

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Є. В. Левус, Т. А. Марусенкова, О. О. Нитребич

# ЖИТТЄВИЙ ЦИКЛ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Навчальний посібник

*Рекомендувала Науково-методична рада  
Національного університету “Львівська політехніка”*

Львів  
Видавництво Львівської політехніки  
2017

УДК 519.8(07)

Л 52

Рецензенти:

Федасюк Д. В., доктор технічних наук, професор, проректор з науково-педагогічної роботи Національного університету "Львівська політехніка";

Савула Я. Г., доктор фізико-математичних наук, професор, завідувач кафедри прикладної математики Львівського національного університету імені Івана Франка;

Рак Т. Є., доктор технічних наук, доцент, проректор з науково-дослідної роботи Львівського державного університету безпеки життєдіяльності

*Рекомендувала Науково-методична рада  
Національного університету "Львівська політехніка"  
як навчальний посібник для студентів спеціальності  
"Інженерія програмного забезпечення"  
(протокол № 24 від 21.12.2016 р.)*

Левус С. В.

Л 52 Життєвий цикл програмного забезпечення: навч. посібник /  
Є. В. Левус, Т. А. Марусенкова, О. О. Нитребич. - Львів : Видавництво  
Львівської політехніки, 2017. - 208 с.  
ISBN 978-966-941-098-6

Навчальний посібник призначений для вивчення розділу "Життєвий цикл програмного забезпечення" дисципліни "Вступ до інженерії програмного забезпечення". У посібнику, крім теоретичних відомостей, наведено чотири лабораторні роботи за основними етапами життєвого циклу: аналізу і визначення вимог, проектування й програмної реалізації, тестування, супроводу. Понад 100 тестових завдань, наведених у посібнику, є важливим засобом підготовки студентів до поточного й семестрового контролю.

Видання призначено для студентів спеціальності "Інженерія програмного забезпечення", а також може використовуватися для інших спеціальностей галузі знань "Інформаційні технології" для вивчення дисциплін, пов'язаних із розробленням програмного забезпечення.

УДК 519.8 (07)

ISBN 978-966-941-098-6

© Левус С. В., Марусенкова Т. А,

Нитребич О. О., 2017

© Національний університет

"Львівська політехніка", 2017

## ПЕРЕДМОВА

“Інженерія програмного забезпечення” — достатньо молодий в Україні освітній фах, виокремлений зі спеціальності “Комп’ютерні науки”, поява якого зумовлена потребою сучасної індустрії програмного забезпечення.

Термін “Інженерія програмного забезпечення” є перекладом відповідного англомовного - “Software Engineering” (SE), який був озвучений вперше в 1968 році на конференції НАТО з питань програмного забезпечення. Аналогічно вживають терміни “програмна інженерія” й “технології програмного забезпечення”.

У 2016 році в Україні до переліку освітніх спеціальностей було внесено спеціальність з оновленою назвою “Інженерія програмного забезпечення” (на заміну не зовсім коректній назві “Програмна інженерія”), яка входить до галузі знань “Інформаційні технології”. Випускники спеціальності “Інженерія програмного забезпечення” отримують професійну кваліфікацію “фахівець з розроблення та тестування програмного забезпечення” і можуть працювати на посадах: програміст, тестувальник програмного забезпечення, архітектор програмних систем, аналітик, менеджер програмних проектів, інженер з якості програмних продуктів.

Переважно ця спеціальність зорієнтована на промислове програмування, яке пов’язане з усіма аспектами виробництва програмного забезпечення: від початкових стадій створення специфікації до підтримки системи після здавання її в експлуатацію.

Спеціальність “Інженерія програмного забезпечення” охоплює і типові питання інженерної діяльності (аналіз та визначення вимог до програмних систем, їх проектування, застосування технологій програмування, верифікація й тестування програмного забезпечення, експлуатація й супровід програмних систем), і питання зі сфер економіки, управління, організації виробничих процесів (менеджмент програмних проектів, основи комунікацій у групах розробників, документування етапів розроблення, маркетинг програмних продуктів, удосконалення процесів життєвого циклу програмного забезпечення).

Інженерія програмного забезпечення динамічно розвивається і доповнюється накопиченим досвідом фахівців: з’являються нові технології, платформи й середовища програмування, методології, методи й засоби автоматизації процесів виробництва програмних продуктів.

Навчальна дисципліна “Вступ до інженерії програмного забезпечення” має дати знання й розуміння інженерних фундаментальних основ галузі розроблення програмного забезпечення, однією з яких є життєвий цикл програмного забезпечення (Software Lifetime Cycle, SLC, англ.), або життєвий цикл розроблення програмного забезпечення (Software Development Lifetime Cycle, SDLC, англ.). Історично першим використовували термін SLC. З ростом складності інженерної діяльності вводили точніші поняття щодо конкретної категорії процесів розроблення програмного забезпечення - SDLC. У контексті посібника ці дві назви вважаються аналогічними.

# **МЕТОДИЧНІ РЕКОМЕНДАЦІЇ**

## **Загальна інформація**

Навчальний посібник призначений для студентів першого курсу першого освітнього рівня спеціальності “Інженерія програмного забезпечення” усіх форм навчання для вивчення розділу “Життєвий цикл програмного забезпечення” й виконання відповідних лабораторних робіт з дисципліни “Вступ до інженерії програмного забезпечення”. Може також використовуватися для студентів суміжних спеціальностей “Комп’ютерні науки й інформаційні технології”, “Системний аналіз”, “Прикладна математика”, “Комп’ютерна інженерія” для вивчення навчальних дисциплін, пов’язаних з програмним забезпеченням.

Дисципліна “Вступ до інженерії програмного забезпечення” належить до циклу професійної та практичної підготовки і є базовою для вивчення інженерних основ процесу програмного забезпечення.

Логічно-змістовими складовими навчальної дисципліни є розділи:

- “Означення інженерії програмного забезпечення”,
- “Життєвий цикл програмного забезпечення”,
- “Моделі життєвого циклу програмного забезпечення”,
- “Елементи моделювання програмного забезпечення”.

Пререквізитом дисципліни “Вступ до інженерії програмного забезпечення” є дисципліна “Основи програмування”. Також для виконання лабораторних робіт необхідні теоретичні знання та елементарні вміння роботи з пакетом програм Microsoft Office, отримані на уроках інформатики у школі.

У посібнику наведено усе необхідне для вивчення розділу дисципліни навчально-методичне забезпечення:

- теоретичні матеріали;
- список питань для самоперевірки знань;
- завдання до лабораторних робіт;
- індивідуальні варіанти завдань;

- тестові завдання для підготовки до поточного й семестрового контролю;
- теми й короткі теоретичні відомості до самостійної роботи;
- перелік питань для підготовки за темами самостійної роботи;
- перелік використаної й рекомендованої літератури.

Теоретичні матеріали за змістом поділяють на підрозділи. У додатках посібника наведено приклади й довідкову інформацію.

Наведені у посібнику чотири лабораторні роботи пов'язані між собою спільною частиною завдання стосовно розроблення конкретної програми. Програму переважно розробляють у межах лабораторної роботи з дисципліни “Основи програмування”.

Теми самостійної роботи пов'язані з основами роботи із програмним забезпеченням, необхідним для документування процесів й результатів основних етапів життєвого циклу програмного забезпечення.

## **Компетентності, які забезпечуються вивченням розділу “Життєвий цикл програмного забезпечення”**

Внаслідок вивчення розділу студент повинен бути здатним продемонструвати такі результати навчання:

- знання принципів та методів інженерії програмного забезпечення;
- вміння створювати документи, що описують основні результати етапів життєвого циклу програмного забезпечення;
- вміння застосовувати базові поняття інженерії програмного забезпечення й інших дисциплін комп’ютингу у процесах життєвого циклу програмного забезпечення.

Вивчення розділу передбачає формування та розвиток у студентів фахових компетентностей:

- здатність застосовувати знання і вміння інженерії вимог для аналізу предметної області та розроблення, верифікації і атестації специфікації вимог до програмного забезпечення;
- розуміння принципів людино-машинної взаємодії та здатність створювати програмні продукти з ефективним інтерфейсом користувача;

- здатність створювати технічну документацію для програмних систем відповідно до прийнятих стандартів та практик.

Результати вивчення розділу деталізують такі **програмні результати навчання**:

- здатність демонструвати знання та розуміння основ інформаційних технологій та математики як теоретичної бази для програмного забезпечення;
- здатність демонструвати знання інженерних основ у галузі програмного забезпечення, зокрема процесів життєвого циклу програмного забезпечення, їхніх моделей, стандартів інженерії програмного забезпечення;
- здатність демонструвати знання методів виявлення та аналізу вимог користувачів для визначення вимог до програмної системи, що розробляється;
- застосувати знання та розуміння для отримання та документування результатів основних етапів життєвого циклу програмного забезпечення;
- знання та розуміння для аналізу та специфікації вимог, а також їх верифікації.

**Місце навчальної дисципліни**  
**“Вступ до інженерії програмного забезпечення”**  
**серед інших дисциплін навчального плану спеціальності**  
**“Інженерія програмного забезпечення”**

“Інженерія програмного забезпечення” - достатньо широка область знань галузі “Інформаційні технології”.

Дисципліна “Вступ до інженерії програмного забезпечення” ознайомлює зі спеціальністю й забезпечує знання фундаментальних понять галузі програмного забезпечення, а саме інженерних аспектів цієї діяльності. Для вивчення цієї дисципліни необхідно вивчити усі розділи навчальної дисципліни “Основи програмування”.

Дисципліна є пререквізитом для багатьох інших дисциплін спеціальності, для яких рекомендується узгодження робочих програм.

Кореквізитом є навчальна дисципліна “Об'єктно-орієнтоване програмування”.

**Зв'язок дисципліни “Вступ до інженерії ПЗ”  
з іншими дисциплінами спеціальності  
“Інженерія програмного забезпечення”**

Попередні навчальні дисципліни	Наступні навчальні дисципліни
Основи програмування	<p>Людино-машинна взаємодія</p> <p>Аналіз вимог до програмного забезпечення</p> <p>Моделювання й аналіз програмного забезпечення</p> <p>Конструювання програмного забезпечення</p> <p>Основи командної розробки програмного забезпечення</p> <p>Якість і тестування програмного забезпечення</p> <p>Архітектура і проектування програмного забезпечення</p> <p>Менеджмент програмних проектів</p> <p>Економіка інженерії програмного забезпечення</p>

**Вимоги до виконання лабораторних робіт**

Основна мета виконання лабораторних робіт - засвоїти на практиці основні фази життєвого циклу програмного забезпечення на прикладі опису процесів створення власної програми.

Опис передбачає документування основних результатів процесів життєвого циклу програмного забезпечення, під час якого створюють такі документи:

- технічне завдання;
- блок-схеми алгоритмів й схематичне зображення структур даних;
- текст програми;
- звіти про тестування;
- прототипи графічного інтерфейсу користувача;
- інструкція для користувача.

До кожної лабораторної роботи додають перелік необхідних питань і визначень щодо теоретичних відомостей. Завдання виконують за індивідуальним варіантом. Кількість варіантів дає можливість у повному обсязі охопити академічну групу студентів. Для лабораторних робіт № 2 й № 4 завдання поділено на декілька частин. У разі необхідності можливе незалежне виконання цих частин як окремих робіт.

Для самоперевірки знань необхідно використовувати список контрольних питань, поданих після теоретичних матеріалів відповідних тем, а також тестові завдання, розміщені у відповідному розділі посібника.

Виконання лабораторних робіт передбачає також окрему самостійну роботу, яка полягає у вивченні основних можливостей і прийомів роботи офісних програм, зокрема текстового процесора MS-Word й редактора ділової графіки MS-Visio.

Для захисту лабораторних робіт виконують звіт у текстовому редакторі за такими вимогами: розмір шрифту основного тексту - 14 пт; міжрядковий інтервал - 1.25; абзацний відступ зліва - 1 см; перед- й післяабзацні міжрядкові інтервали - 0; вирівнювання основного тексту - двостороннє; заголовки відцентровані з накресленням - Bold, усі поля сторінки - 2 см, орієнтація аркушів книжкова.

Аркуші звіту формату А4 нумеруються без вказання номера на першій сторінці. За потреби звіт друкують, паперові аркуші скріплюють.

Складовими звіту є **титульний аркуш, тема, мета, теоретичні відомості, постановка завдання, отримані результати, висновки**. Не допускається використовувати матеріали теоретичних відомостей до лабораторних робіт для формування теоретичних відомостей власного звіту.

## **Рекомендована послідовність виконання лабораторної роботи**

1. Ознайомитися з темою й метою роботи.
2. Вивчити необхідний теоретичний матеріал та розглянути приклади, які реалізують аналогічні задачі до тематики лабораторної роботи.
3. Отримати індивідуальний варіант завдання. Проаналізувати його й скласти план виконання для дотримання термінів виконання та захисту роботи.
4. Ознайомитися з необхідним для виконання завдання програмним забезпеченням.
5. Створити необхідні файли за умовою завдання за допомогою необхідних програмних засобів.
6. Перевірити свої знання за списком контрольних питань та за допомогою відповідних тестів.
7. Захистити роботу за комп'ютером.

8. Проаналізувати отриманий результат й оформити письмовий звіт про роботу такого змісту:

- *тема та мета лабораторної роботи;*
- *теоретичні відомості;*
- *постановка завдання;*
- *отримані результати;*

• *висновки*, в яких необхідно зазначити: які знання отримано в процесі виконання цієї роботи; для чого призначена розроблена документація (вказати галузь її застосування).

### **Рекомендована система оцінювання**

№ роботи	Максимальна оцінка (відсоткова частина від загальної суми балів за дисципліну, %)	Час виконання	
		Аудиторні заняття, год	Самостійна робота, год
1	3	4	2
2	5	6	4
3	6	6	6
4	6	6	8
Разом	20	22	20

**Примітка.** Розподіл для студентів першого курсу денної форми навчання спеціальності “Інженерія програмного забезпечення”.

### **Розподіл годин за видами занять**

	Вид заняття	Години	Примітка
1	Лекції	12	Аудиторні години
2	Лабораторні роботи	22	Аудиторні години
3	Самостійна робота	20	Позааудиторні години
		20	
		18	
Разом		92	>50 % загального обсягу годин

**Примітка. Розподіл для навчальної дисципліни обсягом шість кредитів.**

## ПЕРЕЛІК СКОРОЧЕНЬ

- БД - база даних  
ІПЗ - інженерія програмного забезпечення  
ІТ - інформаційні технології  
ЖЦ ПЗ - життєвий цикл програмного забезпечення  
КІ - комп'ютерна інженерія  
КН - комп'ютерні науки  
ООП - об'єктно-орієнтоване програмування  
ОС - операційна система  
ПЗ - програмне забезпечення  
СУБД - система управління базами даних  
ТЗ - технічне завдання

# ТЕМА 1. АНАЛІЗ ТА ВИЗНАЧЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 1.1. Означення життєвого циклу програмного забезпечення

Інженерія програмного забезпечення/програмна інженерія (*англ.* Software Engineering) - це галузь інформаційних технологій (ІТ), завданням якої є створення методів розроблення якісного програмного забезпечення (ПЗ) в умовах обмежених часових та фінансових ресурсів.

Програмне забезпечення (*англ.* Software) - сукупність програм для обробки інформації і програмних документів, необхідних для експлуатації цих програм. Програмне забезпечення, що належить до категорії системи програмування, є інструментом для створення нового програмного забезпечення різного типу (рис. 1.1).



Рис. 1.1. Класифікація ПЗ

Своєю чергою, комп’ютерна програма - це набір інструкцій у вигляді слів, цифр, кодів, схем, символів чи у будь-якому іншому вигляді у формі, придатній для зчитування комп’ютером, які приводять його у дію для досягнення певного результату. Здебільшого комп’ютерну програму визначають як низку команд для комп’ютера, які становлять запис алгоритму (методу розв’язування задачі) однією з мов програмування. Альтернативним підходом до трактування програми є декларативна парадигма, коли програма є лише формалізованим описом самої задачі, а не методу її розв’язання.

Якісне ПЗ насамперед задоволяє вимоги замовника (користувачів) щодо функціональності, а також володіє додатковими властивостями, такими як надійність, зручність користування, ефективність, супроводжуваність. Програма може працювати правильно, але не бути необхідної якості. Наприклад, із двох версій програми, що відповідають вимогам стосовно виконуваних функцій, якісною вважають ту, яка працює швидше, не спричиняє аварійних ситуацій у випадку введення некоректних вхідних даних, має зручний інтерфейс користувача тощо.

Основною відмінністю програмування як абстрактної діяльності від інженерії ПЗ є обмеження стосовно витрат часу й задіяних фінансів. Час і фінанси є рівноправними, взаємозалежними ресурсами проектів з розроблення ПЗ.

Інженерія ПЗ (промислове програмування) зазвичай асоціюється з розробленням великих і складних програм колективами розробників. Становлення і розвиток цієї галузі діяльності зумовлені низкою проблем, пов'язаних із високою вартістю програмного забезпечення, складністю його створення, необхідністю управління і прогнозування процесів розроблення. Ця галузь поступово виокремлювалась серед інших комп'ютерних галузей (рис. 1.2).

#### До становлення ПЗ



#### Після становлення ПЗ

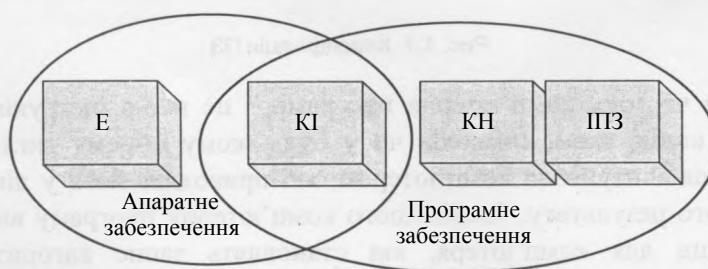


Рис. 1.2. Розвиток комп'ютерних галузей: Е - електроніка;  
КІ - комп'ютерна інженерія; КН - комп'ютерні науки;  
ІПЗ - інженерія програмного забезпечення

Інженерія ПЗ характеризується досить динамічним розвитком і продовжує інтенсивно розвиватися переважно відповідно до нових завдань побудови складних програмних систем для промисловості, урядового управління, військової галузі\* фінансової та банківської справи, комерції, медицини, освіти, сфери розваг.

Основний принцип інженерії програмного забезпечення (ІПЗ) полягає в тому, що програми створюють у результаті виконання декількох взаємопов'язаних етапів (аналіз та специфікація вимог, проектування, розроблення, тестування, впровадження, супровід). Ці етапи становлять життєвий цикл програмного забезпечення (англ. Software Development Lifetime Cycle). Термін “життєвий цикл виробництва”, який використовують у промисловості, запозичено та адаптовано для інженерії програмного забезпечення. Характер взаємозв'язаності окремих етапів життєвого циклу розроблення визначається моделлю життєвого циклу. Тобто життєвий цикл програмного забезпечення (ЖЦ ПЗ) вказує на всі можливі процеси розроблення ПЗ, а модель ЖЦ ПЗ - на необхідні для заданого проекту і, що важливо, на послідовність виконання цих процесів. Приклади деяких типових моделей ЖЦ ПЗ наведено у Додатках А, Б.

Життєвий цикл розбивають на окремі складові - процеси, а процеси - на підпроцеси, роботи, дії. Тобто відбувається ієрархічна декомпозиція життєвого циклу розроблення ПЗ.

Процес — сукупність дій і завдань, які мають на меті досягнення значимого результату. Процес - це важлива складова будь-якої інженерної діяльності. Кожен процес має чіткий опис, визначені вхідні й результати. Кажуть, що процес в інженерній діяльності має бути встановлений.

Повне встановлення процесу передбачає:

- Опис процесу - детальний опис дій і операцій процесу;
- Навчання процесу - проведення занять з персоналом з освоєння процесу;
- Введення метрик - встановлення кількісних показників процесу виконання;
- Контроль виконання - вимірювання метричних показників і оцінювання процесу виконання;
- Удосконалення - зміна процесу за змінних умов застосування.

Важливі базові процеси інколи називають етапами або фазами життєвого циклу. Серед основних процесів прийнято виокремлювати інженерні процеси і ті, що стосуються роботи із замовником.

До основних етапів життєвого циклу ПЗ належать (рис. 1.3):

1. Аналіз та специфікація вимог (початковий і фундаментальний етап, на якому збирають та аналізують вимоги замовника і подають їх у нотації, яка є зрозумілою і для замовника, і для виконавця. Okрім визначення вимог, на цьому етапі планують якість, оцінюють майбутні ризики тощо);
2. Проектування (перетворення вимог до програмної системи на послідовність проектних рішень, які визначають структуру й поведінку системи, а також інтерфейс користувача);
3. Кодування (програмна реалізація проектних рішень, власне розроблення програмного коду);
4. Тестування (перевірка програмного продукту на Наявність помилок і відповідність встановленим вимогам);
5. Експлуатація (безпосереднє використання ПЗ, а також дії з обслуговування системи під час її використання - консультації користувачів, вивчення їхніх побажань тощо);
6. Супровід (дії з керування модифікаціями, підтримка актуального стану та функціональності придатності, інсталяція нових та вилучення попередніх версій програмних систем у користувача).
7. Зняття системи з експлуатації (дії з поступового припинення використання програми, перехід до нової програми, навчання персоналу нової програми).



Рис. 1.3. Етапи життєвого циклу ПЗ

Окрім основних, існує багато додаткових, допоміжних процесів, пов'язаних не безпосередньо зі створенням продукту, а з організацією робіт та підтримкою основних чи й неосновних процесів. До таких процесів, наприклад, належать створення інфраструктури, документування, управління якістю, навчання, вирішення протиріч тощо.

Основні процеси життєвого циклу взаємодіють із допоміжними (супровідними) процесами, тоді як організаційні процеси діють протягом життєвого циклу і пов'язані з основними процесами.

У загальнену класифікацію етапів життєвого циклу ПЗ наведено на рис. 1.4. Детально класифікацію процесів ЖЦ ПЗ описано у міжнародних стандартах інженерії програмного забезпечення.



Рис. 1.4. Класифікація процесів ЖЦ ПЗ

Ключове завдання інженерії ПЗ - не просто використання технологій, а комбінування технологічних знань, людських ресурсів і процесів для досягнення бізнес-цілей організації. Це завдання вирішують на основі поняття ЖЦ ПЗ.

Під час розроблення ПЗ створюють багато артефактів. Артефакти - це створювані людиною інформаційні сутності - документи, які у доволі загальному сенсі беруть участь як вхідні дані й отримувані як результати

різних видів діяльності. Серед артефактів розроблення ПЗ - технічне завдання, задокументовані проектні рішення, прототипи інтерфейсу користувача, тексти програм, набори тестових даних, звіти про тестування, інструкція користувача та інші.

## 1.2. Аналіз та специфікація вимог

**Вимоги до програмного забезпечення** - набір заданих властивостей програмного забезпечення, яке буде розроблено, що стосуються функцій та якості програмного забезпечення. Вимоги визначають під час аналізу вимог та фіксують у специфікації вимог. Специфікація вимог до ПЗ (англ. Software Requirements Specification) - це структурований технічний документ, в якому записують функціонал (що робитиме система) та важливі властивості системи. Крім того, у документах міститься загальна інформація концептуального рівня про проект.

Визначення вимог є нетривіальним завданням, яке проводять, як правило, обговорюючи погляди замовника на систему з майбутніми її розробниками.

Етап аналізу та специфікації вимог є фундаментальним, адже без чіткого визначення вимог неможливо перейти на наступні етапи розроблення програмної системи. Саме правильно визначені вимоги дають уявлення про те, чого хоче замовник і як кінцевий користувач використовуватиме майбутній програмний продукт.

Вимоги - це те вихідне розуміння завдання розробниками, яке є основою всієї розробки.

Існують труднощі взаєморозуміння замовника і розробників через непорозуміння між програмістами та іншими людьми. По-перше, щоб добре розібратися, якою має бути система інформатизації певної діяльності чи бізнесу, треба попрацювати у відповідній галузі достатній час або якось інакше навчитися бачити проблеми цієї предметної області зсередини. По-друге, позначається специфічність програмування як сфери діяльності. Для більшості користувачів і замовників вкрай непросто сформулювати точне знання, яке потрібне програмістам. Наприклад, на питання, скільки типів аналізів існує у медичній лабораторії, лікар, подумавши, відповідає - 25. І вже потім випадково програміст уточнив, - а чи немас інших типів? Звичайно, є, - відповів лікар, -

тільки вони трапляються рідко, і можуть бути у певному сенсі якими завгодно. Спочатку він назвав лише типові, але, звичайно, інформаційна система повинна зберігати інформацію про всі аналізи, проведені в лабораторії.

Ще одним важливим поняттям в інженерії ПЗ є управління вимогами (англ. Software Requirements Management) - це безперервний процес, який містить визначення, документування, аналіз, відстеження, пріоритетизацію вимог та контроль над їхніми змінами. Мета управління вимогами полягає в тому, щоб переконатися, що програмне забезпечення відповідає потребам і очікуванням своїх клієнтів.

У 2008 році консалтингова компанія IAG Consulting проаналізувала дані понад 100 IT-компаній і надала звіт про вплив визначення вимог на успіх технічних проектів. У дослідженні брали участь лише великі фірми із прибутком понад 250 тис. доларів США. Середня вартість проектів, які розробляли ці компанії, становила 3 млн доларів США. Згідно зі звітом, компанії, які чітко не визначили вимог до проектів, витратили на 50 % більше часу та коштів для їхньої реалізації.

Отже, важливо чітко та своєчасно ідентифікувати всі вимоги до програмного продукту, а також постійно відстежувати зміни та уточнення з боку замовника. Розбіжності між уявленням про майбутню програмну систему з боку замовника (майбутнього користувача) й, відповідно, з боку розробника спочатку є переважно значними. Остаточним узгодженням відмінностей позбавляються й у письмовому вигляді ухвалюють остаточне рішення про те, якою має бути програмна система. Інакше немає змісту братися за розроблення ПЗ.

Схематично процес узгодження можна відобразити як графік на площині з осями, що задають витрати ресурсів і очікувану якість, зокрема функціональність, програмної системи (рис. 1.5). Стримувальним фактором для необґрунтованих вимог до програмної системи з боку замовника є вартість їх реалізації.

У зведенні знань з інженерії програмного забезпечення SWEBOK визначено такі види діяльності під час роботи з вимогами (Додаток В):

- Видобування вимог (англ. Requirements Elicitation), націлене на виявлення всіх можливих джерел вимог й обмежень на роботу системи і витяг вимог з цих джерел.
- Аналіз вимог (англ. Requirements Analysis), метою якого є усунення протиріч і неоднозначностей у вимогах, їх уточнення та систематизація, зокрема виявлення взаємозв'язків між вимогами.

- Опис вимог (англ. Requirements Specification). У результаті цієї діяльності вимоги повинні бути оформлені у вигляді структурованого набору документів і моделей, які можна систематично аналізувати, оцінювати з різних позицій, внаслідок чого документ затверджують як офіційне формулювання вимог до системи.

- Валідація вимог (англ. Requirements Validation), яка вирішує завдання оцінки зрозуміlosti сформульованих вимог та їхніх характеристик, необхідних, щоб розробляти ПЗ на їх основі, насамперед, несуперечності і повноти, а також відповідності корпоративним стандартам на технічну документацію.

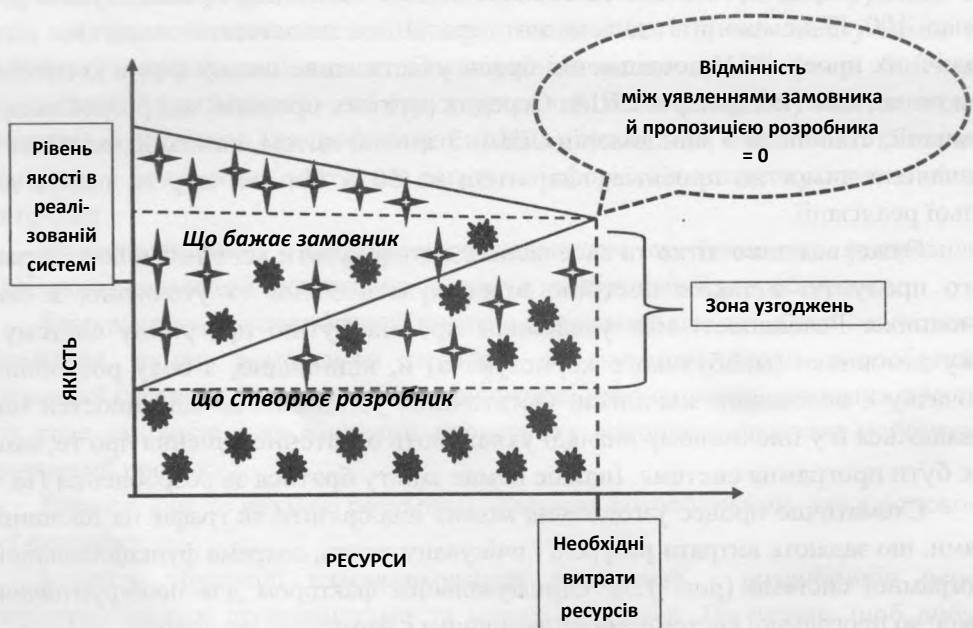


Рис. 1.5. Відмінністі між уявленнями замовника й пропозицією розробника

Всі вимоги до програмного забезпечення поділяють на функціональні та нефункціональні.

**Функціональні вимоги** (англ. Functional or Behavioral Requirements) регламентують функціонування або поведінку системи. Вони визначають те, що система повинна вміти робити. Функціональні вимоги відповідають на питання “що повинна робити система у тих чи інших ситуаціях”. Функціо-

нальні вимоги визначають основний “фронт робіт” команди розробників і встановлюють цілі, завдання та сервіси, які надає система замовнику.

Щодо виявлення функціональних вимог, то варто акцентувати увагу саме на властивості програми виконувати корисну дію, розв’язувати конкретну задачу. Тому “вихід з програми”, “відкриття інтерфейсного вікна” тощо не є варіантами використання програми й не можуть фіксуватися як функціональні вимоги.

Крім того, також важливі **нефункціональні** вимоги. Нефункціональні вимоги не є описом функцій системи. Цей вид вимог описує такі характеристики системи, як надійність, особливості поставки (наявність інсталлятора, документації), певний рівень якості. Такими можна вважати й вимоги до засобів і процесу розроблення системи, вимоги до портативності, відповідності стандартам тощо. Вимоги цього виду часто ставлять до всієї системи. На практиці, особливо фахівці-початківці, часто забувають про деякі важливі нефункціональні вимоги. Це часто додаткові обмеження або умови, з якими необхідно рахуватися під час розроблення програмного продукту, зокрема:

- аппаратні вимоги (необхідне обладнання, платформа тощо);
- продуктивність (наскільки ефективно функціонує програмне забезпечення, тобто потреби в обчислювальних ресурсах);
- документація (які документи потрібно створити для кінцевих користувачів);
- безпека (наприклад, резервне копіювання);
- заданий дизайн інтерфейсу користувача (переважно вигляди елементів графічного інтерфейсу).

Зазвичай вимоги використовують як засіб комунікації між різними зацікавленими сторонами (розробники, замовник, користувачі). Тому вимоги повинні бути простими й зrozуміlimi. Крім цього, усі вимоги до програмного забезпечення повинні мати такі властивості:

1. Коректність (без вживання технічного жаргону, вимоги повинні містити об’єктивні факти);
2. Необхідність (вимоги повинні відповідати певній потребі замовника або користувача);
3. Завершеність (вимоги повністю визначені в одному місці документа і присутня вся необхідна для розуміння інформація);
4. Верифікованість (здатність протестувати програмне забезпечення, щоб упевнитись, чи воно відповідає цим вимогам);

5. Недвозначність (можливість тільки однієї інтерпретації вимоги, зрозумілість);
  6. Послідовність (вимоги не суперечать одна одній).
- Вимога, яка відповідає перерахованим вище вимогам, є “хорошою” вимогою (правильно сформульованою).

