МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет кораблебудування
імені адмірала Макарова

О. М. ДУДЧЕНКО, С. О. КАРПОВА

ОСНОВИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Методичні вказівки
до виконання лабораторних робіт

Рекомендовано Методичною радою НУК

Миколаїв • НУК • 2019

УДК 004.4(076)
Д 81

Автори: О. М. Дудченко, канд. техн. наук, професор НУК;
С. О. Карпова, старший викладач

Рецензент П. С. Носов, канд. техн. наук, доцент

Рекомендовано Методичною радою НУК

Дудченко О. М.

Д 81 Основи програмної інженерії : методичні вказівки до виконання лабораторних робіт / О. М. Дудченко, С. О. Карпова. – Миколаїв : НУК, 2019. – 84 с.

Метою циклу лабораторних робіт є практичне ознайомлення студентів з основами програмування, а також з базовими концепціями програмної інженерії. Після вивчення даної дисципліни студенти повинні оволодіти основними навика ми моделювання та проектування програмних продуктів, а також розуміти основні вимоги до розробки програмного забезпечення, до програмної архітектури та тестування програмних продуктів.

Призначено для студентів І курсу спеціальності 121 "Інженерія програмного забезпечення" Херсонської філії НУК.

УДК 004.4(076)

© Дудченко О. М., Карпова С. О., 2019
© Національний університет кораблебудування імені адмірала Макарова, 2019

ВСТУП

Розробка та використання комп'ютерних програм у теперішній час стало масовим явищем. Практично немає жодної сфери діяльності (економіка, медицина, бізнес, комерція, промисловість і так далі), де б програмне забезпечення (ПЗ) не використовувалося для автоматизації і поліпшення цієї діяльності.

Програмна інженерія (Software Engineering) є галуззю Computer science, яка вивчає питання побудови комп'ютерних програм, відображає закономірності її розвитку, узагальнює досвід програмування у вигляді комплексу загальних знань і правил регламентації інженерної діяльності розробників ПЗ.

Ця інженерна дисципліна надає всю необхідну інформацію та стандарти для вибору найбільш відповідного методу проектування практичних завдань. Не виключається й творчий неформальний підхід до створення ПЗ.

Як інженерна дисципліна, вона охоплює всі аспекти створення ПЗ, починаючи від формування вимог до створення, супроводу та зняття з експлуатації ПЗ, а також включає інженерні методи оцінки трудовитрат, вартості, продуктивності та якості.

Фахівці з програмної інженерії – це інженери, які виконують практичні роботи по реалізації програм із застосуванням теорії, методів і засобів комп'ютерних наук.

Програмна інженерія робить головний акцент на підвищення якості та продуктивності ПЗ за рахунок застосування нових і вдосконалених:

– методів проектування ПЗ;

– готових компонентів і методів їх генерації;

– методів еволюції ПЗ;

– методів верифікації та тестування ПЗ;

– інструментальних засобів підтримки;

– методів управління проектами, методів оцінки якості, продуктивності, вартості тощо;

– стандартизації процесів розробки ПЗ (ISO/ IEC 12207, ISO/ IEC 15504, ISO 9126 і ін.), що регламентують етапи життєвих циклів (ЖЦ); підходів до оцінки продуктів і процесів.

Методичні вказівки містять перелік тем лабораторних робіт. Кожна лабораторна робота складається з коротких теоретичних відомостей, варіантів завдань і прикладу виконання роботи.

Виконання лабораторних робіт дає можливість:

– вивчити і створити діаграму зв'язків (карту пам'яті), ознайомитися з програмою MindManager та її елементами;

– створити тимчасову діаграму для представлення графіка робіт;

– освоїти основні поняття про вимоги, які пред'являються до програмного забезпечення, та показати на прикладі різні способи представлення цих вимог;

– ознайомитися з нотаціями в уніфікованій мові моделювання UML і з тим, як вони використовуються при розробці різних типів моделей систем;

– знати інструментальні CASE-засоби, використовувані при моделюванні систем;

– ознайомитися з основними принципами проектування інтерфейсу користувача, з основними правилами проектування засобів підтримки користувача, вбудованими в програмне забезпечення.

Виконання лабораторних робіт дає також можливість:

– ознайомитися з методами розробки програмного забезпечення, які допомагають уникнути збоїв у програмній системі;

– ознайомитися з методами тестування програмного забезпечення, які використовуються для виявлення помилок і дефектів у програмі;

– ознайомитися з різними методами оцінки вартості та витрат, необхідних для створення програмного продукту.

1-Й СЕМЕСТР
ЛАБОРАТОРНА РОБОТА № 1

Побудова карти пам'яті (діаграми зв'язків). Ознайомлення з програмою MindManager

Мета роботи: вивчення та застосування діаграм зв'язків як засобу зображення процесу загального системного мислення за допомогою схем; ознайомлення з програмою Mind Manager та її елементами.

Короткі теоретичні відомості

Діаграма зв'язків, відома також як інтелект-карта (психолог Тоні Бьюзен запропонував методику візуального представлення процесу мислення і структуризації інформації – інтелект-карти (mind maps, карти розуму, ментальні карти), що дозволяє перевести процеси мислення на якісно новий рівень) – спосіб зображення процесу загального системного мислення за допомогою схем. Діаграма зв'язків може також розглядатися як зручна техніка альтернативного запису.

Діаграма зв'язків реалізується у вигляді деревовидної схеми, на якій зображені слова, ідеї, завдання або інші поняття, зв'язані гілками, що відходять від центрального поняття або ідеї. В основі цієї техніки лежить принцип "радіантного мислення", що відноситься до асоціативних розумових процесів, відправною точкою або точкою прикладання яких є центральний об'єкт. (Радіант – точка небесної сфери, з якої якби виходять видимі шляхи тіл з однаково направленими швидкостями, наприклад, метеорів одного потоку). Це показує нескінченну різноманітність можливих асоціацій і, отже, невичерпність можливостей мозку. Подібний спосіб запису дозволяє діагра-

мам зв'язків необмежено рости і доповнюватися. Діаграми зв'язків використовуються для створення, візуалізації, структуризації і класифікації ідей, а також як засіб для навчання, організації, вирішення завдань, ухвалення рішень, при написанні статей.

Спочатку інтелект-карти рисувалися вручну. Річ у тому, що в методі Тоні Бьюзена велика увага приділяється образному мисленню, тому у багатьох випадках замість слів на карті зображуються картинки. Фактично інтелект-карти можуть взагалі не містити слів – тільки рисунки, образи. В Інтернеті є приклади інтелект-карт, що є справжніми витворами мистецтва (рис. 1.1.). Природно, автоматизувати на комп'ютері рисування таких інтелект-карт до ладу неможливо.

Рис. 1.1. Карта пам'яті

Комп'ютерний метод створення інтелект-карт має безліч інших переваг. Можливості масштабування, організації багаторівневих зв'язків, створення гіперпосилань між картами,

швидкість створення карт, зручність редагування, зручні засоби колективної роботи, наявність різних способів представлення однієї і тієї ж карти – список переваг дуже великий.

Одним з визнаних лідерів в області створення комп'ютерних інтелект-карт є програма MindManager – комерційне ПЗ для управління картами думок, розроблене Mindjet. Mindjet описує карти, створені за допомогою MindManager як "бізнес-карти" для використання на підприємствах, замінюючи ними рукописні карти пам'яті.

Карти MindManager можуть бути експортовані в Microsoft Word, PowerPoint, Visio і Project, збережені як веб-сторінки чи PDF-документи.

Робота в MindManager

Використання MindManager можливо практично в будь-якій області діяльності – для планування, аналізу, вироблення рішень або просто для впорядковування та запам'ятовування будь-яких ідей і планів.

Рис. 1.2. Головне вікно запуску програми MindManager

Доведено, що інформація краще сприймається в графічному вигляді. Одна картинка або схема часом може замінити декілька листів нудного тексту.

Програма MindManager, перш за все, асоціюється з інтелект-картами, хоча за її допомогою можна створювати звичайні схеми та діаграми зв'язків для нарад, планів, різних документів і просто для організації власного життя та роботи.

Програма при першому ж відкритті створить центральний блок, в якому вводиться головний об’єкт схеми, що створюється.

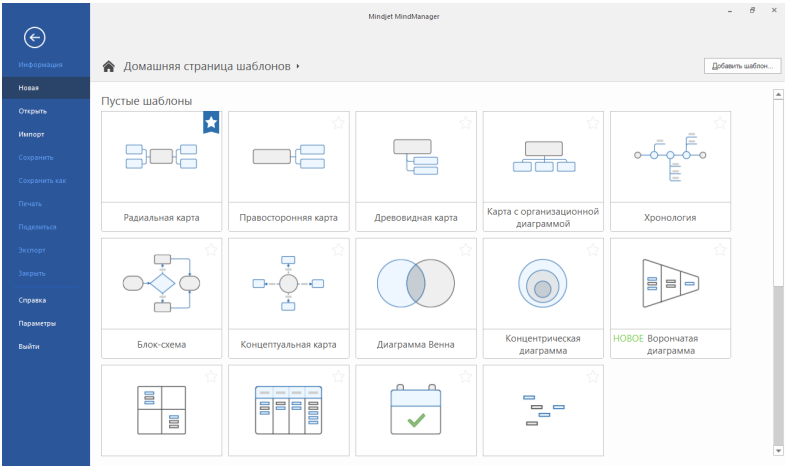


Рис. 1.3. Вікно створення шаблону в середовищі MindManager

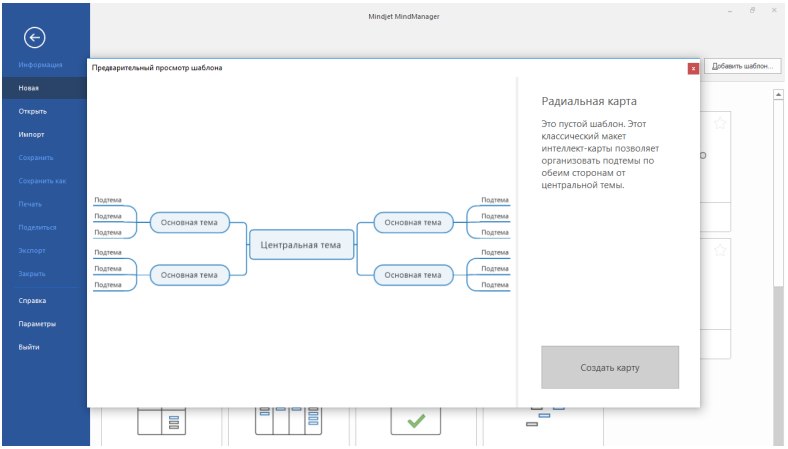


Рис. 1.4. Попередній перегляд шаблону

Зручна вкладка "Главная" програми MindManager (рис. 1.5) містить всі необхідні інструменти (кнопки, опції), щоб приступити до створення карт.

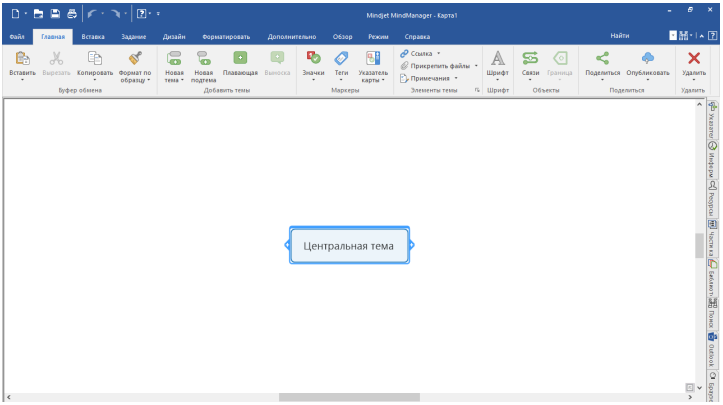


Рис. 1.5. Вкладка "Главная" програми MindManager

Як приклад побудуємо карту "Створення програмного забезпечення". Щоб додати до центрального блоку дочірній блок (блок другого рівня), необхідно виділити батьківський блок і натиснути кнопку "Новая подтема" (рис. 1.6). Дану дію можна також виконати за допомогою кнопки "Insert" на клавіатурі.

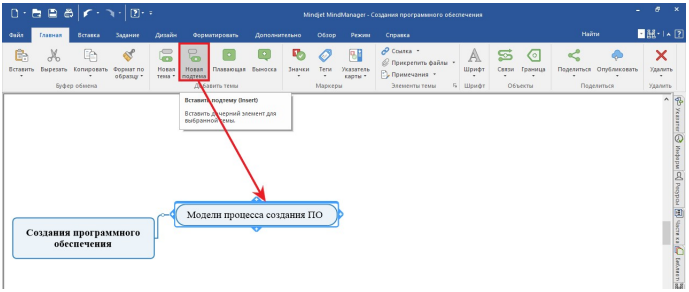


Рис. 1.6. Створення другого рівня об’єктів
у програмі MindManager

MindManager вставляє блоки, як їй надумається, тому, додавши необхідну кількість блоків другого рівня, можнао розмістити їх як подобається. Для цього потрібно навести мишку на блок і, не відпускаючи, перемістити його на нове місце.

Вирівнявши схему, введемо текст, клацаємо потрібний блок мишкою і вводимо необхідний текст. Також є можливість вибрати розмір шрифту, зображення, колір та інші параметри (рис. 1.7).

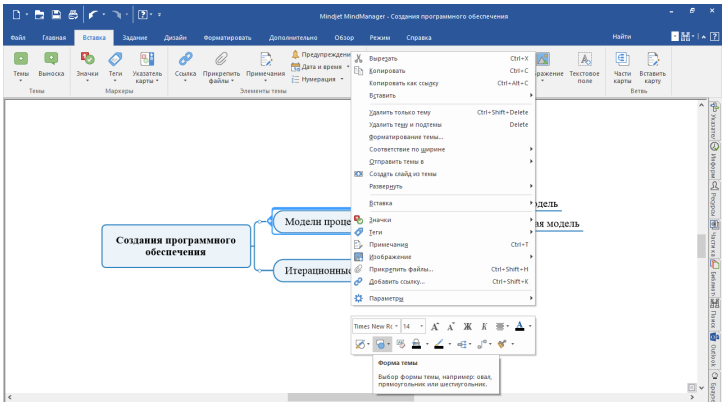


Рис. 1.7. Зміна параметрів у програмі MindManager

За допомогою вкладки "Форматировать" є можливість змінити колір, форму та інше.

Після того, як схема створена і введена вся необхідна інформація, є можливість зберегти карту MindManager. Для цього служить пункт "Сохранить" головного меню. При виборі даного пункту схема буде збережена у форматі програми MindManager з розширенням .mmap. Якщо треба зберегти схему в іншому форматі, вибирається пункт "Сохранить как". MindManager зберігає схеми в різних форматах, включаючи формати документів Microsoft Word і зображень JPEG та PNG.

