні вимоги (Рис.2.9.).

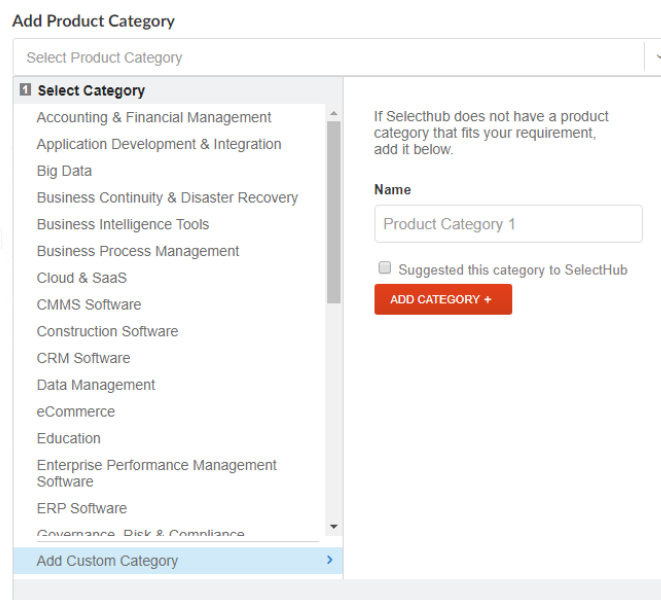


Рис. 2.8. Наявні категорії

AUTOMATED SYSTEM

ADD GROUP +

FUNCTIONAL REQUIREMENTS

ADD SUB-GROUP + ADD REQUIREMENT +

Requirement	Your Priority	Team Priority
This group is empty. You can add requirements or sub-groups above.		

NON-FUNCTIONAL REQUIREMENTS

ADD SUB-GROUP + ADD REQUIREMENT +

Requirement	Your Priority	Team Priority
This group is empty. You can add requirements or sub-groups above.		

Рис. 2.9. Створені категорія та групи вимог

У засобі Requirements Hub є можливість вказати такі атрибути як:

1. назва вимоги;
2. опис до вимоги;
3. пріоритет вимоги.

Даний функціонал відображено на Рис. 2.10:

FUNCTIONAL REQUIREMENTS

ADD SUB-GROUP + ADD REQUIREMENT +

Requirement

Describe the requirement

Your Priority

How important is this requirement to you? The votes from you and other stakeholders are averaged into the "Team Priority."

N/A

Low

Med

High

Override Team Priority

Description

Further explain the requirement

CANCEL

ADD REQUIREMENT +

Рис. 2.10. Створення нових вимог

До списку вимог було додано певні нові вимоги, їх було поміщено в нові підкатегорії під назвою «Newly Created» (Рис. 2.11).

Newly Created			ADD REQUIREMENT +
Requirement	Your Priority	Team Priority	
Використання СУБД SQL Server	Med	Med (1)	
Використання шаблону MVC при створенні архітектури системи	High	High (1)	
Проведення перевірки нових файлів через антивірус	Low	Low (1)	

Рис. 2.11 Нові функціональні вимоги

Requirements Hub надає можливість змінювати назву, опис та пріоритет створених вимог. Так, зміни даних параметрів було проведено для вимоги “Використання СУБД SQ (Рис. 2. 12).

Requirement <div>Використання СУБД SQL Server</div>	Your Priority How important is this requirement to you? The votes from you and other stakeholders are averaged into the "Team Priority." N/A Low Med High <div></div> <input type="checkbox"/> Override Team Priority
Description <div>SQL Server має тісну підтримку .Net Framework, що робить його найраціональнішим варіантом для розроблюваного ПЗ</div>	
<div>CANCEL</div>	<div>SAVE CHANGES</div>

Рис. 2.12 Зміни у вимозі “Використання СУБД SQL Server”

Література: [1], [2], [13].

14. ЛАБОРАТОРНА РОБОТА 2.7

ТЕСТУВАННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Мета – дослідити методи тестування вимог до програмного забезпечення.