*Таблиця 1.1*

### **Приклад “хороших” та “поганих” вимог**

“Погана” вимога	“Хороша” вимога
Основний фон головного вікна програми має бути синього кольору. (відтінків кольору є багато. Який код синього кольору?)	Код кольору основного тла головного вікна має бути #000080.
Після відкриття головної сторінки сайту має на деякий час відобразитись рекламне зображення. (“на трошки” - це на скільки?)	Після відкриття головної сторінки сайту має відобразитись зображення-реклама протягом 10с
Програмна система повинна відображати точні показники температури кожних 15 с (як точно?)	Програмна система повинна відображати показники температури з точністю $\pm 0,5$ °C кожних 15 с

Приклад “хорошої” вимоги:      “Програмна система для роботи з одновимірним масивом повинна реалізувати операції сортування (бульбашки, вставки), пошуку середнього елемента та суми парних елементів масиву. Розмірність масиву не більша за 50 і не менша за 10 елементів, кожен елемент є цілим числом у діапазоні [0, 100]”. Ці вимоги є коректними (зрозумілі виконавцю, граматично правильно написані), необхідними (вимоги описані коротко та чітко, немає зайвої інформації), завершеними (вимоги описані одна за однією, немає розриву думки, не виникає жодного додаткового запитання, адже вся інформація, необхідна для реалізації програми, присутня), верифікованістю (можна протестувати програму на основі вимог: перевірити мінімальну і максимальну довжини масиву; чи можуть бути елементами масиву не цілі числа, символи, цілі числа в межах від  $(-\infty; 0) \cup (100; \infty)$ ; чи реалізовані усі необхідні функції), недвозначністю (кожну вимогу можна однозначно інтерпретувати, адже чітко вказано алгоритми сортування, розмірність масиву, його довжина та тип елементів), послідовність (усі вимоги не суперечать одна одній).

Назвемо помилки, що трапляються при складанні технічних завдань та інших документів з вимогами.

- Опис можливих рішень замість вимог. Користувачеві не потрібно знати, як отримують необхідний йому результат. Важливо описати, що отримується в результаті роботи програми, а не як програма працює. Наприклад, не потрібно Описувати сам метод сортування замість вимоги “сформувати відсортований список студентів за рейтингом”.

- Нечіткі вимоги, які не допускають однозначної перевірки, залишають недомовленості, мають відтінок порад, обговорень, рекомендацій.

Наприклад, недоречно використовувати такі фрази: “Можливо, що має сенс реалізувати також.....”, “і т. д.”.

- Ігнорування аудиторії, для якої призначено представлення вимог.

Наприклад, якщо специфікацію складає інженер замовника, то часто трапляється надлишок інформації про обладнання, з яким повинна працювати програмна система, немає глосарію термінів і визначень основних понять, використовуються численні синоніми тощо. Або допущено занадто значний ухил у бік програмування, що робить таку специфікацію незрозумілою для всіх непрограмістів.

Вимоги до ПЗ можна документувати у текстовому або графічному вигляді. Документ опису вимог є основним документом, який визначає вимоги та порядок створення програмного забезпечення та прийняття його під час введення в експлуатацію.

Для опису вимог часто використовують такі типи документів:

- список вимог - найпростіший тип документа, в якому коротко описано усі основні вимоги до програмного продукту;
- прототип - макет системи, документ, який здебільшого використовують для визначення вимог до дизайну програмного продукту;
- сценарії використання - текстовий опис усіх способів взаємодії користувача з системою через деталізацію усіх кроків чи дій, необхідних для досягнення мети (приклад наведено у Додатку Г);
- діаграми прецедентів - діаграми, що зображають зв’язки між користувачами системи та її основними функціями (приклад наведено у Додатку Д);
- технічне завдання (специфікація вимог) - вихідний документ для нового програмного забезпечення, в якому формулюються основні цілі розроблення, список принципових вимог до продукту, визначено терміни та етапи розроблення і регламентуються характеристики готового до експлуатації програмного продукту (приклад структури документа наведено у Додатку Е);

Зазвичай найчастіше використовують для опису вимог саме **технічне завдання** (ТЗ), в розробленні якого беруть участь і представники замовника, і виконавці. Аналіз й визначення вимог здебільшого є ітеративним процесом. Кожну ітерацію для масштабного програмного проекту фіксують окремим документом (Додатки Ж, З).

### **1.3. Технічне завдання - документування результатів аналізу та визначення вимог**

**Спрошеній зміст** технічного завдання складається з таких пунктів:

1. **Загальні положення:** назва роботи, умовне позначення, імена (назви) замовника та розробника, терміни початку та закінчення робіт.

2. **Призначення системи:** цілі створення програмного забезпечення, основні очікувані результати, область застосування певного продукту.

3. **Об'єкти даних:** характеристика об'єктів, які мають опрацьовуватися за допомогою розробленого програмного продукту.

4. **Вимоги до програмного забезпечення:** функціональні та нефункціональні вимоги до програми.

5. **Стадії розробки:** зміст робіт зі створення ПЗ (перелік етапів з короткою характеристикою кожного).

6. **Вимоги до програмної документації:** перелік документів, які підлягають розробленню.

7. **Постання:** перелік літератури та корисних посилань.

Детальніше кожний пункт описано нижче.

**Пункт 1** задається чітко, конкретно та стисло. Назва роботи коротко передає суть роботи (3-7 слів), умовне позначення - це коротка символічна назва (наприклад, library 1, detMatrix2012).

**Пункт 2** містить короткий опис того, навіщо та для кого створюють програму (3-4 речення). Мету створення ПЗ зазначають узагальнено (наприклад, опрацювання бухгалтерських документів підприємства, автоматизація роботи бібліотеки тощо).

**Пункт 3** описує дані, які опрацьовуватиме програма. Потрібно вказати суть та форму їх представлення.

Наприклад, інформація про працівників підприємства зберігається у текстовому файлі: ідентифікаційний код (восьмизначне натуральне число),

прізвище, ім'я, по батькові, дата народження (день, місяць, рік), освіта (базова середня, повна середня, неповна вища, базова вища, повна вища), назва навчального закладу, рік закінчення навчального закладу, стаж роботи (повні роки), посадовий оклад (грн).

Або ж інформацію про масив подають у відповідній формі інтерфейсу через поля: кількість елементів та відповідно значення елементів масиву, задані безпосередньо або ж випадково згенеровані (у межах від -500 до 600).

**Пункт 4** містить:

4.1. Частини (компоненти, файли), з яких може складатися програмна система;

4.2. Чіткий перелік функцій, які виконуватимемо розроблене ПЗ, тобто що корисного для користувача робить програма. Кожну функцію варто називати. Це функції програми з погляду користувача, а не програмної реалізації.

Наприклад:

R1. Відкриття файла з даними про працівників підприємства у вигляді таблиці у спеціальному вікні програми;

R2. Внесення змін до даних про працівників в існуючому файлі у разі введення правильного коду доступу до даних;

R3. Обчислення заробітної плати працівника за посадовим окладом та надбавкою за стаж. Надбавки для працівників записано у файлі bonus.txt, де рядок містить ідентифікаційний код — відсотки;

4.3. Необхідне для роботи розроблюваної програми апаратне забезпечення (обчислювальні характеристики комп'ютера, інше обладнання) та програмне забезпечення (операційна система, спеціальне програмне забезпечення: середовища, бібліотеки і т. п.).

**Пункт 5** містить перелік етапів життєвого циклу розроблюваного ПЗ із зазначенням і основних, і неосновних етапів. Цей пункт передбачає початкове планування робіт, необхідних для отримання результату.

**Пункт 6** містить перелік документів, які необхідно створити під час розроблення програми.

**Пункт 7** оформлюють згідно з державним стандартом.

Запропонований зміст технічного завдання має концептуальний характер.

Фахівці вважають, що грамотне ТЗ - це понад 50 % успіху у вирішенні завдання, а час, витрачений на його підготовку, - одне з кращих вкладень, які фірма може зробити в період аналізу та специфікації вимог. Недарма складати ТЗ доручають провідним фахівцям: керівникам проектів і робіт, системним аналітикам, головним проектувальникам та ін.

Згідно із стандартом IEEE 830 описано такі переваги добре складеного технічного завдання:

1. Технічне завдання — це основа для угоди між замовниками та виконавцями щодо розроблення програмного забезпечення. Визначення повного опису функцій, які повинні бути реалізовані, допоможе потенційним користувачам визначити, чи програмне забезпечення відповідає їхнім потребам, чи воно необхідно змінити, щоб задоволити їхні потреби. [Примітка: технічне завдання використовують як основу договору з клієнтами протягом усього часу розроблення програмної системи].

2. Скорочення зусиль, необхідних на розроблення програмного забезпечення. Підготовка чіткого технічного завдання змушує усі зацікавлені сторони розглянути ретельно всі вимоги, перш ніж розпочати етап розроблення дизайну та кодування. Такий аналіз вимог може виявити упущення, непорозуміння і протиріччя ще на перших етапах життєвого циклу програми, коли ці проблеми легше виправити.

3. Технічне завдання - базовий документ, за яким розраховують витрати і час для реалізації програмного забезпечення.

4. Технічне завдання є основою валідації та верифікації програмного продукту (верифікація і валідація - це методи аналізу, перевірки специфікацій і правильності виконання програм відповідно до заданих вимог і формального опису програм). [Примітка: технічне завдання використовують для створення плану тестування].

5. Оскільки у технічному завданні описано характеристики продукту, воно слугує основою для подальшого вдосконалення програмного забезпечення. [Примітка: необхідно намагатися оновлювати технічне завдання згідно з побажаннями клієнтів або розробників].

Після формування ТЗ на його основі аналізують поставлену задачу на розроблення ПЗ, визначають структуру вхідних і вихідних даних, оцінюють альтернативні методи розв'язання поставлених завдань, а також обирають оптимальний алгоритм для вирішення певного завдання.

**Висновок.** Аналіз і визначення вимог - один з основних етапів ЖЦ ПЗ. Складність цього етапу полягає у поєднанні формальних й неформальних методів ПЗ, врахуванні знань предметної області, необхідності спілкування із замовником та майбутніми користувачами. Існують різні типи вимог: функціональні й нефункціональні. Основним результатом цього етапу є задокументовані узгоджені із замовником вимоги у формі спеціального документа. Аналізують і визначають вимоги досвідчені фахівці: аналітики, керівники проектів. Участь замовника, майбутніх користувачів обов'язкова на етапі аналізу й специфікації вимог.

## **Контрольні питання для самоперевірки**

1. Що таке інженерія програмного забезпечення?
2. Що таке ІТ?
3. Яке місце ПЗ серед споріднених комп'ютерних галузей?
4. Чим програмування відрізняється від інженерії програмного забезпечення?
5. Що таке програмне забезпечення? Наведіть класифікацію ПЗ.
6. Наведіть приклади ПЗ, що належать до різних класів згідно з вищенаведеною класифікацією.
7. Чим інженерія ПЗ відрізняється від комп'ютерних наук?
8. Чим комп'ютерна інженерія відрізняється від інженерії ПЗ?
9. Які існують обмеження в проектах з розроблення ПЗ?
10. Яке ПЗ вважають якісним?
11. Що таке життєвий цикл програмного забезпечення?
12. Яка історія виникнення терміна “життєвий цикл” (ЖЦ) в інженерії ПЗ?
13. Що таке артефакт розроблення ПЗ? Наведіть приклади.
14. Що є складовими ЖЦ ПЗ? Наведіть приклади.
15. Що таке процес ЖЦ ПЗ? Наведіть приклади.
16. Що означає встановлення процесу?
17. Що таке декомпозиція ЖЦ ПЗ?
18. Які процеси ЖЦ ПЗ є основними? Як поділяють основні процеси?
19. Які процеси ЖЦ ПЗ є неосновними? Як поділяють неосновні процеси?
20. Де наведено класифікацію процесів ЖЦ ПЗ? Наведіть приклад.
21. Чим ЖЦ ПЗ відрізняється від моделі ЖЦ ПЗ? Наведіть приклади відомих моделей ЖЦ ПЗ.
22. Які роботи щодо вимог виконують спочатку?
23. Як створюють опис вимог (технічне завдання)?
24. Назвіть основні пункти спрошеного варіанта технічного завдання.
25. У якому розділі ТЗ задають вимоги до документування процесів розроблення ПЗ?
26. Що таке вимога до програмного забезпечення?
27. Які є типи вимог до ПЗ?
28. Що таке функціональні вимоги до програми? Наведіть приклади.
29. Які документи використовують для опису вимог?
30. Які характеристики правильної вимоги?
31. Що таке специфікація вимог?
32. Як пов'язані специфікація вимог і технічне завдання?
33. Наведіть приклади нефункціональних вимог.
34. Чи у ТЗ записують нефункціональні вимоги? Відповідь поясніть.
35. Що передбачає управління вимогами?
36. Що таке валідація вимог?

37. Що таке пріоритетизація вимог?
38. Що означає така властивість вимоги, як завершеність?
39. Яку вимогу вважають верифікованою?
40. Назвіть форми документування вимог до ПЗ.
41. Наведіть приклади апаратних вимог до ПЗ.
42. Назвіть функціональні вимоги до найпростішого текстового редактора.
43. Чи у ТЗ описують, як працюватиме ПЗ? Відповідь поясніть.
44. Який етап ЖЦ ПЗ слідує за визначенням вимог? Яка його основна суть?
45. У якій частині ТЗ відображене початкове планування робіт?
46. Чи допускається використання у специфікації вимог термінів-синонімів?

Відповідь поясніть.

47. Що зображено на діаграмі прецедентів? Як ця діаграма пов'язана із ТЗ?
  48. Наведіть приклад некоректних вимог до ПЗ.
  49. Наведіть приклади вимог до документації?
  50. Які фахівці ПЗ беруть участь у специфікації вимог?
  51. Яка роль замовника у визначені вимог до програмної системи?
  52. Що є вхідними даними та результатами етапу визначення вимог до ПЗ?
  53. Як визначення вимог до програмної системи пов'язане з її тестуванням?
  54. У чому полягає складність отримання результатів аналізу й визначення вимог до ПЗ?
55. Поясніть принципи взаємодії замовника й розробника у ракурсі визначення вимог на основі рис. 1.4.
56. Яка частика вартості етапу аналізу й специфікації вимог у загальній вартості типового проекту з розроблення ПЗ?
57. Для якого етапу ЖЦ ПЗ результати аналізу й специфікації вимог є вхідними даними?
58. Чи можуть змінюватися вимоги до ПЗ під час розроблення? Відповідь обґрунтуйте.

## **ЛАБОРАТОРНА РОБОТА № 1**

**Тема.** Формування технічного завдання як результат аналізу та визначення вимог.

**Мета.** Навчитися складати найпростіше технічне завдання до розроблення програми.

### **Перелік питань для теоретичних відомостей**

Вимоги до ПЗ. Класифікація вимог. Зміст функціональних вимог. Приклади нефункціональних вимог. Означення етапу аналізу й визначення вимог. Властивості вимог до ПЗ. Завдання під час роботи з вимогами (згідно з Swebook). Управління вимогами. Способи опису вимог. Типові помилки визначення вимог. Ітеративність процесу визначення вимог. Учасники процесу аналізу й визначення вимог. Специфікація вимог. Технічне завдання. Переяви добре складеного технічного завдання. Зміст спрощеного варіанта технічного завдання. Основні результати етапу аналізу й визначення вимог до ПЗ. Місце й роль етапу аналізу й визначення вимог до ПЗ у життєвому циклі ПЗ.

**Необхідні програмні засоби для виконання завдання.** Текстовий редактор/процесор.

### **Завдання**

**Умова.** Скласти технічне завдання (концептуальний рівень) до програми згідно з індивідуальним варіантом (№ 1-28). Крім описаного функціоналу, у варіанті задати дві функціональні вимоги, що можуть бути корисними для потенційного замовника й дві нефункціональні вимоги, важливі для розроблення.

Технічне завдання оформити за поданим вище планом. У п. 7 вказати три інформаційні джерела за темою програмування: і з інтернет-ресурсів, і друковані.

**Примітка.** Крім змісту ТЗ, важливою є форма представлення цього документа. Для цього використайте можливості текстового редактора для оформлення документа в читабельному й наочному вигляді.

## **Варіанти для складання ТЗ**

### **Варіанти 1-14**

З клавіатури ввести послідовність записів, які містять дані про результати сесії студентів групи: <Прізвище>, <Ім'я>, <Дата народження>, <Список екзаменаційних оцінок>. Роздрукувати введені дані у вигляді таблиці, а також подати інформацію згідно з варіантом.

1. Відсортувати дані за прізвищами студентів за алфавітом. Визначити двох студентів з найвищим середнім балом. Вилучити зі списку дані про студентів із рейтинговим балом, нижчим за середній у групі.

2. Відсортувати дані за прізвищами студентів за послідовністю, протилежною алфавітному. Визначити п'ять студентів з найнижчим середнім рейтинговим балом. Вилучити зі списку дані про студентів з оцінками 2.

3. Відсортувати дані за віком студентів за зростанням. Роздрукувати список студентів з рейтинговим балом, нижчим за середній бал у групі. Вилучити зі списку дані про студентів, які не мають оцінки 5.

4. Відсортувати дані за віком студентів у спадному порядку. Роздрукувати список студентів з рейтинговим балом, вищим за середній бал у групі. Вилучити зі списку дані про студентів з рейтинговим балом, нижчим за 4,5.

5. Роздрукувати список студентів, які отримали оцінки 2 на іспитах, за алфавітом за прізвищем. Вилучити зі списку дані про трьох студентів із найнижчим рейтинговим балом.

6. Роздрукувати список студентів, які отримали лише оцінки 5 на іспитах, за зростанням за віком. Вилучити зі списку дані про студентів з рейтинговим балом, нижчим за 3,5.

7. Роздрукувати список студентів за зростанням за рейтинговим балом. Вилучити зі списку дані про студентів з двома оцінками 2.

8. Роздрукувати список студентів, молодших за середній вік у групі, упорядкований за алфавітом за прізвищем. Вилучити зі списку дані про студентів, які не мають оцінок 4 і 5.

9. Роздрукувати список студентів, старших за середній вік у групі, упорядкований за зростанням рейтингового балу. Вилучити зі списку дані про студентів, які отримали на другому іспиті оцінку 3.

10. Роздрукувати список студентів, які отримали оцінки 4 і 5 на іспитах за спаданням за віком. Визначити двох наймолодших студентів серед них. Вилучити зі списку дані про студентів, які не мають оцінки 2.

11. Роздрукувати список студентів, які не отримали жодної оцінки 2 на іспитах за алфавітному порядку за прізвищем. Визначити двох найстарших студентів серед них. Вилучити зі списку дані про студентів, які отримали на першому іспиті оцінку 2.

12. Роздрукувати список студентів, які не отримали жодної оцінки 5 на іспитах, упорядкований за середнім рейтинговим балом. Визначити серед них найстаршого та наймолодшого студентів. Вилучити зі списку дані про студентів, які отримали на першому та третьому іспитах оцінки 3.

13. Роздрукувати список студентів, народжених восени, впорядкований за алфавітом за прізвищем. Вилучити зі списку дані про студентів, які під час екзаменаційної сесії отримали оцінки 2, 3, 4, 5.

14. Роздрукувати список студентів, народжених влітку, впорядкований за зростанням рейтингового балу. Вилучити зі списку дані про студентів з рейтинговим балом, вищим за 4,5.

### **Варіанти 15-30**

З клавіатури ввести послідовність записів, які містять дані про книгу: <Автор>, <Назва книги>, <Рік видання>, <Кількість сторінок>, <Вартість>. Роздрукувати введені дані у вигляді таблиці, а також подати інформацію згідно з варіантом.

15. Відсортувати дані за прізвищами авторів за алфавітом. Визначити дві книги з найбільшою кількістю сторінок. Вилучити зі списку дані про книги ціною, меншою за 50 грн.

16. Відсортувати дані за прізвищами авторів у послідовності, протилежній за алфавітну. Визначити п'ять найновіших книг за роком видання. Вилучити зі списку дані про книги з кількістю сторінок, меншою за 50.

17. Відсортувати дані за назвою у послідовності, протилежній до алфавітної. Визначити шість найстаріших книг за роком видання. Вилучити зі списку дані про книги, видані раніше 1991 року.

18. Відсортувати дані за назвою за алфавітом. Визначити три найдорожчі книги за вартістю. Вилучити зі списку дані про книги з назвою, що починається на букву Д.

19. Відсортувати за зростанням дані за роком видання. Визначити три книги з найменшою кількістю сторінок. Вилучити зі списку дані про книги авторів, прізвище яких на букву Я.

20. Відсортувати за зростанням дані за вартістю. Визначити книги авторів, прізвище яких на букву А. Вилучити зі списку дані про книги з вартістю, меншою за середню у бібліотеці.

21. Відсортувати у спадному порядку дані за роком видання. Визначити книги з назвою, що починається з букви А. Вилучити зі списку дані про книги з кількістю сторінок, меншою за середню у бібліотеці.

22. Відсортувати за зростанням дані за кількістю сторінок. Визначити книги з кількістю сторінок, більшою за середню в бібліотеці. Вилучити зі списку дані про книги, видані раніше 1980 року, з кількістю сторінок, меншою за 90.

23. Відсортувати у спадному порядку дані за вартістю. Визначити книги, видані після 1980 року. Вилучити зі списку дані про книги, вартістю більшою від середньої та з кількістю сторінок, меншою від середньої у бібліотеці.

24. Відсортувати за спаданням за вартістю дані про книги, видані до 1975 року. Вилучити зі списку дані про книги, видані за останні 5 років.

25. Відсортувати за назвою в алфавітному порядку дані про книги, вартість яких більша за середню в бібліотеці. Вилучити зі списку дані про книги з назвою, що починається на букви П, К, Л.

26. Відсортувати за прізвищем за алфавітом дані про книги, вартість яких менша за середню в бібліотеці. Вилучити зі списку дані про книги авторів, прізвища яких починаються з букв А, Б, В, Г.

27. Відсортувати за зростанням за роком видання дані про книги з кількістю сторінок, меншою за середню в бібліотеці. Вилучити зі списку дані про книги, видані раніше 2000 року, з кількістю сторінок, меншою за 150.

28. Відсортувати за зростанням за вартістю дані про книги, видані від 1991 року. Вилучити зі списку дані про книги вартістю більшою за 50 грн та видані за останні три роки.

29. Відсортувати дані за прізвищами авторів у послідовності, протилежній до алфавітної. Визначити п'ять найдорожчих книг. Вилучити зі списку дані про книги з кількістю сторінок, меншою за 25.

30. Відсортувати дані за назвою у послідовності, протилежній ніж алфавітна. Визначити три книги з найменшою кількістю сторінок. Вилучити зі списку дані про книги з ціною понад 1200 грн.

### Варіанти завдань до теоретичних відомостей у звіті

Номер варіанта	Перелік контрольних питань
1	4, 10, 29
2	3, 15, 24
3	2, 20, 19
4	1, 25, 17
5	5, 30, 36
6	6, 26, 34
7	7, 22, 32
8	8, 27, 42
9	11, 33, 40
10	12, 18, 41
11	13, 29, 37
12	16, 20, 1
13	17, 28, 39
14	18, 32, 2
15	24, 34, 3
16	25, 35, 4
17	38, 21, 1
18	39, 22, 5
19	36, 19, 6
20	37, 20, 7
21	35, 18, 8
22	42, 17, 9
23	44, 16, 10
24	43, 15, 11
25	14, 33, 39
26	15, 34, 45
27	24, 12, 2
28	48, 11, 3
29	47, 10, 4
30	46, 9, 5

## **Вимоги до звіту**

Звіт оформлюють за загальними вимогами, наведеними у методичних рекомендаціях.

У теоретичних відомостях згідно з індивідуальним завданням необхідно розгорнуто відповісти на три контрольні питання з наведеної вище списку.

У розділі “Отримані результати” роздрукувати сформоване технічне завдання.

У висновках зазначити й охарактеризувати отримані під час виконання лабораторної роботи результати.

## **ТЕМА 2. ПРОЕКТУВАННЯ Й РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

### **2.1. Означення етапу проектування**

Проектування є невід'ємною частиною будь-якого виробництва. У загальному випадку мають на увазі створення сукупності документів, розрахунків, креслень, ескізів, необхідних для виготовлення виробу; а також планування, задум заснування, організації будь-чого.

Цей етап життєвого циклу програмного забезпечення є найбільш працемістким. Але, на відміну від попереднього етапу, перевагою є формалізація процесів та можливість використання готових інструментів і зразків (шаблонів), що дає змогу автоматизувати розроблення ПЗ.

На етапі проектування необхідно описати, з яких частин складатиметься ПЗ як воно працюватиме, тобто необхідно задати структуру й поведінку ПЗ. Отримані проектні рішення надалі переводять у програмні коди обраною мовою програмування, крім того, можливий автоматизований перехід від проектних рішень до кодів програми.

У строгому значенні архітектура ПЗ (*англ.* Software Architecture) - опис підсистем і компонентів програмної системи, а також зв'язків між ними. Архітектура насамперед визначає внутрішню структуру системи, задаючи спосіб, в який систему буде організовано або сконструйовано.

Перших розробників програмних систем можна вважати вільними творцями, художниками від програмування, які створювали все з нуля. Проте проектування в сучасній інженерії ПЗ - це здебільшого використання готових рішень, шаблонів проектування, - так званих "патернів" (*англ.* Pattern - шаблон). "Винахід велосипеда" під час розроблення архітектури ПЗ є справою невдячною, і будь-який проектувальник зараз зобов'язаний знати як мінімум базовий набір стандартних рішень.

Шаблони проектування (*англ.* патерн, Design Pattern) - це багато разів застосовувана архітектурна конструкція, за допомогою якої вирішують загальну проблему проектування в межах конкретного контексту. Патерн не є закінченим зразком проекту, який можна безпосередньо перетворити на код; це опис або зразок для пояснення, як вирішити завдання так, щоб це можна було

використовувати в різних ситуаціях. Об'єктно-орієнтовані шаблони часто показують відношення і взаємодії між класами або об'єктами, без визначення того, які кінцеві класи чи об'єкти додатка використовуватимуть (Додаток I). Шаблони проектування не залежать від застосованої мови програмування.

Патерни проектування в програмуванні розглядають як стандартні будівельні блоки для архітекторів програмного забезпечення і допомагають їм легко знаходити спільну мову.

Тому проектувальник повинен мати достатній досвід програмування і підходити доожної нової задачі, враховуючи кращі зразки та встановлену методику проектування. На жаль, внаслідок ускладнення програмних систем дуже важко проектувати так, щоб уникнути помилок під час майбутнього розроблення програм. Тому зазвичай складна задача проектування складається з двох етапів (рівнів):

1. *Архітектурне проектування, або високорівневий дизайн* (англ. Software Architectural Design/Top-level Design), протягом якого визначають так звані компоненти високого рівня: зв'язок між найбільшими і загальними частинами програмного забезпечення;

2. *Детальне проектування компонентів* (англ. Software Detailed Design), протягом якого визначають деталізовану архітектуру, що описує кожну компоненту у тому обсязі, який необхідний для конструювання.

Отже, на етапі проектування системні архітектори й програмісти, керуючись вимогами, розробляють високорівневий дизайн та детальну архітектуру системи.

Архітектура системи називають внутрішню структуру продукту (компоненти програми і зв'язки між ними), а також основи користувальського інтерфейсу продукту.

Переважно проектування архітектури програмного продукту передбачає виконання таких завдань:

- трансформація вимог до програмного забезпечення в архітектуру (визначення структури програмної системи та її проектування);
  - розбиття програмної системи на окремі компоненти та їх проектування з визначенням ключових елементів структур даних;
  - розроблення і документування інтерфейсів програми і баз даних.
- Проектування полягає в створенні уявлення про:
- складові програми;

- модульну структуру програмного продукту;
- алгоритмічну структуру;
- структури даних;
- вхідний і вихідний інтерфейс (вхідні та вихідні формати даних).

Будь-яку програмну систему можна розглядати з різних аспектів - наприклад, *структурного* (статичного), *поведінкового* (динамічного), *логічного* (задоволення функціональних вимог), *фізичного* (розділеність за обчислювальними вузлами), *реалізації* (як деталі архітектури подано в коді) тощо. У результаті отримують різні архітектурні подання (*англ. View*). Архітектурне подання можна визначити як частковий аспект програмної архітектури, що відображає специфічні властивості програмної системи. Своєю чергою, дизайн системи можна розглядати як комплекс архітектурних подань, достатній для реалізації системи й задоволення вимог до системи.

Для спрощення візуалізації процесу проектування використовують так звані нотації - схематичне зображення характеристик розроблюваної програмної системи (блок-схеми, макети, діаграми тощо).

У зводі знань з інженерії програмного забезпечення SWEBOK визначено основні розділи галузі знань, пов'язані з проектуванням ПЗ (Додаток К).

## 2.2. Алгоритми - поведінкова складова результатів проектування

Якщо використовувати алгоритмічний підхід до розроблення ПЗ, на стадії проектування необхідно розробляти алгоритми й відповідні структури даних.

Недарма відомий швейцарський учений Вірт Ніклаус написав книгу “Алгоритми + Структури даних = Програми”, де сама назва книги відображає важливість правильного вибору й алгоритмів, і структур даних.

Алгоритми за своєю суттю також є шаблонами, але не проектування у повному змісті, а обчислення, оскільки вирішують обчислювальні завдання.

**Алгоритм** — це опис послідовності дій, які потрібно виконати для досягнення поставленої мети або здобуття заданого результату. Тобто алгоритм задає, як вхідні дані перетворюються на вихідні.

Кожен алгоритм має мати такі характеристики:

1. Зрозумілість (виконавець повинен уміти виконувати кожну вказівку алгоритму, тобто розуміти кожну його команду);
2. Детермінованість або однозначність (усі дії алгоритму мають трактуватись однозначно, будь-які розбіжності в тлумаченні виключаються);
3. Скінченість або результативність (алгоритм повинен приводити до результату за скінченну кількість дій);
4. Дискретність (поділ алгоритму на послідовність дій);
5. Масовість або універсальність (можливість застосовувати алгоритм для однотипних задач).

Для подання алгоритмів застосовують різні способи. Кожний з них надає певні засоби для опису послідовних дій, які треба виконати.

1. У повсякденному житті найчастіше застосовують словесний спосіб, який полягає в поданні алгоритму у вигляді окремих занумерованих пунктів, кожний з яких містить команду на виконання певної дії. Усі команди записуються словами.

2. Формула - це математичний спосіб запису алгоритму. За формулою визначають послідовність обчислень для отримання числового результату.

3. Якщо виконують серію розрахунків за однаковими формулами, то для запису алгоритму іноді використовують розрахункову таблицю.

4. Поширеним способом наочного подання алгоритму є блок-схема, яка складається з геометричних фігур, з'єднаних напрямленими лініями.

Найчастіше під час побудови алгоритму майбутньої програмної системи використовують блок-схеми. Для побудови блок-схеми алгоритму прийнято стандартну систему геометричних фігур, які використовують для різних видів операторів (табл. 2.1).