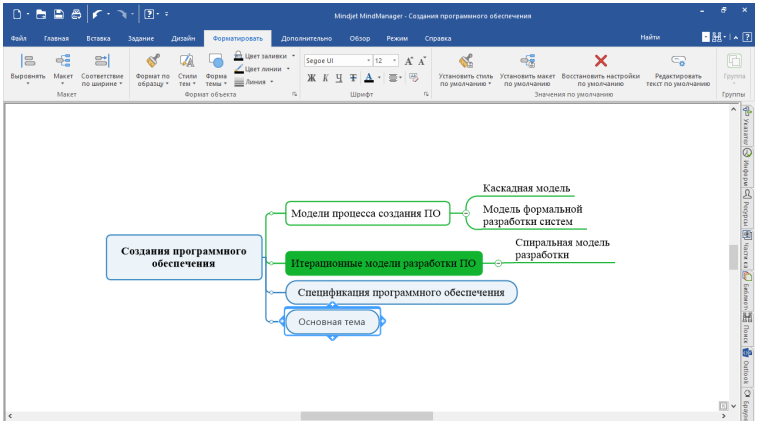


Рис. 1.8. Вкладка "Форматировать"

Більш детальна інформація у файлі "Керівництво користувача – MindManager".

Завдання

Ознайомитися з інтерфейсом програми MindManager. Побудувати за допомогою MindManager діаграму зв’язку двох рівнів для заданого процесу (див. табл. 1). Номер варіанта завдання взяти за списком журналу відвідування занять. Приклад оформлення лабораторної робіти представлено у додатку.

Таблиця 1. **Варіанти завдання**

№ варіанта	Назва процесу
1	Книжковий магазин
2	Магазин промислових товарів
3	Написання програми
4	Центр замовлень
5	Поштова служба, включаючи службу доставки
6	Поштове відділення
7	Управління персоналом в компанії (відділ кадрів)
8	Управління компанією (заводом)
9	Центр міжміських (міжнародних) переговорів
10	Залізничний вокзал
11	Автовокзал
12	Автотранспортні послуги
13	ЗВО або факультет ЗВО
14	Школа
15	Спортивний клуб (організація й управління)
16	Бібліотека
17	Банк
18	Процес навчання деякому предмету (організація й управління)
19	Аеропорт
20	Склад

ЛАБОРАТОРНА РОБОТА № 2

Побудова карти пам'яті (діаграми зв'язку) для заданого процесу в програмі MindManager

Мета роботи: оволодіння навиками побудови діаграм зв'язків за допомогою програми Mind Manager і створення діаграми для заданого процесу.

Короткі теоретичні відомості

MindManager – найпотужніша в світі програма для створення інтелект-карт і вирішення безлічі інших завдань.

Провідне в індустрії рішення для візуалізації інформації і створення інтелект-карт:

– створення чудових інтерактивних карт, графіків і діаграм;

– ефективні процеси обробки і систематизації даних;

– структуризація інформації та внесення ясності в концепції, плани і проекти;

– можливість одночасно спостерігати за загальною картиною та її складовими елементами та багато іншого.

Переваги для користувача:

– можливість колективного обговорення різних питань і вирішення виникаючих проблем;

– використання індивідуального виробничого потенціалу кожного співробітника, а також можливість управління завданнями;

– можливість управління нарадами;

– можливість планування проектів;

– аналіз виробничих процесів;

– здійснення стратегічного планування та бізнес-планування;

– створення презентацій і документів;

– організація дослідницької роботи.

Основні функціональні можливості:

– кращий в галузі майнд-меппінг;

– шаблони й інструменти для планування та розробки проектів, включаючи можливість виведення даних у вигляді діаграм Ганта й експорту даних в Microsoft Project;

– шаблони та графіки для бізнес-аналізу;

– тимчасові шкали, блок-схеми та діаграми;

– інструменти для складання бюджету та проведення фінансових розрахунків;

– інтеграція з Microsoft Outlook та Microsoft Office;

– інтеграція з більш ніж 800 інтернет-додатками, серед яких Asana, Trello, Salesforce.com, GoogleDocs, Evernote, OneNote та багато інших, через сервіс Zapier;

– інтегрування даних і прикладні програмні інтерфейси (API).

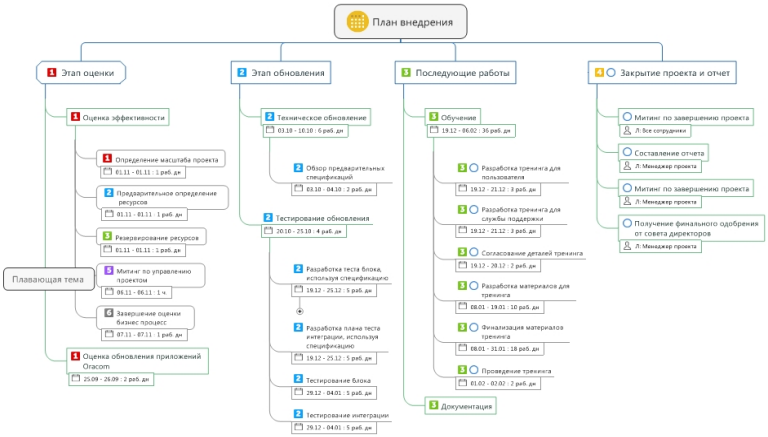


Рис. 2.1. Приклад створення карти "Планування проекту" [7]

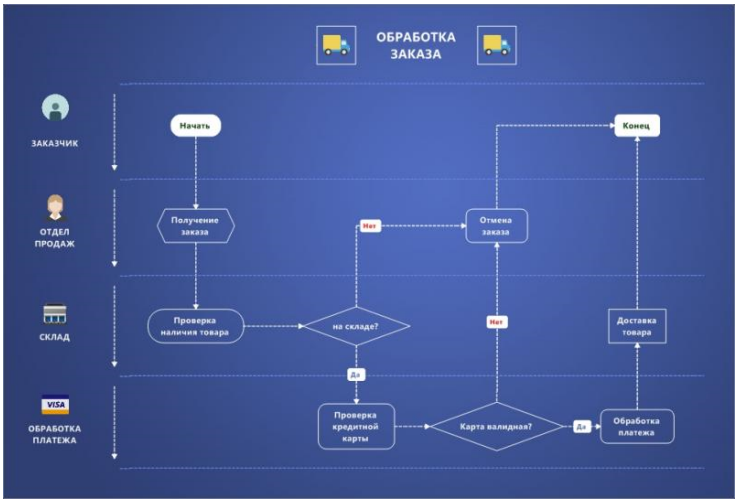


Рис. 2.2. Приклад блок-схеми в програмі MindManager [7]

Завдання

Для заданого процесу (див. табл. 1) побудувати за допомогою MindManager діаграму зв'язку. Номер варіанта завдання взяти за списком журналу відвідування занять. Приклад оформлення лабораторної роботи представлено у додатку.

ЛАБОРАТОРНА РОБОТА № 3

Створення презентації

Мета роботи: за допомогою Microsoft Office PowerPoint створити презентацію програми, що розробляється, з вказівкою вимоги до програмного забезпечення.

Короткі теоретичні відомості

Презентація (англ. presentation) – це процес ознайомлення слухачів з певною темою. Зазвичай це демонстрація, лекція чи промова з метою поінформувати або переконати когось [8].

Презентація може містити три компоненти:

– промова доповідача: те, що доповідач розповідає;

– слайди: те, що бачить аудиторія на екрані;

– роздаткові матеріали: те, що роздається кожному в аудиторії окремо і містить деталі інформації, що презентується. Наприклад, список літератури та інших посилань чи статистичні таблиці.

Часто, щоб допомогти створити та передати зміст презентації, використовуються програми для показу презентацій, такі як Microsoft PowerPoint, Apple Keynote, OpenOffice.org Impress чи Prezi. Сучасні веб-застосування, такі як Google Docs, SlideRocket та emaze також дозволяють створювати презентації спільними зусиллями кількох географічно розділених людей. Веб-розробники часто створюють презентації як звичайні HTML-сторінки, доповнені скриптом для зміни слайдів.

Щоправда, головним в презентації є все-таки те, що розповідає доповідач, а зображення на екрані – лише допоміжний інструмент.

Однією з прикладних програм для підготовки виступів або створення презентацій є Microsoft PowerPoint, що входить в комплект Microsoft Office.

Кожна сторінка презентації називається слайдом. Презентація складається з безлічі слайдів, які зберігаються в одному файлі. Розширення файлу ".ppt". Презентації можна представляти в електронному вигляді, роздруковувати на папері (копії одного або усіх слайдів) або поширювати через Інтернет тощо.

Засобами PowerPoint слайди можна подавати як у чорно-білому форматі, так і у вигляді кольорових зображень. Для цього використовують шаблони оформлення, створені професійними дизайнерами. Шаблони можна створювати і за власною ініціативою.

Слайди PowerPoint можуть містити такі елементи: текст (написи, колонтитули, об'єкти WordArt, символи, вбудовані об'єкти, дату, час, нумерацію); таблиці; зображення (рисунки, графіку, фотоальбоми, фігури, SmartArt фігури, діаграми); медіакліпи (фільми та звуковий супровід); гіперпосилання на інші слайди та документи (презентації, таблиці, діаграми та ін., які знаходяться на даному комп'ютері або в Інтернеті). Окремі об'єкти слайдів можуть мати ефекти анімації. Готуючи презентацію, можна використовувати фрагменти документів Word, електронних таблиць і діаграм Excel та ін. Створені в PowerPoint слайди можна відразу переглянути і, при необхідності, змінити.

Презентації, створені в PowerPoint, можна продемонструвати: на моніторі для невеликого кола осіб (у тому числі в Інтернеті); на екрані за допомогою мультимедійного проектора; як матеріали на папері.

Якщо відкрити програму PowerPoint, відобразяться вбудовані теми й шаблони. Тема – це оформлення слайда в єди-

ному стилі, що містить кольори, шрифти й спеціальні ефекти, як-от тіні, відбиття тощо [9].

На вкладці "**Файл**" на стрічці натисніть кнопку **Створити** і натисніть кнопку теми.

PowerPoint буде виконано попередній перегляд теми, з чотирьох зазначений на вибір праворуч.

Натисніть кнопку **Створити** або виберіть варіант кольору, а потім натисніть кнопку **Створити** (рис. 3.1).

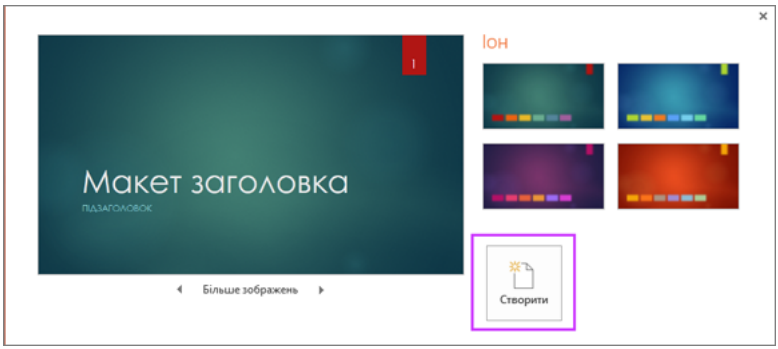


Рис. 3.1. Створення тем у програмі Microsoft PowerPoint

На вкладці **Основне** клацніть нижню половину кнопки **Створити** слайд і виберіть макет слайда (рис. 3.2).

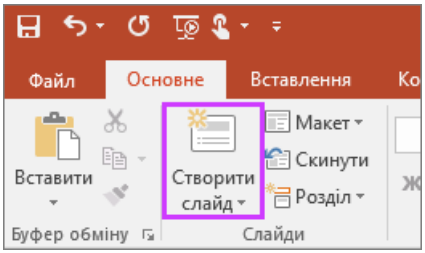


Рис. 3.2. Створення слайдів
у програмі Microsoft PowerPoint

Збереження презентації

На вкладці **Файл** клацніть елемент **Зберегти**. Виберіть папку або перейдіть до неї. У полі **Ім'я файлу** введіть ім'я презентації й натисніть кнопку **Зберегти** (рис. 3.3).

Примітка: Якщо вам часто доводиться зберігати файли в певній папці, можна "закріпити" цей шлях, щоб він завжди був доступний (як на знімку екрана нижче).

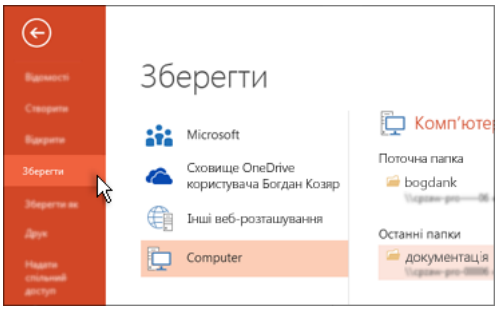


Рис. 3.3. Збереження файлу в програмі Microsoft PowerPoint

Додавання тексту

Клацніть місце для тексту та почніть вводити дані (рис. 3.4).

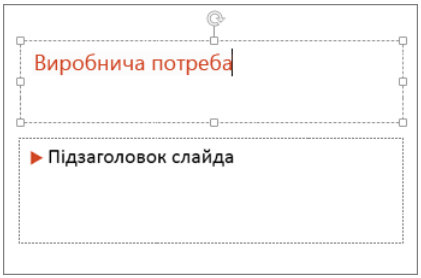


Рис. 3.4. Додавання тексту в програмі Microsoft PowerPoint

Форматування тексту

1. Виділіть текст.
2. У групі **Засоби креслення** виберіть вкладку **Формат**.
3. Виконайте одну з таких дій:

– щоб змінити колір тексту, натисніть кнопку **Заливка тексту**, а потім виберіть колір;

– щоб змінити колір контуру, натисніть кнопку **Контур тексту**, а потім виберіть колір;

– щоб застосувати тіні, відбиття, світіння, рельєф, об'ємне обертання, перетворення, натисніть кнопку **Текстові ефекти**, а потім виберіть потрібний ефект.

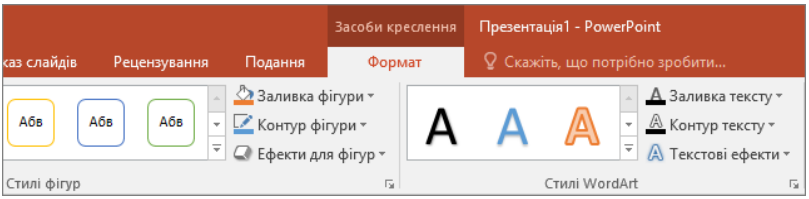


Рис. 3.5. Форматування тексту в програмі Microsoft PowerPoint

Також є можливість:

– зміни шрифтів;

– зміни кольору тексту на слайді;

– перетворення тексту на маркірований або нумерований список;

– форматування тексту як надрядкового або підрядкового.