Завдання

1. Здійснити перевірку документації, створеної у лабораторній роботі 2.4, за допомогою checklist.
2. Здійснити аналіз поведінки системи за допомогою достатнього покриття системи тест-кейсами та юз-кейсами, як з позитивними так і з негативними сценаріями.

Основні теоретичні відомості

Тестування вимог дозволяє зменшити кількість допрацювань і змін. Перед початком тестування вимог необхідними є наступні умови: вимоги проаналізовані й задокументовані, тобто є щось, з чим можемо працювати. Аналітик, як роль, самостійно проводить певну оцінку та перевірку цих вимог. Під час тестування вимог відбувається наступний процес: всі зацікавлені особи підтверджують, що вимоги коректні, зрозумілі і достатні для того, щоб розпочинати роботу. А користувачі підтверджують, що вимоги відображають їхні потреби в роботі. Основним критерієм готовності вимог є те, чи містять вимоги достатньо інформації для того, щоб розпочинати розробку. І важливо відзначити, що і замовник, і користувачі, і проектна команда, якою планується робота над проектом, — всі беруть участь у тестуванні вимог і враховують різні пов'язані з тестуванням вимог аспекти.

Розрізняють три методи тестування вимог:

1. *Перевірка документації*: дозволяє перевірити вимоги на правильність, повноту, відслідковуваність, важливість, зрозумілість, однозначність та вимірюваність. Найпоширенішим способом є створення формального checklist, що містить перелік параметрів, аспектів, критеріїв тощо — всього, на що необхідно перевіряти вимоги.

2. *Аналіз поведінки системи*: метод формалізації події, тобто формування вимог у форматі «вхід — вихід», «подія — наслідок», «умова — відповідь». Щось подається на вхід до системи, з системою щось відбувається, і вона має дати щось на виході. Найпоширеніший спосіб, у який це відбувається, — це або тест-кейси, або юз-кейси. Тест-кейси готують тестувальники, юз-кейси — аналітики. Приклад специфікації тест-кейсу наведено у табл. 2.5, а юз-кейсу у табл. 2.6.

Стандартом ДСТУ ISO/IEC 12119:2003 встановлено в якості мінімального базового елемента тестування тестовий варіант, який є документованою інструкцією, що містить мету тесту, функції що тестуються, середовище тестування, дані, процедуру та очікувані результати тесту. Визначена сукупність тестових варіантів формує план тестування. Тестові дії передбачають виконання тестів в тестовому середовищі відповідно до плану тестування та отримання результатів тестування, в тому числі і негативних, тобто таких, що не відповідають очікуваним. Результати тестування оформлюються як тестові записи.

Таблиця 2.5.

Специфікація тестового варіанта	
Назва взаємодіючих класів: TCommandQueue, TCommand	Назва тесту: TCommandQueueTest1
Опис тесту: тест перевіряє можливість створення об'єкта типу TCommand та додавання його в чергу при визові методу AddCommand	
Початкові умови: черга команд пуста	
Очікуваний результат: у чергу буде додана одна команда	
Отриманий результат: в чергу додана команда	

Таблиця 2.6. Приклад юз-кейсу «Розблокувати обліковий запис користувача»

Розблокувати обліковий запис користувача

Діючі особи	Адміністратор, система
Мета	Змінити статус облікового запису користувача на «активний»
Передумова	Обліковий запис користувача не активний
Успішний сценарій: <ol style="list-style-type: none"> Адміністратор вибирає користувача і активує «Розблокувати»; Система переключає обліковий запис користувача у статус «активний» і 	

надсилає повідомлення користувачу на email.	
Результат	Обліковий запис користувача був переведений в статус «активний»

3. *Прототипування*: застосовується для перевірки вимог на повноту, правильність, реалізовуваність і на зручність використання користувацького інтерфейсу (якщо говорити про розробку інформаційної системи). Є різні приклади прототипів: *горизонтальний та вертикальний*. У першому, будується модель якомога більшої кількості спектрів системи, тоді як у другому — докладно прототипується певна вузько спрямована модель або функція системи.