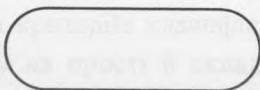
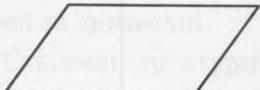
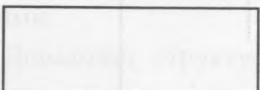
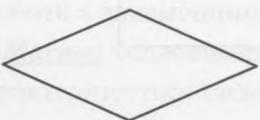
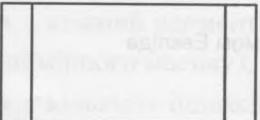
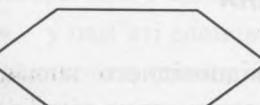
На рис. 2.1 зображено блок-схему алгоритму пошуку найбільшого спільного дільника (НСД) чисел  $M$  і  $N$  алгоритмом Евкліда.

З визначенням алгоритму пов'язано багато інших базових термінів й визначень, які описують суть цього фундаментального поняття й є взаємопов'язаними (Додаток Л).

Інший приклад блок-схеми алгоритму з використанням коментарів і спеціальних блоків для позначення циклу наведено в Додатку М.

Таблиця 2.1

### Основні конструкції блок-схем

Оператор	Опис
	Термінатор. Початок/кінець
	Оператор введення, виведення даних
	Процес обчислень, арифметична дія
	Розгалуження. Оператор умови
	Підпрограми (функції)
	Оператор циклу

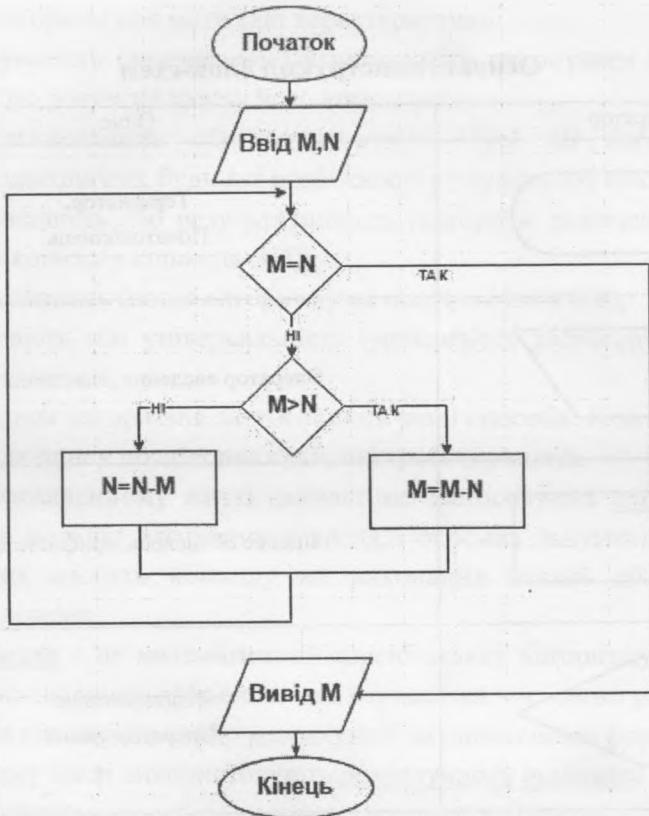


Рис. 2.1. Блок-схема пошуку НСД за алгоритмом Евкліда

### 2.3. Структури даних - структурна складова результатів проектування

Крім розроблення алгоритмів розв'язування відповідного завдання, на етапі проєктування визначають структури даних, які оптимально підходять для цієї задачі.

Структури даних - це спосіб організації даних у пам'яті комп'ютера. Важливо, що зі структурою даних пов'язують і специфічний перелік операцій, які можна виконати над даними, організованими в таку структуру.

Тому правильно вибрати структуру даних надзвичайно важливо для ефективного функціонування відповідних алгоритмів їх обробки. Добре побудовані структури даних дають змогу оптимізувати використання машинного часу та пам'яті комп'ютера для виконання найважливіших операцій. Існує безліч критеріїв класифікації структур даних. Найпростіший з них стосується поділу на прості й складені структури даних. Складені структури даних ще називають агрегованими.

У загальному випадку всі складені структури даних можна поділити на статичні та динамічні.

Статичні структури даних розміщаються у статичній пам'яті, яку виділяє компілятор під час компіляції програми. Зв'язки елементів та їхня кількість, розміщення структур завжди залишаються сталими під час виконання програми.

Динамічні структури даних розміщаються у динамічній пам'яті, яка виділяється на етапі виконання програми. Розміщення, зв'язки елементів та їхня кількість можуть динамічно змінюватися в процесі виконання програми. Для роботи з динамічними даними використовують вказівники.

**Масиви** (одновимірні, двовимірні) - об'єднання однотипних елементів, що характеризується такими властивостями:

- елементи одного типу;
- фіксований набір елементів;
- кожний елемент має унікальний індекс (або набір індексів у випадку багатовимірного масиву);
- кількість індексів визначає розмірність масиву;
- звернення до елемента масиву виконують за ім'ям масиву і значенням індексів для цього елемента;
- у пам'яті елементи масиву розміщено підряд.

Під кожен елемент масиву виділено певну пам'ять, відповідно добуток кількості елементів масиву на розмір типу елемента визначають розмір пам'яті для зберігання масиву.

Масив як структуру даних обирають тоді, коли необхідно часто здійснювати операції пошуку елемента за ключем, а також у задачах сортування елементів. Це структура даних прямого доступу.

Приклад ініціалізації масивів мовою C++:

```
int nVector[10] = { 5, 5, 5, 5, 5, 5, 5, 5, 5, 5 };
int nTable[3][5] =
{
{ 1,2, 3,4,5 }, //row 0
{ 6, 7, 8, 9,10 }, // row 1
{ 11,12,13, 14, 15 } //row2
};
```

**Структури** - об'єднання різновидних елементів (поля структури).

Найважливішою операцією для структури є операція доступу до вибраного поля - операція кваліфікації.

Приклад опису структури в C++:

```
struct Saddress {
    int nFlatNumber;
    char *chStreetName;
    char *chCityName;
    char *chPostalCode;
};
```

**Перелічуваний тип** - тип даних, що описується переліченням усіх можливих значень (кожне з яких позначають власним ідентифікатором), яких можуть набувати об'єкти цього типу. Тобто перелічуваний тип - це набір ідентифікаторів, які відіграють ту саму роль, що і звичайні іменовані константи, але пов'язані з одним типом.

Приклад ініціалізації перелічуваного типу в C++:

```
enum eMark { one, two, three, four, five };
```

**Черга** (FIFO - first in, first out, **англ.**) - динамічна структура даних, що працює за принципом для опрацювання елементів “перший прийшов - перший пішов”. У черги є голова та хвіст. Елемент, що додається до черги, опиняється в її хвості. Елемент, що видаляється з черги, знаходиться в її голові. Основними операціями, які можна провести з чергою, є:

- операція додавання елемента в “хвіст” черги. При цьому довжина черги збільшується на одиницю;

- операція, яка повертає елемент з “голови” та видаляє його з черги, встановлюючи “голову” на наступний за видаленим елементом та зменшуючи довжину на одиницю.

Чергу варто використовувати за потреби швидкого доступу до елементів, яких було додано найпершими. На рис. 2.2 зображене графічний вигляд черги.

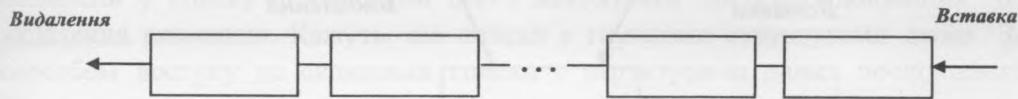


Рис. 2.2. Графічне зображення черги

Приклад опису черги в C++:

```
struct Squeue {
    int nData;
    Squeue *pNext;
} *pHead, *pTail;
```

**Стос/Стек** (англ. FILO - first in, last out) - динамічна структура даних, що працює за принципом “перший прийшов - останній пішов”. Усі операції (наприклад, видалення, вставка елемента) зі стеком можна проводити тільки з одним елементом, який знаходиться на “верхівці” стеку та був доданий останнім. Стос можна розглядати як певну аналогію до стосу тарілок, з якої можна взяти верхню і на яку можна покласти верхню тарілку. Основними операціями, які можна провести зі стосом, є:

- операція додавання елемента на верхівку стосу. При цьому довжина стосу збільшується на одиницю;
- операція, яка повертає останній доданий елемент та видаляє його зі стосу, встановлюючи “верхівку” на наступний за видаленим елементом та зменшуючи довжину стосу на одиницю.

Стос варто використовувати для перевірки останніх доданих елементів, видалення останніх нових елементів.

Цікавим фактом є те, що існує велика кількість “стеко-орієнтованих” мов програмування (Forth, PostScript), які використовують стек як базову структуру даних під час виконання багатьох операцій (арифметичних, логічних, введення-

виведення тощо). Також компілятори мов програмування використовують стос для передавання параметрів у процесі виклику підпрограм, процедур та функцій.

На рис. 2.3 зображене графічний вигляд стосу.

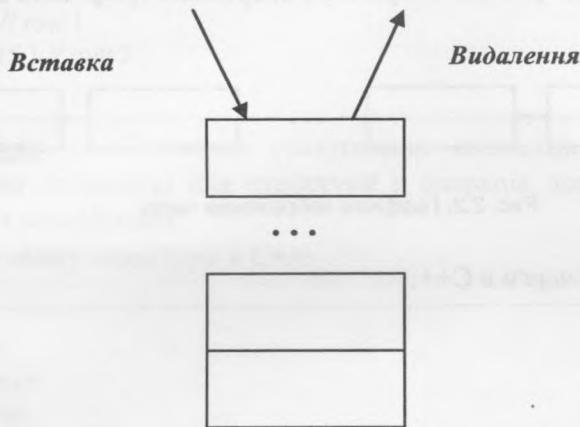


Рис. 2.3. Графічне зображення стосу

Приклад опису стеку в C++:

```
struct Sstack {  
    int nData;  
    Sstack *pNext;  
} *pHead;
```

Зв'язний список - одна з найважливіших структур даних, в якій елементи лінійно впорядковані, але порядок визначається не номерами елементів, а вказівниками, які входять до складу елементів списку та вказують на наступний (в однозв'язних списках (ланцюг), рис. 2.4) або на наступний та попередній елементи (у двозв'язних списках, рис. 2.5).

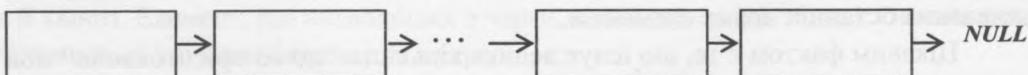


Рис. 2.4. Графічне зображення однозв'язного списку

Двозв'язний список потребує більше пам'яті на елемент (зберігає два вказівники) та більших обчислювальних затрат на виконання елементарних операцій. Але такими списками легше маніпулювати, тому що вони дають змогу проходити по списку в обох напрямах, що є необхідною умовою функціонування деяких алгоритмів.

Основною перевагою зв'язних списків є те, що незалежно від кількості елементів у списку зі списками легко виконувати операцію додавання або видалення елементів. Кажуть, що списки є гнучкими структурами даних. За способом доступу до складових списки є структурами даних послідовного доступу до елементів.



Рис. 2.5. Графічне зображення двозв'язного списку

Приклад опису двозв'язного списку в C++:

```
struct Slist {  
    int nData;  
    Slist *pNext, *pPrevious;  
};
```

Сьогодні розроблено багато бібліотек для різних мов програмування, які містять реалізацію основних структур даних (і статичних, і динамічних). Наприклад, для мови C++ розроблено стандартну бібліотеку Standart Template Library (STL), що містить реалізацію списків, черг, множин тощо.

Варто зауважити, що за об'єктно-орієнтованим підходом до програмування, фундаментальне поняття “клас” містить і набір структур даних ( поля класу), і алгоритми роботи з ними (методи класу). Клас - абстрактний тип даних, який визначає користувач і є моделлю реального об'єкта у вигляді даних та функцій для роботи з ними. Створений та ініціалізований екземпляр класу називають об'єктом класу.

Приклад опису класу “Студент” в C++:

```
class Cstudent {  
private:
```

```
int nID;  
string sName;  
double dCourse;  
int nMarks[5];  
public:  
Student();  
Student(int nID, string sName, double dCourse, int nMarks[5]);  
void setStudent(int nID, string sName, double dCourse, int nMarks[5]);  
int getID();  
string getName();  
double getCourse();  
void print();  
};
```

Вибір правильного представлення даних часто є ключем до ефективного програмування і може більшою мірою впливати на продуктивність програми, ніж деталі реалізації використованого алгоритму. Проте, ймовірно, так і не сформується загальна теорія оптимального вибору структур даних; у кожному конкретному випадку потрібно підходити до цього творчо.

## 2.4. Зміст етапу реалізації (розроблення, кодування)

Після того, як вимоги та архітектуру програмного продукту затверджено, переходять до наступної фази життєвого циклу розроблення - програмної реалізації. Програмісти починають писати код програми відповідно до раніше прийнятих архітектурних рішень.

Основними складовими цього етапу є:

- створення алгоритмів і кодів окремих модулів вибраною мовою програмування;
- створення вихідного тексту програми;
- налагодження вихідного тексту.

Основний результат цього етапу - вихідний код. Вихідний код усієї програмної системи створюють, інтегруючи її окремі модулі.

Правильно оформленій текст програми є якісною характеристикою, що дає змогу легко вносити зміни, виправляти помилки та супроводжувати

програмне забезпечення. Тому прийнято писати тексти програм, зважаючи на загальноприйняті вимоги до коду.

**Стандарт оформлення коду** (Coding Conventions, англ.) - це сукупність вказівок щодо конкретної мови програмування, які встановлюють правила стилювого оформлення коду, практики та методи написання програм цією мовою. Метою прийняття та використання стандарту є спрощення сприйняття програмного коду людиною, зменшення навантаження на пам'ять та зір під час читання програми.

Код програми повинен бути настільки простим, наскільки це можливо. Досвідчені розробники дотримуються правил “Не треба ускладнювати завдання”, “Вибираєте найпростіше з працюючих рішень”, “Не треба робити того, що не передбачено технічним завданням”.

Простий код зазвичай коротший і зрозуміліший, а отже, імовірно, містить менше помилок.

## 2.5. Правила оформлення коду на C++/C

1. *Послідовність методів у файлах.* Рекомендують таку послідовність методів у кожній секції (private, protected, public) класу:

- ^ Статичні
- ^ Конструктори
- ^ Деструктори
- ^ Змінні-члени
- ^ Оператори
- ^ Методи-члени

2. *Коментарі.* Зверніть увагу на розділові знаки, орфографічні та граматичні правила під час написання коментарів. Коментарі повинні бути чіткими, читабельними, інформативними.

^ *Стилі коментарів.* Рекомендується використовувати стиль однострічкових коментарів через //. Для коментування великого фрагмента коду у процесі відлагодження використовують /\* \*/.

^ *Коментування опису класу.* Кожен опис класу повинен містити супровідний опис, який дає змогу зрозуміти, як і для чого використовують цей клас. Рекомендують у коментарях вказувати, як можна створити об'єкт цього класу та використовувати його основні функції.

```
// Iterates over the contents of a GargantuanTable.  
// Example:  
// GargantuanTableIterator* iter = table->NewIterator();  
// for (iter->Seek("foo");!iter->done(); iter->Next()) {  
// process(iter->key(), iter->value());  
//}  
// delete iter;  
class GargantuanTableIterator {  
...  
};
```

^ **Коментування функцій.** Кожна функція програми повинна містити коментарі безпосередньо перед її реалізацією. Коментарі до функцій повинні давати вичерпну інформацію про вхідні та вихідні параметри, призначення функції, чи потрібно очищувати пам'ять (у випадку, якщо функція виділяє динамічну пам'ять);

```
// Returns true if the table cannot hold any more entries.  
Bool IsTableFull();
```

```
// Returns an iterator for this table. It is the client's  
// responsibility to delete the iterator when it is done with it,  
// and it must not use the iterator once the GargantuanTable object  
// on which the iterator was created has been deleted.  
//  
// The iterator is initially positioned at the beginning of the table.  
//  
// This method is equivalent to:  
// Iterator* iter = table->NewIterator();  
// iter->Seek("");  
// return iter;  
// If you are going to immediately seek to another place in the  
// returned iterator, it will be faster to use NewIterator()  
// and avoid the extra seek.  
Iterator* GetIterator() const;
```

**^ Коментування змінних.** Загалом імені змінної має бути достатньо для розуміння її призначення. Відмінність полягає в коментуванні глобальних змінних - для них коментар повинен бути завжди.

```
// The total number of tests that we run through in this regression test,  
const int gNUM_TEST_CASES = 6;
```

**3. Форматування.** Набагато простіше читати код проекту, якщо його відформатовано з використанням пробілів, табуляції, пропусків.

**^ Табуляція.** Для читабельності коду використовуйте відступи та табуляцію.

Форматування умов if-else (кожна команда починається після відступу у два пробіли, else пишуть у тому самому рядку, що і закриваюча дужка):

```
if(condition) { // no spaces inside parentheses  
... // 2 space indent.  
} else if (...) { // The else goes on the same line as the closing brace.  
...  
} else {  
...  
}  
}
```

Форматування умов switch (кожна команда починається після відступу у чотири пробіли, а case - після відступу у два пробіли):

```
switch (var) {  
case 0: // 2 space indent  
... // 4 space indent  
break;  
}  
case 1: {  
...  
break;  
}  
default: {  
assert(false);  
}  
}
```

Форматування циклів (кожна команда починається після відступу у два пробіли, відкриваюча дужка пишуть у рядку заголовка циклу):

```
for (int i = 0; i < 100; i++) {  
    instruction 1; // 2 space indent  
    instruction2;  
}
```

Форматування функцій (відкриваюча дужка методу знаходиться у рядку назви методу, за замовчуванням відступи перед командами методів становлять два пробіли):

```
void Function() {  
    Instruction 1; // 2 space indent  
    Instruction2;  
}
```

^ **Довжина рядка.** Максимальна довжина рядка коду повинна не має перевищувати 80 символів (це традиційний стандарт для багатьох мов програмування. Вважають, що саме така кількість символів поміщається на екрані повністю). Винятками є URL, назви заголовних файлів (*англ. Header File*) та рядки, які не можна розбити на декілька у випадку, якщо їхня довжина перевищує 80 символів.

^ **Опис методів.** Тип повернення методу, назву та параметри задають в одному рядку. Якщо не вистачає 80 символів для опису декларації методу, необхідно кожен параметр методу описувати з нового рядка, відступивши чотири пробіли.

```
ReturnType LongClassName::ReallyReallyReallyLongFunctionName(  
    Type par_name1, // 4 space indent  
    Type par_name2,  
    Type par_name3) {  
    DoSomething(); // 2 space indent  
    ...  
}
```

**S Розділення методів.** Імплементація методів та функцій розділяють рядком завдовжки 80 символів

```
//-----
void Function() {
    instruction;
    instruction;
}
//-----
void Function2() {
    instruction;
    instruction;
}
//-----
```

4. **Назви.** Назви змінних і методів мають бути такими, щоб спростити складність розуміння коду. Наприклад, краще вживати GetNextStudent() замість GetNextArrayElement(). Назви мають бути достатньо довгими, але такими, щоби це не ускладнило читання коду. Дуже часто під час називання відповідних конструкцій програми використовують угорську нотацію.

**Угорська нотація** - метод найменування змінних у програмуванні, коли до ідентифікатора змінної або функції додається префікс, що вказує на його семантику або тип. Перевагою угорської нотації є системність, що полегшує читання програм і зменшує ймовірність неправильного використання змінної.

Угорську нотацію було розроблено з метою використання її для різних мов програмування. Основним її правилом є використання малих букв (префіксів) на початку кожної назви для позначення типу (призначення) змінної або відповідного об'єкта коду.

Основними перевагами використання угорської нотації є:

^ тип змінної зрозумілій за її назвою. Це корисно, якщо проект дуже великий, і над його реалізацією працює велика кількість розробників;

^ у мовах високого рівня програмування, в яких використовують динамічну типізацію, переважно, не вказують тип змінної. Тому єдиним ключем до розуміння того, які операції можна робити з цією змінною, є “підказки” в її назві;

^ форматування імен змінних може спростити деякі аспекти рефакторингу коду;

^ декілька змінних з одинаковими іменами (але різними типами даних) з погляду семантики можна використати в блоці коду: dwWidth, iWidth, fWidth, dWidth;

^ недоречність приведення типів і операції з використанням несумісних типів можна легко виявити під час читання коду.

Отже, згідно з угорською нотацією, варто враховувати такі правила під час кодування:

#### ***^ Назви класів.***

о Назви класів починаються з префікса C, приклади: CsomesClass, CsomesOtherClass.

о Великі літери вживають для розділення слів,

о Не варто вживати назв, що складаються більше ніж з трьох слів.

```
class CtableInfo {  
...  
}
```

#### ***^ Назви методів і функцій.***

о Якщо функція виконує якусь дію, то краще назвати: CheckForErrors() замість ErrorCheck(), DumpDataToFile() замість DataFile(). Це зробить код зрозумілішим.

о Якщо здійснюють певні дії над об'єктом, назvu варто формувати як дієслово+іменник CalculateInvoice().

о Іноді допомагає вживання суфіксів: Max - максимальне значення, Cnt - поточна кількість, Key - ключове значення.

о Використовують префікси для функцій чи методів: Is - використовується для перевірки (IsVisible(), IsHitRetryLimit()), Get - повертає значення, Set - встановлює значення.

о Не використовуйте такі самі назви методів, як назва класу Book, BookTitle. Застосуйте Book, Title.

```
class CnameOneTwo  
{  
public:  
int DoIt();  
void HandleError();  
};
```

### **^ Назви змінних.**

о Як уже згадували, вводьте тип до назви змінних.

```
int      nTimeCount;
DWORD   dwWeightKg;
Cstring csFileName;
LPCSTR  pszUserName;
BOOL    bFileIsFound;
void*   pvList;
Cstring& rcsName;
```

о Додавайте за потреби кваліфікатори (Avg, Sum, Min, Max, Index) укінець назви.

о Використовуйте пари змінних, що відповідають min/max, begin/end, open/close.

о Як індекс у циклах можна використовувати змінну “i”. Варто використовувати в умовах циклу константи. Замість

```
for(int i = 1;i<7;i++){
...
}
```

краще писати:

```
for(int i = 1; i < NUM_DAYS_IN_WEEK; i++){
...
}
```

### **^ Назви атрибутів класу.**

Атрибути класу (змінні-члени) повинні завжди починатися з ‘m\_’.

```
class CnameOneTwo {
private:
int m_nVarAbc;
int m_nErrorNumber;
LPCSTR m_pszUserName;
Cstring mcsFileName;

public:
int VarAbc();
int GetErrorNumber();
};
```

### **^ Назви вказівників.**

- о Вказівники починаються з ‘р’ у більшості випадків,
- о Розміщуйте знак \* ближче до назви, ніж до типу змінної.

```
clistBox *pList = new clistBox();
```

```
CHAR *pczName;
```

### **^ Посилання.**

- Посилання завжди починаються з ‘r’.

```
class Ctest {
public:
void DoSomething(CstatusInfo& rStatus);
CstatusInfo& LoadStatus();
const CstatusInfo& GetStatus();

private:
CstatusInfo& mrStatus;
};
```

### **^ Глобальні змінні.**

- Глобальні змінні завжди починаються з ‘g’.

```
Logger g_Log;
Logger *g_pLog;
```

### **^ Статичні змінні.**

- Статичні змінні можуть починатися з ‘s’.

```
class Ctest {
public:
static CstatusInfo m sStatus;
};
```

### **^ Константи, Enum, #define і Macro.**

- Назви мають бути великими літерами і розділятися ‘\_’.

```
const int A_GLOBAL_CONSTANT = 5;  
  
#defme NUMBEROFELEMENTS 5  
  
enum { STATE_ERR, STATE_OPEN, STATE_RUNNING, STATE_DYING};
```

## 2.6. Застосунок для підвищення якості коду

Для того, щоб легше було дотримуватися стандартів написання коду, існують допоміжні інструменти. Наприклад, для Microsoft Visual Studio варто використовувати ReSharper C++, який має такі переваги:

^ Аналіз та вдосконалення якості коду — можливість швидкого виправлення помилок і зауважень, видалення надлишкових елементів коду. Також ReSharper “підказує” назви певних об’єктів коду.

^ Навігація за кодом - пошук місць використання об’єкта програми, операції швидкого переходу до зв’язаних об’єктів і т. д.

^ Генерація шаблонів коду - генерація пропущених перевизначених функцій, реалізація методів інтерфейсу.

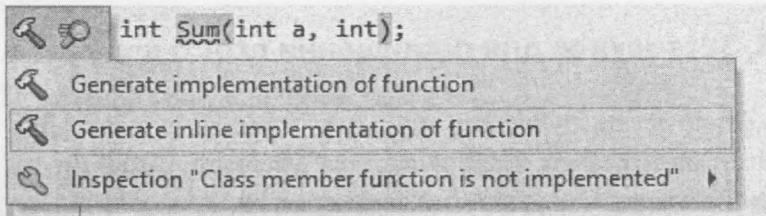
^ Рефакторинг коду - один із типових процесів, що полягає у перетворенні програмного коду, зміні внутрішньої структури програмного забезпечення для полегшення розуміння коду і легшого внесення подальших змін без зміни зовнішньої поведінки самої системи. Назва “рефакторинг” походить від терміна “факторинг” у структурному програмуванні, який означає декомпозиція) програми на максимальні автономні та елементарні частини.

**Статичний аналіз коду.** Як тільки відкривається Visual Studio з проектами, ReSharper C++ починає аналізувати проекти і пропонувати виправлення проблем, які знаходить, і рекомендувати ті або інші покращання. Наприклад, якщо не додано бібліотеки std, ReSharper C++ запропонує її додати:

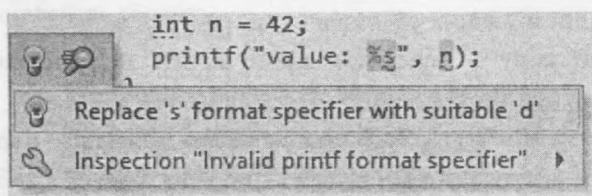
```
int main()  
{  
    ② 'std::cout' (#include <iostream>)? (Alt+Enter)  
    std::cout << i
```

Для того, щоб прийняти пропозицію ReSharper, необхідно натиснути комбінацію клавіш Alt+Enter.

Якщо є прототип функції, але не вистачає її опису, ReSharper C ++ запропонує згенерувати тіло функції або в цьому самому, або в окремому файлі:



Під час використання функції printf(), ReSharper C ++ перевірити аргументи форматування і запропонує рішення, якщо знайде проблеми:



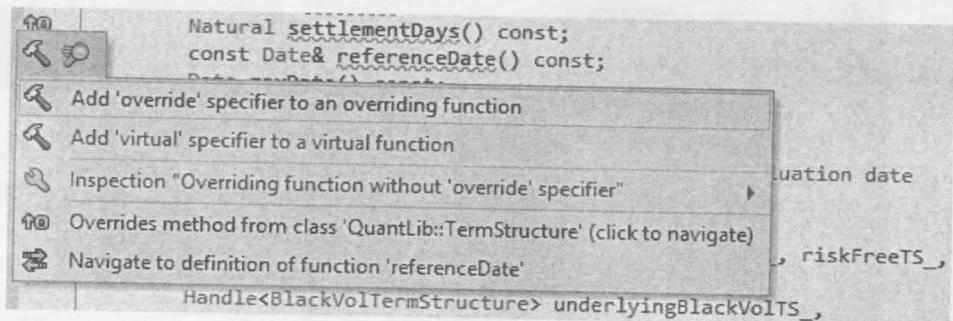
**Навігація та пошук.** ReSharper C++ дає змогу дуже швидко здійснювати навігацію за великими проектами і шукати в них потрібний код. Ось деякі з можливостей навігації:

^ Go to Everything знаходить за назвою будь-який тип сутності (клас, член класу, файл або папку) у всьому проекті відразу. Результати також можна фільтрувати підкомандами, такими як Go to Type, Go to File і Go to Symbol.

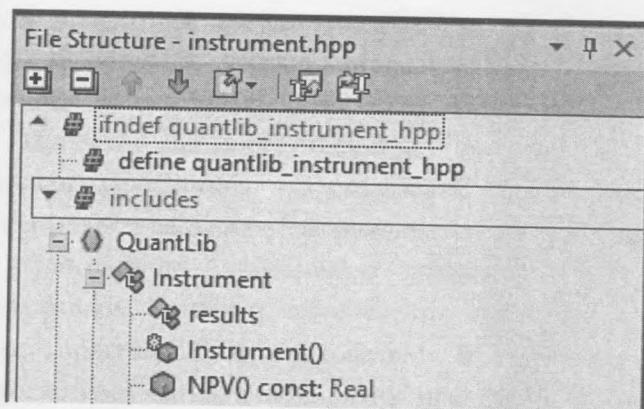
^ Go to File Member дає змогу швидко знайти символ у тому файлі, де ви зараз працюєте.

^ Go to Base / Derived допомагає шукати нащадків і батьків класів або елементи класу. Go to Definition дає змогу швидко перейти до місця оголошення певного об'єкта.

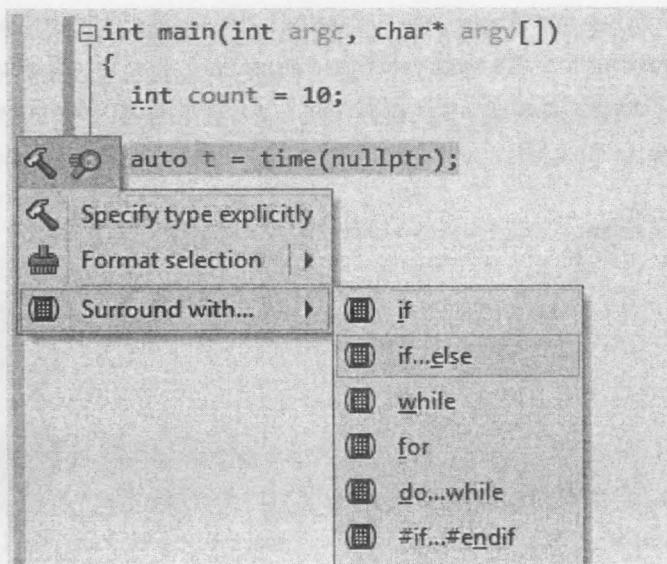
^ Go to Related Files застосовують для переходу до файлів, які мають якесь відношення до поточного. Це можуть бути включені заголовки або суп-файли. Крім того, ReSharper дає змогу отримати доступ між заголовним і відповідним суп-файлами (Ctrl + B).



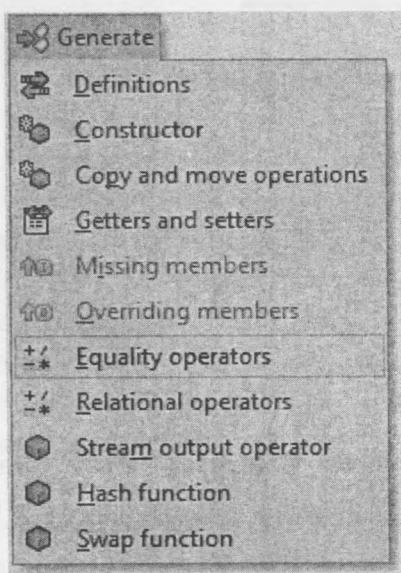
ReSharper C++ також містить кілька вікон для навігації у великих складних проектах. Одне з цих вікон називається File Structure та дає змогу швидко зорієнтуватися в структурі поточного файла:



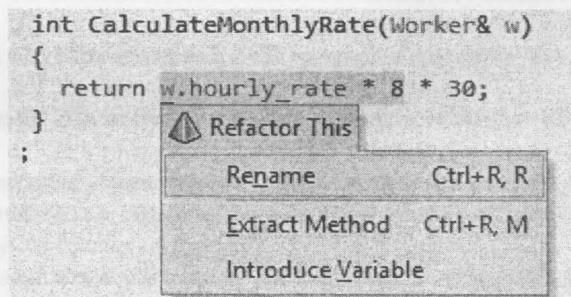
**Генерація шаблонів коду.** За допомогою ReSharper можна виділити частину коду і трансформувати його в деяку конструкцію, наприклад, в if або try-catch вираз.