Додавання зображень

На вкладці **Вставлення** виконайте одну з наведених нижче дій.

– Щоб вставити зображення, збережене на локальному диску або внутрішньому сервері, натисніть кнопку **Зображення**, знайдіть зображення, а потім натисніть кнопку **Вставити**.

– Щоб вставити зображення з Інтернету, натисніть кнопку **Онлайнові зображення** та знайдіть потрібне зображення, використовуючи поле пошуку.

Додавання нотаток доповідача

Найкраще слайди виглядають, коли вони не перевантажені надмірною кількістю інформації. Корисні факти й примітки можна додати до нотаток доповідача та звертатися до них під час презентації.

Щоб відкрити область нотаток, у нижній частині вікна клацніть елемент **Нотатки**. Клацніть в області **Нотатки** під слайдом і почніть вводити нотатку.

Проведення презентації

На вкладці **Показ слайдів** виконайте одну з наведених нижче дій:

1. Щоб запустити презентацію з першого слайда, у групі **Розпочати показ слайдів** клацніть пункт **З початку**.

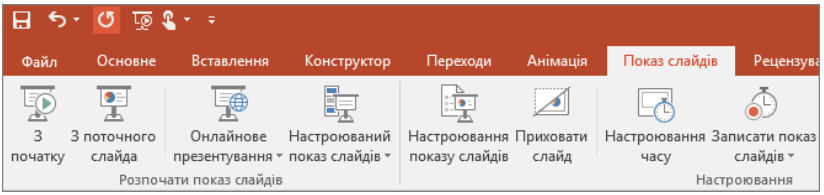


Рис. 3.6. Показ слайдів у програмі Microsoft PowerPoint

2. Якщо зараз відображається не перший слайд, а потрібно почати саме з нього, клацніть пункт **З поточного слайда**.

3. Якщо потрібно провести презентацію для осіб, які перебувають в іншому розташуванні, натисніть кнопку **Онлай-**

нове презентування та налаштуйте презентацію через Інтернет, вибравши один із наведених нижче параметрів:

– онлайнове презентування за допомогою служби презентацій Office;

– онлайнове презентування в PowerPoint за допомогою "Skype для бізнесу".

Щоб вийти з подання показу слайдів, у будь-який момент натисніть на клавіатурі клавішу **Esc**.

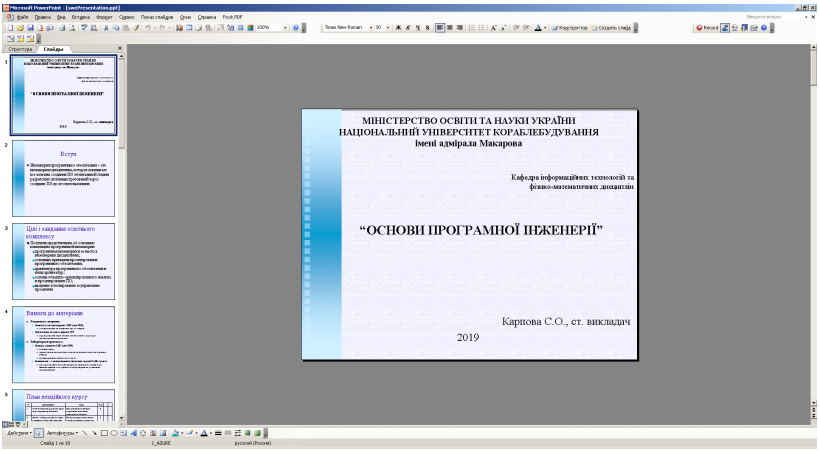


Рис. 3.7. Приклад оформлення презентації
в програмі Microsoft PowerPoint

Завдання

За допомогою Microsoft Office PowerPoint створити презентацію програми для заданого процесу (див. табл. 1). Номер варіанта завдання взяти за списком журналу відвідування занять.

ЛАБОРАТОРНА РОБОТА № 4

Тестова документація, розробка test case, розробка матриці вимог

Мета роботи: засвоєння основних понять про тестову документацію; створення Check List та розробка матриці вимог.

Короткі теоретичні відомості

Вимоги (Software requirements specification) – це документ, основа основ того, що буде реалізовано. У загальному вигляді вимоги описують перелік побажань замовника і того, що повинен продукт робити [11].

Збір вимог є початковим та невід'ємним етапом процесу розробки програмних систем. Він полягає у визначенні набору функцій, які необхідно реалізувати у продукті. Процес збору вимог частково реалізується у спілкуванні із замовником, частково штурмом мізків розробників. Результатом є формування набору вимог до системи, що називається технічним завданням.

Технічне завдання (ТЗ) дозволяє донести суть того, що слід створити команді. Допомагає зрозуміти, яким саме функціоналом повинен володіти продукт, іноді із зазначенням використовуваних технологій і методів його реалізації. Технічне завдання допомагає зрозуміти програму краще. У свою чергу, нерозуміння призводить до помилок у реалізації продукту чи помилкової публічної інтерпретації програмного продукту аудиторії. Технічне завдання дає можливість приблизно оцінити витрати ресурсів на розробку та тестування ще до початку робіт.

Технічна документація зазвичай містить повний опис логіки конкретної частини продукту, що розробляється, і варіанти, сценарії використання предмета розробки користувачами. Маючи під рукою технічну документацію, співробітник з легкістю зможе знайти в ньому необхідну інформацію, відразу приступивши до тестування. Технічна документація є частиною вихідного коду. Тестова документація повинна документувати все, що повинно бути зроблено *ДО* або *ПІД ЧАС* самого процесу тестування програмного забезпечення.

План тестування (**test plan**): Документ, що описує цілі, підходи, ресурси та графік запланованих тестових активностей [12]. Він визначає об'єкти тестування, властивості для тестування, завдання, відповідальних за завдання, ступінь незалежності кожного тестувальника, тестове оточення, методи проектування тестів, визначає використовувані критерії входу і критерії виходу та причини їх вибору, а також будь-які ризики, що вимагають планування на випадок надзвичайних обставин [IEEE 829].

План тестування відповідно до IEEE 829 [13]:

1) Analyze the product (Аналіз продукту);

2) Develop Test Strategy (Розробка тестової стратегії):

2.1) Define Scope of Testing (Визначення обсягу тестування);

2.2) Identify Testing Type (Визначення типу тестування);

2.3) Document Risk & Issues (Задокументуйте ризики та проблеми);

2.4) Create Test Logistics (Створення тестової логістики).

3) Define Test Objective (Визначення цілей тесту);

4) Define Test Criteria (Визначення критеріїв тестування);

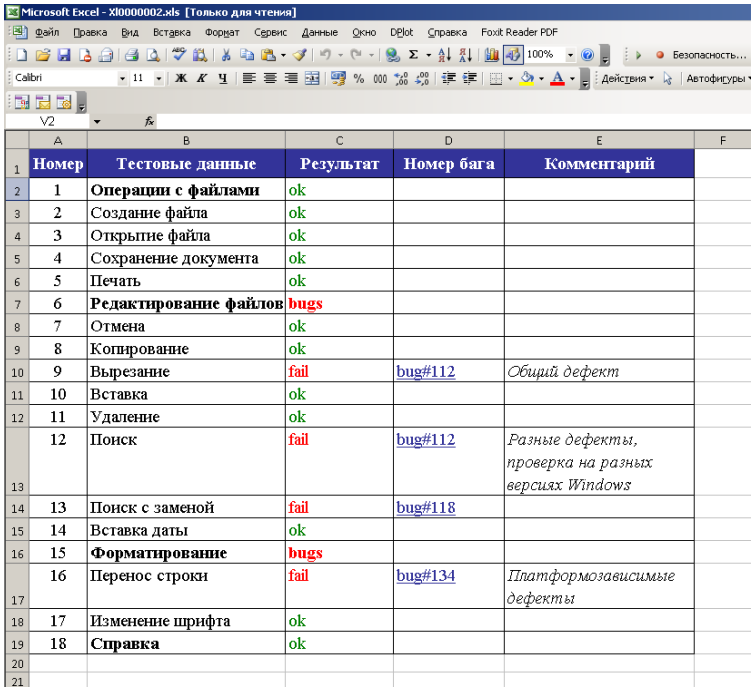
5) Resource Planning (Планування ресурсів);

6) Plan Test Environment (Планування середовища тестування);

7) Schedule & Estimation (Розклад і оцінка (тестова));

8) Test Deliverables (Результати тесту).

Check List – це частина test plan, конкретний список того, що потрібно перевірити (це документ, який описує, що має бути протестовано). Він допомагає планувати терміни закінчення робіт у майбутньому й сьогоденні. У ньому можна відмічати, скільки часу необхідно для перевірки та скільки часу було витрачено. Відображає ступінь готовності продукту. Зберігає історію раніше проведених тестів. Не дає забути, які тести необхідно виконати в першу чергу, які в другу, які в третю тощо. Також є можливість легко звірити, які саме тести проходили з помилками, і перепровірити їх ще раз.



	A	B	C	D	E	F
1	**Номер**	**Тестовые данные**	**Результат**	**Номер бага**	**Комментарий**	
2	1	**Операции с файлами**	ok			
3	2	Создание файла	ok			
4	3	Открытие файла	ok			
5	4	Сохранение документа	ok			
6	5	Печать	ok			
7	6	**Редактирование файлов**	**bugs**			
8	7	Отмена	ok			
9	8	Копирование	ok			
10	9	Вырезание	fail	bug#112	*Общий дефект*	
11	10	Вставка	ok			
12	11	Удаление	ok			
13	12	Поиск	fail	bug#112	*Разные дефекты, проверка на разных версиях Windows*	
14	13	Поиск с заменой	fail	bug#118		
15	14	Вставка даты	ok			
16	15	**Форматирование**	**bugs**			
17	16	Перенос строки	fail	bug#134	*Платформозависимые дефекты*	
18	17	Изменение шрифта	ok			
19	18	**Справка**	ok			
20						
21						

Рис. 4.1. Створення чек-листа

Error – помилка користувача, тобто він намагається використовувати програму іншим способом (вводить букви в поля, де потрібно вводити цифри (вік, кількість товару тощо)). У якісній програмі передбачені такі ситуації і видаються повідомлення про помилку (error message).

Bug (defect) – помилка програміста (або дизайнера, того, хто бере участь у розробці), тобто коли в програмі щось іде не так, як планувалося і програма виходить з-під контролю. Наприклад, якщо не контролюється введення даних користувача, в результаті невірні дані викликають "кращі" чи інші "радощі" в роботі програми. Або всередині програма побудована так, що не відповідає тому, що від неї очікувалось.

Failure – збій (причому не обов'язково апаратний) в роботі компонента, всієї програми або системи. Тобто, існують такі дефекти, які призводять до збоїв (A defect caused the failure), й існують такі, які не призводять. UI-дефекти, наприклад. Також апаратний збій, який не пов'язаний з software, теж є failure.

Тестовий сценарій (Test scenario) – повідомляє про те, яку ділянку й у якому порядку в програмі буде перевірено. Тестові сценарії використовуються, щоб ефективно протестувати все передбачене покриттям. Залежно від величини та складності програми, тестових сценаріїв може бути від одного до кількох сотень. Терміни "тестовий сценарій" та "тестові випадки" використовуються інколи взаємозамінно, проте тестовий сценарій має кілька етапів, тоді як тестовий випадок має один крок. З цієї точки зору, тестові сценарії є тестовими випадками, але вони містять кілька тестів і послідовність їх виконання. Окрім цього, кожен тест залежить від результатів попереднього тесту.

Тестові випадки включають набір кроків, передумов та вхідних умов, які можуть бути використані під час виконання тестових перевірок. Головною умовою є забезпечення

працездатності програми з точки зору її функціональності та інших аспектів. Існує багато типів тестових випадків, таких як функціональні, негативні, баги, логічні тестові випадки, логічні помилки, випадки від фізичних тестів на навантаження чи проникнення до системи, випадки тестування UX/UI інтерфейсу користувача тощо. Крім того, існують тестові випадки для відстеження тестового покриття програмного забезпечення.

Проте наступні компоненти радимо завжди включати у перевірку кожного тестового випадку:

– ідентифікатор тестового випадку, аби потім не потонути у морі невизначених випадків;

– модуль продукту;

– версія продукту;

– історія редакції;

– що викликало припущення перевірити саме цей тестовий випадок;

– передумови перевірки;

– виконання кроку;

– очікуваний результат;

– фактичний результат;

– пост-умови.

Багато тестів може бути розроблено з одного сценарію тестування. Крім того, іноді існує декілька різних тестів для перевірки одного й того самого моменту програмного забезпечення, які спільно називаються **тестовими об'єктами**.

Тест-кейс (Test case) – це описана послідовність певних дії (кроків) й очікуваного результату для перевірки роботи певного функціоналу системи. Необхідно, щоб опис кейса був таким, щоб виконати його змогла будь-яка людина (тестувальник, розробник, аналітик, замовник).

Структура тест-кейса

Важливі:

1. *ID* – унікальний ідентифікатор. Якщо пишеться вручну, то бажано робити його осмисленим. Часто генерується автоматично.

2. *Priority* (пріоритет тест-кейса) – атрибут, який вказує пріоритетність тест-кейса і приймає значення, відповідно до прийнятої в компанії класифікації (A-B-C-D-E, High-Medium-Low та ін.).

3. *Test description* (опис тест-кейсів) – важливий атрибут, в якому вказується заголовок (суть тесту), умови для його виконання, кроки та дії після проходження тесту. Test description, відповідно, може та повинен містити precondition, steps, postcondition.

4. *Steps* – це по суті крок (дія) в сценарії, який повинен бути зрозумілим і логічно випливати з попереднього кроку (що зробити для того, щоб перевірити).

5. *Expected result* (очікуваний результат тест-кейса) – даний атрибут відображає очікуваний результат на кожному кроці.

6. *Result* – сюди заносяться дані про результат пройденого тесту (passed / failed та ін.).

Додаткові:

7. *Req. ID* – атрибут, який вказує на пов'язаний з тестом пункт вимоги.

8. *Module* – вказується пов'язаний з тест-кейсом модуль.

9. *Sub-Module* – аналогічно попередньому пункту, тільки вписується дрібніша структурна одиниця.

10. *Comment* – сюди можна вносити свої примітки до тесту (питання замовнику, будь-які спостереження або деталі).

Головне, тест-кейси повинні бути незалежними (навіть в одному тест-комплекті). Будь-який кейс потрібно пройти

від початку до кінця. (Наприклад, якщо потрібно залогінитися на початку та вилогінитися в кінці, потрібно це вказати в перед- і пост-умовах). Всередині одного кейса не повинно бути посилань на інші кейси. Кейси можна виконувати в різному порядку навіть всередині тест-комплекту.