Література: [1], [3], [9].

СПИСОК ЛІТЕРАТУРИ

1. *Леффингуэлл Д.* Принципы работы с требованиями к программному обеспечению. Унифицированный подход / Д. Леффингуэлл, Д. Уиндрилг. – М.: «Вильямс», 2002. – 448с.
2. *Вигерс К.* Разработка требований к программному обеспечению / К. Вигерс. – М.: «Русская редакция», 2004. – 576 с.
3. *Халл Э.* Разработка и управление требованиями. Практическое руководство пользователя / Э. Халл, К. Джексон, Д. Дик. Лондон.: Спрингер Сайнс, 2005. – 229 с.
4. *Кармайл Э.* Быстрая и качественная разработка программного обеспечения / Э. Кармайл, Д. Хейвуд. – М.: «Вильямс», 2009. – 263с.
5. IEEE Standard for Developing Software Life Cycle Processes: IEEE Std. 1074-1997. – N.Y.: The Institute of Electrical and Electronics Engineers, 1997. – 88 p.
6. IEEE Recommended Practice for Software Requirements Specifications: IEEE Std. 830-1998. – N.Y.: The Institute of Electrical and Electronics Engineers, 1998. – 31 p.
7. *Кватрани Т.* Визуальное моделирование с помощью Rational Rose 2002 и UML – Пер. с англ. – М.: «Вильямс», 2003. – 192 с.
8. *Кендал С.* UML, основные концепции / С. Кендал. – М.: «Вильямс», 2009. – 275 с.
9. *Брауде Э.* Технология разработки программного обеспечения / Э. Брауде. – СПб.: Питер, 2004. – 655 с.
10. Data-Flow-Diagram-Notations [Электронный ресурс]. – Режим доступа: <http://www.smartdraw.com/resources/tutorials>.
11. IDEF0 Function Modeling Method. [Электронный ресурс]. – Режим доступа: <http://www.idef.com/IDEF0.htm>
12. IDEF3 Process Description Capture Methodod. [Электронный ресурс]. – Режим доступа: <http://www.idef.com/IDEF3.htm>
13. Using Rational RequisitePro. [Электронный ресурс]. – режим доступа <http://pic.dhe.ibm.com/infocenter/rqmhelp/v2r0>.
14. *Ноэл М.* Microsoft SharePoint 2010. Полное руководство / М. Ноэл, К. Спенс. – М.: «Вильямс», 2011. – 800 с.
15. <https://habr.com/ru/post/245625/>
16. <https://evergreens.com.ua/ru/articles/checklist-tz.html>
17. http://foranalysts.blogspot.com/2011/08/blog-post_17.html
18. <https://www.youtube.com/watch?v=CiIUNg49fgw>
19. <https://www.youtube.com/watch?v=c8bmvwWx80Q>

20. <https://habr.com/ru/post/215837/>
21. <https://dou.ua/lenta/articles/use-cases/>
22. <https://systems.education/use-case/>
23. <https://blog.business-analyst.info/2009/12/25/use-cases-for-use-cases/>
24. http://www.dsc.ufcg.edu.br/~garcia/cursos/ger_processos/seminarios/Crystal/Use%20Cases,%20Ten%20Years%20Later.htm
25. <https://habr.com/ru/post/231961/>
26. <https://www.youtube.com/watch?v=XR33OQUv17U>
27. <https://www.youtube.com/watch?v=ZQI0vrB7Fok>
28. Марка Д., МакГоуен К. Методологія структурного аналізу і проектування. - М.: МетаТехнологія, 1993.
29. ГОСТ 34.601-90. Інформаційна технологія. Автоматизовані системи. Стадії створення.
30. Білі сторінки MSF. <http://www.microsoft.com/rus/msdn/msf>