Крім шаблонів, ReSharper C++ також має механізми генерації коду в меню Generate (наприклад, згенерувати конструктор класу, перевантажені методи):



**Рефакторинг коду.** Рефакторинг коду дуже полегшує роботу програмістів та зменшує кількість помилок у коді. Наприклад, можна перейменовувати змінну у всьому проекті:



Після завершення етапу програмної реалізації розпочинається робота з перевірки правильності функціонування програми, а саме етап тестування ПЗ.

**Висновок.** Безпосередньо розроблення ПЗ у загальному випадку передбачає два основні етапи ЖЦ ПЗ: проектування й створення програмного коду за проектними рішеннями. Під час розроблення ПЗ використовують уже готові рішення й зразки, автоматизовані засоби, що значно полегшує виконання типових задач цього етапу ЖЦ.

Проектні рішення стосуються насамперед структури системи та її поведінки. Проектують ПЗ спеціальні фахівці з інженерії ПЗ: архітектори систем, проектувальники, дизайнери.

Програмною реалізацією займаються розробники, програмісти, кодувальники. З метою забезпечення супроводжуваності ПЗ важливо, щоб тексти програм були читабельні й зрозумілі. Для цього необхідно дотримуватися загальноприйнятих правил стосовно написання програмного коду.

## Контрольні питання для самоперевірки

1. Які завдання вирішуються на етапі проектування ПЗ?
2. У чому полягає етап проектування для найпростішої програми?
3. Що розуміють під архітектурою програмної системи?
4. Які є рівні проектування ПЗ? Чим вони відрізняються?
5. Що є вхідними даними для етапу проектування ПЗ? Для якого етапу ці вхідні дані є результатом?
6. Для якого етапу ЖЦ ПЗ проектні рішення є вхідними? Наведіть приклади проектних рішень.
7. Що таке шаблони проектування? Яка мета використання патернів проектування?
8. Які структури даних використовуються у вашій програмі? Які є альтернативні структури даних?
9. Порівняти переваги/недоліки у застосуванні масивів чи однозв'язних списків для власної програми.
10. Чим статичні структури даних відрізняються від динамічних?
11. Описати основну суть структури даних - черга та базові операції над нею.
12. Описати основну суть структури даних - стек та базові операції над ним.
13. Що таке алгоритм? Навести приклад.
14. Які властивості алгоритмів?
15. Поясніть властивість дискретності алгоритму на прикладі розв'язування квадратного рівняння.
16. Які є відомі способи представлення алгоритмів?
17. Як виглядає вербальне представлення алгоритму? Наведіть приклад.
18. З чого складається блок-схема алгоритму?
19. Скільки входів та виходів має блок розгалуження? Відповідь пояснити.
20. Скільки входів та виходів має блок циклу? Відповідь пояснити.
21. Побудуйте блок-схему алгоритму знаходження півсуми мінімального й максимального з трьох дійсних чисел.
22. Що таке клас в ООП? Яку принципову відмінність має такий тип даних порівняно зі складеними типами даних, що використовуються в алгоритмічному програмуванні?
23. Що виконують на етапі інтеграції?
24. Що таке стандарт оформлення коду?
25. Які вимоги до запису коментарів у тексті програми?
26. Як зробити текст програми читабельним?
27. Які правила форматування конструкцій умов та циклів?
28. Які правила найменування параметрів у параметричному циклі?
29. Що таке угорська нотація змінних? Навести три приклади.
30. Які правила найменування вказівників? Навести три приклади.
31. Які правила запису назв функцій? Навести п'ять прикладів.

32. Як правильно розділяти функції у тексті програми?
33. Які правила найменування змінних? Навести три приклади.
34. Які правила найменування констант? Навести три приклади.
35. Яка послідовність методів у кожній секції класу у мові C++? Навести приклад.
36. Як записуються класи та їх складові у мові C++?
37. Як можна підвищити продуктивність роботи на етапі кодування?
38. Назвіть переваги використання ReSharper C++.
39. Як можна провести навігацію за кодом?
40. Що означає термін “факторинг”?
41. Що таке рефакторинг коду? Навіщо його виконувати?
42. Що таке статичний аналіз коду?
43. Що є основним результатом наступного після проектування етапу? Які роботи виконують після цього етапу?
  44. Які переваги/недоліки модульної структури програми над монолітною?
  45. Що таке проектування виробу (у загальному випадку)? Порівняйте проектування ПЗ і проектування деякого матеріального виробу.
  46. Назвіть типові вигляди програмної системи (view), які формуються на етапі проектування.
  47. Які фахівці галузі ПЗ задіяні на етапі розроблення ПЗ?
  48. Чим мова програмування С принципово відрізняється від мови програмування С+?
  49. Перерахуйте найуживаніші мови програмування.
  50. Які є парадигми програмування? Яка відмінність між ними?
  51. Охарактеризуйте найновіші перспективні технології програмування.
  52. Охарактеризуйте базові ідеї процедурного (алгоритмічного) програмування.
- Назвіть мови програмування, що ґрунтуються на цій парадигмі.
  53. Охарактеризуйте базові ідеї об'єктно-орієнтованого програмування. Назвіть мови програмування, що ґрунтуються на цій парадигмі.
  54. Які є відомі Постоб'єктні технології програмування?
  55. Яка є альтернативна до імперативної парадигма програмування? Назвіть відповідні мови програмування.

## ЛАБОРАТОРНА РОБОТА № 2

**Тема.** Документування етапів проектування та кодування програми.

**Мета.** Навчитися документувати основні результати етапів проектування та кодування найпростіших програм.

### **Перелік питань для теоретичних відомостей.**

Означення етапу проектування ПЗ. Архітектура ПЗ. Зміст етапу проектування ПЗ. Рівні проектування ПЗ. Патерн проектування. Архітектурні подання програмної системи (view). Алгоритм: означення, властивості, форми представлення. Структури даних: означення, класифікація, задачі застосування. Зміст етапу кодування ПЗ (програмної реалізації). Стандарти оформлення текстів програми. Рефакторинг програмного коду. Застосунок для підвищення якості коду. Учасники процесу проектування й програмної реалізації системи. Основні результати етапу розроблення ПЗ. Місце й роль етапу розроблення ПЗ у життєвому циклі ПЗ.

### **Завдання**

#### **Необхідні програмні засоби для виконання завдання:**

- ^ середовище програмування мовою С/С++ (довільна версія),
- ^ текстовий редактор/процесор,
- ^ редактор ділової та інженерної графіки MS-Visio.

**Умова.** Для контролю проміжних результатів лабораторної роботи пропонується поділити завдання на частини.

**Частина I.** У розробленій раніше програмі до лабораторної роботи з дисципліни “Основи програмування” внести зміни - привести її до модульної структури, де модуль - окрема функція-підпрограма. Як такі функції запрограмувати алгоритми зчитування та запису у файл, сортування, пошуку, редагування, видалення елементів та решту функцій згідно з варіантом.

**Частина ІІ.** Сформувати пакет документів до розробленої раніше власної програми:

1. Схематичне зображення структур даних, які використовуються для збереження інформації;
2. Блок-схема алгоритмів - основної функції й двох окремих функцій-підпрограм (наприклад, сортування та редагування);
3. Текст програми з коментарями та оформленний згідно з наведеними вище рекомендаціями щодо забезпечення читабельності й зрозумілості.

Для схематичного зображення структур даних, блок-схеми алгоритму використати редактор MS-Visio.

**Частина ІІІ.** У редакторі MS-Visio розробити зразки фігур, які було використано для схематичного зображення структур даних програм як готові трафарети до використання. Сформувати свою бібліотеку фігур - окремий користувацький файл із використаними зразками.

### Варіанти завдань до теоретичних відомостей у звіті

Номер варіанта	Перелік контрольних питань
1	2
1	3, 15, 24
2	4, 10, 29
3	1, 25, 17
4	2, 20, 19
5	5, 30, 36
6	7, 22, 32
7	6, 26, 34
8	8, 27, 42
9	13, 29, 37
10	12, 18, 41
11	11, 33, 40
12	17, 28, 39
13	16, 20, 1
14	18, 32, 2
15	25, 35, 4
16	24, 34, 3
17	38, 21, 1
18	39, 22, 5
19	37, 20, 7

1	2
20	36, 19, 6
21	35, 18, 8
22	42, 17, 9
23	40, 15, 11
24	41, 16, 10
25	43, 34, 38
26	14, 33, 39
27	26, 11, 3
28	24, 12, 2
29	22, 13, 1
30	20, 14, 4

## **Вимоги до звіту**

Звіт оформлюють за загальними вимогами, викладеними у методичних рекомендаціях.

У теоретичних відомостях згідно з індивідуальним завданням необхідно дати відповідь на три контрольні питання з наведеного вище списку.

У розділі “Отримані результати” роздрукувати:

- *схематичне зображення використаних структур даних* (масив, список);
- *блок-схеми алгоритмів*;
- *сформований користувацький visio-файл фігур*;
- *текст програми*.

У висновках зазначити й охарактеризувати отримані під час виконання лабораторної роботи результати.

# ТЕМА 3. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 3.1. Означення та роль тестування

Випробування будь-якої програми є одним із найвідповідальніших етапів її розроблення й часто буває пов'язано з найбільшими труднощами і найбільшими втратами часу. У сучасних проектах на тестування відводиться понад 50 % загальних витрат.

Можна визначити такі основні цілі тестування програмного забезпечення:

- оцінювання якості програмного забезпечення;
- підвищення якості програми;
- запобігання появі помилок у майбутньому.

Цілі тестування можуть відрізнятися залежно від етапу розроблення ПЗ, на якому його проводять. Наприклад, на етапі кодування метою тестування буде пошук якомога більшої кількості помилок у роботі програми, що дасть змогу їх локалізувати і виправити. Водночас у процесі приймання програми замовником тестирують програму для того, щоби підтвердити, що система працює правильно. У період супроводу тестування переважно необхідно для того, щоб переконається у відсутності нових помилок, що з'явилися під час внесення змін до нових її версій.

Головне ж завдання тестування - пошук помилок. Успішним вважають тестування, коли, власне, знайдено помилки, а не навпаки.

Цей етап тестування варто розпочинати якомога раніше, адже згідно з результатами досліджень відомого фахівця інженерії ПЗ Б. Боема, вартість виправлення помилок, знайдених на ранніх етапах життєвого циклу ПЗ, є значно меншою, порівняно із вартістю її виправлення, наприклад, на етапі експлуатації (рис. 3.1).

Варто пам'ятати, що тестування — процес ітераційний. Після виявлення і виправленняожної помилки обов'язково слід повторити виконання тестів, щоб переконатися в працевздатності програми.

Недосвідчені програмісти часто не розрізняють етапів відлагодження та тестування програми, підміняючи одне поняття іншим.

Відлагодження і тестування програмного забезпечення - це два чітко відмінні етапи, у яких різні цілі.



Рис. 3.1. Графік вартості виправлення помилок на різних етапах життєвого циклу програмного забезпечення

**Відлагодження** - усунення синтаксичних помилок та помилок кодування. Синтаксичні помилки (порушення синтаксису мови програмування) порівняно легко “відфільтрувати” компілятором. Натомість семантичні помилки (логічні помилки) можна не виявити, особливо в ділянках коду, яким рідко передається управління.

Згідно з міжнародною термінологією програмної інженерії, *тестування програмного забезпечення* (англ. Software Testing) - це процес перевірки готової програми в статиці (перегляди, інспекції, налагодження вихідного коду) і в динаміці прогону скінченного набору тестових даних для різних способів виконання програми та порівняння отриманих результатів із заздалегідь запланованими.

Тестування ПЗ - це техніка контролю його якості. У вужчому сенсі, тестування - це перевірка відповідності фактичної поведінки програми очікуваній, що здійснюється на обмеженому наборі перевіркових даних, які було вибрано певним чином (не довільно).

Тестування ПЗ передбачає: планування тестування, проектування тестів, проведення тестування та аналіз результатів тестування.

На ранніх етапах розроблення ПЗ виявляти помилки допомагає статичний аналіз коду. *Статичний аналіз коду* - це уважна перевірка коду (найчастіше - вихідного коду програми) з метою виявлення в ній семантичних помилок без виконання програми. Найефективнішою і найдорожчою є перевірка коду командою програмістів (англ. техніка перегляду, Code Review).

Вартість такої перевірки зумовлена значними затратами часу і тим, що увага людини доволі швидко притупляється. Частково працю людей заміняє спеціальне ПЗ (одним з прикладів такого ПЗ є PCLint). Статичні аналізатори коду допомагають зекономити час, вказуючи розробникам на ділянки коду з потенційними помилками. Зокрема, статичні аналізатори допомагають “виловити” помилки, що виникають при неуважному редагуванні скопійованих фрагментів коду, наприклад:

```
dist1 = sqrt(x1^2 + y1^2 + z1^2);  
dist2 = sqrt(x2^2 + y2^2 + z1^2);
```

На відміну від статичного, *динамічний аналіз коду* - перевірка працевздатності під час виконання програмного коду на реальному чи віртуальному процесорі.

Якість ПЗ передбачає низку характеристик і коректність є лише однією з них. Програму, яка задовольняє специфікації, тобто видає очікувані відповіді на визначені комбінації значень вхідних даних, називають *коректною*.

З-поміж інших характеристик, які входять до поняття якості, є також і такі:

*Ефективність* - міра використання системних ресурсів (зокрема обсягу пам'яті, завантаженість процесора);

*Надійність* - здатність системи функціонувати як у наперед визначених умовах, а також у випадку нестандартних, непередбачуваних ситуацій чи збоїв у роботі. Одним із показників надійності є середній інтервал між відмовами програмної системи;

*Практичність* - простота у вивчені та застосуванні програмної системи;

*Цілісність* - здатність системи запобігти неавторизованому або некоректному доступу до програм і даних;

*Адаптованість* - можливість використання системи без її зміни у середовищах, для яких її спочатку не призначали.

*Живучість* - здатність системи функціонувати у разі введення недопустимих даних або ж у напруженіх умовах.

Як бачимо, надійність програми - це властивість програми, суворіша, ніж коректність, оскільки програма може бути коректною, але не надійною. Програма є *надійною*, якщо вона коректна, прийнятно реагує на неточні вхідні дані і задовільно функціонує в незвичайних умовах.

Програму не можна здавати в експлуатацію доти, поки не буде впевненості в її надійній роботі.

Етап тестування має на меті визначення коректності та надійності програми. Тобто тестування дає відповідь на запитання, чи програма насправді вирішує поставлене перед нею завдання і видає правильний результат за будь-яких умов - типових і нетипових. Варто пам'ятати, що тестування доводить наявність помилок, але не їхню відсутність.

На ранніх етапах розвитку інженерії ПЗ этап тестування програм не розглядали окремо від реалізації. Вважали, що перевіряти програму можна на інтуїтивному рівні, і це схоже на мистецтво. Сьогодні, безперечно, тестування є одним із найважливіших етапів життєвого циклу програмного забезпечення. Підхід до тестування лише як до мистецтва є неактуальним, оскільки тестування - це наука зі своїми моделями, методами та інструментами.

У зведенні знань з інженерії програмного забезпечення SWEBOK визнано основні розділи області знань, пов'язані з тестуванням ПЗ (Додаток Н).

Серед основних принципів тестування є такі:

1. Про тестування необхідно думати протягом усього періоду розроблення програми.
2. Повнота тестування забезпечується не кількістю тестів, а правильним вибором тестових даних для перевірки всіх можливих варіантів роботи програми.
3. У кожному наступному тесті необхідно використовувати клас даних, відмінний від попереднього.

Перший принцип означає, що необхідно писати програми так, щоб легко потім було знайти і виправити помилки. Щонайменше у такому випадку потрібне документування. Більших зусиль розробників вимагає врахування можливості введення некоректних даних, передбачення нетипових ситуацій у роботі програми. Другий принцип вказує на те, що протестована програма, наприклад, на 100 наборах даних, не є обов'язково якісно перевіrenoю. Все залежить від якості тестових даних, тобто як було спроектовано тести. Третій принцип саме вказує, як необхідно обирати дані для проведення тестування.

Виокремлюють різні види тестування залежно від обраного критерію.

**За об'єктом тестування** можна виділити функціональне та нефункціональне тестування. У першому випадку йдеться про перевірку правильності виконання функцій програмою, в іншому - про перевірку нефункціональних характеристик програми, а саме: тестування ефективності, безпеки, зручності використання тощо.

За часом проведення розрізняють димове тестування, санітарне, регресійне тестування, альфа- та бета-тестування. Димовим тестом є перевірка,

чи програма запускається взагалі (якщо вона, наприклад, працює з базою даних, то чи є правильне підключення). Тобто, димовий тест дає відповідь на питання, чи є сенс переходити до подальшого тестування, але не дає змоги зробити висновок про правильність роботи програми. Термін походить з інженерного середовища: якщо із ввімкненням приладу не з'явився дим, можна вважати, що першочергове випробування прилад пройшов успішно. Змістом димового тестування ПЗ є елементарна, дуже поверхнева перевірка, чи ключові модулі програми виконують свої основні функції. Санітарне тестування, або перевірка узгодженості - вузьконапрямлене тестування, достатнє для доведення того, що конкретна функція працює згідно із заявленими у специфікації вимогами. Таке тестування є підмножиною регресійного тестування і використовується для визначення працездатності певної частини програми після змін, вироблених у ній або навколоїшньому середовищі. Регресійне тестування орієнтоване на повторне вибіркове тестування системи або її компонентів після внесення в них змін на тих самих тестах, що і до модифікації. Альфа-тестування - це тестування системи групою тестування організації-розробника, а бета-тестування системи проводять "зовнішні" користувачі.

Згідно з тим, як враховують *інформацію про програмну систему*, розрізняють тестування "чорної скриньки" англ. (black box Testing,), тестування "білої скриньки" (white box Testing, англ.). Тестування "чорної скриньки" - це перевірка роботи програми без врахування внутрішньої структури програми (тільки як працюють функції), а тестування на підставі аналізу структури програми - тестування "білої скриньки". Тому тестування "чорної скриньки" - це функціональне тестування, а тестування "білої скриньки" - структурне тестування. Є ще тестування "сірої" скриньки - це комбінація методів "білої" і "чорної" скриньки. При цьому тестувальникам відомі деякі (але не всі) деталі реалізації системи, що є об'єктом тестування.

**За рівнем тестування** розрізняють модульне, інтеграційне, системне та приймальне тестування. Рівні відображають компонентний підхід до розроблення ПЗ. Модульне тестування (англ. Unit Testing) - це метод тестування програмного забезпечення, який полягає в окремому тестуванні кожного модуля коду програми. Модулем називають найменшу частину програми, яку можна протестувати. У процедурному програмуванні модулем вважають окрему функцію, в об'єктно-орієнтованому програмуванні - інтерфейс, клас. Інтеграційне тестування (англ. Integration Testing) - це фаза тестування програмного забезпечення, під час якої окремі модулі програми комбінуються та тестиються

разом, у взаємодії. Системне тестування тестує інтегровану систему для перевірки її відповідності всім вимогам. Приймальне тестування (*англ.* Acceptance Testing) - це фінальний етап тестування програми перед здаванням в експлуатацію, який проводять для перевірки на відповідність до вимог замовника.

**За методом тестування** розрізняють ручне, автоматизоване та напівавтоматизоване тестування. Ручне тестування - це процес пошуку дефектів у роботі програми, коли тестувальник перевіряє працездатність програми, якщо б він був користувачем. Автоматизоване тестування використовує програмні засоби для виконання тестів і перевірки коректності результатів виконання, що спрощує тестування і скороочує його тривалість. Головна перевага автоматизованого тестування полягає в можливості повторного прогону тестів без участі людини. Очевидно, напівавтоматизоване тестування - це поєднання ручного та автоматизованого тестування.

Наведена вище класифікація не є вичерпною. Найчастіше використовувані види тестування наведено на рис. 3.2.

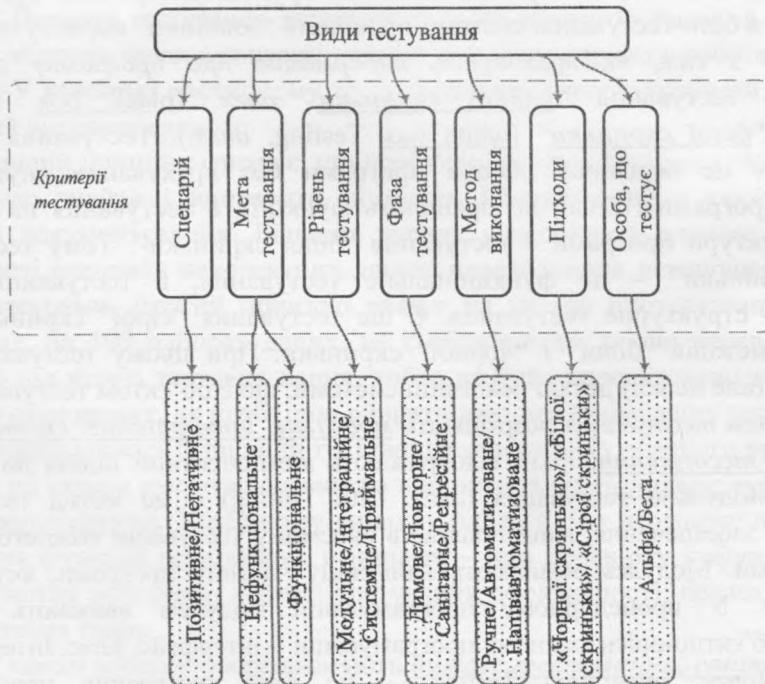


Рис. 3.2. Класифікація видів тестування

Окремі види тестування на прикладі крісла як об'єкта тестування наведено в табл. 3.1.

**Таблиця 3.1**

**Різні види тестування на прикладі використання крісла**

Вид тестування	Випробування крісла
1	2
Автоматизоване тестування	<p>Розробити роботів для випробування. Нехай з ним розбереться робот-порохотяг.</p>
Альфа-тестування	Меблярі самі випробовують свій виріб
Бета-тестування	Розсадити гостей перед тривалою бесідою.
Верифікація	<p>Є сидіння чи тільки каркас?</p> <p>Розміри крісла відповідають зросту та вазі середньостатистичної людини?</p> <p>Пружини не тиснуть?</p>
Димовий тест	<p>Сісти в крісло та й відразу встati. Не розпалося? Вдалося сісти?</p>
Інтеграційне тестування	<p>Сидіння, спинка, ніжки, підлокітники спiввiмiрнi? Непропорцiйностi немає?</p> <p>Все разом добре з'єднано?</p>
Модульне тестування	<p>Окремо посидимо на подушках.</p> <p>Потягнемо-роздягнемо тканину оббивки.</p> <p>Покладемо книгу на підлокітник.</p> <p>Упремося на спинку.</p>
Негативне тестування	<p>Плюхнутися з розмаху.</p> <p>Сісти на спинку крісла. Не зламається?</p> <p>Поставити чорнильну пляму. Чи вiдчиститься?</p> <p>Помити з садового шлангу. Не розпадеться?</p> <p>Дати котові поточити кігті.</p>
Тестування “чорної скриньки”	<p>Невідомо, як воно влаштоване.</p> <p>Керуємося інтуїцією</p>
Функціональне тестування	<p>Людина вміщається? Великої ваги? Дитина? А якщо відкинутися на спинку - не перекинеться крісло?</p> <p>Висота регулюється?</p> <p>Легко пересувати та переносити? Матеріал не токсичний?</p>
Нефункціональне тестування	
<i>Тестування безпеки</i>	Протягти крісло пiдлогою. Коліщата не драпають покриття?
<i>Тестування документацiї</i>	Оббивка не забруднює і не псує одяг?
	Інструкцiя зрозумiла та вичерпна?

1	2
<i>Тестування зручності</i>	Зручно сидіти? М'язи не напружені?
	Зручно чистити? Є доступ до місць, де збиратиметься бруд?
	Є підлокітники? Руки справді відпочивають?
<i>UI -тестування</i>	Дизайн цього століття?
	Зовнішній вигляд естетичний?
	Колір не надто кричущий?
<i>Тестування на сумісність</i>	Де можна поставити? У кабінеті? У вітальні? На кухні?
	Можна скласти? Засунути під стіл?
	Скільки років зможе слугувати?
Тестування продуктивності	Термін придатності?
	Чи витримає людину з надвеликою вагою? Не сильно прогинається?
	Чи витримає постійне використання?
	Тканина не протирається?

Серед усіх видів тестування докладніше розглянемо функціональне та структурне.

### 3.2. Функціональне тестування

Функціональне тестування - це перевірка того, чи програма правильно виконує усі свої функції, корисні з погляду кінцевого споживача. Функціональне тестування ще називають тестуванням “чорної скриньки”, оскільки для його проведення не потрібно знати внутрішньої організації коду, а лише як взаємодіяти з програмним продуктом. Функціональне тестування тісно пов’язане з поняттям верифікації.

Верифікація (англ. Verification) - перевірка, чи програмне забезпечення відповідає технічному завданню (специфікації вимог), тобто, чи насправді програму реалізовано саме так, як описано у документації на неї. Крім того, процес верифікації передбачає перевірку правильності специфікацій вимог на їх відповідність, несуперечність, повноту і здійсненність, а також на дотримання вимог стандартів.

Перший тест має бути максимально простий, оскільки початкова мета тестування полягає в тому, щоб перевірити, чи працює програма взагалі. Таку перевірку часто називають **димовий тест**.

Для складних систем неможливо перебрати усі комбінації вхідних даних, особливо якщо вхідні дані не є дискретними. Часом використовують випадкове (стохастичне) тестування, коли програму перевіряють на наборі випадково згенерованих даних. Для забезпечення надійності ПЗ з відмовою  $10^{-5}$  і помилкою, не більшою за 5 %, потрібно згенерувати 299572 тестів. Тести при цьому мають бути **незалежними**. Як бачимо, випадкове тестування потребує надто багато зусиль.

Для зменшення кількості тестів було запропоновано поняття класу еквівалентності (термін запропонував Майєрс).

**Клас еквівалентності** - набір даних із загальними властивостями. Обробляючи різні елементи класу, програма має поводитися однаково.

Розрізняють “правильні” та “неправильні” класи еквівалентності. Нехай, наприклад, програма обробляє оцінки студентів. “Правильним” класом еквівалентності будуть значення від 0 до 100 (за стобальною шкалою) включно, а двома “неправильними” класами еквівалентності: 1) від’ємні значення; 2) значення більші за 100. Значення на границях підлягають окремій перевірці.

Існують такі правила формування класів еквівалентності:

- 1) якщо умова введення задає діапазон  $n \dots m$ , то визначається один допустимий і два неприпустимі класи еквівалентності;
- 2) якщо умова введення задає конкретне значення, то визначається один допустимий і два неприпустимі класи еквівалентності;
- 3) якщо умова введення задає множину значень  $\{a, b, c\}$ , то визначають один допустимий і один неприпустимий клас еквівалентності;
- 4) якщо умова введення задає логічне значення, наприклад, true, то визначається один допустимий і один неприпустимий клас еквівалентності.

Отже, для функціонального тестування необхідно визначити “правильні” та “неправильні” класи еквівалентності та виділити граничні (екстремальні) значення.

У контексті перевірки функціонування програми в різних умовах етап тестування можна поділити на три частини (рис. 3.3):

1. Тестування програми в нормальніх умовах (допустимі значення вхідних даних).

2. Тестування програми в екстремальних умовах (границі/крайні значення вхідних даних).
3. Тестування програми у виняткових ситуаціях (значення вхідних даних за межами допустимого діапазону).



Рис. 3.3. Вибір вхідних даних для функціонального тестування відповідно до його типу

**Перевірка в нормальних умовах** передбачає тестування на основі даних, які характерні для реальних умов функціонування програми. Випадки, коли програма має працювати зі всіма можливими даними, надзвичайно рідкісні. Зазвичай існують конкретні обмеження на область зміни даних, в якій програма має зберігати свою працездатність.

**Перевірка в екстремальних умовах** має проводитися відразу після перевірки програми в нормальних умовах. Тестові дані цього етапу містять граничні значення області зміни вхідних даних, які програма має сприймати як правильні дані.

Для числових даних тестиють програму за початкового та кінцевого значень допустимої області зміни даних, у разі зміни довжини відповідного поля від мінімальної до максимальної. На практиці перевіряють не лише самі граничні значення, але й такі, що незначно від них відрізняються. Наприклад, якщо програма приймає дискретне число від 0 до 100, то потрібно перевірити її роботу при введених значеннях 0, 1, 99, 100. Якщо ж очікуване число є неперервним у діапазоні від 0 до 100, то перевіряють поведінку програми при введених значеннях в околі 0 та 100 відповідно.

Контроль граничних вхідних значень дає змогу виявити проблеми переповнення типів. Наприклад, якщо функція приймає два аргументи типу char і повертає їхню суму, теж оголошенну як тип char, то перевіркою правильності функції на прикладах всередині діапазону 0..255 рідко можна виявити проблему. Натомість передавання функції значень 255 і 255 одразу виявить переповнення типу.

Для файлів, які можуть мати від 0 до N записів, необхідно переконатися, що програма працює коректно, коли файл порожній, коли є тільки один запис, коли кількість записів у ньому є максимально допустимою або ж на одиницю меншою за максимально допустиму.

Загалом для впорядкованих даних (наприклад, для масивів чи динамічних списків) необхідно ретельно перевіряти перший і останній з елементів. Особливу увагу потрібно звертати на правильність обробки індексів масивів, особливо для кодів, написаних мовами, що не передбачають контролю виходу за межі масиву.

Типовими прикладами таких екстремальних значень є дуже великі числа, дуже малі числа і відсутність інформації. Для нечислових даних як екстремальні умови, окрім їх відсутності, потрібно використовувати спеціальні, нетипові символи, що охоплюють всі можливі ситуації. Процес використання екстремальних значень як тестових даних має назву *граничні випробування*.

Необхідно обов'язково перевірити усі виявлені граничні умови.

Ще один тип екстремальних умов - це граничні обсяги даних, коли вони складаються з надто малої чи, навпаки, надто великої кількості записів. Необхідно встановити, що відбувається з програмою, якщо їй на оброблення не надходить жодного елемента даних або лише один, і чи збережеться працевдатність програми при дуже великих наборах даних.

Особливий інтерес становлять так названі нульові приклади. Для числового введення - це звичайні нульові значення, для послідовності символів - це ланцюжок пропусків, для вказівників - нульове значення адреси. Якщо такого тестування не виконують, то згодом часто доводиться стикатися з незрозумілою поведінкою програми.

Перевірка у виняткових ситуаціях передбачає використання даних, значення яких знаходяться за межами допустимої області змін. Відомо, що всі програми розробляють з розрахунку на оброблення обмеженого набору даних. Найгіршим є варіант, коли програма сприймає некоректні дані як правильні і видає неправильний, але правдоподібний результат. Програма має відкидати будь-які дані, які вона не в стані опрацювати правильно.

Щодо перевірки програми в екстремальних і виняткових умовах, то тут можна дати такі рекомендації:

1. Дані, які містять пропуски, цифри і букви, необхідно випробовувати в найрізноманітніших комбінаціях;
2. Інколи найбільші ускладнення спричиняють дані, які містять відразу декілька помилок;
3. Особливими є помилки, спричинені неправильним використанням первого та останнього елементів опрацьованої інформації.