Тест-кейси зручно відображати у вигляді таблиці. Популярні спеціалізовані інструменти для ведення тест-кейсів: Zephyr, Jira (з інтегрованими інструментами), Redmine, Meliora, TestRail, Confluence, TFS та багато інших Test management system tools.

Фіксація вимог (requirement capturing), з однієї сторони, визначається бажаннями замовника в реалізації тієї чи іншої властивості. З іншої сторони, в процесі збору вимог може виявитися помилка, яка призведе до деяких наслідків, знешкодження яких відбиратиме непередбачені ресурси – додаткове кодування, перепланування.

Чим пізніше помилка виявиться, тим вона буде більш серйознішою, особливо якщо це пов'язано з множиною специфікацій. Тому однією із складових етапу фіксації вимог є верифікація вимог, а саме перевірка їх на суперечність та повноту.

Матриця відповідності вимог (Matrix Traceability Maturity (RTM)) – це таблиця, яка використовується для відстеження вимог під час життєвого циклу розробки програмного забезпечення (відповідність функціональних вимог (functional requirements) продукту та підготовлених тестових сценаріїв (test cases)). Вона може використовуватися для прямого відстеження (тобто від вимог до дизайну або кодування) або навпаки (тобто від кодування до вимог). У процесі збору вимог та проектування програмно-технічних систем, матриці трасування використовується для швидкої оцінки зв'язків між артефактами проектування, такими як:

– вимоги та тести;

– замовник та релізи (спринти);
– вимоги та підсистеми;
– вимоги та функціональні специфікації;
– функціональні та нефункціональні вимоги;
– вимоги та моделі системи;
– варіанти використання (use cases) і підсистеми;
– похибки та тести;
– помилки і модулі системи.

Кожна вимога в документі RTM пов'язана з відповідним тестовим випадком, тому виконання тестування відповідає зазначеним вимогам. Окрім того, ідентифікатор помилки також включений і пов'язаний з відповідною вимогою, тестовим сценарієм та випадком.

Основні цілі створення матриці:

– переконатися, що програма розроблена відповідно до зазначених вимог;

– допомогти знайти причину будь-якої помилки;

– гарний помічник, який підказує, які документи слід відстежувати на різних етапах циклу розробки програмного забезпечення.

Requirements Traceability Matrix for Katy's Cookies

		Objectives		
Req. #	Req. Description	Tastes Good	Big	Safe
1.1.1	Use non-expired ingredients	x		x
1.2.1	Do not use wheat			x
1.3.1	Use gourmet chocolate chips	x		
2.3.1	Cook thoroughly	x		x
3.1.1	Cookie diameter is 6 inches		x	
4.1.1	Cooks take food handler's course			x

Рис. 4.2. Створення матриці вимог [25]

На прикладі показана матриця вимог, яка описує три мети та шість вимог до печива Кеті, вони повинні бути смачними, великими й безпечними. За прикладом видно, що всі три цілі мають вимоги, пов'язані з ними, і немає ніяких вимог, які не можуть бути пов'язані ні з однією із цілей.

Ще один приклад матриці вимог для частини функціоналу калькулятора [13].

Складаємо нумерований список вимог до додатка (для прикладу візьмемо кілька існуючих функцій калькулятора).

ID требования	Формулировка требования
Rq1	Ввод цифр и действий. Можно производить вычисления, нажимая на кнопки калькулятора или вводя символы с клавиатуры. Кроме того, доступен ввод цифр и действий с цифровой клавиатуры, когда нажата клавиша NUM LOCK. Также можно вставлять математические выражения из буфера обмена и получать результат (например, набрать в Блокноте «5*3=», скопировать и вставить в Калькулятор, на «экране» которого появится ответ «15»).
Rq2	Очистка экрана / Удаление символов. Полная очистка экрана осуществляется при нажатии на кнопку «C» калькулятора или клавишу Delete на клавиатуре. Последний введённый числовой символ можно удалить, используя клавишу Backspace на клавиатуре.
Rq3	Калькулятор можно использовать для выполнения операций сложения.

Рис. 4.3. Вимоги до деяких функцій калькулятора

Формуємо список тестів – це може бути список назв детально розписаних покрокових тест-кейсів або ж чек-лист перевірок (рис. 4.4).

Складаємо матрицю вимог, яка може бути представлена різними засобами (рис. 4.5).

Звіт про результати тестування – це письмовий або медійний звіт про виконану роботу та її результати. Історично фіксує інформацію. До неї завжди можна буде повернутися та побачити, що саме було виконано й що саме отримали у результаті. Якщо це не окремий документ, то ним виступає **Bug Report**. Для формування Bug Report використовують Jira, Redmine, Mantis та ін.

ID теста	Проверка
Th1	Ввод значений и операций при нажатии на кнопки калькулятора
Th2	Ввод цифр и действий с цифровой клавиатуры, когда нажата клавиша NUM LOCK
Th3	Ввод цифровых значений с клавиатуры
Th4	Вставка математических выражений из буфера обмена (на одно, два, максимально возможное количество действий)
Th5	Очистка экрана нажатием на кнопку «C» калькулятора
Th6	Очистка экрана нажатием на клавишу Delete калькулятора
Th7	Удаление последнего введённого символа нажатием на клавишу Backspace на клавиатуре
Th8	Сложение однозначных, двузначных, многозначных чисел
Th9	Сложение десятичных чисел
Th10	Сложение отрицательных чисел

Рис. 4.4. Список тестів

TRACEABILITY MATRIX												
	ID теста	Th1	Th2	Th3	Th4	Th5	Th6	Th7	Th8	Th9	Th10	Кол-во тестов, которые покрывают требование
ID требования												
Rq1		X	X	X	X							4
Rq2						X	X	X				3
Rq3									X	X	X	3

Рис. 4.5. Матриця вимог

Bug Tracking System повідомляє автоматично про результати всіх, кому важливо про них знати. Наприклад, для співробітників відділу підтримки повідомляється про вихід нової версії програми, що розробляється, а для розробників – про найкритичніші проблеми тощо.

Звіт про тестування допомагає приймати важливі рішення про подальші дії (наприклад, чи варто випускати версію програми в поточному стані?).

Завдання

Створити Check List (список перевірок) і розробити матрицю вимог для заданого процесу (див. табл. 1). Номер варіанта завдання взяти за списком журналу відвідування занять. Приклад оформлення лабораторних робіт представлено у додатках.

ЛАБОРАТОРНА РОБОТА № 5

Схематична розробка моделі потоків даних

Мета роботи: набуття навичок створення моделей потоків даних.

Короткі теоретичні відомості

Вимоги для користувачів зазвичай пишуться на природній мові, оскільки вони повинні бути зрозумілі не тільки фахівцям в області розробки ПЗ. Проте більш деталізовані системні вимоги повинні описуватися більш "технічним" засобом. Однією з широко використовуваних методик документування системних вимог є побудова ряду моделей системи. Ці моделі використовують графічні уявлення, що показують рішення як початкової задачі, для якої створюється система, так і системи, що розробляється. Як правило, графічні уявлення зрозуміліші, ніж детальний опис системних вимог на природній мові. Моделі є сполучною ланкою між процесом аналізу початкового завдання та процесом проектування системи.

Моделі можна використовувати в процесі аналізу існуючої системи, яку потрібно або замінити, або модифікувати, а також для формування системних вимог:

Приведемо типи системних моделей, які можуть створюватися в процесі аналізу систем:

– модель обробки даних. Діаграми потоків даних показують послідовність обробки даних в системі;

– композиційна модель. Діаграми "сутність-зв’язок" показують, як системна сутність складається з іншої сутності;

– архітектурна модель. Ці моделі показують основні підсистеми, з яких будується система;

– класифікаційна модель. Діаграми спадкоємства класів показують, які об'єкти мають загальні характеристики;

– модель "стимул-відповідь". Діаграми зміни станів показують, як система реагує на внутрішні і зовнішні події.

У лабораторній роботі розглядається модель потоків даних.

Моделі потоку даних (діаграми потоків даних(Data flow diagramming, DFD) – це інтуїтивно зрозумілий спосіб показу послідовності обробки даних усередині системи. Нотації, використовувані в цих моделях, описують обробку даних за допомогою системних функцій, а також зберігання та переміщення даних між системними функціями.

Моделі потоків даних використовуються для показу послідовності кроків обробки даних. Ці кроки обробки або перетворення даних виконуються програмними функціями. По суті, діаграми потоків даних використовуються для документування програмних функцій перед проектуванням системи.

У основі моделі лежать поняття зовнішньої сутності, процесу, сховища (накопичувача) даних і потоку даних.

Зовнішня сутність – матеріальний об'єкт або фізична особа, що виступають в якості джерел або приймачів інформації, наприклад, замовники, персонал, постачальники, клієнти банк і тому подібне.

Процес – перетворення вхідних потоків даних у вихідні відповідно до певного алгоритму. Кожен процес у системі має свій номер, пов'язаний з виконавцем, який здійснює дане перетворення. Як у разі функціональних діаграм, фізично перетворення може здійснюватися комп'ютерами, вручну або спеціальними пристроями. На верхніх рівнях ієрархії, коли процеси ще не визначені, замість поняття "процес" використовують поняття "система" та "підсистема", які позначають відповідно систему в цілому або її функціонально закінчену частину.

Сховище даних – абстрактний пристрій для зберігання інформації. Тип пристрою та способи розміщення, вилучен-

ня й зберігання для такого пристрою не деталізують. Фізично це може бути база даних, файл, таблиця в оперативній пам'яті, картотека на папері й тому подібне.

Потік даних – процес передачі деякої інформації від джерела до приймача. Фізично процес передачі інформації може відбуватися по кабелях під управлінням програми або програмної системи, або вручну за участю пристроїв, або людей зовні проектованої системи.

Таким чином, діаграма ілюструє як потоки даних, породжені деякими зовнішніми сутностями, трансформуються відповідними процесами (або підсистемами), зберігаються накопичувачами даних і передаються іншій зовнішній суті – приймачам інформації. В результаті отримуємо мережеву модель зберігання/обробки інформації. Для зображення діаграм потоків даних традиційно використовують два види нотацій: нотації Йордана і Гейна-Сарсона (рис. 5.1).

Елемент	Опис	Нотація Йордона-Де Марко	Нотація Гейна-Сарсано
Функція	Робота	Ім'я функції Номер	Ім'я функції Номер
Поток даних	Об'єкт над яким виконується робота. Може бути логічним або керуючим (пунктирна лінія зі стрілкою)	Ім'я об'єкта	Ім'я об'єкта
Сховище даних	Структура для зберігання інформаційних об'єктів	Ім'я об'єкта	Ім'я об'єкта
Зовнішня сутність	Зовнішній по відношенню до системи об'єкт, обмінюється з нею потоками	Ім'я зовнішнього об'єкта	Ім'я зовнішнього об'єкта

Рис. 5.1. Зображення потоків даних у нотаціях

У діаграмах потоків даних використовуються наступні позначення: закруглені прямокутники відповідають етапам обробки даних; стрілки, забезпечені примітками з назвою даних, представляють потоки даних; прямокутники відповідають сховищам або джерелам даних.

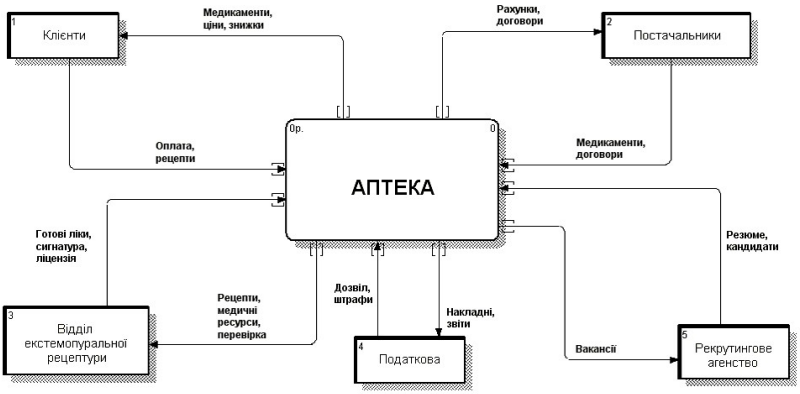


Рис. 5.2. Діаграма потоків даних нульового рівня (DFD0) для процесу "Аптека"

Моделі потоків даних цінні тим, що вони простежують і документують переміщення даних по системі, допомагаючи тим самим аналітикам зрозуміти цей процес. Перевага діаграм потоків даних у тому, що вони, на відміну від інших моделей, прості та інтуїтивно зрозумілі.

Моделі потоків даних показують функціональну структуру системи, де кожне перетворення даних відповідає одній системній функції. Іноді моделі потоків даних використовують для опису потоків даних у робочому оточенні системи. Така модель показує, як різні системи та підсистеми обмінюються інформацією. Підсистеми оточення не зобов'язані бути

простими функціями. Наприклад, одна підсистема може бути сервером бази даних з досить складним інтерфейсом.

Завдання

Побудувати діаграму потоків даних нульового рівня для заданого процесу (див. табл. 1). Номер варіанта завдання взяти за списком журналу відвідування занять.

ЛАБОРАТОРНА РОБОТА № 6

Схематична розробка об'єктних моделей

Мета роботи: набуття навичок створення об'єктних моделей.

Короткі теоретичні відомості

Об'єктно-орієнтований підхід широко використовується при розробці програмного забезпечення, особливо для розробки інтерактивних систем. У цьому випадку системні вимоги формуються на основі об'єктної моделі, а програмування виконується за допомогою об'єктно-орієнтованих мов, таких, як Java або C++.

Об'єктні моделі, розроблені для формування вимог, можуть використовуватися як для представлення даних, так і для процесів їх обробки. В цьому відношенні вони об'єднують моделі потоків даних і семантичні моделі даних. Вони також корисні для класифікації системної суті й можуть представляти суть, що складається з іншої суті.

Для деяких класів систем об'єктні моделі – природний спосіб відображення реально існуючих об'єктів, які знаходяться під управлінням системи. Наприклад, для систем, що оброблюють інформацію щодо конкретних об'єктів (таких, як автомобілі, літаки, книги і так далі), які мають чітко певні атрибути. Абстрактніша високорівнева суть, наприклад, бібліотеки, медичні реєструючі системи або текстові редактори, які важче моделювати у вигляді класів об'єктів, оскільки вони мають достатньо складний інтерфейс, що складається з незалежних атрибутів і методів.

Об'єктні моделі, розроблені під час аналізу вимог, поза сумнівом, спрощують перехід до об'єктно-орієнтованого проектування та програмування. Клас об'єктів – це абстракція безлічі об'єктів, які визначаються загальними атрибутами й сервісами або операціями, які забезпечуються кожним об'єктом. Об'єкти – це виконувана суть з атрибутами та сервісами класу об'єктів. Об'єкти є реалізацією класу, на основі одного класу можна створити багато різних об'єктів. Зазвичай при розробці об'єктних моделей основна увага зосереджена на класах об'єктів і їх відносинах.