Згідно з загальноприйнятою термінологією в галузі інженерії ПЗ, перевірку у виняткових ситуаціях для непередбачених вхідних даних називають негативним тестуванням. А позитивним тестуванням, відповідно, називають перевірку роботи програми для передбачених вхідних даних. Варто зазначити, що до таких даних належать і не зовсім типові вхідні дані, але ще дозволені, наприклад, файл з максимально можливою кількістю даних чи файл з одним записом, який буде видалено. Тобто тестування для нормальних та екстремальних вхідних даних називають позитивним тестуванням.

Як було розглянуто вище, правильний відбір даних для тестування тієї чи іншої програми може значно полегшити процес знаходження в ній помилок.

До способів перевірки правильності комп'ютерних результатів належать обчислення вручну, отримання результатів з документації та інших інформаційних джерел, отримання результату за допомогою деякої іншої аналогічної програми.

### 3.3. Документування результатів тестування

Якщо відлагоджують ПЗ його розробники, то тестиє готові програми команда тестувальників. Оскільки у разі функціонального тестування програма - це "чорна скринька", і тестувальник не знає, в якій ділянці коду міститься помилка, то важливим є не лише пошук помилок, але й їхній правильний опис. У розвинених компаніях (згадаймо модель СММ) для відстеження дефектів застосовуються спеціальні багтрекінгові (*англ.* Bugtracking) системи. У таких системах фіксують опис дефекту, достатній для його відтворення програмістом, особу, відповідальну за виправлення дефекту, рівень критичності дефекту, пріоритет і статус (відкритий, виправлений, закритий тощо; статус змінюють як тестувальник, так і розробник).

Згідно з тим, чи відбувається повний опис (документування) тестування, розрізняють види тестування - формальне чи неформальне тестування.

У будь-якому випадку (використовується система відстеження дефектів чи ні), результати тестування варто документувати. Оскільки знаходить помилку одна людина, а виправляє її інша, то ймовірність виправлення помилки залежить від того, наскільки вона задокументована (взаємне розуміння учасників проекту є однією з основних проблем під час розроблення ПЗ).

Складаючи звіт про знайдені помилки, необхідно максимально повно, просто і доступно її описати, причому одразу після виявлення, не покладаючись на пам'ять.

Зважаючи на сукупний практичний досвід компаній-розробників ПЗ, потрібно відобразити у звіті про знайдені дефекти таку інформацію:

1. Назва програми.

2. Версія програми. Потреба в зазначені версії зумовлена тим, що після виправлення попередніх дефектів створюють нову версію; версія дає змогу уникнути плутанини.

3. Опис дефекту. Для складних програм з розгалуженим інтерфейсом і багатьма переходами між графічними екранами необхідно описати повну послідовність дій, що дала б змогу виявити дефект (можна долучити екранні знімки, якщо вони інформативні). Для консольної програми достатньо описати набір вхідних даних.

4. Тип звіту може бути зазначений як:

- Помилка кодування - (програма поводиться неправильно, на думку тестувальника);
- Помилка проектування;
- Пропозиція — тестувальник викладає свою ідею, як покращити програму (це не є дефект);
- Невідповідність документації - при цьому потрібно дати посилання на конкретні сторінки документації, помилка може бути у програмі, що спільно з'ясовують різні учасники команди;
- Запитання - програма робить щось, чого тестувальник не розуміє і хоче уточнення;

5. Ступінь важливості (критичності) - суб'єктивний критерій, типово вказують рівень (низький, середній, високий).

6. Статус дефекту - широко вживане поняття, використовується для поліпшення взаємодії між розробниками та тестувальниками під час моніторингу роботи над виявленням і виправленням дефектів, зокрема у багтрекінгових системах. Типові приклади статусу дефекту: New (щойно описаний), In Process (опис дефекту прочитаний розробником, зрозумілий йому і над усуненням дефекту вже ведеться робота), Fixed (розробник встановлює цей статус, коли вважає, що йому вже вдалося усунути дефект), Verified (тестувальник перевірив роботу розробника і визнав, що дефект насправді усунуто), Cannot Verify (тестувальник не може перевірити результати виправлення дефекту, оскільки наявних у нього ресурсів недостатньо для відтворення умов, за яких виявлено дефект), Deferred (виправлення дефекту відкладено на деякий час).

Під час виконання лабораторної роботи потрібно оформити результати функціонального тестування у вигляді табл. 3.2.

*Таблиця 3.2*

**Результати функціонального тестування програми  
<Назва програми>, версія <версія>**

№ з/п	Тестові дані	Фактичні результати	Очікувані результати	Ступінь критичності	Тип звіту	Примітки
Тест нормальних умов						
1						
2						
Тест граничних умов						
1						
2						
Тест виняткових ситуацій						
1						
2						

### 3.4. Структурне тестування

*Структурне тестування* (тестування “білої” або “скляної” скриньки) - це відлагодження алгоритму програми, яке, звісно, передбачає доступ до вихідного коду.

Зміст структурного тестування полягає у перевірці вихідного коду програми. Є такі різновиди структурного тестування:

- тестування на основі потоку управління програми;
- тестування на основі потоку даних програми;
- мутаційне тестування.

*Тестування на основі потоку керування програми* полягає у виконанні таких дій. Вихідний код програми подають у вигляді графу, в якому вершинами є оператори, а дугами - переходи між ними. Такий граф називають *графом управління*.

Для повного структурного тестування потрібно перевірити усі шляхи у графі управління (“пройтися” усіма можливими шляхами від початкової до кінцевої вершини принаймні один раз). Граф управління зручно будувати на основі блок-схеми програми.

Розглянемо побудову графу управління на прикладі програми для розв’язання квадратного рівняння. Блок-схему зображенено на рис. 3.4.

Як бачимо, кожен блок позначено великою латинською літерою (літери не повторюються). Порядок позначення не є принциповим, однак зручно “пройтися” усіма блоками згори донизу зліва направо. Для побудови графу заміняємо блоки у блок-схемі умовним позначенням вершин, внаслідок чого отримаємо граф управління (рис. 3.5).

Отже, необхідно скласти тести для проходження таких шляхів у графі управління:

1. [Початок]->[A]->[B]->[C]->[E]->[I]->[Кінець]
2. [Початок]->[A]->[B]->[C]->[G]->[Кінець]
3. [Початок]->[A]->[B]->[D]->[F]->[G]->[Кінець]
4. [Початок]->[A]->[B]->[D]->[F]->[H]->[J]->[L]->[M]->[Кінець]
5. [Початок]->[A]->[B]->[D]->[F]->[H]->[K]->[N]->[Кінець]

Очевидно, що якщо кожна з двох чи більше вершин зв’язані тільки з наступною та попередньою (вершини “нанизані” на кшталт намистинок, наприклад, як вершини на рис. 3.5), то вони неодмінно належатимуть одному шляху, відповідно, граф можна спростити, поєднавши такі вершини в одну (як показано на рисунку штриховою лінією).

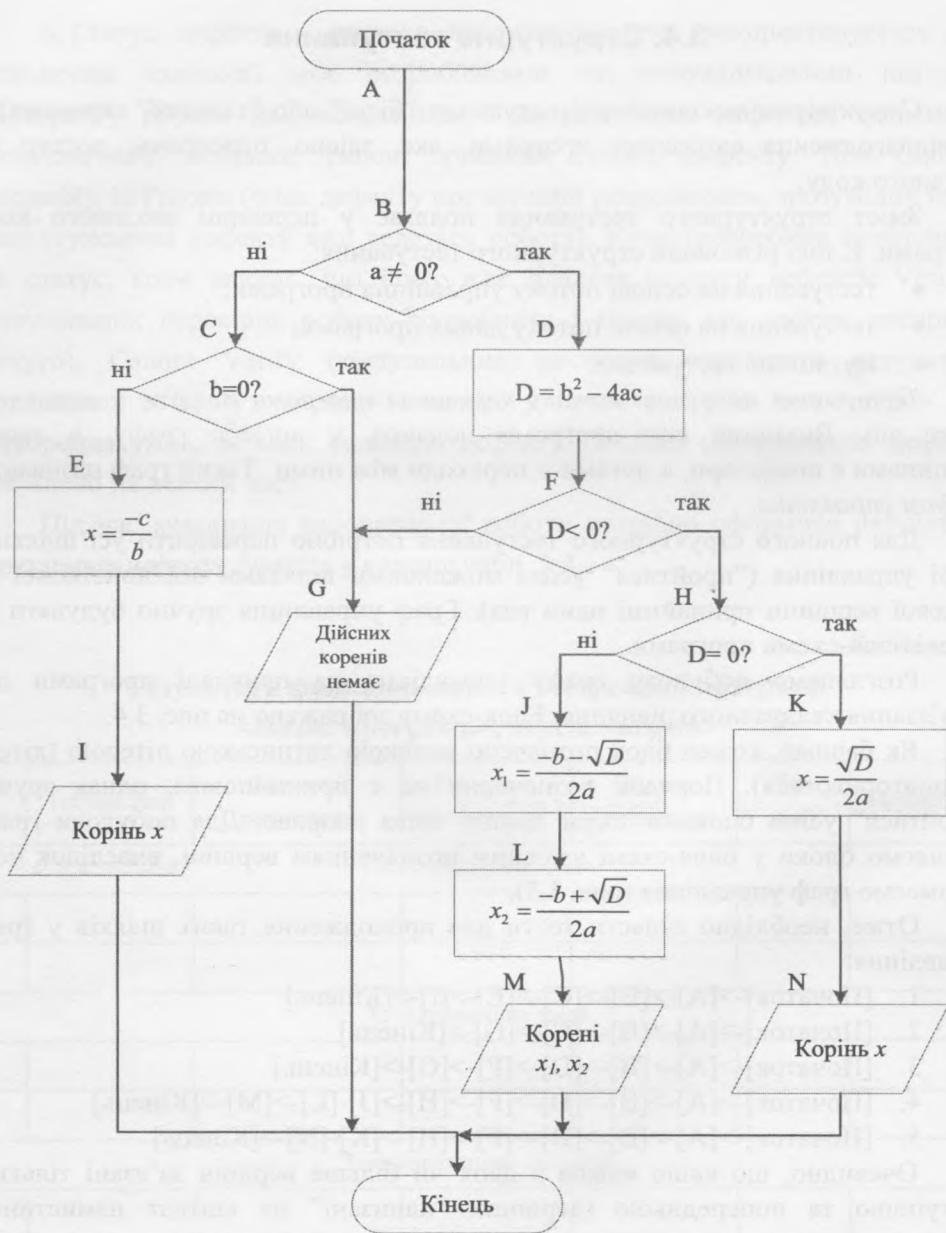


Рис. 3.4. Блок-схема алгоритму розв'язання квадратного рівняння

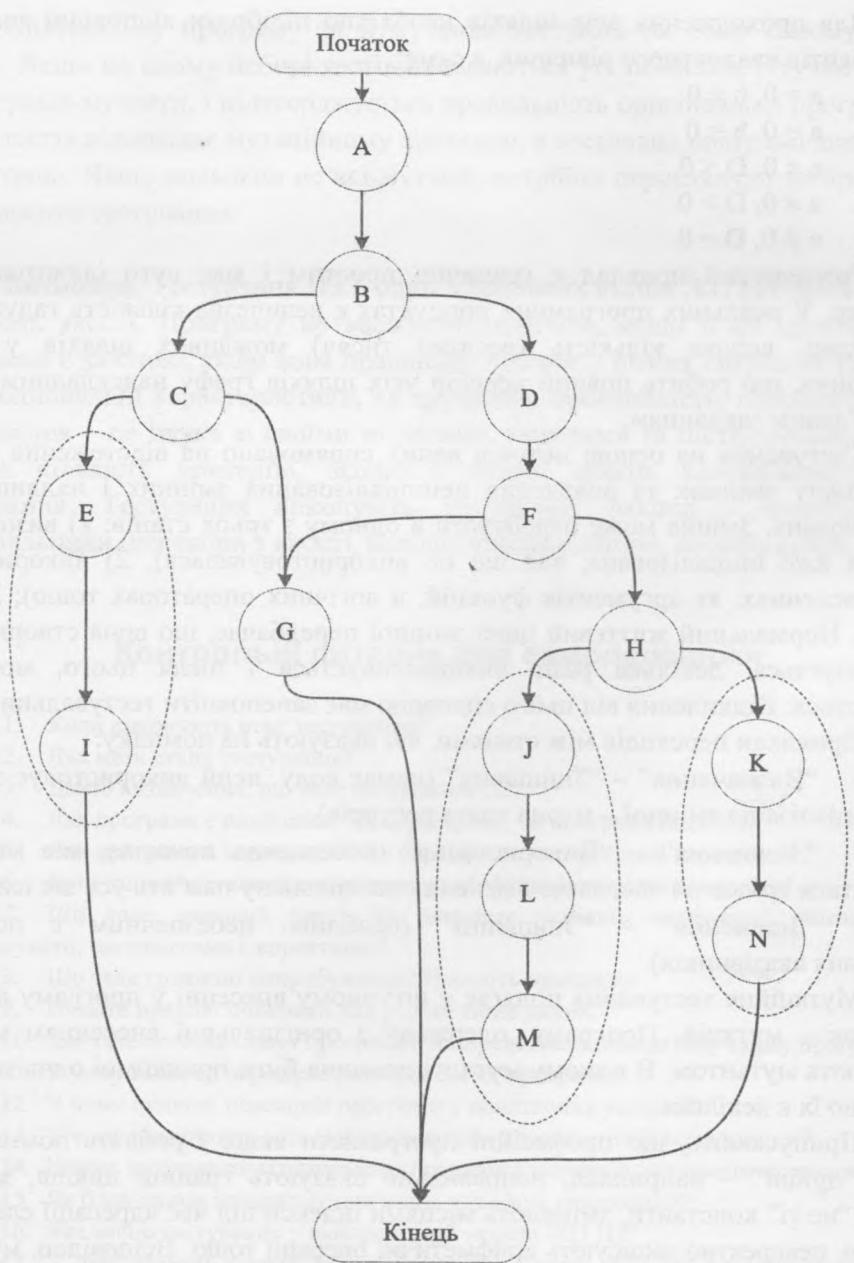


Рис. 3.5. Граф управління програми

Для проходження всіх шляхів необхідно підібрати відповідні значення коефіцієнтів квадратного рівняння, а саме:

1.  $a = 0, b^0$
2.  $a = 0, b = 0$
3.  $a \neq 0, D < 0$
4.  $a \neq 0, D > 0$
5.  $a \neq 0, D = 0$

Розглянутий приклад є гранично простим і має суто ілюстративний характер. У реальних програмних продуктах є величезна кількість галужень і, відповідно, велика кількість (десятки тисяч) можливих шляхів у графі управління, що робить повний перебір усіх шляхів графу надскладним або ж нерозв'язним завданням.

Тестування на основі потоків даних спрямовано на відстеження життєвого циклу змінних та виявлення неініціалізованих змінних і надлишкових присвоювань. Змінна може перебувати в одному з трьох станів: 1) визначення (zmінна вже ініціалізована, але ще не використовувалася), 2) використання (в обчисленнях, як аргументів функцій, в логічних операторах тощо); 3) знищення. Нормальний життєвий цикл змінної передбачає, що вона створюється, ініціалізується, декілька разів використовується і після цього, можливо, знищується. Відхилення від цього сценарію має занепокоїти тестувальника.

Приклади переходів між станами, які вказують на помилку:

- “Визначення” - “Знищенння” (немає коду, який використовує змінну, тож ініціалізація змінної - марна трата ресурсів),
- “Знищенння” - “Використання” (небезпечна помилка, яка може не виявлятися тривалий час, якщо вказівник на звільнену пам’ять усе ще існує),
- “Знищенння” - “Знищенння” (особливо небезпечним є подвійне знищенння вказівників).

Мутаційне тестування полягає у штучному внесенні у програму дрібних помилок - мутацій. Програму, одержану з оригінальної внесенням мутації, називають мутантом. В одному мутанті повинна бути принаймні одна мутація, а типово їх є декілька.

Припускають, що професійні програмісти якщо і роблять помилки, то лише “дрібні” - наприклад, неправильно вказують граници циклів, застосовують “не ті” константи, змінюють місцями індекси під час адресації елементів масивів, некоректно записують арифметичні операції тощо. Відповідно, мутації - це змінені граници циклів, змінені індекси і т.д.

Оригінальну програму та її мутанти тестиють на тому самому наборі тестів. Якщо на цьому наборі тестів виявляються усі помилки, штучно внесені у програми-мутанти, і підтверджується правильність оригінальної програми, то набір тестів відповідає мутаційному критерію, а тестована програма вважається коректною. Якщо виявлено не всі мутації, потрібно переглянути набір тестів і продовжити тестування.

**Висновок.** Тестування ПЗ - один з основних етапів ЖЦ ПЗ, мета якого - контроль якості. Програму не вважають робочою, якщо її не протестовано. Програма є якісною, якщо вона правильно працює у різних ситуаціях і має такі нефункціональні характеристики, як зручність, ефективність, надійність тощо. Тестування - це наука зі своїми моделями, методами та інструментами. Існує велика кількість критеріїв, згідно з якими можна класифікувати види тестування. Тестування виконують спеціальні фахівці з інженерії ПЗ: тестувальники, інженери з якості, валідатори, інспектори, верифікатори.

## Контрольні питання для самоперевірки

1. Коли виконують етап тестування?
2. Яка мета етапу тестування?
3. Дайте визначення, що таке тестування ПЗ.
4. Яка програма є надійною? Як перевірити, чи програма надійна?
5. Яка програма є коректною? Як перевірити, чи програма коректна?
6. Які є типи тестування в контексті умов функціонування програми?
7. Що таке димовий тест? Чи дозволяє успішно пройдений димовий тест стверджувати, що програма є коректною?
8. Що таке граничні випробування? Наведіть приклади.
9. Назвіть нульові приклади для різних типів даних.
10. Що таке логічна схема програми? Як представляють логічну схему програми?
11. У чому полягає перевірка логічної схеми програми?
12. У чому полягає перевірка програми у виняткових умовах?
13. У чому відмінність між відлагодженням програми та її тестуванням?
14. Назвіть інструменти відлагодження програми у типовому середовищі програмування.
15. Як блок-схема алгоритму пов'язана з графом управління?
16. Яке місце тестування у найпростішій моделі ЖЦ ПЗ?
17. У якій моделі паралельно до основних етапів ЖЦ ПЗ розробляються плани випробувань та тести?

18. У якій моделі враховано недолік каскадної моделі - неможливість повернення на попередні етапи у разі виявлення помилок?
19. Які основні принципи тестування?
20. Як можна визначити, чи правильні отримано за допомогою програми результати?
21. Чим відрізняється функціональне тестування від нефункціонального?
22. Що означає властивість “практичність ПЗ”? Як її можна перевірити?
23. Що означає властивість “адаптованість ПЗ”? Як її можна перевірити?
24. Що означає властивість “живучість ПЗ”? Як її можна перевірити?
25. Сформулюйте рекомендації стосовно тестування на основі власного досвіду програмування, про які не згадувалося вище?
26. До якої групи та категорії належить процес тестування згідно із стандартом ISO15504?
27. Чим відрізняється тестування “чорної” та “білої” скриньки?
28. Що таке клас еквівалентності?
29. Як визначити повноту структурного тестування?
30. Що таке регресійне тестування?
31. Що таке приймальне тестування?
32. Що таке модульне тестування?
33. Чим відрізняється інтеграційне тестування від системного тестування?
34. Наведіть приклади нефункціонального тестування.
35. Як класифікують тестування за методом проведення тестування?
36. Яке тестування називають позитивним?
37. Яке тестування називають негативним? Наведіть приклад.
38. Хто проводить бета-тестування?
39. Що таке альфа-тестування?
40. Яке тестування називають формальним, а яке — неформальним?
41. Які є різновиди структурного тестування?
42. У чому суть мутаційного тестування? Наведіть приклади мутацій.
43. Що таке верифікація ПЗ?
44. Як описується тип звіту у документуванні тестуванні?
45. Які фіксуються типи дефектів під час тестування ПЗ?
46. Як вказується ступінь критичності помилок у звіті про тестування?
47. Як проводять статичний аналіз коду?
48. Чим динамічний аналіз коду відрізняється від статичного?
49. Що таке якість програмного продукту?
50. Наведіть характеристики якості ПЗ.
51. Назвіть документи, згідно з якими означено якість ПЗ.
52. Чим валідація відрізняється від верифікації?
53. Які категорії фахівців ПЗ задіяно у процесі тестування ПЗ?
54. Які роботи виконують після того, як складено звіт про тестування?
55. Яка частка вартості тестування ПЗ у загальній вартості типового масштабного проекту з розроблення ПЗ?
56. Що передбачає якісне тестування ПЗ?

## ЛАБОРАТОРНА РОБОТА № 3

**Тема роботи.** Функціональне й структурне тестування ПЗ.

**Мета роботи.** Отримати практичні навички застосування основних методів та принципів тестування на прикладі перевірки правильності власної програми.

### Перелік питань для теоретичних відомостей.

Означення етапу тестування ПЗ. Відлагодження програми. Принципи тестування ПЗ. Зміст етапу тестування ПЗ. Контроль якості ПЗ. Характеристики якості ПЗ. Класифікація та критерії класифікації видів тестування ПЗ. Верифікація ПЗ. Означення різних видів тестування. Різновиди функціонального тестування: тестування програми в нормальних умовах, тестування програми в екстремальних умовах, тестування програми у виняткових ситуаціях. Формальне тестування. Структурне тестування. Граф управління програми. Рекомендації до оформлення звіту про тестування ПЗ. Учасники процесу тестування ПЗ. Основні результати етапу тестування ПЗ. Місце й роль етапу тестування ПЗ у життєвому циклі ПЗ.

### Завдання

#### Необхідні програмні засоби для виконання завдання:

- ^ середовище програмування мовою C/C++ (довільна версія),
- ^ текстовий редактор,
- ^ редактор ділової та інженерної графіки MS-Visio.

#### Умова.

1. Провести **функціональне тестування** програми (варіанти 1-30 лабораторної роботи № 1), використавши такі тести:

- a) димовий тест;
- b) тести нормальних умов;
- c) тести екстремальних(границьких) умов;
- d) тести виняткових умов.

Коротко описати, в чому полягає димовий тест для тестованої програми.

Результати тестів а, б, с, д оформити у вигляді таблиці (за зразок взяти табл. 3.2).

2. Знайдені дефекти у роботі програми виправити. Провести повторне тестування, результати якого оформити окремою аналогічною таблицею.

3. Окремо провести тестування логічної схеми програми. Для цього зобразити її у вигляді графа управління, використавши редактор ділової та інженерної графіки MS-Visio. Вершини графу пронумерувати. Записати усі отримані маршрути роботи програми, використовуючи номери вершин. Для кожного маршруту провести тестування і записати результати тестування у звітній таблиці.

### *Примітки.*

1) формуючи тести для проведення функціонального тестування, використайте складене раніше технічне завдання.

2) для побудови графу логічної схеми програми використайте раніше створені блок-схеми алгоритмів.

### Варіанти завдань до теоретичних відомостей у звіті

Номер варіанта	Перелік контрольних питань
1	2
1	2, 20, 19
2	4, 10, 29
3	1, 25, 17
4	3, 15, 24
5	7, 22, 32
6	5, 30, 36
7	8, 27, 42
8	6, 26, 34
9	11, 33, 40
10	12, 18, 41
11	13, 29, 37
12	16, 20, 1
13	17, 28, 39
14	18, 32, 2
15	25, 35, 4
16	24, 34, 3
17	37, 20, 7
18	38, 21, 1
19	39, 22, 5
20	36, 19, 6
21	35, 18, 8
22	40, 15, 11
23	41, 16, 10

## **Продовження таблиці**

1	2
24	42, 17, 94
25	24, 12, 2
26	14, 33, 39
27	26, 11, 3
28	3, 34, 38
29	1, 25, 44
30	2, 23, 43

### **Вимоги до звіту**

Звіт оформляють згідно із загальними вимогами, викладеними у методичних вказівках.

У теоретичних відомостях згідно з індивідуальним завданням необхідно дати відповідь на 3 контрольні питання з наведеного вище списку.

У розділі “Отримані результати” роздруковувати результати функціонального тестування програми, результати структурного тестування із зображенням графу управління.

У висновках зазначити й охарактеризувати отримані під час виконання лабораторної роботи результати.

## ТЕМА 4. ЗАДАЧА СУПРОВОДУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

### 4.1. Основні означення етапу супроводу

Коли програма протестована і в ній більше не залишилося серйозних дефектів, приходить час передавання її користувачам. Розпочинається експлуатація готової програми, а також її супровід.

Після випуску нової версії програми до роботи залучається відділ технічної підтримки. Його працівники забезпечують зворотний зв'язок з користувачами, їх консультування та підтримку. У разі виявлення користувачами тих чи інших помилок інформація про них передається у вигляді звітів команді розроблення, яка, залежно від серйозності проблем, або негайно їх виправлює, або відкладає їх до наступної версії програми.

Сучасний стан інженерії ПЗ характеризується зростанням складності програмних систем. Тому сьогодні для розробників надзвичайно важливо розробляти програмне забезпечення так, щоби в майбутньому його було легко супроводжувати: підвищувати продуктивність, додавати певний функціонал, адаптувати до нового середовища функціонування тощо.

Історично фаза супроводу ПЗ (рис. 4.1) стримувала менше, ніж інші фази життєвого циклу. Зараз це змінюється, оскільки організації прагнуть отримувати максимальну віддачу від своїх інвестицій у розроблення ПЗ, зберігаючи його працездатність якнайдовше.

У зводі знань з інженерії програмного забезпечення SWEBOK визначено основні розділи області знань, пов'язані зі супроводом ПЗ (Додаток П).

Етап супроводу (*англ. Maintenance*) розглядають з погляду задоволення вимог споживача у готовому програмному продукті.

Іноді проектні менеджери вирішують розробляти програмне забезпечення з нуля, змінюючи його архітектуру, інші технічні характеристики, оскільки внесення змін у готовий продукт стає дуже складним і ресурсомістким процесом.

Проблема супроводу насамперед пов'язана з обмеженим розумінням ПЗ, яке супроводжується: наприклад, якщо ПЗ розробляли одні програмісти, а супроводжують інші. Як спосіб вирішення такої проблеми розглядають використання документації, а також те, що компанії-замовники для супроводу звертаються до одних і тих самих програмістів.



Рис. 4.1. Зміст етапу супроводу ПЗ

Отже, етап супроводу — це процес створення і впровадження нових версій програмного продукту.

Причинами випуску нових версій можуть бути:

- необхідність виправлення помилок, виявлених у процесі експлуатації попередніх версій;
- необхідність удосконалення попередніх версій, наприклад, покращення інтерфейсу, розширення складу виконуваних функцій або підвищення продуктивності розробленої системи;
- зміна середовища функціонування, наприклад, поява нових технічних засобів та/або програмних продуктів, з якими взаємодіє супроводжуване програмне забезпечення.

На цьому етапі в програмний продукт вносять необхідні зміни, які так само, як в інших випадках, можуть вимагати перегляду проектних рішень, прийнятих на будь-якому попередньому етапі. Роль цього етапу істотно зросла, оскільки продукти тепер створюють ітераційно: спочатку випускають порівняно просту версію, потім наступну з великими можливостями, потім наступну і т. д. Саме це є причиною виділення етапу супроводу й експлуатації в окремий процес життєвого циклу програмного забезпечення.

Загалом можна виділити дві групи робіт етапу супроводу:

- усунення дефектів (*англ. Fixing*);
- удосконалення програми (*англ. Enhancing*).

Перший вид робіт спричиняє недостатньо повне тестування системи і відповідно не оплачується додатково, виконується у межах раніше виділеного бюджету вже виконаного проекту. Другий вид робіт спричиняють зміни факторів використання програми.

Прикладом задач вдосконалення програми є підвищення її якості.

У різних джерелах термінологія характеристик якості ПЗ різна. Створено різні моделі якості зі своїм набором характеристик й атрибутів, і вони можуть бути корисні у різних ситуаціях для обговорення, планування й оцінювання якості програмних продуктів.

Якість ПЗ (*англ. Software Quality*) - це сукупність характеристик ПЗ, які належать до його здатності задовольнити встановлені й передбачувані потреби користувача ПЗ.

Сьогодні найпоширеніша і багаторівнева модель якості ПЗ, наведена в наборі стандартів ISO 9126 (рис. 4.2). На верхньому рівні виділено шість основних характеристик якості ПЗ, кожну з яких визначають набором атрибутів, що мають відповідні метрики для подальшого оцінювання.

Функціональність (*англ. Functionality*) визначається здатністю ПЗ вирішувати завдання, які відповідають зафіксованим і очікуваним потребам користувача при заданих умовах використання ПЗ. Тобто ця характеристика відповідає за те, що ПЗ працює справно і точно, функціонально сумісно, відповідає стандартам галузі і захищено від несанкціонованого доступу.

Надійність (*англ. Reliability*) - здатність ПЗ виконувати необхідні завдання в позначеных умовах протягом заданого проміжку часу або вказану кількість операцій. Атрибути цієї характеристики - це завершеність і цілісність всієї системи, здатність самостійно і коректно відновлюватися після збоїв у роботі, відмовостійкість.

Зручність використання (*англ. Usability*) - можливість легкого розуміння, вивчення, використання і привабливості ПЗ для користувача.

Ефективність (*англ. Efficiency*) - здатність ПЗ забезпечувати необхідний рівень продуктивності відповідно до виділених ресурсів, часу та інших визначених умов.

Зручність супроводу/супроводжуваність (*англ. Maintainability*) - легкість, з якою ПЗ можна аналізувати, тестувати, змінювати для виправлення дефектів, для реалізації нових вимог, для полегшення подальшого обслуговування та адаптувати до певного оточення.

Портативність (*англ. Portability*) - характеризує ПЗ з погляду легкості його перенесення з одного оточення (software / hardware) в інше.



Рис. 4.2. Модель якості програмного забезпечення (ISO 9126-1)

Так само в кожній компанії з розроблення ПЗ можуть існувати свої стандарти якості, що відповідають конкретній специфіці роботи та вимогам до нього.

## 4.2. Зручність використання програми

Однією з ознак якісної програми є зручність її використання, або практичність ПЗ.

Зручність використання залежить від того, наскільки правильно побудовано інтерфейс користувача.

Інтерфейс користувача - сукупність засобів для обробки та відображення інформації, які максимально зручні для користувача.

У графічних системах інтерфейс користувача реалізується багатовіконним режимом, змінами кольору, розміру, видимості (прозорість, напівпро-

зорість, невидимість) вікон, їхнім розташуванням, сортуванням елементів вікон, гнучкими налаштуваннями і самих вікон, і окремих їхніх елементів (піктограми, ярлики, вкладки, кнопки, шрифти тощо), доступністю багатокористувацьких налаштувань.

Графічний інтерфейс користувача (*англ. GUI*, Graphical user interface) - інтерфейс між комп'ютером та його користувачем, що використовує піктограми, меню і вказівний засіб для вибору функцій та виконання команд. Зазвичай можливе відкриття більше ніж одного вікна на одному еkranі.

GUI - система засобів для взаємодії користувача з комп'ютером, основана на представленні всіх доступних користувачеві системних об'єктів і функцій у вигляді графічних компонентів еkrana (вікон, значків, меню, кнопок, списків і т. п.). При цьому, на відміну від інтерфейсу командного рядка, користувач має довільний доступ (за допомогою клавіатури або пристрою координатного введення) до всіх видимих еcranних об'єктів.

Вперше концепцію GUI запропонували вчені з дослідницької лабораторії Xerox PARC у 1970-х, але отримала комерційне втілення вона лише в продуктах корпорації Apple Computer.

Зручність використання користувацького інтерфейсу (*англ. Usability*) - показник його якості, що визначає кількість зусиль, необхідних для вивчення зasad роботи з програмною системою з допомогою цього інтерфейсу, її використання, підготовки вхідних даних і інтерпретації вихідних. Тобто, зручність використання виявляє міру простоти доступу користувача до функцій системи, наданих через людино-машинний (користувацький) інтерфейс.

На зручність використання користувацького інтерфейсу впливають такі чинники:

- легкість навчання (чи швидко людина навчається використовувати систему);
- ефективність навчання (чи швидко людина працює після навчання);
- запам'ятовуваність навчання (чи легко запам'ятується все, чого людина навчилася);
- наявність помилки (чи часто людина припускається помилок у роботі);
- загальна задоволеність (чи є загальне враження від співпраці з системою позитивним).

Усі ці фактори, незважаючи на неформальність, можна вимірювати. Для таких вимірювань набирають групу типових користувачів системи, і під час її

роботи вимірюють показники роботи і системи (наприклад, кількість допущених помилок, витрачений час на виконання завдання), а також користувачам пропонується висловити враження від системи за допомогою опитувальних карток.

Тестування зручності використання користувацького інтерфейсу, власне кажучи, не належить до класичних методів тестування програмних систем. Фахівець із тестування користувацького інтерфейсу має поєднувати знання і з інженерії програмного забезпечення, і з фізіології, психології та ергономіки.

Найбільш цитованими у книгах з людино-машинного інтерфейсу є так звані евристичні правила, які спільно розробили відомий американський фахівець у галузі проектування інтерфейсів Якоб Нільсен (Jakob Nielsen) та дослідник Рольф Моліч (Rolf Molich). Евристичні правила не мають строгого доведення, - їх виведено з практичного досвіду за мінімальними критеріями, яким має відповідати будь-який інтерфейс.

Фактично це “десять заповідей” будь-якого розробника інтерфейсів комп’ютерних програм:

1. Сповіщення про поточний стан (**англ.** Visibility of system status)
2. Близькість до реального світу (**англ.** Match between system and real world)
3. Управління свободою дій користувача (**англ.** User control and freedom)
4. Цілісність та стандарти (**англ.** Consistency and standards)
5. Допомога користувачам у розпізнаванні, діагностиці та усуненні помилок (**англ.** Help users recognize, diagnose and recover from errors)
6. Запобігання помилкам (**англ.** Error prevention)
7. Розпізнавання, а не згадування (**англ.** Recognition rather than recall)
8. Гнучкість та ефективність використання (**англ.** Flexibility and efficiency of use)
9. Естетичний і мінімально необхідний дизайн (**англ.** Aesthetic and minimalist design)
10. Допомога та документація (**англ.** Help and documentation)

Наведені принципи є базовими і розширяються в документах, присвячених розробленню інтерфейсу користувача для певних класів систем. Написано цілі томи про те, як треба створювати інтерфейс для застосунків на платформі Android, для застосунків на платформі iOS і т.д.

## 4.3. Зміст принципів побудови графічного інтерфейсу

### 1. Сповіщення про поточний стан.

Система завжди повинна повідомляти користувача про те, що вона у цей час робить, причому її час відгуку має бути в розумних межах. Інакше кажучи, реакцією на будь-яку дію користувача, крім виконання основного функціоналу, має бути **помітна зміна вигляду екрана**. Якщо операція тривала і результати її не можуть бути показані миттєво, слід змінювати вигляд екрана поетапно - спочатку повідомляємо, що дія виконується і потребує часу, а тоді вже сам результат, але в жодному випадку система не повинна застигнути і “мовчати”.

Згадаймо класичні способи демонстрування користувачеві стану речей. Коли через оглядач завантажується доволі об'ємний файл, цей процес відображається індикатором виконання (прогрес-баром). Копіювання файлів, запис диску, процес інсталяції програмного забезпечення ілюструють індикатором виконання з приближною оцінкою часу, який лишився до завершення операції. Можливо, що користувачеві не повідомляють, скільки зроблено, скільки залишилося зробити, а просто показують, що дія у процесі, наприклад, пісочним годинником чи анімованим колом. Якщо графічні засоби застосовувати недоцільно через обмеження ресурсів, як це буває у вбудованих системах, стан системи показують простим текстовим повідомленням. Описані типові способи відображення стану системи показано на рис. 4.3.

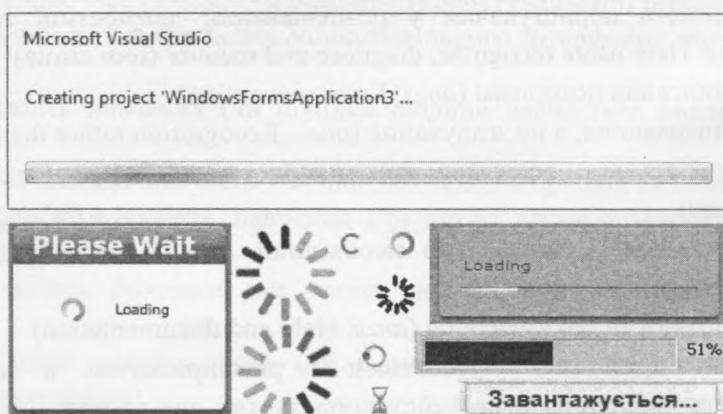


Рис. 4.3. Приклади способів відображення стану системи

Типовим прикладом реалізації принципу “Сповіщення про поточний стан” є реакція елементів графічного інтерфейсу на дії миші - наведення та відведення курсора, натиск елемента тощо. Якщо користувач натискає кнопку, вона має змінити вигляд аби запевнити користувача, що натиснення відбулося. Якщо текстове поле отримує фокус, у ньому має близити курсор, до того ж, може змінитися рамка. Під час наведення миші на елемент за “правильного” інтерфейсу він теж змінює вигляд (варіанти показано на рис. 4.4). У графічних редакторах, де точна позиція курсора миші є важливою, поточні координати відображають у статус-рядку.



Рис. 4.4. Приклади відгуку інтерфейсу на події миші

Прийнято повідомляти користувачеві, де він перебуває зараз. Зокрема, якщо інтерфейс має декілька закладок, потрібно виділити ту, що є активною в цей момент. Якщо виконується якийсь багатоетапний процес, варто показати, на якому кроці користувач є зараз. Зокрема, якщо користувач читає онлайн параграф якоїсь книги, йому буде зручно бачити весь зміст і місце цього параграфа у змісті. У графічному редакторі з інтерфейсу видно, який колір, товщину лінії чи інші ефекти вибрано зараз, у текстовому - які властивості шрифту вибрані зараз і т. д. (рис. 4.5).

У повідомленнях потрібна розумна міра. Якщо користувач видаляє елемент списку, правильною реакцією буде показати йому оновлений список, в якому більше немає видаленого ним елемента. У цьому випадку додаткові текстові повідомлення на кшталт “Елемент успішно видалений” є зайві.

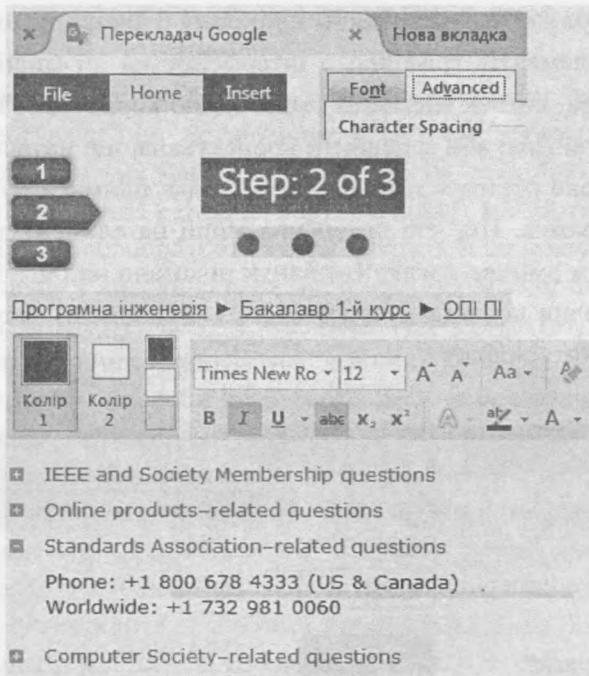


Рис. 4.5. Відображення поточного активного фрагмента інтерфейсу

**2. Близькість до реального світу.** Система повинна “спілкуватися” з користувачем його мовою, використовуючи слова, поняття та фрази, зрозумілі й близькі користувачеві. Термінологія, використана в інтерфейсі системи, повинна співвідноситися з користувацьким світом, тобто це має бути термінологія проблемної області користувача, а не технічна термінологія. Наприклад, якщо програмний продукт призначений для медиків, термінологія має бути зрозуміла медикам. Крім того, інформацію потрібно подавати у природному, логічному порядку. Нехай програма працює зі списком студентів, дозволяючи створення, редагування списку та видалення студента зі списку, а інтерфейс користувача містить панель з піктограмами, що позначають ці дії. Нелогічно було б розмістити кнопки у такому порядку: “видалення”, “редагування”, “створення”, бо неможливо видалити або редагувати те, чого ще не існує. Згадаймо знайомі інтерфейси: файл спочатку створюємо або відкриваємо, а тоді вже зберігаємо, тому піктограми розміщують у відповідному порядку.

Користувач почуватиметься комфортніше, якщо тексти наближені до звичайної розмовної мови. Якщо користувач повинен авторизуватися в системі і забув пароль, “правильне” рішення - дати посилання “Забули пароль?”, а якщо користувач ще не зареєстрований, тоді варто запросити його зареєструватися просто зараз - так, як би це зробила людина-співрозмовник. Замість відправляти користувача читати розділ FAQs (*англ.* Frequently Asked Questions), варто написати “Чим ми можемо допомогти?” і надати багаторядкове текстове поле для введення запитання.

Приклади людино-орієнтованих повідомлень показано на рис. 4.6.

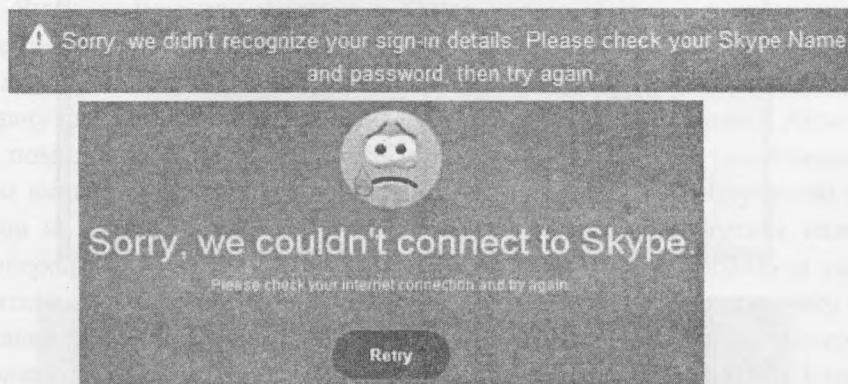
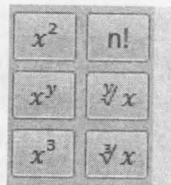


Рис. 4.6. Приклади повідомлень природною мовою

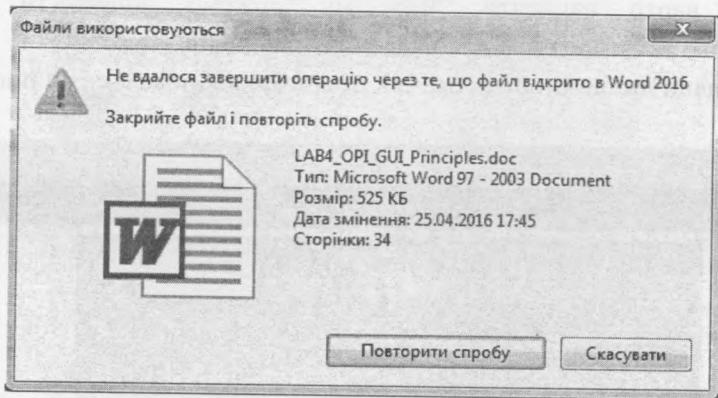
Окрім зображень, що імітують емоції людини (рис. 4.6), природності повідомленням системи додають “персональні” звертання “ви, ваш, ваші...” (рис. 4.7, в).

Природно, якщо написи на кнопках калькулятора (рис. 4.7, *a*) відповідають загальноприйнятим позначенням — знак кореня замість  $\sqrt{}$ , математичний запис піднесення до степеня замість  $pow$  і т. д.

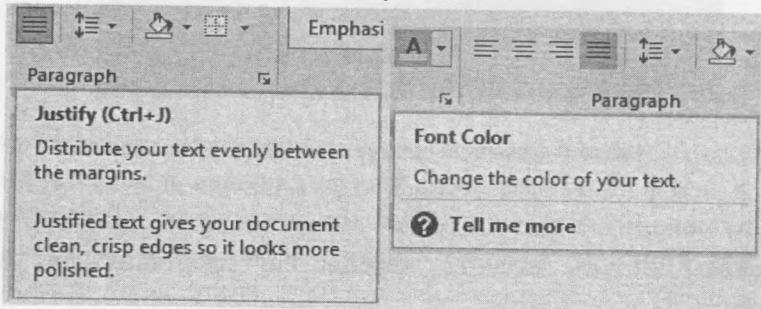
Прикладом застосування принципу “Близькість до реального світу” є і колірне оформлення. Для повідомлень про помилки застосовують червоний колір, а правильність введених даних позначають зеленим (аналогія зі світлофором). Для попереджень теж застосовують знаки, що навіюють асоціації з дорожнім рухом (рис. 4.7, *б*).



*a*



*b*



*c*

Рис. 4.7. Використання слів, символів та зображенень з реального світу

**3. Управління свободою дій користувача.** Користувачі часто вибирають окремі елементи інтерфейсу й використовують функції системи з помилкою. І тут необхідно надавати чітко визначений “аварійний вихід”, з якого можна повернутися до попереднього нормальногго стану. Цей самий принцип використовують не лише для аварійних ситуацій, але й для можливості скасувати зміни. Що більше елементів здатний вмістити журнал дій, то вільніше почував-

тиметься користувач. Яскравим прикладом є текстовий редактор Word. Користувач може перебувати у творчих пошуках, намагаючись добирати влучні слова, друкувати речення, потім пробувати перефразувати текст і повернутися до попереднього варіанта. Усі ці маніпуляції зі словами можливі завдяки командам Undo і Redo. Краще дозволити користувачеві робити, що він хоче, і скасувати зміни, ніж набридати запитаннями типу “Ви впевнені, що хочете виконати цю дію?”. Інакше кажучи, *потрібно за можливості уникати незворотних дій*, залишаючи користувачеві шлях до відступу.

Ось ще приклади реалізації принципу “Управління свободою дій користувача”.

Якщо користувач написав у Skype повідомлення з помилкою, це повідомлення можна редагувати (по свіжих слідах). Якщо повідомлення/файл було надіслано не тому абоненту, що треба, повідомлення можна видалити, а передачу файла - скасувати (якщо файл ще не був прийнятий). Якщо на друк було помилково відправлено зайві сторінки, друк можна призупинити або й зовсім видалити файл з черги на друк. Якщо користувач, блукаючи по сайту, зайдов на сторінку, яка йому нецікава, він може повернутися назад або ж безпосередньо перейти на головну сторінку. Тобто, розробляючи сайти, слід пам'ятати, що з кожної сторінки повинно бути посилання на головну сторінку. Неправильно, якщо за посиланням сторінка раптово, без попередження, відкривається в новому вікні - всупереч “почуттю навігації” користувача. Якщо випадково почало встановлюватися якесь застосування, цей процес можна перервати.

Для дій, які важко скасувати, потрібно перепитати користувача, щоб запобігти незворотним наслідкам (це якраз той випадок, коли запитання не буде здійснено). З цієї причини, коли ми випадково закриваємо незбережений документ, програма перепитує, потрібно його зберігати чи ні. Коли видаляємо файл, система теж просить підтвердити бажання видалити.

У будь-якому випадку користувачеві потрібно дати впевненість у тому, що він не зайде у глухий кут.

**4. Цілісність і стандарти.** Зміст цього принципу полягає в тому, щоб не плутати користувача. Інтерфейс користувача не повинен конфліктувати зі вже сформованим досвідом користувачів. Інакше кажучи, якщо користувач звик, що команди Undo (Скасувати) і Redo (Повторити) слід шукати в меню Edit/Редагування, то не варто ці команди розміщати в іншому меню. Не варто

до того ж, називати пункт меню, що містить команди Скасувати, Повторити, Знайти тощо, словом, відмінним від Редагування.

Для позначення одних і тих самих понять слід застосовувати одні й ті самі терміни, для позначення одних і тих самих дій - ті самі піктограми у всіх частинах графічного інтерфейсу, причому кожному графічному елементові відведено своє стало місце на екрані. Наприклад, верхнім елементом головного вікна будь-якої програми є титульний рядок з назвою програми та кнопками мінімізації, розгортання та закриття вікна. Нижче розміщено меню, причому у різних програмних продуктах різних виробників першим у меню є Файл, останнім - Довідка. Команди меню Файл “Створити”, “Відкрити”, “Зберегти”, “Зберегти як...” розташовано у певній послідовності, останнім пунктом цього меню є вихід з програми. Незалежно від характеру та призначення програмного продукту світло-сірий колір шрифту означає, що елемент неактивний, “...” в кінці назви команди - те, що буде викликаний діалог, стрілка - що меню галузиться і буде показане підменю. У модальних вікнах кнопку “OK” розміщено лівіше, ніж кнопку “CANCEL”, і саме такого взаємного розташування кнопок дотримуються у більшості графічних інтерфейсів. Винятком є випадки, коли потрібно забезпечити нетиповий вибір користувача за замовчуванням (часто користувач машинально натискає кнопку, не замислюючись над змістом).

Якщо на одній зі сторінок інтерфейсу для позначення дії “Повернутися назад” використано синю стрілку, яка вказує ліворуч, і цю стрілку розміщено зліва внизу, то неправильно на інших сторінках для позначення цієї самої дії застосувати кнопку з текстом “Назад” або ж стрілку іншого кольору, розміру, форми чи з іншим розташуванням. Якщо на головній сторінці сайту є посилання “Контактні дані”, то далі по сайту не слід застосовувати назву “Наши контакти” чи якусь іще. Якщо в меню використовується назва “Папка”, то не варто в допоміжних діалогових вікнах підмінити назву “папка” назвами “каталог” або “директорія”. Якщо в одних частинах інтерфейсу ліворуч показано колонку з підписами елементів, а праворуч - з елементами введення даних, то не варто у інших частинах інтерфейсу показувати підписи справа від елементів введення даних. Якщо текст на кнопках починається з великої літери, то кнопки, на яких написи починаються з малої, виглядатимуть “чужими” у цьому інтерфейсі.

**5. Допомога користувачам у розпізнаванні, діагностиці та усуненні помилок.** Повідомлення про помилки мають бути написані природною мовою, а не замінюватися кодами помилок або ж текстами, сповненими слів-агнонімів. Наприклад, користувачеві, далекому від програмування, навряд чи буде зрозумілий текст на кшталт того, що показаний на рис. 4.8 (мабуть, кожен програміст-початківець бачив цей тип помилки за перших невпевнених спроб роботи зі вказівниками). Такі модальні вікна зі шістнадцятковими кодами допустимі для середовищ програмування, але їх не повинно бути в інтерфейсах, призначених для людей, далеких від програмування.

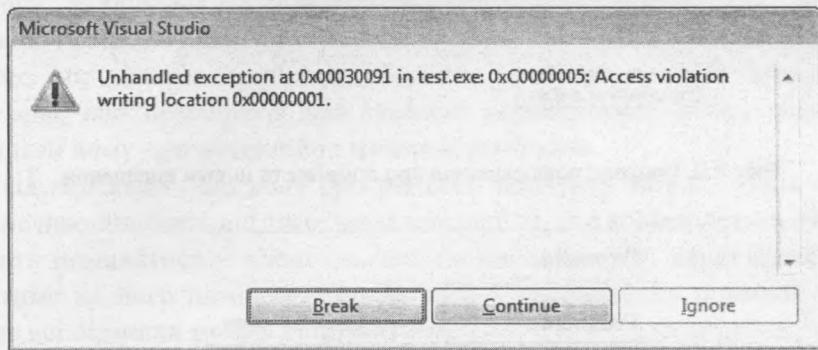


Рис. 4.8. Приклад повідомлення, розрахованого лише на програмістів

Мало констатувати факт настання помилки - користувачеві нецікава проблема, йому потрібне конструктивне рішення. Нехай, наприклад, у банкоматі немає суми, запитуваної користувачем. Типова система повідомляє користувача про це і пропонує варіанти дій: 1) продовження роботи і виконання якоїсь іншої дії (наприклад, перевірка стану рахунка) або ж 2) вихід. Інший приклад - спробуймо набрати неіснуючу адресу в адресному рядку браузера Mozilla Firefox. Як бачимо з рис. 4.9, користувачеві не просто кажуть, що сторінки не знайдено, але й пропонують способи вирішення проблеми.

Якщо абонент перебуває поза зоною, ми не лише дізнаємося про це, але й отримуємо шлях вирішення - зателефонувати пізніше. За можливості потрібно завершувати повідомлення про помилку на "позитивній ноті" — пропозицією вирішення проблемної ситуації.

Повідомлення про критичні помилки виводяться в окремих модальних вікнах. Повідомлення, пов'язані з валідацією форм, часто виводять поряд з "проблемним" графічним елементом.



# Сервер не знайдено

Firefox не може знайти сервер [www.franko.lvi.ua](http://www.franko.lvi.ua).

- Перевірте, чи не допущена помилка при введенні адреси, наприклад [www.example.com](http://www.example.com) замість [www.example.com](http://www.example.com)
- Якщо жодна сторінка не завантажується — перевірте налаштування з'єднання з інтернетом.
- Якщо комп'ютер або мережа захищені фірмовим або проксі-сервером — переконайтесь, що Firefox дозволено виходити в інтернет.

[Спробувати знову](#)

Рис. 4.9. Приклад повідомлення про помилку та шляхи вирішення

The screenshot shows a password input field with the placeholder "Password". Below it is a red error message: "Your password isn't strong enough, try making it longer." To the right of the input field are two "show" buttons, one above the placeholder and one next to the error message, both of which are currently inactive (grayed out).

a

Email

Значення атрибуту Email не може бути порожнім.

Пароль

Довжина Password має бути не менше ніж 6.

b

Рис. 4.10. Приклади повідомлень про помилки при валідації форми

На рис. 4.10 показано один зі сценаріїв обробки дій користувача. “У стані спокою” поле лише містить підказку, які дані туди потрібно вписати.

Отримуючи фокус, поле змінює рамку (згідно з принципом про сповіщення про поточний стан). У процесі набору символів користувач бачить підказку, що пароль надто слабкий. Це попередження щезає після введення достатньої кількості літер. Існує, однак, думка, що червоний колір потрібно резервувати для дуже критичних помилок. Як бачимо з рис. 4.10, текст попередження не лише констатує факт, що користувач зробив щось недобре, але й дає “рецепт”, що робити у цьому випадку. Приклад взято з реєстраційної форми пошти yahoo.

**6. Запобігання помилкам.** Продуманий дизайн користувацького інтерфейсу, який **запобігає появі помилок** користувача, кращий, ніж добре продумані повідомлення про помилки. Під час проектування інтерфейсу необхідно або цілком прибрati елементи, у яких можуть бути помилки користувача, або перевіряти дані, введені користувачем у тих елементах, і повідомляти йому про потенційно можливі проблеми.

Слід пам'ятати, що доля програмного продукту та його успіх на ринку безпосередньо залежать від того, чи задоволені ті, хто користуються ним. Ніхто не любить помиллятися - користувачеві не сподобається, якщо система часто вказуватиме на його помилки. Більше того - виправлення помилок потребує часу, і не всі помилки можна виправити.

Розглянемо прості приклади запобігання помилкам. Нехай очікується введення числового значення, що є натуральним або цілим невід'ємним числом у визначених межах (скажімо, оцінки студента у деякій шкалі). Замість зробити просте текстове поле і дозволяти користувачеві ввести що завгодно, а потім перевіряти дані, краще застосувати елемент, який не сприйматиме недозволених даних. Зокрема, якщо шкала 5-бальна, то для виставлення оцінки можна використати лічильник /спінбокс (рис. 4.11) або ж випадаючий список з можливістю вибору лише одного елемента. Для 100-бальної шкали це є гіршим рішенням, оскільки суперечитиме важливому правилу: виконання будь-якої дії має потребувати мінімуму зусиль від користувача, а в ідеальному випадку - “виконуватися у 3 кліки”. У цьому випадку краще використати текстове поле, встановивши для нього обмеження за кількістю символів і заборонивши введення не-цифр.

Якщо введені користувачем дані типу рядок символів зберігатимуться у базі даних, то текстове поле не повинно сприймати більше символів, ніж визначено у відповідному полі таблиці бази даних. Інакше при неграмотно написаному коді може трапитися помилка, пов'язана з роботою з пам'яттю.

Або ж, у “кращому” випадку, останні, “зайві” символи у тексті будуть втрачені, можливо, користувач згодом побачить ці “урізані” дані.

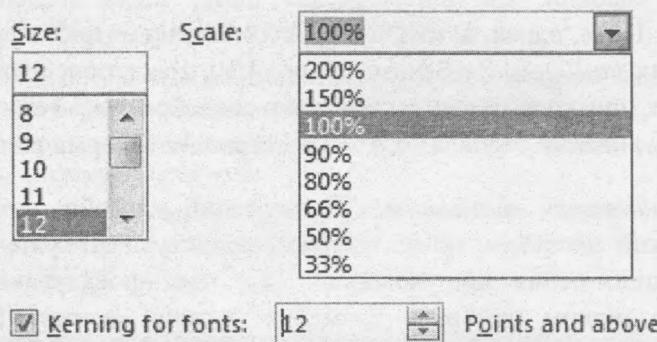


Рис. 4.11. Використання лічильника й випадаючого списку з можливістю вибору лише одного елемента

Для введення дат у сучасних інтерфейсах застосовують календар (рис. 4.12). Для швидкого вибору дня і місяця довільного року зручним є календар, показаний на рис. 4.12, б. Вибирати календар потрібно обережно у випадках, коли потрібно забезпечити підтримку мов, які використовують розширені латинські та кириличні символи або ієрогліфи. Якщо календар недоступний, можна використати групу з трьох графічних елементів (наприклад, три випадаючі списки). Враховуючи, що місяць і день є у визначених межах, можна використати для них лічильник (спінбокс), а для року - випадаючий список.

Якщо ж для введення дати застосовуються текстові поля, повинен бути у тому чи іншому вигляді підписаний формат.

Якщо потрібно запитати в користувача номер його мобільного телефону, варто запропонувати шаблон введення номера. Часто код оператора оформляють випадаючим списком, а користувачеві залишається ввести тільки сам номер. Взагалі, вважається хорошим стилем, якщо якомога більше даних вводяться за допомогою вибору із запропонованих варіантів (на противагу ручному введенню у текстові поля). Зокрема, у більшості сучасних реєстраційних форм країни та міста вибирають з випадаючих списків (рис. 4.13).



4



6



6

Рис. 4.12. Способи введення даних

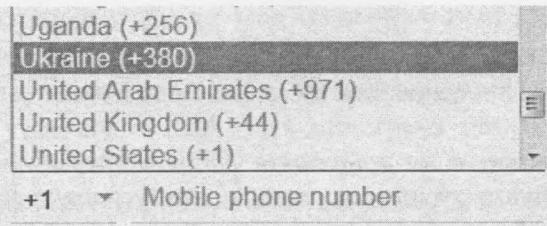


Рис. 4.13. Список кодів і країн

Можуть бути і змішані способи введення: типові значення даних одразу представлені на екрані, а нетипові вводяться вручну. Наприклад, добре спроектований банкомат запропонує користувачеві на вибір з десятків типових варіантів, серед яких пункт “Інша сума”. І знову приклад запобігання помилкам в дії: на екран виводиться повідомлення про те, що запитана сума повинна бути кратна N (переважно — 5). Інакше операцію неможливо буде виконати, і користувач змушений буде повторювати ті самі кроки ще раз, але з “правильною” сумою. Якщо система тимчасово не здатна виконати дію, якої від неї природно очікувати, потрібно попередити користувача завчасно. Наприклад, якщо платіжний термінал не видає решти, потрібно сповістити про це користувача і запропонувати йому, скажімо, переказати решту на мобільний рахунок. Якщо ж банкомат/платіжний термінал тимчасово не видає чек, потрібно попередити користувача до того, як той почне виконання якоїсь дії.

Якщо немає іншого способу ввести дані, окрім як за допомогою текстових полів (наприклад, коли запитуємо в користувача його ім'я чи електронну адресу), варто написати приклад, як у формі, показаній на рис. 4.14.

The screenshot shows a registration form with the following fields:

- Display Name:** J. Doe
- Email (required, but never shown):** you@example.org
- Password:** A masked password consisting of several asterisks.

At the top, there are social media sharing buttons for Google+ and Facebook, followed by an "OR" separator, and a "Create Account" button at the bottom right.

Рис. 4.14. Реєстраційна форма stackoverflow з прикладами правильного заповнення

**7. Розпізнавання, а не згадування.** Обидві операції - і розпізнавання, і згадування - потребують зусиль користувача, але зусилля ці різні. Розпізнавання - визнання чогось знайомим, згадування - видобування даних з пам'яті. Коли ми бачимо людину на вулиці, спочатку усвідомлюємо, знайомі ми з нею чи ні

(розвідання). “Витягти” з пам’яті ім’я та прізвище - складніша задача (згадування). Коли нам потрібно дати відповідь типу “так/ні” або ж вибрати правильну відповідь з множини запропонованих варіантів (розвідання), це простіше зробити, ніж вписати відповідь вручну, без підказок (згадування). Коли ми читаємо текст іноземною мовою, віднайти знайомі слова простіше, ніж витягти вчасно з пам’яті ті самі слова у “живій” бесіді. Коли ми пробуємо знайти сайт, який переглядали нещодавно і який хотіли б відвідати ще, то звертаємося переважно до пошукових систем. Простіше віднайти те, що треба, з-поміж результатів пошуку, ніж пригадати точну адресу сайту. Чимало користувачів Інтернету повертаються до вже відвіданих сайтів через механізм закладок або історію браузера частіше, ніж шляхом набору URL сайту в адресному рядку браузера. Часто, редакуючи файли, створені вчора (кілька днів тому, минулого тижня), ми не пригадуємо точної назви файла, але, переглянувши список файлів, за датою редагування віднайшовмо те, що нам потрібно. Принцип “розвідання замість згадування” означає, що необхідно мінімізувати навантаження на пам’ять користувача, за можливості заміняючи необхідність згадувати необхідністю віднайходити. Працюючи з командним рядком, доводиться пам’ятати, як правильно записати ту чи іншу команду. З графічним інтерфейсом все набагато простіше: команди лежать перед користувачем, підписані та візуалізовані, залишається лише віднайти з-поміж них потрібну і вибрати її. Згідно з принципом “Розвідання, а не згадування”, під час створення інтерфейсу *об’єкти, дії і опції роблять зрозумілими, доступними та очевидними*.