Моделі систем, що розробляються при формуванні вимог, повинні відображати реальну суть, що належить класам об'єктів. Класи не повинні містити інформацію про окремі системні об'єкти. Можна розробити різні типи об'єктних моделей, що показують, як класи зв'язані один з одним, як об'єкти агрегуються з інших об'єктів, як об'єкти взаємодіють з іншими об'єктами і так далі. Ці моделі розширюють розуміння системи, що розробляється.

Різні методи об'єктно-орієнтованого аналізу були запропоновані в 1990-х роках Бучем (Booch), Кодом і Джордоном (Goad and Yourdon), а також Рамбо (Rumbaugh). Ці методи мали багато загального, тому три головні розробники – Буч, Рамбо і Якобсон (Jacobson) вирішили інтегрувати їх для розробки уніфікованого методу. В результаті розроблений ними уніфікована мова моделювання (Unified Modeling Language – UML) стала фактичним стандартом для моделювання об'єктів. У UML є нотації для різних типів системних моделей. У UML клас об'єктів представлений вертикально орієнтованим прямокутником з трьома секціями (рис. 6.1).

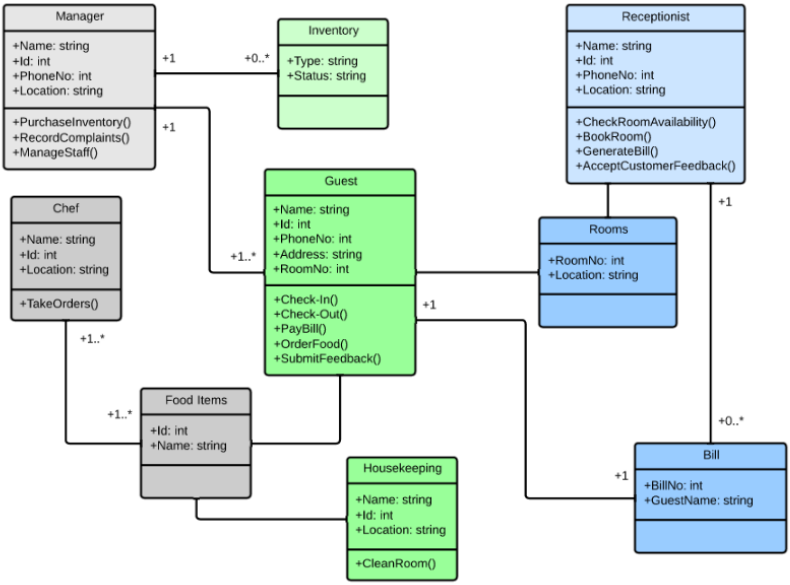


Рис. 6.1. Діаграма класів управління готелем [21]

Складові класа:

– у верхній секції розташовується ім’я класу;

– атрибути класу знаходяться в середній секції;

– операції, пов’язані з класом, приводяться в нижній секції прямокутника.

Завдання

Розробити об’єктну модель даних для заданого процесу (див. табл. 1). Номер варіанта завдання взяти за списком журналу відвідування занять.

ЛАБОРАТОРНА РОБОТА № 7

Створення елементів графічного інтерфейсу користувача

Мета роботи: набуття навичок створення елементів графічного інтерфейсу користувача.

Короткі теоретичні відомості

Проектування обчислювальних систем охоплює широкий спектр проектних дій – від проектування апаратних засобів до проектування інтерфейсу користувача. Грамотно спроектований інтерфейс користувача вкрай важливий для успішної роботи системи. Складний у застосуванні інтерфейс, як мінімум, приводить до помилок користувача. Іноді вони просто відмовляються працювати з програмною системою, не дивлячись на її функціональні можливості.

Таблиця 2. **Елементи графічних інтерфейсів користувача**

Елементи	Опис
Вікна	Дозволяють відображати на екрані інформацію різного роду
Піктограми	Представляють різні типи даних. У одних системах піктограми представляють файли, в інших – процеси
Меню	Введення команд замінюється вибором команд з меню
Вказівники	Миша використовується як пристрій вказівки для вибору команд з меню і для виділення окремих елементів у вікні
Графічні елементи	Можуть використовуватися спільно з текстовими

На рис. 7.1 зображений ітераційний процес проектування призначеного для користувача інтерфейсу. На початку процесу прототипування створюються паперові макети інтерфейсу, потім розробляються екранні форми, що моделюють взаємодію з користувачем.

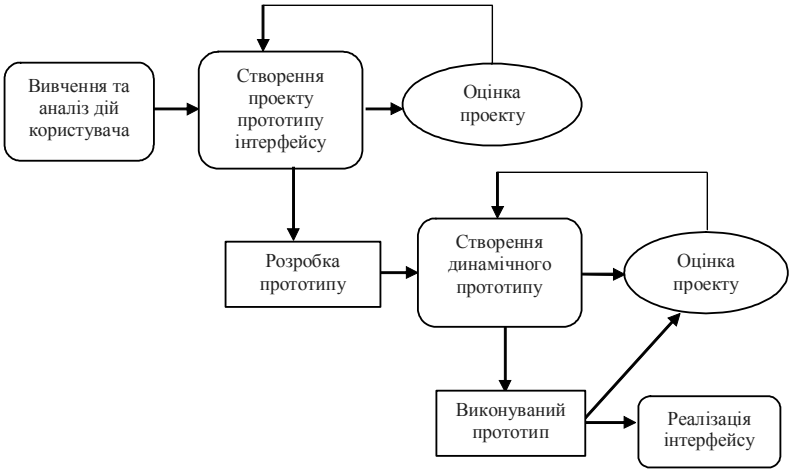


Рис. 7.1. Процес проектування інтерфейсу користувача

Важливим етапом процесу проектування інтерфейсу користувача є аналіз діяльності користувачів, яку повинна забезпечити обчислювальна система. Не вивчивши того, що, з погляду користувача, повинна робити система, неможливо сформувати реалістичний погляд на проектування ефективного інтерфейсу. Для аналізу потрібно застосовувати різні методики, а саме: аналіз завдань, етнографічний підхід, опитування користувачів і спостереження за їх роботою.

Рекомендації при створенні інтерфейсу користувача:

– Процес проектування інтерфейсу повинен орієнтуватися на користувача. Інтерфейс повинен взаємодіяти з кори-

стувачем на його "мові", бути логічним і послідовним. У інтерфейсі повинні бути довідкові засоби, що допомагають користувачам при роботі з системою, і засоби відновлення після помилок.

– Існує декілька стилів взаємодії з програмними системами: безпосереднє маніпулювання, системне меню, заповнення форми, командні мови і природна мова.

– Для відображення тенденцій числових даних і їх приблизних значень слід використовувати графічні уявлення. Числове уявлення повинне застосовуватися тільки тоді, коли потрібно відобразити точні значення даних.

– Кольори в інтерфейсі користувача повинні використовуватися обережно і послідовно. Розробники повинні завжди пам'ятати, що багато хто не розрізняє кольори.

– Повідомлення про помилки не повинні містити звинувачень в адресу користувача. Вони повинні пропонувати варіанти виправлення помилки і забезпечувати зв'язок з довідковою системою.

– У документації користувача повинне бути керівництво для початкуючих і досвідчених користувачів. Для системного адміністратора повинні бути окремі документи.

– У системній специфікації бажано мати кількісні значення для показників зручності використання інтерфейсу, а процес його оцінювання повинен перевіряти систему на відповідність цим вимогам.

Завдання

Ознайомитися з ключовими поняттями при створенні інтерфейсу користувача. Створити прототип графічного інтерфейсу користувача. Варіанти завдань (див. табл. 1) взяти за списком журналу відвідування занять.

2-Й СЕМЕСТР
ЛАБОРАТОРНА РОБОТА № 8

Розробка вимог до програмного забезпечення

Мета роботи: освоїти основні поняття про вимоги, що пред'являються до програмного забезпечення, і показати на прикладі різні способи представлення цих вимог.

Короткі теоретичні відомості

Опис функціональних можливостей та обмежень, що накладаються на програмну систему, називається *вимогами* до цієї системи, а сам процес формування, аналізу, документування та перевірки цих функціональних можливостей й обмежень – *розробкою вимог* (requirements engineering).

Термін *вимоги* (до програмної системи) може трактуватися по-різному. В деяких випадках під вимогами розуміються високорівневі узагальнені твердження про функціональні можливості й обмеження системи. Інша крайня ситуація – деталізований математичний формальний опис системних функцій. Розрізняють три види вимог:

1. *Призначені для користувача вимоги* – опис на природній мові (плюс пояснюючі діаграми) функцій, що виконуються системою, й обмежень, що накладаються на неї.

2. *Системні вимоги* – деталізований опис системних функцій й обмежень, який іноді називають функціональною специфікацією. Вона служить основою для укладення контракту між покупцем системи та розробниками ПЗ.

3. *Проектна системна специфікація* – узагальнений опис структури програмної системи, який буде основою для більш деталізованого проектування системи та її подальшої реалі-

зації. Ця специфікація доповнює та деталізує специфікацію системних вимог.

Вимоги до програмної системи часто класифікуються як функціональні, нефункціональні та вимоги наочної області.

1. *Функціональні вимоги.* Це перелік сервісів, які повинна виконувати система, причому повинно бути вказано, як система реагує на ті або інші вхідні дані, як вона поводиться в певних ситуаціях і так далі. В деяких випадках указується, що система не повинна робити.

2. *Нефункціональні вимоги.* Описують характеристики системи та її оточення, а не поведінку системи. Тут також може бути приведений перелік обмежень, що накладаються на дії і функції, що виконуються системою. Вони включають *тимчасові* обмеження, обмеження на процес розробки системи, стандарти і так далі.

3. *Вимоги наочної області.* Характеризують ту наочну область, де система буде експлуатуватись. Ці вимоги можуть бути функціональними та нефункціональними.

Документ, що містить вимоги, також званий специфікацією системних вимог, – це офіційне розпорядження для розробників програмної системи. Він містить призначені для користувача вимоги та деталізований опис системних вимог. В деяких випадках, призначені для користувача системні вимоги, можуть не розрізнятися, виступаючи спільно у вигляді однорідного опису системи. В інших випадках призначені для користувача вимоги наводяться у введенні документа-специфікації. Якщо загальна кількість вимог велика, деталізовані системні вимоги можуть бути представлені у вигляді окремого документа.

Системну специфікацію читає безліч різних людей, починаючи від вищого керівництва компанії-замовника системи й закінчуючи рядовим розробником системи.

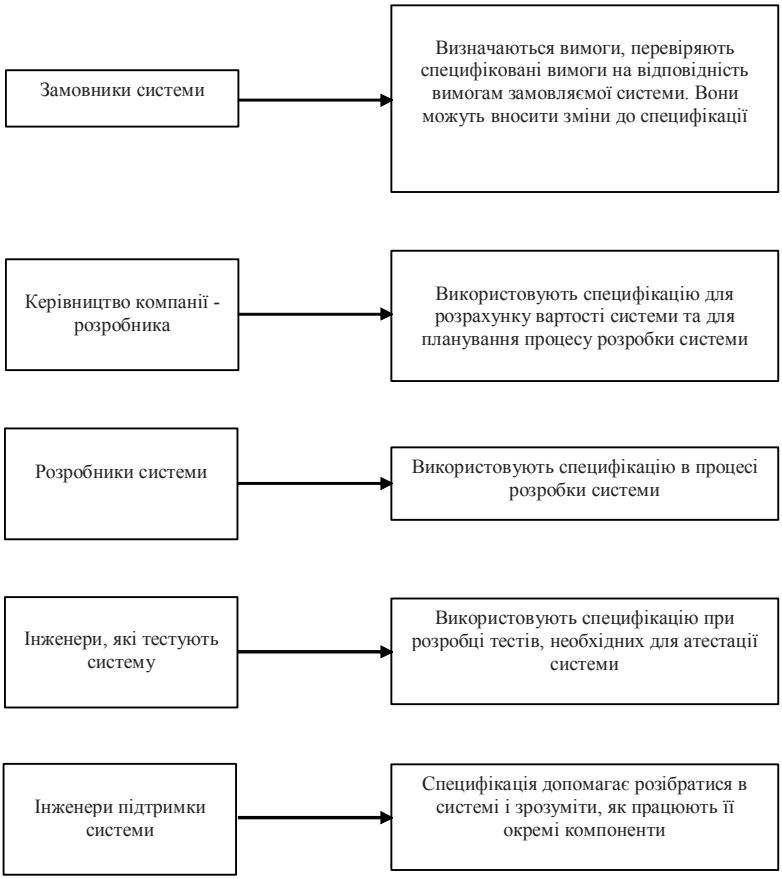


Рис. 8.1. Читачі системної специфікації

Багато організацій, таких як Міністерство оборони США й Інститут інженерів по електротехніці і радіоелектроніці IEEE, розробили власні стандарти документування специфікацій. Найбільш відомий стандарт розроблений IEEE і називається IEEE/ANSI 830-1993. Даний стандарт припускає наступну структуру специфікації.

1. **Введення**

1.1. Цілі документа

1.2. Призначення програмного продукту

1.3. Визначення акроніму й абревіатури

1.4. Список літератури й інших джерел

1.5. Огляд специфікації

2. **Загальний опис**

2.1. Опис програмного продукту

2.2. Функції програмного продукту

2.3. Призначені для користувача характеристики

2.4. Загальні обмеження

2.5. Обґрунтування, припущення і допущення

3. **Специфікація вимог** охоплює функціональні, нефункціональні й інтерфейсні вимоги. Це найбільш значуща частина документа, але внаслідок украй широкого діапазону можливих вимог, що пред'являються програмним системам, у стандарті не визначена структура цього розділу. Тут можуть бути документовані зовнішні інтерфейси, описані функціональні можливості системи, приведені вимоги, що визначають логічну структуру баз даних, обмеження, що накладаються на структуру системи, описані інтеграційні властивості системи або її якісні характеристики.

4. **Додатки**

5. **Покажчики**

Завдання

Розробити вимоги для заданого програмного забезпечення. Варіанти завдань (див. табл. 1) взяти за списком журналу відвідування занять.

ЛАБОРАТОРНА РОБОТА № 9

Розробка інтерфейсу та виконання розрахунку в середовищі Visual Fortran для заданого процесу. Огляд методів розробки критичних систем

Мета роботи: оволодіння навиками розробки інтерфейсу в середовищі Visual Fortran; знати про методи розробки програмного забезпечення, які допомагають уникнути збоїв у програмній системі.

Короткі теоретичні відомості

Зазвичай відмова систем, керованих за допомогою ПЗ, викликає незручності, але вони не приводять до тривалих наслідків. Проте є системи, відмови яких можуть приводити до значних економічних втрат, фізичних пошкоджень або створювати загрозу людському життю. Такі системи зазвичай називають *критичними*. Функціональна надійність – необхідна вимога до критичних систем, і всі її складові (працездатність, безвідмовність, безпека і захищеність) дуже важливі. Не менш важливий для критичних систем і високий рівень надійності.