Щоб користувачам було легко здогадатися, яку дію приховано за тією чи іншою піктограмою, розробники інтерфейсу намагаються використовувати зображення об’єктів реального світу, що викликають “правильні асоціації”. Недарма команда “Вирізати” зображають ножицями, “Знайти” - біноклем або лупою, “Викинути у смітник” - смітником, “Повернутися на домашню сторінку” - хатинкою, “Зберегти у хмарі” - стрілкою, направленою на хмаринку, а загалом хмарні сервіси зображуються хмаркою, “Зберегти” - дискетою, незважаючи на те, що цей носій інформації відійшов у історію (рис. 4.15). Зображення конверта викликає асоціацію з поштою, надсиланням листа, скріпки - з приєднанням, прикріпленим (так позначають приєднані файли), канцелярської кнопки - з фіксацією чогось у певній позиції (така піктограма позначає фіксацію частин вікна), дзвіночка - зі сповіщенням, цегляної стіни - з брандмауером або ж іншими програмами для безпеки комп’ютера, візка (господарської сумки) - з покупками. Зображення гайкового ключа або шестерні “міцно закріпилося” за налаштуваннями параметрів, встановленням преференцій користувача. Прямими стрілками

“вліво” та “вправо” прийнято позначати напрям “вліво” і “вправо”, а конкретне значення “уточнюється контекстом”. Наприклад, в оглядачі ці стрілки означають переміщення між вже відвіданими сторінками, у плеєрі - між аудіофайлами тощо. Стрілка вліво означає перехід до попереднього елемента, а якщо елемент не має попередника (тобто є першим), то стрілка “впирається у стіну” (рис. 4.15, д). Дугоподібні стрілки означають операції **Скасувати** та **Повторити**, однак, у оглядачі піктограма Повторити означає перезавантаження/новлення сторінки. Ключ і замок асоціюються з захищеністю, безпекою і вживаються для позначення цих понять окремо або ж спільно з об'єктами, які підлягають захисту (наприклад, піктограма з рис. 4.15, р зображує захист документа). Лупа зі знаками “+” або “-” - поширені піктограми для збільшення/зменшення масштабу. Розімкнутим колом традиційно позначають вимкнення живлення.



Рис. 4.15. Поширені піктограми для часто виконуваних дій:

а - “Вирізати”; б - “Знайти”; в - “Викинути у смітник”; г - “Навігація”;

д - “Зафіксувати елемент інтерфейсу”; е - “Повернутися на домашню сторінку”;

ж - “Зберегти у хмарі”; з - “Скасувати”/“Повторити”; і - “Прикріпити документ”;

й - “Параметри”; к - “Брандмауер”; л - “Відкрити/Закрити”; м - “Змінити масштаб”;

н - “Сповіщення”; о - “Аутентифікація”; п - “Покупки”; р - “Лист”; с - “Допомога”; т - “Вимкнути”

Якщо об'єкт має матеріальний аналог, його зображення і застосовують у інтерфейсі. Годинник, калькулятор, клавіатуру, гучномовець, маніпулятор мишу, дисплей, календар, комп'ютер, мікрофон, CD/DVD, дискету, друк/принтер, телефон, файл, папку, пензель, олівець, ручку, людину/користувача представляють так, як користувач звик бачити їх "вживу" у повсякденному житті (рис. 4.16).

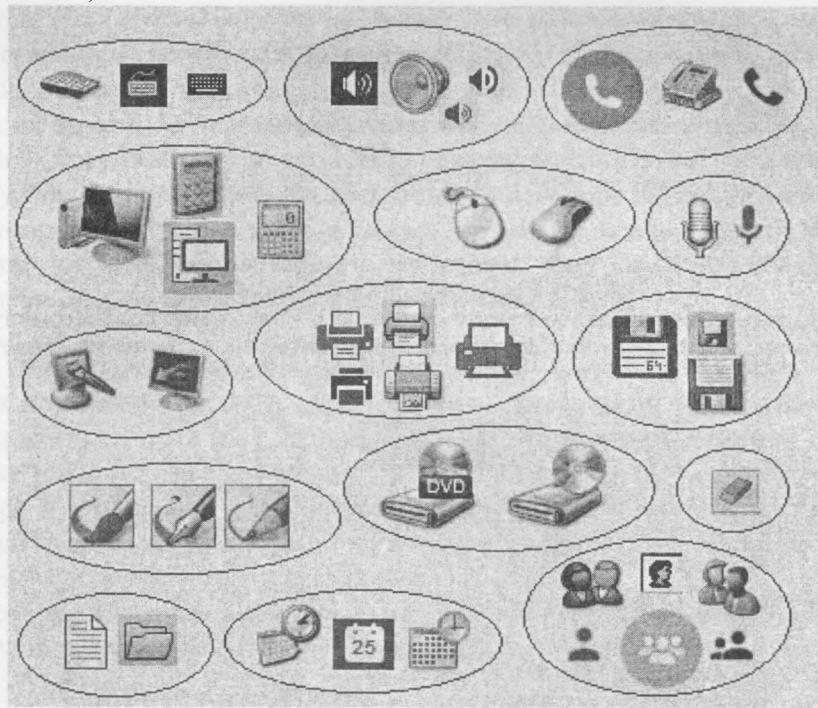


Рис. 4.16. Зображення об'єктів, що мають прямі матеріальні аналоги

Якщо дію можна стисло, схематично, але наочно зобразити у піктограмі, то так і роблять (рис. 4.17, а). Наприклад, знаком "+" позначають дію "додати", а сполученням якогось об'єкта зі знаком "+" - додавання цього об'єкта (приклади показані на рис. 4.17, б). Поєднання олівця/ручки і предмета означає редагування даних цього предмета. Часто дію просто показують її результатом (рис. 4.17, в).

Деякі піктограми, якщо і не є настільки красномовними, то принаймні прижилися і є широко застосовуваними. Наприклад, трикутник означає запуск

на виконання або початок відтворення, хрестик - закриття вікна, літера "і" - дані, довідку (перша літера слів "info", "information"), "\*" справа від графічного елемента введення даних - те, що ці дані обов'язкові (рис. 4.18).

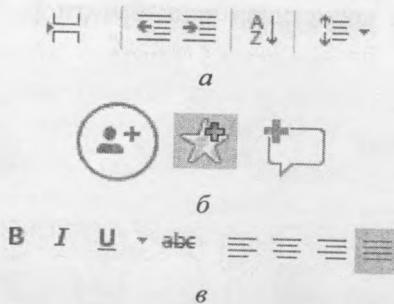


Рис. 4.17. Піктограми, що позначають дію:

"*a*" - приклади піктограм, де дію зображують стрілкою; *b* - піктограми "додати користувача"- "додати в закладки"; "додати коментар"; *c* - приклади піктограм, що показують результат дії

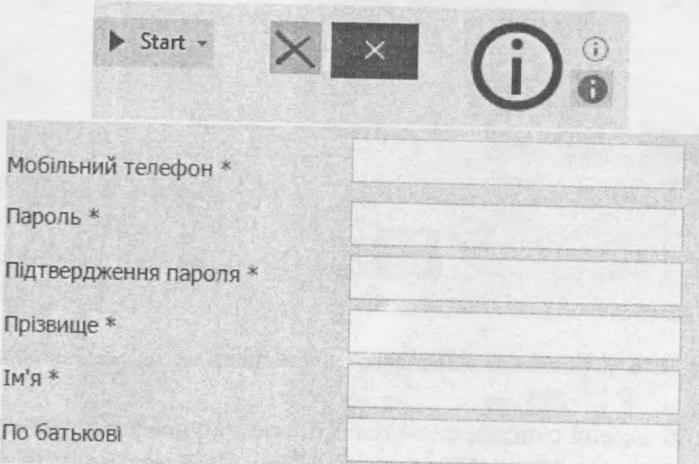


Рис. 4.18. Загальноприйняті символи

Навіть якщо графічний елемент зрозумілий, варто дати підказку, щоб користувач був цілком впевнений щодо призначення цього елемента.

Попри бажання зробити інтерфейс оригінальним, пріоритетом має бути зручність у використанні і простота навчання. Відтак, якщо якась дія має

усталене, добре відоме позначення, варто використати саме його в розроблюваному графічному інтерфейсі, замість вигадувати щось нове, незвичне.

**Користувач не зобов'язаний нічого пам'ятати під час переходу між вікнами/діалогами.** Усі дані, потрібні йому для роботи, мають бути перед очима. В усіх необхідних місцях мають бути доступні контекстні інструкції з використання інтерфейсу.

Якщо користувач один раз вже відповів на якесь запитання системи, буде грубою помилкою запитувати його про це ще раз. Ще один важливий аспект (особливо це актуально у платіжних терміналах) - користувач почуватиметься комфортно, якщо бачитиме при переході між екранами “історію спілкування зі системою”. Наприклад, якщо користувач бажає поповнити рахунок мобільного телефону і на першому кроці вводить номер, після чого підтверджує введення, на наступному екрані має відобразитися цей номер. Інакше користувач може нервувати, чи його кошти будуть зараховані саме на потрібний рахунок. Результатом операції поповнення має бути чек і/або SMS.

**8. Гнучкість та ефективність використання.** Система повинна надавати два способи роботи: для новачків і досвідчених користувачів. Саме дотримання цього принципу дозволяє одному і тому ж програмному продукту мати таке розмаїття користувачів. Зокрема, немає окремих версій текстового процесора Word для користувачів різного віку, різного рівня освіти і т.д. Один і той самий Word можуть використовувати школярі або ж професійні секретарі чи верстальники.

Новачки йдуть до мети повільніше, довшим шляхом, часто з істотною підтримкою з боку системи. Досвідчені користувачі знають прийоми, які “не лежать на поверхні”, і тому досягають тих самих результатів швидше. Найпростіший приклад - “гарячі клавіші”, що дають змогу прискорити виконання типових, часто застосованих операцій (відкриття файла, збереження файла тощо). Хоч комбінації клавіш і записано у підказках та у пунктах меню (рис. 4.19), вони, тим не менш, не заважають новачкам.

Інша реалізація цього ж принципу - приховання рідкісних функцій і повсякчасна присутність “на видному місці” інтерфейсу тих графічних елементів, що пов’язані з найтиповишим, найчастіше застосовуваними командами. Наприклад, у Word “на поверхні” лежать кнопки для вибору гарнітури та кеглю шрифту, вирівнювання, створення/відкриття/збереження файла, встановлення декору шрифту (жирний, курсив, підкреслений), задавання абзацного відступу, задавання кольору шрифту і фону тощо (рис. 4.20).

Практично кожен користувач щоразу виконує мінімальне форматування тексту за допомогою зазначених інструментів. Однак, нечасто застосовувані функції, наприклад, представлення всіх літер як великих чи середнього розміру (*англ. Small Caps*), задавання нетипового декору (наприклад, перекреслення тексту подвійною лінією) тощо, не винесено на панель інструментів.

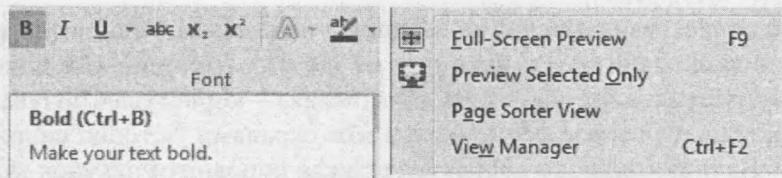


Рис. 4.19. “Гарячі клавіші” в інтерфейсі користувача

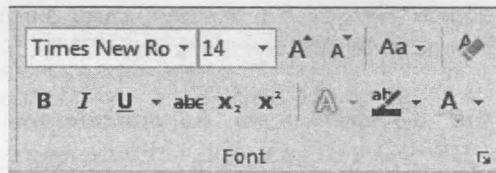


Рис. 4.20. Найпоширеніші команди “на виду”

У діалозі “Шрифт” (*англ. “Font”*) є дві вкладки. В одній згруповано елементи для задавання часто вживаних властивостей (рис. 4.21), в іншій (рис. 4.22) - інструменти, які можуть знадобитися небагатьом користувачам. Однією з таких “рідкісних” функцій є кернінг (процес зміни розмірів міжбуквених пропусків (інтервалів) між сусідніми буквами для поліпшення зовнішнього вигляду і легкості читання тексту). Отже, нечасто вживані інструменти “не плутаються” під руками, більше того, чимало користувачів текстового процесора ніколи не чули про кернінг.

Ще один приклад - можливість одержати результат у Excel самостійно (вводячи формулу вручну) або ж з використанням Майстра, який приводить користувача до бажаного результату крок за кроком і буквально не дає зробити помилку.

Може бути і “розшарування” графічних інтерфейсів користувача. Приклад - калькулятор у Windows, який має “варіації на тему” - “Звичайний”, “Інженерний”, “Програміст”, “Статистика”. Отже, кожен користувач отримує, що близче до його потреб і очікувань; інтерфейс при цьому не засмічений зайвими елементами.

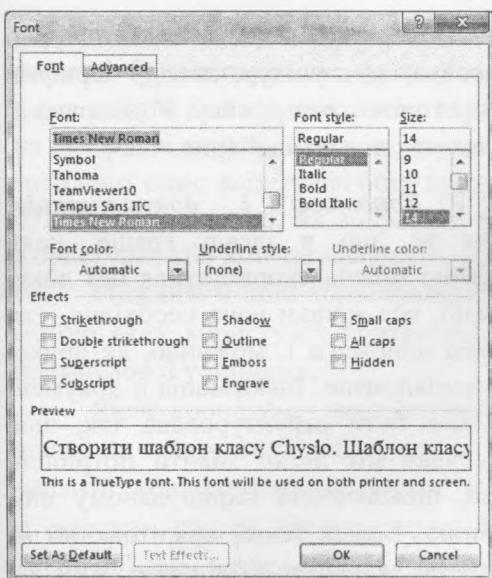


Рис. 4.21. Вкладка для налаштування часто вживаних властивостей шрифту

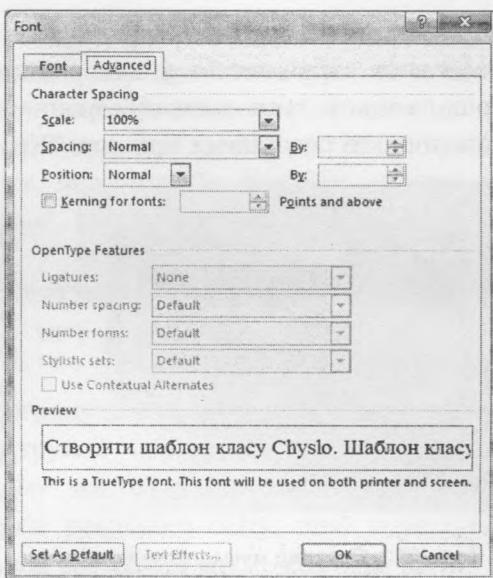


Рис. 4.22. Вкладка для задавання зрідка вживаних властивостей шрифту

Загалом система “має вміти” більше, ніж видно на перший погляд, але її вигляд варто робити зумисне “спрощеним”, щоб не відлякувати початківців. Досвідчені ж користувачі знають, де знайти те, що їм потрібно.

**9. Естетичний і мінімально необхідний дизайн.** Вікна не повинні відображати того, що не стосується справи чи зрідка використовується. Кожен інтерфейсний елемент, у якому є зайва інформація, виконує функцію інформаційного шуму та відволікає користувача від справді корисних інтерфейсних елементів. Непотрібні графічні елементи “конкурують” з потрібними графічними елементами та зменшують “відносну видимість” потрібних графічних елементів. Розробники графічного інтерфейсу повинні перевіряти кожен елемент, запитуючи себе, чи можна без нього обйтися.

Загострюючи це питання, можна сказати - якщо систему використовують для поповнення мобільного рахунку, непотрібно запитувати користувача, яка в нього група крові. На деяких сайтах під час заповнення адреси користувачеві пропонується задати країну, область, місто, район міста, вулицю, будинок, квартиру та індекс, хоча очевидно, що область та район міста є явно залежими елементами, оскільки ніколи не фігурують у поштовій адресі.

Естетичності, незважаючи на те, що це доволі суб'єктивне поняття, часто досягають стриманістю у використанні кольорів, гарнітур, декору шрифтів тощо, а також стилювим об'єднанням усіх сторінок інтерфейсу. У досягненні естетичності помічним є принцип “Краще менше, але краще” (рис. 4.23).

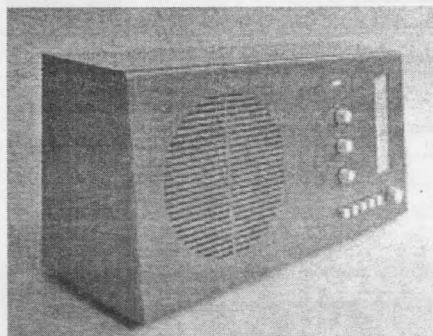


Рис. 4.23. Приклад мінімалістичного та естетичного дизайну

## 10. Допомога документація.

Попри те, що в ідеалі краще, коли системою можна скористатися без документації, все ж таки вона необхідна - як система допомоги і, можливо, як друковане керівництво. Інформація в документації має бути структурована так, щоб користувач міг легко знайти потрібний розділ, присвячений вирішуваному ним завданню. Кожен такий орієнтований на конкретне завдання розділ має, крім загальної інформації, містити покрокові

інструкції з виконання завдання і не бути надто довгим. Приклад вікна довідкової системи наведено в Додатку Р.

Для ефективного використання розробленого програмного забезпечення необхідно створити інструкцію користувача.

**Методи і стиль написання.** Залежно від особливості програми і цільової аудиторії, інструкція користувача може наблизатися за способом написання до навчального посібника або до довідника. Порядок написання матеріалу в інструкції визначається користувацькою перспективою програми.

Якщо програма є інструментом, який дає змогу вирішувати практичні завдання з деякого кінцевого набору, то в інструкції користувача наводять типові процедури вирішення кожного з них. Наприклад, користувачу поштового клієнта потрібно знати, як написати і відправити повідомлення, як завантажити нові повідомлення з сервера, як відповісти на повідомлення і т.д. Кожну з цих дій можна розкласти на послідовні елементарні кроки, принаймні для типових ситуацій. У великих програмах таких користувацьких задач може бути багато. Інструкція користувача, побудована за принципом користувацьких задач, нагадує посібник, але при цьому відсутній притаманний посібникам методичний апарат: перевірка знань, запитання, вправи.

Якщо програму застосовують у широкій предметній області, в межах якої користувач може вирішувати завдання, поставлені ним самим, інструкція

користувача має бути близькою до довідника. У ній послідовно і методично повинні бути описані всі функції програми і порядок їх використання. Що з ними робити і на що направити, користувач вирішуватиме сам (або запишеться на навчальні курси). Так, в інструкції користувача для графічного редактора ми знайдемо опис всіх графічних примітивів, інструментів, фільтрів, але там не буде прямо сказано, як зобразити будівлю, автомобіль, чи, скажімо, собаку. Користувач або вміє це малювати, або ні.

Можливі й інші користувацькі перспективи. Бувають програми, за допомогою яких користувач контролює стан того чи іншого об'єкта, наприклад, промислової установки.

Якщо користувач використовує програму для вирішення завдань у нетривіальних предметних областях, до інструкції користувача рекомендується вводити концептуальний розділ. В ньому має бути описаний реалізований у програмі спосіб представлення об'єктів реального світу, щоб користувач добре розумів, з яким із них і на якому рівні абстракції він може працювати.

Створенням документації для кінцевих користувачів займаються спеціальні фахівці - технічні письменники.

#### **4.4. Типова структура інструкції користувача**

Через велике розмаїття програм важко уявити собі універсальну структуру інструкції користувача. У кожному конкретному випадку вона переважно визначатиметься особливостями програми. Тим не менш, типова структура інструкції користувача є такою:

1. Загальні відомості.
2. Встановлення і перше налаштування.
3. Основні поняття і визначення.
4. Інтерфейс користувача.
5. Робота з програмою.
6. Користувацькі налаштування.
7. Повідомлення про помилки.

Розділ “Робота з програмою” часто заміняють декількома послідовними розділами, які описують великі групи користувацьких задач чи функцій.

Як правило, на практиці використовують скорочений варіант інструкції користувача, який має таку структуру:

1. Компоненти ПЗ.
2. Встановлення ПЗ.
3. Налаштування ПЗ.
4. Базові функції ПЗ.
5. Аналіз можливих помилок.

Розглянемо рекомендації до створення інструкції користувача.

**1. Компоненти** ПЗ. У цьому підрозділі потрібно коротко описати розроблене ПЗ: використані технології, середовища розробки, мови програмування тощо.

#### Приклад.

Пакет розроблено мовою програмування C# 3.0 у середовищі розробки Microsoft Visual Studio.NET 2012 і може експлуатуватися під управлінням сімейства операційних систем Windows. Під час проектування підсистем відбувалося поєднання об'єктно-орієнтованого підходу до програмування з процедурно-орієнтованим. Всі класи документувались інформаційно і семантично.

Крім цього, подати вимоги до програмно-апаратних засобів, базовий набір файлів ПЗ та використані компоненти сторонніх розробників (фреймворки, бібліотеки тощо).

#### Приклад.

Для коректної роботи пакета необхідна користувальська машина з процесором не меншим за 200 MHz, оперативною пам'яттю, не меншою за 256 Mb. Для експлуатації пакета під управлінням сімейства операційних систем Windows необхідно встановити збірку класів .NET Framework 3.5, мати всі необхідні файли бібліотек і налаштувань. Перелік необхідних файлів, пояснення їх призначення та інформацію про приналежність до конкретного проекту наведено у табл. 4.1.

Підсистема кластеризації образів використовує засоби проекту з відкритим кодом (*open source*) *Image Processing Lab (IPLab)* [джерело], який є віконним проектом для взаємодії з бібліотекою *Aforge.NET* і призначений для обробки зображення. Своєю чергою, проект *IPLab* використовує такі компоненти з відкритим кодом: *DockManager Control* (розробник *Weifen Luo*) [джерело]; *SourceGrid* (розробник *Davide Icardi*) [джерело]; *A Simple C# Toolbar Docking Framework* (розробник *Rogerio Paulo*) [джерело].

Таблиця 4.1

## Набір файлів для коректної роботи ППП “Кластеризація”

№ з/п	Файл	Призначення	Належить проекту
1	<i>Aforge.dll</i>	Бібліотека ядра	<i>Aforge.NET</i>
2	<i>Aforge.Imaging.dll</i>	Бібліотека для обробки зображень	
4	<i>app.config</i>	Файл налаштувань підсистеми кластеризації образів	ППП “Кластеризація”
5	<i>ClustersCore.dll</i>	Бібліотека ядра	
6	<i>ImageClustering.exe</i>	Виконавчий файл підсистеми кластеризації образів	
7	<i>log.txt</i>	Текстовий файл, що містить записи журналу	

**2. Встановлення ПЗ** - перелік необхідного спеціалізованого ПЗ сторонніх виробників та порядок дій для встановлення ПЗ.

Приклад.

Для роботи пакета необхідно встановити *SQL Server 2012*.

Якщо необхідно здійснити базові налаштування, то файл з переліком налаштувань додається до диску з ПЗ. Якщо налаштування можна коротко охарактеризувати - тоді подати в цій інструкції.

Приклад.

Для роботи пакета необхідно встановити *SQL Server 2012*. При цьому потрібно встановити повнотекстовий пошук.

Приклад.

1. Встановити *SQL Server 2012*.

2. Встановити пакет *ImageProcessingDB*, запустивши на виконання файл *ipDB setup.exe*.

3. Запустити на виконання файл *ipdb.exe*, який створить базу даних *IMGPROCDB* та всі необхідні таблиці.

**3. Налаштування ПЗ** - опис налаштувань, які підтримуються в системі для коректної роботи програми.

Приклад.

Для коректної роботи програми виберіть у пункті *Memory* меню *Settings* параметр *Standard*.

**4. Базові функції ПЗ** - опис базових функцій ПЗ. Потрібно коротко описати функцію розробленого ПЗ та щонайменше один спосіб доступу до неї.

### Приклад.

Для збереження результатів перетворення зображення в головному меню виберіть пункт — “Зберегти результати ”.

Для запуску обчислення результатів виберіть почергово такі пункти головного меню:

1. “Згенерувати ”.
2. “Поділити простір ”.
3. “Обчислити відстань ”.

**5. Аналіз помилок** - опис можливих помилок та проблем і шляхи їх виправлення. Зазвичай розроблене ПЗ містить файл журналу, куди записують всі події, спричинені користувачем.

### Приклад.

У випадку збою програми переглянути файл *log.txt*.

При виникненні повідомлення “Memory 2 error” закрити програму, почистити кеш в папці *Cache*, запустити програму.

## 4.5. Інтерфейсні правила побудови повідомлень

Під час проектування інтерфейсу значну увагу звертають на дизайн різних типів повідомлень, адже саме завдяки повідомленням про помилки, додаткову інформацію або повідомленням-підтвердженням користувач взаємодіє зі системою у виняткових ситуаціях.

Усі повідомлення для віконних програмних систем поділяють на чотири типи (табл. 4.2).

**Таблиця 4.2**

### Типи повідомлень

Назва типів повідомлень	Мета повідомлень
Повідомлення про помилки ( <i>error message</i> , англ.)	Повідомляє користувача про збій у програмі, який вже відбувся.
Повідомлення-попередження ( <i>warning message</i> , англ.)	Повідомляє користувача про можливий збій програми внаслідок виконання певних дій.
Повідомлення-підтвердження ( <i>confirmation message</i> , англ.)	Запитує користувача про необхідність продовження його дій.
Інформаційне повідомлення ( <i>notification message</i> , англ.)	Інформує користувача про деяку ситуацію, що виникла в системі (наприклад, наявність оновлення програми). Рідко пов'язане з діями користувача.

Повідомлення про помилки. Цей тип повідомень виникає у випадку збоїв програми або неможливості завершення деякої операції. Зображення такі повідомлення у віконних програмних системах у вигляді модальних вікон, які складаються з назви вікна, іконки (хрестик у червоному кружку), тексту повідомлення та кнопки “Закрити” (рис. 4.24).

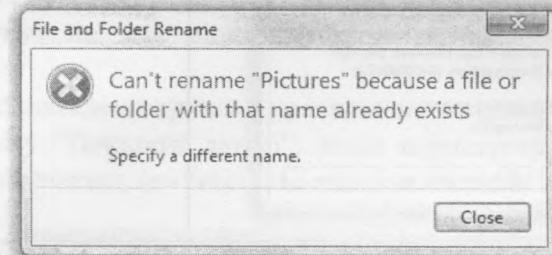


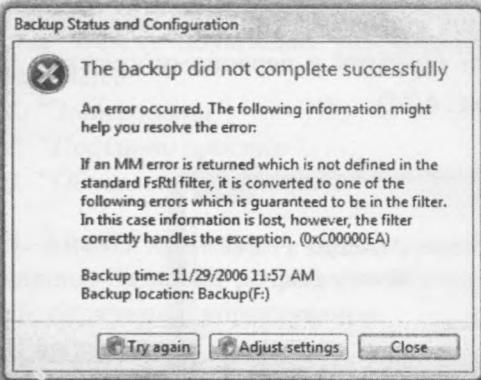
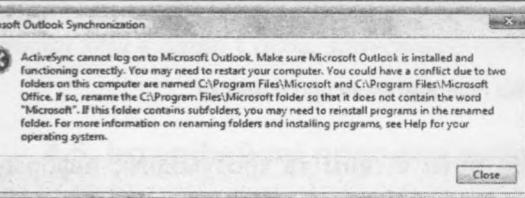
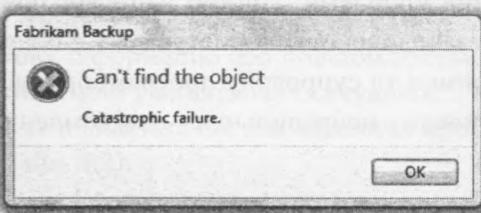
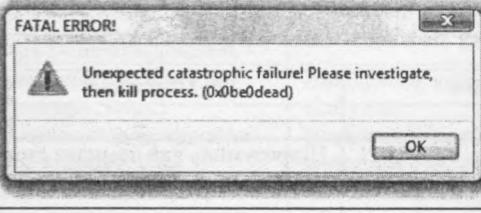
Рис. 4.24. Приклад повідомлення про помилку

Текст повідомлення повинен бути чітким та зрозумілим, інформувати користувача про збій у системі, пояснювати, чому стала така ситуація і коротко описувати способи вирішення проблеми. Погано написані повідомлення про помилки призводять до незадоволеності користувача і є основною причиною витрат під час етапу підтримки та супроводу програмного забезпечення. У табл. 4.3 наведено приклади неправильних повідомень про помилку.

**Таблиця 4.3**

**Приклади неправильних повідомень про помилку**

Приклад неправильного повідомлення	Пояснення
1	2
A screenshot of an Internet Explorer modal dialog box. The title bar says 'Internet Explorer'. The main content area contains an icon of a red circle with a white cross, followed by the text: 'Unknown error'. At the bottom right of the dialog is an 'OK' button.	Незрозуміло, яка помилка стала і чому

1	2
	<p>Пояснення помилки з погляду розробника, а не користувача.</p> <p>Користувач не повинен знати, що таке FsRtl фільтр, код помилки</p>
	<p>Сильно деталізоване повідомлення.</p> <p>Повідомлення має бути чітким і стисливим, адже користувач не має бажання витрачати свій час на некорисну інформацію. Для детальнішої інформації варто використовувати додаткову кнопку "Показати деталі"</p>
	<p>Повідомлення, що це катастрофічний збій, для користувача не є інформаційним. Крім цього, незрозуміло, чому кнопка має підпис "Ок", а не "Закрити"</p>
	<p>Неправильна іконка, невідповідний текст, наявність коду помилки, неправильна назва вікна.</p>

Повідомлення про помилки складаються з трьох частин:

1) опис помилки;

- 2) причина збою;
- 3) виправлення помилки.

Крім цього, властивостями правильних повідомлень про помилки є:

- 1) повідомлення повинне бути зрозуміле користувачу (варто уникати технічних слів, кодів помилки);
- 2) повідомлення про помилки повинне з'являтись рідко, лише у випадках складних збоїв програми;
- 3) користувач не має відчувати вини за свої дії, читуючи текст повідомлення.

На рис. 4.25 наведено приклад правильного повідомлення про помилку, яке містить кнопку “Показати деталі”, якщо користувач захоче детальніше ознайомитись із помилкою, що виникла в системі.

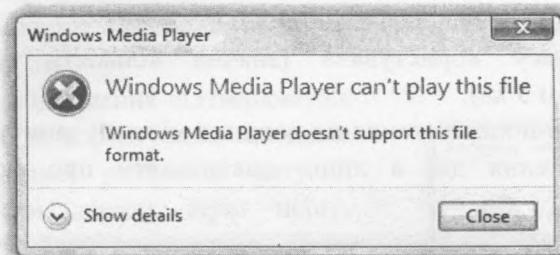


Рис. 4.25. Приклад правильного повідомлення про помилку

Повідомлення-попередження. Цей тип повідомлень необхідний для застереження користувача від дій, що в майбутньому можуть привести до збоїв системи. Зображеніся такі повідомлення у віконних програмних системах у вигляді модальних вікон, які складаються з назви вікна, іконки (знак оклику у жовтому трикутнику), тексту повідомлення та кнопки “Закрити” (рис. 4.26).

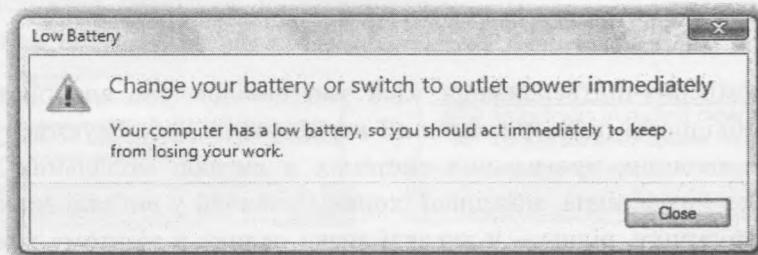


Рис. 4.26. Приклад повідомлення-попередження

Властивостями правильних повідомлень-попереджень є:

- 1) чіткий і зрозумілий текст попереджень;
- 2) вплив помилки на роботу програми;
- 3) відомості про те, що користувач повинен зробити;
- 4) інформація про те, що буде у випадку невиконання необхідних дій користувачем.