Існує три основні типи критичних систем:

– системи, критичні по забезпеченню безпеки. Системи, відмова яких приводить до руйнувань, створює загрозу життю людини або завдає шкоди навколишньому середовищу. Як приклад можна привести систему управління виробництвом на хімічному заводі;

– системи, критичні для цільового призначення. Системи, відмова яких може привести до помилок в діях, направлених на забезпечення певної мети. Прикладом може служити навігаційна система космічного корабля;

– системи, критичні для бізнесу. Відмова таких систем може завдати шкоди справі, в якій вони використовується. Прикладом є система, обслуговуюча рахунки клієнтів у банку.

Ціна помилки критичної системи часто дуже велика. Вона включає прямі витрати, пов'язані з внесенням змін до системи або її заміною, непрямі витрати, наприклад, судові, і витрати, пов'язані з втратами в бізнесі. З високої можливої ціни відмови системи виходить, що якість методів розробки та сам процес створення ПЗ зазвичай важливіші, ніж вартість застосування цих методів.

Тому при створенні критичних систем зазвичай використовуються випробувані методи розробки, а не нові, такі, що ще не мали великого практичного застосування. Тільки порівняно недавно такі відносно нові методи, як, наприклад, об'єктно-орієнтовані, почали використовуватися для розробки критичних систем. Разом з тим до цих пір при розробці багатьох критичних систем все ще застосовуються функціонально-орієнтовані методи.

Процес створення надійного програмного забезпечення переслідує мету розробки безвідмовного ПЗ, тобто такого, що точно відповідає специфікації системних вимог. Але точна відповідність системи своїй специфікації не гарантує, що ПЗ завжди поводитиметься так, як очікується користувачами. У специфікації можуть бути помилки, які відіб'ються в програмному забезпеченні, або користувачі можуть невірно тлумачити або неправильно експлуатувати систему. Безвідмовне ПЗ не обов'язково гарантує відсутність відмов у роботі системи. Але, з іншого боку, мінімізація помилок програмного забезпечення значно зменшує число відмов системи та повинна виконуватися при розробці критичних систем.

Існує ряд вимог до розробки безвідмовного програмного забезпечення:

– Повинна бути точна (переважно формальна) специфікація системних вимог, що визначає систему, що розробляється.

– Організація-розробник ПЗ повинна мати високу культуру управління якістю, оскільки якість є головною в процесі створення критичних систем. У ідеалі передбачається, що програмісти створюють програми, в яких відсутні помилки.

– Методи проектування та реалізації ПЗ повинні ґрунтуватися на захованні й інкапсуляції інформації. Об'єктно-орієнтовані мови задовольняють цим умовам (як приклад Java).

– В процесі реалізації програмного коду повинні використовуватися мови програмування із строгим контролем типів даних, наприклад Java або Ada. У таких мовах багато помилок програмування будуть виявлено на етапі компіляції програм.

– Скрізь, де можливо, слід уникати використання тих програмних конструкцій, які потенційно можуть привести до помилок.

– Повинна бути визначена чітка технологія розробки ПЗ, і розробники повинні бути навчені застосуванню цієї технології.

Помилки в програмах і, як наслідок, збої в роботі систем часто є результатом помилок людини. Програмісти роблять помилки, тому що втрачають зв'язок між взаєминами змінних станів. Вони пишуть програми, які приводять до непередбаченої поведінки та непередбаченої зміни станів системи.

Використання оператора безумовного переходу goto приводить до логічних конструкцій, істотно схильним до помилок програмування. Вони також ускладнюють локалізацію змін стану системи. Це спостереження привело до появи так званого структурного програмування. Мова йде про викори-

стання, наприклад, оператора циклу while замість використання оператора goto у сполученні з умовним оператором if. Окрім операторів безумовного переходу, існують інші мовні конструкції та методи програмування, схильні до помилок:

– числа з плаваючою комою;
– вказівники;
– динамічний розподіл пам'яті;
– паралельність процесів;
– рекурсія;
– переривання;
– спадкоємство;
– поєднання імен;
– введення даних без перевірки.

Деякі стандарти проектування систем, критичних по забезпеченню безпеки, повністю забороняють використання перерахованих програмних конструкцій.

Завдання

Ознайомитися з ключовими поняттями по розробці критичних систем користувача. Розробити програму і створити для неї інтерфейс для математичного розрахунку. Варіанти завдання можуть бути запропоновані студентом.

Приклади варіантів завдань:

– Розрахунок моменту інерції балки складеного перетину;
– Розрахунок частоти малих поперечних коливань;
– Розрахунок перекриття;
– Розрахунок світлової віддачі електричної лампочки;
– Апроксимація функцій;
– Розрахунок напружено-деформованого стану в точці тіла;
– Розрахунок координат центра ваги криволінійної трапеції;
– Розрахунок періоду коливань маятника.

ЛАБОРАТОРНА РОБОТА № 10

Побудова тимчасової діаграми етапів робіт для заданого процесу

Мета роботи: за допомогою Microsoft Excel побудувати тимчасову діаграму для заданого процесу.

Короткі теоретичні відомості

Складання графіка – одна з найвідповідальніших робіт, що виконуються менеджером проекту. В процесі складання графіка весь масив робіт, необхідних для реалізації проекту, розбивається на окремі етапи і оцінюється час, потрібний для виконання кожного етапу. Зазвичай багато етапів виконуються паралельно. Графік робіт повинен передбачати це та розподіляти виробничі ресурси між ними найбільш оптимальним чином. Брак ресурсів для виконання якого-небудь критичного етапу – часта причина затримки виконання всього проекту.

Тривалість етапів зазвичай повинна бути не менше тижня. Якщо вона буде менша, то виявиться нижчою за точність тимчасових оцінок етапів, що може привести до частого перегляду графіка робіт. Також доцільно (в аспекті управління проектом) встановити максимальну тривалість етапів, що не перевищує 8 або 10 тижнів. Якщо є етапи, що мають велику тривалість, їх слід розбити на етапи меншої тривалості.

При розрахунку тривалості етапів потрібно враховувати, що виконання будь-якого етапу не обійдеться без великих або маленьких проблем і затримок. Розробники можуть допускати помилки або затримувати свою роботу, техніка може вийти з ладу або апаратні чи програмні засоби підтримки процесу розробки можуть поступити із запізненням. Якщо

проект інноваційний і технічно складний, це стає додатковим чинником появи непередбачених проблем і збільшення тривалості реалізації проекту в порівнянні з первинними оцінками.

Графік робіт за проектом зазвичай представляється у вигляді набору діаграм і графіків, що показують розбиття проектних робіт на етапи, залежності між етапами та розподіл розробників по етапах (рис. 10.1).

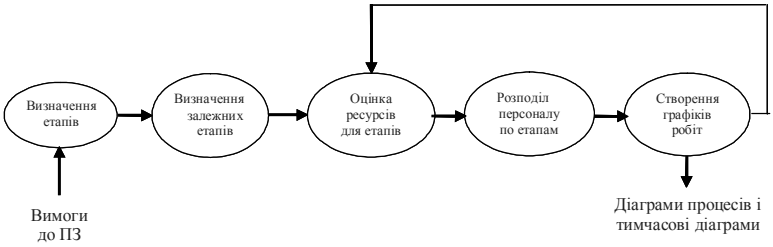


Рис. 10.1. Процес складання графіка робіт

Тимчасові та мережеві діаграми

Тимчасові та мережеві діаграми корисні для представлення графіка робіт. Тимчасова діаграма показує час початку та закінчення кожного етапу і його тривалість. Мережева діаграма відображає залежності між різними етапами проекту. Ці діаграми можна створити автоматично за допомогою програмних засобів підтримки управління на основі інформації, закладеної в базі даних проекту.

Розглянемо етапи якогось проекту (табл. 10.1), з якого видно, що етап Т3 залежить від етапу Т1. Це означає, що етап Т1 повинен завершитися раніше, ніж почнеться етап Т3. Наприклад, на етапі Т1 виконується аналіз створюваного програмного продукту, а на етапі Т3 – проектування системи.

На основі приведених значень тривалості етапів і залежності між ними будується мережевий графік послідовності етапів (рис. 10.2). На цьому графіку видно, які роботи можуть виконуватися паралельно, а які повинні виконуватися послідовно одна за одною. Етапи позначені прямокутниками. Контрольні відмітки та контрольні проектні елементи показані у вигляді овалів і позначені (табл. 3) буквою М з відповідним номером. Дати на даній діаграмі відповідають початку виконання етапів. Мережеву діаграму слід читати зліва направо й зверху вниз.

Таблиця 3. **Етапи проекту**

Етап	Тривалість (дні)	Залежність
Т1	8	
Т2	15	
Т3	15	Т1(М1)
Т4	10	
Т5	10	Т2, Т4 (М2)
Т6	5	Т1,Т2(М3)
Т7	20	Т1(М1)
Т8	25	Т4 (М5)
Т9	15	Т3, Т6 (М4)
Т10	15	Т5, Т7 (М7)
Т11	7	Т9 (Мб)
Т12	10	Т11(М8)

Будь-який етап не може початися, поки не виконані всі етапи на всіх шляхах, що ведуть від початку проекту до даного етапу. Наприклад, етап Т9 не може початися, поки не будуть завершені етапи Т3 і Т6. Відзначимо, що в даному випадку досягнення контрольної відмітки М4 говорить про те, що ці етапи завершені.

Мінімальний час виконання всього проекту можна розрахувати, підсумувавши в мережевій діаграмі тривалість етапів на щонайдовшому шляху від початку проекту до його закін-

чення (це так званий критичний шлях). У нашому випадку тривалість проекту складає 11 тижнів або 55 робочих днів. На рис. 10.3 критичний шлях показаний товщими лініями, ніж решта шляхів. Таким чином, загальна тривалість реалізації проекту залежить від етапів робіт, що знаходяться на критичному шляху. Будь-яка затримка в завершенні будь-якого етапу на критичному шляху приведе до затримки всього проекту.

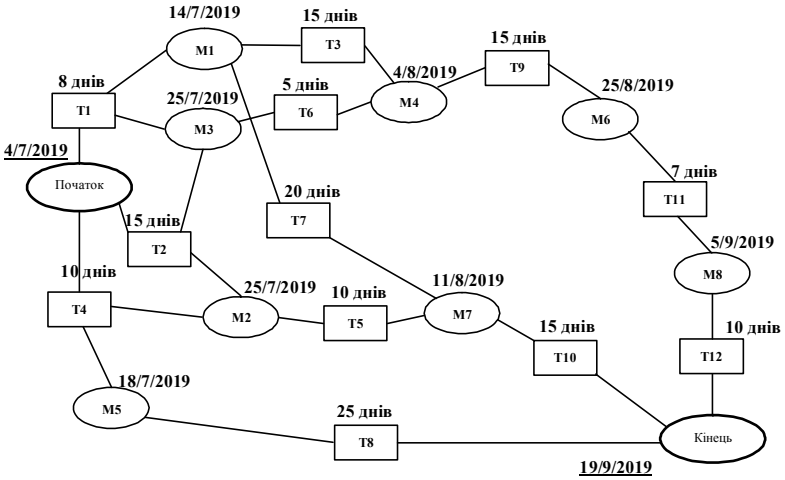


Рис. 10.2. Мережева діаграма етапів

Затримка в завершенні етапів, що не входять в критичний шлях, не впливає на тривалість всього проекту до тих пір, поки сумарна тривалість цих етапів (з урахуванням затримок) на якому-небудь шляху не перевищить тривалості робіт на критичному шляху. Наприклад, затримка етапу Т8 на термін, менший 20 днів, ніяк не впливає на загальну тривалість проекту. На рис. 10.3 представлена тимчасова діаграма, на якій показані можливі затримки на кожному етапі.

Мережева діаграма дозволяє побачити в залежності етапів значущість того або іншого етапу для реалізації всього проекту. Увага до етапів критичного шляху часто дозволяє знайти способи їх зміни з тим, щоб скоротити тривалість всього проекту. Менеджери використовують мережеву діаграму для розподілу робіт.

На рис. 10.3 показано інше представлення графіка робіт. Це тимчасова діаграма (іноді звана на ім'я її винахідника діаграмою Ганта), яка може бути побудована програмними засобами підтримки процесу управління.

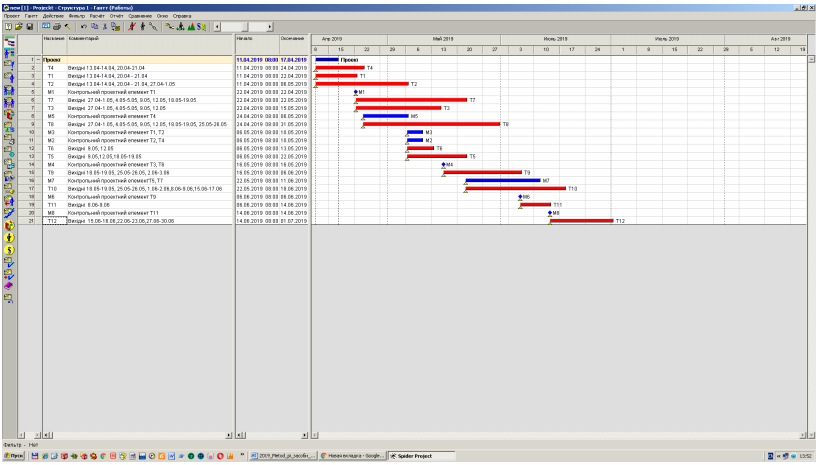


Рис. 10.3. Тимчасова діаграма тривалості етапів

Вона показує тривалість виконання кожного етапу та їх можливі затримки (показані затіненими прямокутниками), а також дати початку та закінчення кожного етапу. Етапи критичного шляху не мають затінених прямокутників; це означає, що затримка із завершенням даних етапів приведе до збільшення тривалості всього проекту.

Первинний графік робіт неминуче містить які-небудь помилки або недоробки. У міру реалізації проекту розраховані оцінки тривалості виконання етапів робіт повинні порівнюватися з реальними термінами виконання цих етапів. Результати порівняння повинні використовуватися як основа для перегляду графіка робіт ще не реалізованих етапів проекту, зокрема для того, щоб спробувати зменшити тривалість етапів критичного шляху.

Завдання

За допомогою Microsoft Excel побудувати тимчасову діаграму етапів робіт для написання програми та заповнити табл. 4. Варіанти завдань (див. табл. 10.2) взяти за списком журналу відвідування занять.

Таблиця 4. **Етапи робіт**

Етапи робіт	Час виконання
1. Аналіз і формування вимог	
2. Проектування системи та програмного забезпечення	
3. Кодування та тестування програмних модулів	
4. Збірка та тестування системи	
5. Експлуатація та супровід системи	

ЛАБОРАТОРНА РОБОТА № 11

Тестування програмного забезпечення

Мета роботи: освоєння основних понять про вимоги, що пред'являються до програмного забезпечення, та демонстрація на прикладі різних способів представлення цих вимог.

Короткі теоретичні відомості

Тестування – невід'ємна складова процесу програмної інженерії, один з методів подальшого поліпшення якості розробленого програмного забезпечення системи за допомогою виявлення дефектів, що залишилися, не виявлених раніше іншими видами перевірок.

Процес тестування складається з декількох етапів:

– тестування компонентів;
– тестування модулів;
– тестування підсистем;
– тестування системи;
– приймальні випробування.

Тестування методом "чорного ящика"

Функціональне тестування, або тестування методом "чорного ящика", базується на тому, що всі тести ґрунтуються на специфікації системи або її компонентів. Система представляється як "чорний ящик", поведінку якого можна визначити тільки за допомогою вивчення її вхідних і відповідних вихідних даних. Інша назва цього методу – "функціональне тестування" – пов'язана з тим, що випробувач перевіряє не реалізацію ПЗ, а тільки його виконувані функції.

Області еквівалентності

Вхідні дані програм часто можна розбити на декілька класів. Вхідні дані, що належать одному класу, мають загальні властивості, наприклад це додатні числа, від'ємні числа, рядки без пробілів і тому подібне. Зазвичай для всіх даних з якогонебудь класу поведінка програми однакова (еквівалентна). Через це такі класи даних іноді називають областями еквівалентності. Один з систематичних методів виявлення дефектів полягає у визначенні всіх областей еквівалентності, що обробляються програмою. Контрольні тести розробляються так, щоб вхідні та вихідні дані лежали в межах цих областей.

Структурне тестування

Метод структурного тестування припускає створення тестів на основі структури системи та її реалізації. Такий підхід іноді називають тестуванням методом "білого ящика", "скляного ящика" або "прозорого ящика", щоб відрізняти його від тестування методом "чорного ящика".

Як правило, структурне тестування застосовується до відносно невеликих програмних елементів, наприклад, до підпрограм або методів, що асоціюються з об'єктами. При такому підході випробувач аналізує програмний код і для отримання тестових даних використовує знання про структурну компоненту. Наприклад, з аналізу коду можна визначити, скільки контрольних тестів потрібно виконати для того, щоб в процесі тестування всі оператори виконалися принаймні один раз.

Тестування гілок

Це метод структурного тестування, при якому перевіряються всі незалежно виконувані гілки компоненту або програми. Якщо виконуються всі незалежні гілки, то й усі оператори

повинні виконуватися принаймні один раз. Більш того, всі умовні оператори тестуються як з дійсними, так і з помилковими значеннями умов. У об'єктно-орієнтованих системах тестування гілок використовується для тестування методів, що асоціюються з об'єктами.

Після того, як протестовані всі окремі програмні компоненти, виконується збирання системи, внаслідок чого створюється часткова або повна система. Процес інтеграції системи включає збирання й тестування отриманої системи, в ході якого виявляються проблеми, що виникають при взаємодії компонентів. Тести, перевіряючи збірку системи, повинні розроблятися на основі системної специфікації, причому тестування збірки слід починати відразу після створення працездатних версій компонентів системи.

Завдання

Ознайомитися з методами тестування програмного забезпечення, які використовуються для виявлення помилок і дефектів. Протестувати створену програму (лабораторна робота № 9) методами "чорного" та "білого" ящиків.

ЛАБОРАТОРНА РОБОТА № 12

Оцінка вартості програмного продукту

Мета роботи: ознайомлення з різними методами оцінки вартості та витрат, необхідних для створення програмного продукту.

Короткі теоретичні відомості

Оцінка вартості проекту та планування графіка робіт проводяться паралельно. Проте деякі попередні розрахунки повинні бути виконані на ранній стадії, ще до початку розробки точного плану проекту. Такі розрахунки необхідні для затвердження бюджету проекту або для виставляння ціни замовникові.

Як тільки проект починає діяти, всі розрахунки повинні регулярно оновлюватися. Це допомагає планувати роботу та сприяє ефективному використанню засобів. Якщо фактичні витрати значно перевищують плановані, менеджерові необхідно зробити які-небудь дії. Це може бути перерахування додаткових коштів на проект або зміна майбутніх етапів робіт відповідно до фактичного бюджету.

Зазвичай для оцінки проекту по розробці програмного забезпечення використовуються три параметри:

– вартість апаратних засобів і програмного забезпечення, включаючи їх обслуговування;

– витрати на відрядження та навчання;

– витрати на персонал, в основному на залучення з боку фахівців з програмного забезпечення.

Оцінка вартості повинна бути об'єктивною, щоб дати компанії-розробникові достатньо точний прогноз собівартості

проекту. Якщо собівартість розраховується для включення в комерційну пропозицію замовникові, слід ухвалити рішення про те, яку ціну призначити за проект. Традиційно в ціну продукту включають витрати виробництва плюс пропонований прибуток. Проте визначити співвідношення собівартості проекту й ціни, що виставляється замовникові, не завжди просто.

На визначення ціни програмного продукту можуть вплинути організаційні, економічні, політичні та комерційні міркування. Таким чином, взаємини ціни таі собівартості зовсім не так прості, як може показатися.

Чинники, що впливають на вартість програмного продукту:

– можливості ринку ПЗ;

– неможливо врахувати всі чинники, що впливають на вартість;

– умови контракту;

– зміна вимог;

– фінансова стабільність.

Існує безліч методів оцінки вартості програмного продукту, які слід використовувати паралельно. Якщо отримані результати мають великі відмінності, значить, для аналізу використана недостатня або невідповідна інформація.

Методи оцінювання

Не існує простого методу визначення майбутніх витрат, необхідних для розробки програмного продукту. Початкове оцінювання можна провести, ґрунтуючись на певних призначених для користувача вимогах високого рівня. Але заздалегідь не відомо, які технології застосовуватимуться при розробці ПЗ і які комп'ютери використовуватимуться. Також

неможливо передбачити, які люди працюватимуть над проектом і які у них будуть навики та досвід. Це показує, що надзвичайно важко провести точну оцінку вартості проекту на початкову етапі. Більш того, основна проблема в оцінці собівартості проектів полягає в низькій точності вживаних методів оцінювання.

Не дивлячись ні на що, організації-розробники обов'язково повинні оцінювати витрати на розробку та собівартість програмного продукту. Для цього можна застосовувати методи, описані в табл. 5.

Таблиця 5. **Методи оцінки собівартості**

Метод	Опис
Алгоритмічне моделювання собівартості	Метод заснований на аналізі статистичних даних про раніше виконані проекти, при цьому визначається залежність собівартості проекту від якого-небудь кількісного показника програмного продукту (звичайно це розмір програмного коду). Проводиться оцінка цього показника для даного проекту, після чого за допомогою моделі прогнозуються майбутні витрати
Оцінка експерта	Проводиться опит декількох експертів з технологій розробки ПЗ, що знають область застосування створюваного програмного продукту. Кожен з них дає свою оцінку собівартості проекту. Потім всі оцінки порівнюються й обговорюються. Цей процес повторюється до тих пір, поки не буде досягнута згода по остаточному варіанту попереднього кошторису проекту
Оцінка по аналогії	Цей метод використовується в тому випадку, коли в даній області застосування створюваного ПЗ вже реалізовані аналогічні проекти. У такому разі при оцінці витрат для порівняння беруться попередні проекти

Прожовж. табл. 5

Метод	Опис
Закон Паркінсона	Згідно цьому закону зусилля, витрачені на роботу, розподіляються рівномірно по виділеному на проект часу. Тут критерієм для оцінки витрат за проектом є людські ресурси, а не цільова оцінка самого програмного продукту. Якщо проект, над яким працює п'ять чоловік, повинен бути закінчений протягом 12 місяців, то витрати на його виконання обчислюються в 60 людино-місяців
Призначення ціни з метою виграти контракт	Витрати на проект визначаються наявністю тих засобів, які є у замовника. Тому собівартість проекту залежить від бюджету замовника, а не від функціональних характеристик створюваного продукту

Алгоритмічні моделі вартості застосовуються для порівняння різних інвестицій в цілях зниження вартості проекту. Це особливо важливо в тих випадках, коли приймаються вирішення відносно покупки апаратних або програмних засобів, або виникає необхідність найму нових співробітників з особливими навиками, потрібними для реалізації проекту.

Формула для обчислення алгоритмічної оцінки вартості записується таким чином:

$$\text{витрати} = A * \text{розмір } B * M,$$

де **А** – постійний коефіцієнт, який залежить від організації виконання проекту та типу програмного забезпечення, що розробляється, середнє значення А = 2.5; показник **розмір** може співвідноситися або з розміром коду програми, або з функціональною оцінкою, вираженою в кількості об'єктних або функціональних точок; показник ступеня **В** може варіюватися в межах від 1 до 1.5, він відображає обсяг робіт, потрібний для реалізації великих проектів; множник **М** відобра-

жає характеристики різних етапів розробки, а також характеристики створюваного продукту. Множник **M** є добутком семи показників, що характеризують проект і процес створення ПЗ, а саме: надійність і рівень складності системи (RCPX), що розробляється, повторне використання компонентів (RUSE), складність платформи розробки (PDIF), можливості персоналу (PERS), досвід персоналу (PREX), графік робіт (SCED) і засобу підтримки (FCIL). Це дозволяє провести оцінювання по шестибальній шкалі, де число 1 відповідатиме найменшим значенням цих показників, а число 6 – найвищим значенням. З іншого боку, їх можна обчислити шляхом комбінування значень більш деталізованих показників, які використовуються на архітектурному для поста рівні.

Таким чином, **оцінка витрат** обчислюється за наступною формулою:

$$PM = A * \text{розмір } B * M + PMm,$$

де M = PERS *RCPX * RUSE * PDIF * PREX * FCIL * SCED.

Останній доданок у формулі (PMm) позначає чинник, використовуваний у випадках, коли значна частина коду генерується автоматично. При цьому частину коду завжди потрібно вводити вручну, але рівень продуктивності все одно буде вищий, ніж при повністю ручному введенні. Витрати PMm розраховуються окремо по приведеній нижче формулі, а потім додаються до оцінки витрат на введений уручну код:

$$PMm = (ASLOC * (AT/100))/ ATPROD,$$

де ASLOC – це кількість рядків коду, проведених автоматичним способом, ATPROD – рівень продуктивності автоматичної генерації коду. Проте слід відмітити, що потрібно також виконання певних робіт для узгодження генерованого коду

рештою частини системи. Обсяг цих робіт залежить від відсотка коду, що автоматично генерований у всій системі (коефіцієнт АТ). Фактично продуктивність залежить від кількості створених програмних модулів. Чим менше обсяг згенерованого коду, тим більший обсяг робіт необхідний для інтеграції його з іншими кодами системи.

Оцінка вартості ПЗ використовується різними моделями. Однією з найпопулярніших, відкритих і добре документованих моделей оцінки вартості ПЗ є COCOMO (Constructive COst MOdel – конструктивна модель вартості), розроблена Баррі Боем [22].

У формулах моделі COCOMO використовуються деякі припущення. Вихідні інструкції кінцевого продукту включають в себе всі (крім коментарів) рядки коду, оброблюваного комп'ютером. Початок життєвого циклу проекту збігається з початком розробки продукту, закінчення – збігається з закінченням приймального тестування, завершальною стадією інтеграції та тестування. (Праця та час, що витрачаються на аналіз вимог, оцінюються окремо, як додатковий відсоток від розробки в цілому.)

Види діяльності включають в себе тільки роботи, спрямовані безпосередньо на виконання проекту, в них не входять звичайні допоміжні види діяльності, такі як адміністративна підтримка, технічне забезпечення та капітальне обладнання. Людино-місяць складається з 152 годин. Проект управляється належним чином, в ньому використовуються стабільні вимоги.

Життєвий цикл у моделі COCOMO складається з п'яти основних стадій: планування та визначення вимог, проектування продукту, детальне проектування, кодування і тестування окремих модулів, інтеграція і тестування.

Модель COCOMO дає рекомендації з розподілу робіт і часу по основних стадіях традиційної "водоспадної" моделі.

Ці рекомендації в деякій мірі залежать від варіанту й масштабу. Модель COCOMO дозволяє оцінити роботу та час, необхідні для отримання рішення (розробки продукту за допомогою ітерацій і тестування). Витрати на формулювання проблеми (планування та визначення вимог) оцінюються у вигляді додаткового відсотка від і понад роботи та часу, що витрачається на розробку.

Таблиця 6. **Параметри моделі COCOMO, що характеризують проект**

Ідентифікатор	Уточнюючий фактор робіт	Діапазоні зміни параметра	Потенційний вплив
RELY	Необхідна надійність	0.75–1.40	1.87
DATA	Розмір бази даних	0.94–1.16	1.23
CPLX	Складність продукту	0.70–1.65	2.36
TIME	Обмеження часу виконання	1.00–1.66	1.66
STOR	Обмеження обсягу основної пам'яті	1.00–1.56	1.56
VIRT	Мінливість віртуальної машини	0.87–1.30	1.49
TURN	Час реакції комп'ютера	0.87–1.15	1.32
ACAP	Здібності аналітика	1.46–0.71	2.06
AEXP	Знання додатків	1.29–0.82	1,57
PCAP	Здібності програміста	1.42–0.70	2.03
VEXP	Знання віртуальної машини	1.21–0.90	1.34
LEXP	Знання мови програмування	1.14–0.95	1.20
MODP	Використання сучасних методів	1.24–0.82	1.51
TOOL	Використання програмних інструментів	1.24–0.83	1.49
SCED	Необхідні терміни розробки	1.23–1.10	1.23

Розглянемо як приклад вбудовану систему для управління експериментом, який проводитиметься в космосі. Устаткування для експериментів, що проводяться в космосі, повинне

бути граничне надійним і підлягає строгим ваговим обмеженням.

Рис. 12.1. Проектні показники і характеристики, що враховуються при розрахунку вартості

Для оцінки показника ступеня використовуються перераховані нижче показники проекту. Вони відраховуються по шестибальній шкалі від найнижчого (5 балів) до найвищого (0 балів) рівня. Значення показників підсумовуються, сума ділиться на 100, результат додається до 1,01, після чого виходить значення показника ступеня.

Показники проекту

– Новизна проекту. Це новий проект для організації, даний показник має низький рівень (оцінюється в 4 бали);

– гнучкість процесу розробки. Немає втручання замовника – рівень показника дуже високий (оцінюється в 1 бал);

– аналіз архітектури системи та ризик. Аналіз не був проведений – рівень даного показника дуже низький (5 балів);

– згуртованість команди. Команда розробників нова, інформація про неї відсутня – рівень цього показника оцінюється як звичайний (3 бали);

– рівень розвитку процесу розробки. Певне управління проектом має місце – показник оцінюється як звичайний (3 бали).

Сума значень всіх цих показників складає 16 балів, тому значення показника ступеня буде рівне 1,17.

Проектні характеристики, використовувані для уточнення попередньої оцінки витрат на архітектурному для поста рівні, розбиваються на чотири групи:

– характеристики програмного продукту, які визначаються системними вимогами;

– характеристики апаратних засобів, які є обмеженнями, що накладаються на ПЗ, яке розробляється, вибраною платформою обчислювальних засобів;

– характеристики персоналу, які враховують досвід і можливості фахівців, що працюють над проектом;

– характеристики проекту, що враховують певні параметри та показники проекту розробки ПЗ.

Вартість проекту складається з трьох компонентів:

– вартість цільових апаратних засобів, на яких функціонуватиме система, що розробляється;

– вартість платформи (обчислювальна техніка плюс програмне забезпечення), використовуваної для розробки системи;

– Вартість витрат на розробку системи.

У табл. 5 показані апаратні та програмні засоби, які забезпечують реалізацію характеристик A–E, представлених на рис. 12.1. Вартість даного проекту оцінюється в 45 людино-місяців, а вартість витрат на один людино-місяць складає $15000.

Вартість програмного продукту (SC) для таких моделей обчислюється наступним чином:

SC = оцінка витрат * RELY * TIME * STOR * TOOL *
* LTEX * $15000,

де RELY – необхідна надійність системи; TIME – показники, що обмежують час виконання; STOR – обмеження об'єму пам'яті; TOOL – використання допоміжних програмних засобів; LTEX – досвід застосування даної мови програмування та засобів розробки.

Приклад розрахунку системи для управління експериментом, який проводитиметься в космосі методом оцінки вартості і витрат за допомогою моделі COCOMO, показаний у табл. 7.

Завдання

Ознайомитися з різними методами оцінки вартості та витрат, необхідних для створення програмного продукту. Розрахувати як зразок вартість свого програмного продукту.

Таблиця 7. **Розрахунок системи**

Характе-ристика	RELY	STOR	TIME	TOOL	LTEX	Загальні витрати	Вартість ПЗ	Вартість апаратних засобів	Підсумкова вартість
А	1,39	1,06	1,11	0,86	1	63	949 393	100 000	1 049 393
Б	1,39	1	1	1,12	1,22	88	1 313 550	120 000	1 402 025
В	1,39	1	1,11	0,86	1	60	895 653	105 000	1 000 653
Г	1,39	1,06	1,11	0,86	0,84	51	769 008	100 000	897 490
Д	1,39	1	1	0,72	1,22	56	844 425	220 000	1 044 159
Е	1,39	1	1	1,12	0,84	57	851 180	120 000	1 002 706

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

Рекомендована література

1. Кознов, Д.В. Методика обучения программной инженерии на основе карт памяти. Системное программирование / Вып. 3 ; под ред. А.Н. Терехова и Д.Ю. Булычева. – СПб.: Изд. СПбГУ, 2008. – с. 121–140. – www.sysprog.info.

2. Липаев, В.В. Программная инженерия. Методологические основы : учебник / В.В. Липаев; Гос. ун-т – Высшая школа экономики. – М.: ТЕИС, 2006. – 608 с.

3. Соммервилл Иан. Инженерия программного обеспечения, 6-е издание : пер. с англ. – М.: Издательский дом "Вильямс", 2002. – 624 с.: ил. – Парал. тит. англ.

4. Лаврищева, Е.М., Петрухин, В.А. Методы и средства инженерии программного обеспечения : учебник. – М., 2006. – 304 с.

5. http://www.intuit.ru/department/se/inprogeng/1/.

6. http://www.ixbt.com/soft/mind-manager.shtml.

7. http://www.coreltuts.com/ru/tutorials/varianty-ispolzovaniya-mindjet-mindmanager.

8. https://uk.wikipedia.org/.

9. https://support.office.com/uk-ua/ .

10. http://www.corel.ru/.

11. https://www.quality-assurance-group.com/vydy-dokumentatsiyi-u-roboti-testuvalnyka/.

12. https://www.istqb.org/.

13. http://tmguru.ru/.

14. http://softwaretestingfundamentals.com/test-plan/.

15. http://www.protesting.ru/.

16. http://iqa.com.ua/.

17. https://veraksoff.info/.

18. https://it.wikireading.ru/58320.
19. https://www.testingexcellence.com/test-strategy-and-test-plan/.
20. https://www.softwaretestinggenius.com/important-attributes-of-a-good-test-plan-and-how-to-create-it/.
21. https://qalight.com.ua/baza-znaniy/test-plan/.
22. https://www.lucidchart.com/pages/uml-class-diagram.
23. https://project.dovidnyk.info/index.php/home/upravlenie proektamiposozdaniyuprogrammnogoobespecheniya/130-modelo cenkistoimostisosomo.
24. https://strongqa.com/qa-portal/knowledge-base.
25. https://tapuniversity.wordpress.com/2009/08/28/requirements-traceability-matrix/.

Додаткова література

1. Andreas Spillner (2014), Software Testing Foundations.
2. Beizer Boris (1990), Software Testing Techniques (2nd edition), Van Nostrand Reinhold.
3. Black, R. (2017), Agile Testing Foundations, BCS Learning & Development Ltd: Swindon UK.
4. Graham, D. and Fewster, M. (2012), Experiences of Test Automation, Pearson Education: Boston MA.
5. Jorgensen, P. (2014), Software Testing, A Craftsman's Approach (4e), CRC Press: Boca Raton FL.
6. Kaner, C., Padmanabhan, S. and Hoffman, D. (2013), The Domain Testing Workbook, Context-Driven Press.
7. Kramer, A., Legeard, B. (2016), Model-Based Testing Essentials: Guide to the ISTQB Certified Model-Based Tester: Foundation Level, Wiley.
8. Shull, F., Rus, I., Basili, V. July 2000, "How Perspective-Based Reading can Improve Requirement Inspections", IEEE Computer.

9. Wiegers, K. (2002), Peer Reviews in Software, Pearson Education: Boston MA.

10. Lorin W. Anderson, David R. Krathwohl (eds.) (2001) A Taxonomy for Learning, Teaching and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, Allyn & Bacon.

11. Carol M. Barnum (2011), Usability Testing Essentials, Elsevier.

12. John Brooke, SUS – A 'Quick and Dirty' Usability Scale, in Patrick W. Jordan, Bruce Thomas, Bernard A. Weerdmeester, Ian L. McClelland (eds.), Usability evaluation in industry, Taylor & Francis.

13. Malcolm Gladwell (2008), Outliers – The Story of Success,, Little, Brown and Company.

14. Rex Hartson, Pardha S. Pyla (2012), The UX Book, Morgan Kaufman.

15. Jon A. Krosnick, Stanley Presser (2010), Question and Questionnaire Designin Peter V. Marsden, James D. Wright (eds.), Handbook of Survey Research, Second Edition, Emerald Group Publishing.

16. Steve Krug (2010), Surgery Made Easy, New Riders.

17. Michael C. Medlock, Dennis Wixon, Mark Terrano, Ramon L. Romero, Bill Fulton (2002), Using the RITE method to improve products: A definition and a case study, Usability Professionals Association 2002 Conference, Orlando Florida.

18. Rolf Molich (2007), Usable Web Design, Nyt Teknisk Forlag.

19. Rolf Molich, Kasper Hornb?k, Steve Krug, Josephine Scott, Jeff Johnson (2008), Recommendations on Recommendations, User Experience Magazine, Issue 4.

20. Adam Goucher and Tim Reilly, editors (2009), Beautiful Testing: Leading Professionals Reveal How They Improve Software, O'Reilly Media.

21. Jakob Nielsen, Heuristic Evaluation, in Jakob Nielsen, Robert L. Mack (eds.) (1994), Usability Inspection Methods, John Wiley & Sons.

22. Tomer Sharon (2012), It's Our Research: Getting Stakeholder Buy-in for User Experience Research Projects, Morgan Kaufman.

23. Chauncey Wilson (2007), Designing Useful and Usable Questionnaires: You Can't Just "Throw a Questionnaire Together",interactions.

Додаток

Приклад оформлення лабораторної роботи

					ЛР 01-12.121.1157.01		
Змін.		*№ докум.*	*Підпис*	*Дата*			
Розробив					Лабораторні роботи з дисципліни “Основи програмної інженерії ”	*Літ.*	*Арк.* 1 / *Аркушів*
Перевірив		Карпова С.О.					
						ХФ НУК	

Лабораторна робота №1

Тема: Побудова карти пам'яті (діаграми зв'язку).
Ознайомлення з програмою MindManager

Мета роботи: вивчення і застосування діаграм зв'язків як засобу зображення процесу загального системного мислення за допомогою схем; ознайомлення з програмою MindManager та її елементами.

Короткі теоретичні відомості

Діаграма зв'язків, відома також як інтелект-карта (психолог Тоні Бьюзен запропонував методику візуального представлення процесу мислення та структуризації інформації – інтелект-карти (mind maps, карти розуму, ментальні карти), що дозволяє перевести процеси мислення на якісно новий рівень) – спосіб зображення процесу загального системного мислення за допомогою схем. Також може розглядатися як зручна техніка альтернативного запису.

Діаграма зв'язків реалізується у вигляді деревовидної схеми, на якій зображені слова, ідеї, завдання або інші поняття, зв'язані гілками, що відходять від центрального поняття або ідеї. В основі цієї техніки лежить принцип "радіантного мислення", що відноситься до асоціативних розумових процесів, відправною точкою або точкою прикладання яких є центральний об'єкт. (Радіант – точка небесної сфери, з якої якби виходять видимі шляхи тіл з однаково направленими швидкостями, наприклад, метеорів одного потоку). Це показує нескінченну різноманітність можливих асоціацій і, отже, невичерпність можливостей мозку. Подібний спосіб запису дозволяє діаграмам зв'язків необмежено рости та доповнюватися. Діаграми зв'язків використовуються для створення, візуалізації, структуризації і класифікації ідей, а також як засіб для навчання, організації, вирішення завдань, ухвалення рішень, при написанні статей.

Одним з визнаних лідерів в області створення комп'ютерних інтелект-карт є програма MindManager – комерційне ПЗ для управління

картами думок, розроблене Mindjet. Mindjet описує карти, створені за допомогою MindManager як "бізнес-карти" для використання на підприємствах, замінюючи ними рукописні карти пам'яті.

Карти MindManager можуть бути експортовані в Microsoft Word, PowerPoint, Visio і Project, збережені як веб-сторінки чи PDF-документи.

Завдання

Ознайомитися з інтерфейсом програми MindManager. Побудувати за допомогою MindManager діаграму зв'язку двох рівнів для процесу "Створення програмного забезпечення"

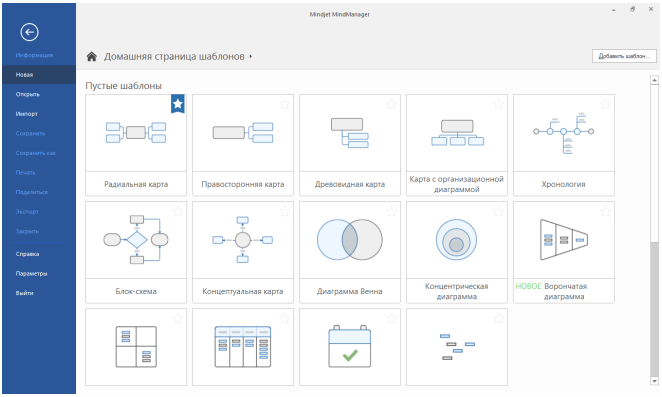


Рисунок 1 - Вікно створення шаблону в середовищі MindManager

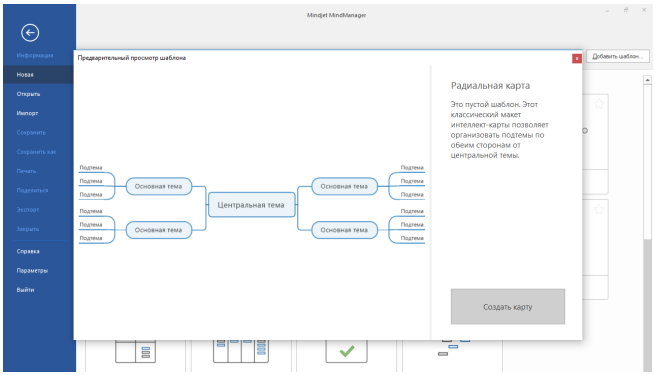


Рисунок 2 - Попередній перегляд шаблону

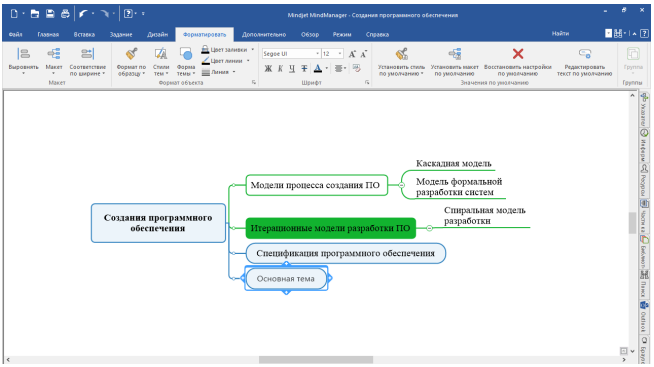


Рисунок 3 - Діаграмма зв'язку для процесу "Створення програмного забезпечення"

Висновок

Програмний продукт MindManager дуже простий у використанні, він замінює ручне створення карт пам'яті, у ньому легко і зручно працювати, набір необхідних для роботи інструментів більш ніж достатній. MindManager інтегрований з MS Office додатками (Word, Excel, Outlook, PowerPoint). Використання MindManager може допомогти в навчанні, організації, запам'ятовуванні, веденні переговорів, тренінгів, зустрічей та нарад, в узагальненні й плануванні, а також у вирішенні багатьох інших завдань, різноманіття яких таке ж велике.

ЗМІСТ

Навчальне видання

ДУДЧЕНКО Олег Миколайович
КАРПОВА Світлана Олегівна

ОСНОВИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Методичні вказівки
до виконання лабораторних робіт

Комп'ютерне складання та верстання *В. В. Москаленко*
Коректор *М. О. Паненко*

Формат 60×84/16. Ум. друк. арк. 4,8. Тираж 15 прим. Вид № 19. Зам. № 108

Видавець і виготівник Національний університет кораблебудування
імені адмірала Макарова
просп. Героїв України, 9, м. Миколаїв, 54025
E-mail : publishing@nuos.edu.ua
Свідоцтво суб'єкта видавничої справи ДК № 6402 від 19.09.2018 р.