Рекомендації щодо оформлення тексту повідомлення залишаються такими, як і для повідомлень про помилки.

Найпоширенішими ситуаціями, у яких варто використовувати такий тип повідомлень, є:

- 1) попередження дій, що можуть привести до фінансових втрат;
- 2) попередження дій, що призводять до порушення цілісності програмної системи або до виникнення помилки;
- 3) захист і контроль конфіденційної інформації;
- 4) втрата часу користувача (значна кількість часу, наприклад, очікування протягом 3 хв).

Повідомлення-попередження не повинне містити запитувань до користувача про продовження дії, а лише повідомляти про можливу помилку (рис. 4.27).

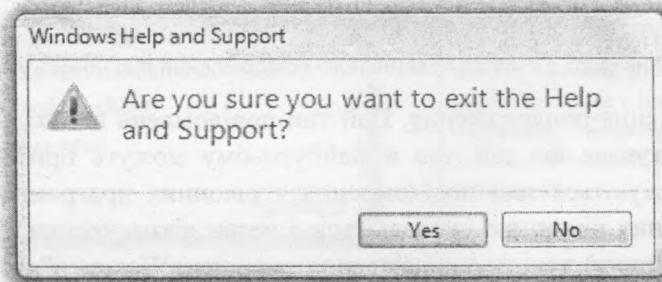


Рис. 4.27. Приклад неправильного повідомлення-попередження

Повідомлення-підтвердження. Цей тип повідомлень використовують у випадку необхідності підтвердження дії користувача. Зображенням є повідомлення у віконних програмних системах у вигляді модальних вікон, які складаються з назви вікна, можливої іконки (зазвичай у вигляді знака питання у синьому кружечку, рідше - у вигляді знака оклику в жовтому трикутнику), тексту запитання та кнопок з альтернативними варіантами відповіді (рис. 4.28).

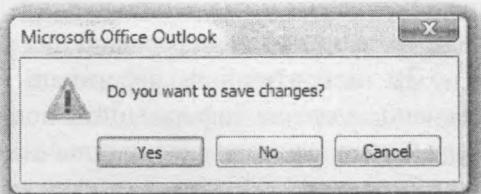


Рис. 4.28. Приклад повідомлень-підтвердженъ

У загальному випадку повідомлення-підтвердження мають такі властивості:

- 1) повідомлення відображають результат дії користувача;
- 2) повідомлення складаються з простого запитання і декількох

можливих відповідей;

- 3) повідомлення зосереджують увагу користувача для правильного вибору подальшої дії.

Зазвичай повідомлення-підтвердження виникають у разі дії користувача, що може привести до деякої ризикової ситуації (наприклад, втрати даних). Крім цього, потрібно уникати частого використання такого типу повідомлень, бо це швидко втомить користувача (рис. 4.29).

Інформаційні повідомлення. Повідомлення такого типу суть інформативні: про стан системи, наявність оновлень тощо. Інформація в таких повідомленнях є корисною, актуальною, але не критичною для користувача. Приклади інформаційних повідомлень наведено на рис. 4.30.

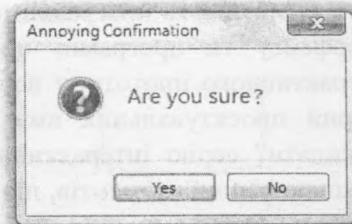


Рис. 4.29. Приклад неправильного повідомлення-попередження

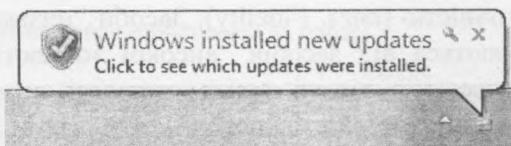
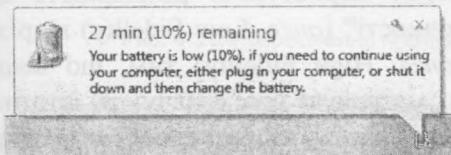


Рис. 4.30. Приклади інформаційних повідомлень

Властивостями правильних інформаційних повідомлень є:

- 1) корисність інформації для користувача;
- 2) некритичність інформації (інакше використовують інші типи повідомлень), адже на інформаційні повідомлення користувач не завжди звертає увагу (адміністратор може відключити інформаційні повідомлення, користувач може бути дуже зайнятим і не мати часу для перегляду повідомлення);
- 3) можливість ігнорування повідомлення. Тобто інформаційні повідомлення не вимагають негайних дій від користувача і тому можуть ігноруватись.

#### **4.6. Програмні продукти для розроблення графічного інтерфейсу**

Існує низка спеціалізованих програмних продуктів, що допомагають створити якісний графічний інтерфейс користувача (розмаїття цих продуктів підкреслює важливість цієї задачі).

Розрізняють програмні продукти для розроблення прототипу графічного інтерфейсу та програмні продукти для “шліфування” деталей. Побудова інтерактивного прототипу вартоє витрачених на неї зусиль. Це і не дивно - добрий проектувальник вміє за кілька годин у професійному середовищі “накидати” серію інтерактивних прототипів з переходами між сторінками і реакцією різних елементів, що дає уявлення про те, як працюватиме майбутній продукт. При цьому він одержує наочне втілення частини специфікації вимог і реальний предмет для узгодження з замовником. Якщо замовникові щось не сподобається і він захоче це змінити, змінювати ескіз не так шкода, як вносити зміни у готовий, відшліфований до дрібниць інтерфейс (zmінювати макет будинку простіше, ніж сам будинок). Із застосуванням прототипів як інструментів для уточнення вимог шанси завершити програмний продукт у межах відведеного часу і бюджету зростуть.

Засоби побудови прототипу інтерфейсу користувача розрізняють за точністю (*англ. Fidelity*). Засоби “низької точності” (*англ. Low Fidelity*) відрізняються від засобів “високої точності” (*англ. High Fidelity*) тим, що вони опускають деталі, використовують дешевші матеріали і забезпечують істотно менше інтерактивності, а то й взагалі дають змогу зробити лише статичні прототипи. За допомогою *high fidelity*-засобів розробляють прототипи, наближені до кінцевих продуктів.

Незважаючи на існування низки “електронних асистентів”, першорядними low fidelity-інструментами для розроблення прототипу інтерфейсу користувача є... олівець і папір. Першу розмітку бажано робити на аркушах паперу однакового розміру, що заміняють екран. Проектуючи екрани, необхідно пам'ятати, що найважливіших функцій необхідно досягти за мінімальну кількість переходів між екранами (і взагалі, мінімальною кількістю рухів користувача). На папері малюють спрощене зображення екрана, при цьому часто блоки замінюють прямокутниками з підписом, що вони позначають (рис. 4.31, *a*).

Кнопки зображають прямокутниками із заокругленими кутами, зарезервовані ділянки (*англ.* placeholders) - прямокутниками з діагоналями, тексти - горизонтальною лінією з вертикальною рискою зліва (рис. 4.31, *b*). На цьому етапі немає змісту відволікатися на деталі, оскільки прототип пройде, ймовірно, не одну ітерацію правок. Однак, вже за “паперовим” прототипом можна оцінити, чи найтиповіші сценарії використання реалізуються найкоротшими шляхами, чи є можливість повернення до попередніх екранів (згадаймо третю евристику Нільсена) тощо.

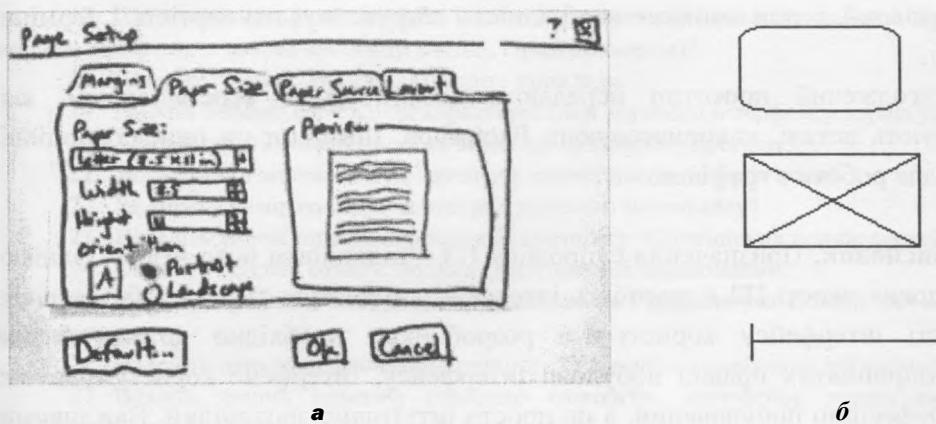


Рис. 4.31. Розроблення прототипу інтерфейсу користувача на папері

З-поміж засобів прототипування інтерфейсу низької точності популярними є Balsamic Mockups та iMockups для мобільних застосунків.

З-поміж high-fidelity засобів популярними є Axure Pro, MS Expression Blend, iRise Studioj, а також давно знайомі нам MS Visio і MS PowerPoint.

Axure Pro підходить для веб-сайтів, настільних та мобільних застосунків. Перевагами Axure Pro є простота і швидкість розроблення прототипу, можливість використання без навичок програмування (вся поведінка ґрунтуються на задаванні реакцій на події), величезний асортимент типових графічних елементів, зокрема і для мобільних застосунків, можливість перегляду готових прототипів у браузері без додаткових плагінів, генерування звітів тощо. Microsoft Expression Blend теж дає змогу організувати інтерактивність без необхідності писати код і пропонує великий вибір графічних елементів, а також надає значну свободу у розробленні власних елементів і підтримку віддаленого перегляду та тестування інтерфейсу користувача.

У засобах прототипування є цікава опція — викривлення контурів графічних елементів, які ми звикли бачити рівними відрізками. Ця опція “імітує” рисування елементів від руки і потрібна для порозуміння з замовниками. Оскільки замовник може бути далекий від інформаційних технологій, є ризик, що інтерактивний прототип у його очах виглядатиме як версія продукту, наблизена до кінцевої (“фасад будівлі” прийматиметься за саму будівлю), і тоді виникне необхідність обґрунтовувати вартість і терміни проекту.

Узгоджений прототип передають дизайнерам і верстальникам, які продумують деталі, використовуючи Photoshop, Illustrator чи інші повноцінні засоби для роботи з графікою.

**Висновок.** Призначення супроводу ПЗ - підвищити його якість. Однією зі складових якості ПЗ є зручність інтерфейсу користувача. Для забезпечення зручності інтерфейсу користувача розробникам необхідно дотримуватися загальноприйнятих правил побудови інтерфейсу. Інтерфейс користувача має бути професійно побудований, а не просто інтуїтивно зрозумілий. Важливими для користування програмною системою є також необхідна користувачеві документація, довідкова система, допомога.

## **Контрольні питання для самоперевірки**

1. Які задачі виконують на етапі супроводу ПЗ?
2. У чому полягає етап експлуатації ПЗ? Що є вхідними даними для цього етапу?
3. У чому складність виконання етапу супроводу ПЗ?
4. Якою має бути програмна система, щоб її було легко супроводжувати?
5. На які дві групи можна поділити задачі супроводу? Яка між ними принципова відмінність?
6. У чому полягає етап вдосконалення ПЗ? Наведіть приклади.
7. Охарактеризуйте модель якості, задану у міжнародному стандарті.
8. Дайте означення якості ПЗ.
9. Дайте означення зручності користування програмою. Яку ще назву використовують для цієї характеристики якості?
10. Що таке інтерфейс користувача?
11. Наведіть приклад неграфічного інтерфейсу користувача.
12. Чи є переваги у використанні неграфічного інтерфейсу? Відповідь обґрунтуйте.
13. На якому етапі ЖЦ ПЗ типово розробляється інтерфейс користувача програми?
14. Що є складовими типового графічного інтерфейсу користувача?
15. Що таке контекстне меню? Яке його основне призначення?
16. Що таке вікно інтерфейсу програми та які його складові?
17. Що означають різні способи написання пунктів у меню (звичайним шрифтом, жирним шрифтом, з трьома крапками в кінці, сірим кольором)?
18. Що таке “гарячі” клавіші? Наведіть приклади.
19. Назвіть основні евристичні характеристики зручного інтерфейсу користувача.
20. У чому полягає евристичність вимог до інтерфейсу програм?
21. Які чинники впливають на зручність користування інтерфейсом?
22. Як можна виміряти характеристики зручного інтерфейсу?
23. Наведіть типові приклади реалізації принципу “Сповіщення про поточний стан”.
24. Опишіть основні ознаки людино-орієнтованих повідомлень.
25. Наведіть приклади використання лічильників і випадаючих списків для введення вхідних даних.
26. Наведіть приклади реалізації принципу “Управління свободою дій користувача”.
27. Назвіть типові рішення стосовно елементів інтерфейсу користувача, які забезпечують характеристику “Цілісність і стандарти”.
28. Які принципи побудови повідомлень про помилкові ситуації під час роботи програм?
29. Які є способи запобігання введенню користувачем неправильної інформації через елементи графічного інтерфейсу користувача?
30. Що мають на увазі, коли говорять про естетичність інтерфейсу користувача?
31. Наведіть приклад елементів інтерфейсу користувача, які забезпечують розпізнавання, а не згадування.

32. Що забезпечує гнучкість інтерфейсу користувача?
33. Що таке “розшарування” інтерфейсу користувача?
34. Наведіть 5 прикладів типових піктограм для позначення дій у програмі.
35. У чому суть правила трьох клацань (кліків) в інтерфейсі користувача?
36. У чому полягає ітеративність прототипування інтерфейсу користувача?
37. Які є відомі типи сучасних інтерфейсів корисувача, окрім неграфічного й графічного?
38. Яка типова структура інструкції користувача?
39. Які позначення використовуються на паперовому прототипі інтерфейсу?
40. Коли інструкція користувача близька за формою до довідника?
41. Коли інструкція користувача близька за формою до посібника?
42. Чим інструкція користувача програмою для нетипової предметної області відрізняється від типової інструкції?
43. Які складові скороченого варіанта інструкції користувача?
44. Як формується розділ “Робота з програмою”?
45. Які складові типової довідкової системи?
46. Які існують типи повідомлень у програмних системах?
47. З яких частин складається повідомлення про помилки?
48. Які властивості мають повідомлення-підтвердження?
49. Наведіть приклади інформаційних повідомлень.
50. Які властивості мають повідомлення про попередження?
51. Які властивості правильних інформаційних повідомлень?
52. Охарактеризуйте програми для розроблення графічного інтерфейсу.
53. Що таке зручність інтерфейсу користувача?
54. Як інструкція користувача пов’язана з ТЗ?
55. Чим відрізняються системи прототипування інтерфейсів категорії high fidelity від low fidelity?
56. Чим відрізняється інструкція користувача з типовою структурою від інструкції користувача зі скороченим змістом?
57. Які фахівці ПЗ задіяні на етапі експлуатації й супроводу ПЗ?
58. Чим експлуатація відрізняється від супроводу?

## ЛАБОРАТОРНА РОБОТА № 4

**Тема.** Підвищення зручності користування програмою як задача супроводу ПЗ.

**Мета.** Навчитися розробляти ескізи інтерфейсу користувача та створювати інструкцію користувача.

### Перелік питань для теоретичних відомостей.

Означення етапу супроводу ПЗ. Зручність користування програмою як якість програми. Графічний інтерфейс програми. Тестування зручності інтерфейсу користувача. Типові складові графічного інтерфейсу програми. Евристичні характеристики зручного інтерфейсу користувача: суть, принципи, приклади. Програмні продукти для розроблення графічного інтерфейсу. Інструкція користувача ПЗ. Зміст спрощеного варіанта інструкції користувача ПЗ. Учасники процесу супроводу ПЗ. Основні результати етапу супроводу ПЗ. Місце й роль етапу супроводу ПЗ у життєвому циклі ПЗ.

### Завдання

#### Необхідні програмні засоби для виконання завдання:

- ^ середовище програмування з візуальними засобами побудови інтерфейсу;
- ^ офісний редактор із засобами графіки;
- ^ текстовий редактор.

**Умова.** Для контролю проміжних результатів лабораторної роботи пропонується завдання поділити на частини.

#### Частина I

1. Запропонуйте три різні вигляди піктограм для об'єктів/понять/дій згідно з варіантом:

- A. Конфіденційність/приватність
- B. Сортувати
- C. Історія/Журнал дій
- D. Ігри
- E. Згрупувати

- F. Соціальна мережа
- G. Квиток/проїзний
- H. Закладка
- I. Нагорода/подарунок
- J. Відправити/надіслати
- K. Відеофайл
- L. Фільтр/Фільтрувати
- M. Задати колір
- N. USB флеш-накопичувач
- O. Поділитися (новиною, файлом)/поділитися з другом
- P. Уподобати
- Q. Поскаржитися на спам
- R. Підписати документ
- S. Сертифікати

2. Знайдіть по три приклади графічних елементів інтерфейсу користувача ПЗ, що ілюструють виконання однієї з десяти евристик. Запропоновані приклади мають відрізнятися від тих, що наведені в теоретичних відомостях.

3. Виберіть довільні 5 діалогів (усі з різних програмних продуктів) і проаналізуйте, чи заданий програмний продукт відповідає десяти принципам побудови інтерфейсу користувача.

## **Частина II**

1. Розробити прототип інтерфейсу до наступної версії програми, яку протестовано у попередній лабораторній роботі. Інтерфейс має забезпечити зручність користування програмою.

Для виконання цього завдання потрібно зобразити ескіз інтерфейсу програми набором рисунків.

2. Скласти інструкцію користувача для програми (з передбаченим інтерфейсом).

Для виконання цього завдання потрібно використати описані рекомендації для створення інструкції користувача зі скроченим змістом.

## Варіанти завдань до теоретичних відомостей у звіті

Номер варіанта	Перелік контрольних питань
1	2, 20, 19
2	4, 10, 29
3	1, 25, 17
4	3, 15, 24
5	7, 22, 32
6	5, 30, 36
7	8, 27, 42
8	6, 26, 34
9	11, 33, 40
10	12, 18, 41
11	13, 29, 37
12	16, 20, 1
13	17, 28, 39
14	18, 32, 2
15	25, 35, 4
16	24, 34, 3
17	43, 20, 7
18	38, 21, 1
19	40, 22, 5
20	36, 19, 6
21	35, 18, 8
22	44, 15, 11
23	41, 16, 10
24	42, 17, 9
25	24, 12, 2
26	14, 39, 45
27	46, 11, 3
28	3, 34, 38
29	2, 27, 45
30	1, 25, 43

### Вимоги до звіту

Звіт оформлюють за загальними вимогами, викладеними у методичних рекомендаціях.

У теоретичних відомостях згідно з індивідуальним завданням необхідно дати відповідь на 3 контрольні питання з наведеної вище списку.

У розділі “Отримані результати” роздрукувати прототипи інтерфейсу користувача, інструкцію користувача.

У висновках зазначити й охарактеризувати отримані результати під час виконання лабораторної роботи.

## ТЕСТОВІ ЗАВДАННЯ ДЛЯ САМОПЕРЕВІРКИ ЗНАНЬ ЗА ТЕМАМИ ЛАБОРАТОРНИХ РОБІТ

Подано типові тестові завдання за чотирима темами лабораторних робіт. До кожного тесту задається набір п'яти варіантів відповіді, з яких лише один правильний.

Тести можуть бути використані і для підготовки до поточного, і до семестрового контролю.

### I. Тести до лабораторної роботи № 1

1. До основних етапів життєвого циклу ПЗ не входить?

- А) документування ПЗ;
- Б) супровід ПЗ;
- В) проектування ПЗ;
- Г) розроблення ПЗ;
- Д) усі відповіді правильні.

2. Неосновними в життєвому циклі ПЗ є такі етапи?

- А) оцінювання якості ПЗ;
- Б) проведення тренінгів для розробників ПЗ;
- В) створення робочих місць;
- Г) оцінювання ризиків виконання проекту;
- Д) усі відповіді правильні.

3. Функціональною вимогою до програмного забезпечення може бути:

- А) ПЗ повинне забезпечити безперебійну роботу 1000 користувачів;
- Б) над проектом повинні працювати лише 10 програмістів та 3 тестувальники;
- В) проект повинен бути виконаний до 1 січня 2020 року;
- Г) щодня надсилати замовнику звіт про виконання роботи розробниками проекту;
- Д) немає правильної відповіді.

4. Оберіть лише функціональні вимоги до ПЗ:

- А) розробити вкладку для побудови графіків тригонометричних функцій;

Б) вікно з повідомленням: “Звіт буде сформований через декілька хв” повинне з’явитись на екрані користувача за 5 хв до закінчення формування звіту;

В) програма передбачає роботу лише з такими графічними форматами: \*.jpg, \*.bmp;

Г) надсилати лист-підтвердження користувачу після замовлення товару;  
Д) усі відповіді правильні.

5. Оберіть лише нефункціональні вимоги до ПЗ:

А) розробити документ - план тестування ПЗ;

Б) розробити ПЗ для роботи під операційними системами сімейства Windows;

В) усі кнопки інтерфейсу повинні мати форму овалу;

Г) забезпечити високу надійність розробленого ПЗ;

Д) усі відповіді правильні.

6. Технічне завдання є результатом етапу ЖЦ:

А) документування ПЗ;

Б) проектування ПЗ;

В) визначення та аналізу вимог до ПЗ;

Г) роботи з замовником;

Д) немає правильної відповіді.

7. Для опису вимог не можуть використовуватися такі види документів:

А) технічне завдання;

Б) прототип ПЗ;

В) блок-схема ПЗ;

Г) діаграма прецедентів;

Д) усі відповіді правильні.

8. До опису технічного завдання ПЗ не входить?

А) терміни розроблення ПЗ;

Б) мета створення ПЗ;

В) оцінювання ризиків;

Г) системні вимоги;

Д) немає правильної відповіді.

9. Чи можна вносити зміни до технічного завдання?

- A) можна завжди;
- Б) не можна;
- В) можна лише на початку кожного етапу ЖЦ ПЗ;
- Г) можна лише за згоди замовника;
- Д) немає правильної відповіді.

10. У якому вигляді не можна описати вимоги до ПЗ?

- A) графічному;
- Б) табличному;
- В) текстовому;
- Г) словесно-усному;
- Д) немає правильної відповіді.

11. Оберіть властивості вимог до ПЗ:

- A) терміни розроблення вимог;
- Б) пріоритетизація;
- В) необхідність;
- Г) рівень функціональності;
- Д) усі відповіді правильні.

12. “Розробити вкладку для побудови графіків тригонометричних функцій” - це приклад:

- A) хорошої вимоги;
- Б) поганої вимоги, порушена умова необхідності;
- В) поганої вимоги, порушена умова послідовності;
- Г) поганої вимоги, порушена умова коректності;
- Д) немає правильної відповіді.

13. Які види діяльності розрізняють під час роботи з вимогами:

- A) документування вимог;
- Б) аналіз узгоджених із замовником вимог;
- В) оцінка зрозуміlosti вимог для майбутньої реалізації;
- Г) виявлення джерел вимог;
- Д) усі відповіді правильні.

14. Вимоги необхідно документувати для:

- А) програмістів, які на їхній основі проектують ПЗ;
- Б) оцінки коштів розроблення ПЗ;
- В) оцінки термінів розроблення ПЗ;
- Г) уникання непорозумінь із замовником;
- Д) усі відповіді правильні

15. Організаційні процеси ЖЦ ПЗ:

- А) діють протягом усього ЖЦ;
- Б) пов'язані із додатковими етапами ЖЦ;
- В) належать до основних процесів ЖЦ;
- Г) завжди присутні під час розроблення ПЗ;
- Д) немає правильної відповіді.

16. Аналізують структури даних для реалізації ПЗ на основі:

- А) прототипу ПЗ;
- Б) вимог замовника;
- В) блок-схеми;
- Г) технічного завдання;
- Д) немає правильної відповіді.

17. Технічне завдання - це документ, що дає можливість:

- А) оцінити терміни виконання проекту;
- Б) уникнути протиріч між замовниками та розробниками;
- В) скласти план тестування;
- Г) розпочати етап проектування ПЗ;
- Д) усі відповіді правильні.

18. Основні етапи ЖЦ ПЗ мають такі властивості:

- А) повинні завжди виконуватись у строгому порядку - аналіз і специфікування вимог, проектування, кодування, тестування;
- Б) обов'язково усі процеси повинні бути виконані під час розроблення проекту;
- В) всі етапи повинні бути задокументованими;
- Г) у кінці кожного етапу отримують певний результат;
- Д) усі відповіді правильні.

19. Які нефункціональні вимоги можна верифікувати?

- А) вимоги до графічного інтерфейсу;
- Б) вимоги щодо обчислювальних ресурсів;
- В) апаратні вимоги;
- Г) вимоги щодо ефективності ПЗ;
- Д) усі відповіді правильні.

20. Вимоги до ПЗ поділяють на:

- А) апаратні та програмні;
- Б) функціональні та нефункціональні;
- В) основні та допоміжні;
- Г) хороші та погані;
- Д) немає правильної відповіді.

21. Нефункціональна вимога до програми - це:

- А) необов'язкова до реалізації вимога;
- Б) неправильно запрограмована вимога;
- В) вимога стосовно характеристики програми, що стосується безпосередньо її поведінки;
- Г) вимога стосовно характеристики програми, що не стосується безпосередньо її поведінки;
- Д) усі відповіді правильні.

22. Вимога стосовно продуктивності програми означає:

- А) скільки розробників бере участь у проекті;
- Б) як швидко буде розроблено програму;
- В) як швидко працюватиме програма;
- Г) як довго використовуватиметься програма;
- Д) немає правильної відповіді.

23. Вимога стосовно супроводжуваності програми означає:

- А) скільки розробників бере участь у проекті;
- Б) як швидко буде розроблено програму;
- В) як швидко працюватиме програма;
- Г) як довго буде використовуватися програма;
- Д) немає правильної відповіді.

24. Модель життєвого циклу ПЗ:

- А) зображає порядок виконання етапів ЖЦ ПЗ;
- Б) є вхідними даними для написання вимог до ПЗ;
- В) будується на етапі проектування ЖЦ ПЗ;
- Г) обов'язково містить усі етапи ЖЦ ПЗ;
- Д) немає правильної відповіді.

25. Процес життєвого циклу ПЗ:

- А) певна множина дій і задач;
- Б) призводить до значимого результату;
- В) є складовою частиною ЖЦ ПЗ;
- Г) іноді є синонімом до етапу ЖЦ ПЗ;
- Д) усі відповіді правильні.

26. Поняття “модель ЖЦ ПЗ” і “ЖЦ ПЗ” - це:

- А) ідентичні поняття;
- Б) зовсім незалежні поняття;
- В) пов’язані поняття через спільну змістовну частину;
- Г) одне є складовою іншого поняття;
- Д) немає правильної відповіді.

27. Визначення вимог належить до категорії/групи процесів:

- А) допоміжні;
- Б) інженерні основні;
- В) інженерні неосновні;
- Г) замовник-постачальник;
- Д) немає правильної відповіді.

28. Специфікацію вимог до ПЗ займається:

- А) інженер-програміст;
- Б) інженер з якості;
- В) замовник;
- Г) технічний письменник;
- Д) немає правильної відповіді.

29. Визначення вимог відбувається:

- А) рекурсивно;
- Б) ітеративно;
- В) лише за вимогою замовника;
- Г) лише за потреби з боку розробників;
- Д) усі відповіді правильні.

30. Ієрархічна декомпозиція життєвого циклу розроблення ПЗ означає:

- А) обов'язковість виконання основних етапів;
- Б) застосування допоміжних процесів;
- В) поділ на основні етапи;
- Г) існування складових, які деталізуються іншими елементами;
- Д) усі відповіді правильні.

31. Термін “життєвий цикл ” у галузі ПЗ:

- А) є оригінальним;
- Б) запозичено з технічних наук;
- В) запозичено з біології;
- Г) є невідомого походження;
- Д) немає правильної відповіді.

32. Визначення вимог відбувається:

- А) інтегрально;
- Б) інтерактивно;
- В) лише за вимогою замовника;
- Г) лише за потреби зі сторони розробників;
- Д) немає правильної відповіді.

33. Аналізом і визначенням вимог до ПЗ займається:

- А) менеджер проекту;
- Б) системний аналітик;
- В)замовник;
- Г) експерт предметної області;
- Д) усі відповіді правильні.

34. Вимога стосовно ефективності програми означає:

- А) чи правильно працює програма;
- Б) чи програмою зручно користуватися;
- В) як співвідносяться її вартість й якість;
- Г) як довго буде використовуватися програма;
- Д) немає правильної відповіді.

35. Аналізом і визначенням вимог до ПЗ не займається:

- А) інженер з якості;
- Б) інженер-програміст;
- В) технічний письменник;
- Г) постачальник;
- Д) усі відповіді правильні.

## **ІІ. Тести до лабораторної роботи № 2**

1. Найбільш працемістким є такий етап ЖЦ:

- А) проектування;
- Б) тестування;
- В) розроблення;
- Г) документування;
- Д) немає правильної відповіді.

2. До етапу проектування можна зарахувати такі види діяльності:

- А) розроблення алгоритму;
- Б) розроблення прототипу графічного інтерфейсу;
- В) створення блок-схем;
- Г) проектування структур даних;
- Д) усі відповіді правильні.

3. Основна мета етапу проектування:

- А) створити блок-схеми;
- Б) розробити архітектуру ПЗ;
- В) розробити окремі компоненти ПЗ;
- Г) проаналізувати технічне завдання;
- Д) немає правильної відповіді.

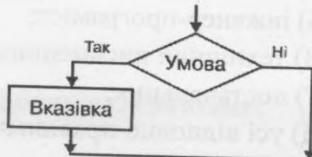
4. Способи подання алгоритму:

- A) блок-схеми;
- Б) словесний;
- В) табличний;
- Г) графічний;
- Д) усі відповіді правильні.

5. Оберіть рисунок, на якому зображенено цикл з післяумовою.



А)



Б)



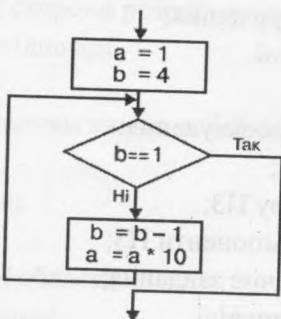
В)



Г)

Д) немає правильної відповіді

6. Обчисліть значення, якого набуває змінна а після виконання вказівок блок-схеми, поданої на рисунку:



- A) 10;
- Б) 100;
- В) 1000;
- Г) 10000;
- Д) немає правильної відповіді.

7. Яка властивість алгоритму означає його розбиття на відповідні кроки:

- А) дискретність;
- Б) Скінченність;
- В) збіжність;
- Г) детермінованість;
- Д) немає правильної відповіді.

8. Для організації назв областей України краще обрати:

- А) перелічуваний тип;
- Б) структуру;
- В) масив;
- Г) динамічний список;
- Д) немає правильної відповіді.

9. Задано  $\text{int } nVector[10] = \{ 5, 5, 5, 5, 5, 5, 5, 5, 5, 5 \}$ ,  $\text{int}$  займає пам'ять в 1 байт. Яку кількість пам'яті займає весь масив?

- А) 1 байт;
- Б) 5 байт;
- В) неможливо визначити;
- Г) 10 байт;
- Д) немає правильної відповіді.

10. Які структури даних можуть об'єднувати різноманітні елементи?

- А) масиви;
- Б) клас;
- В) перелічувані типи;
- Г) примітиви;
- Д) усі відповіді правильні.

11. Пам'ять для статичних типів даних виділяють:

- А) під час етапу виконання програми;
- Б) програміст може задати час;
- В) під час етапу компіляції;
- Г) на етапі кодування;
- Д) немає правильної відповіді.

12. Властивість використовувати алгоритм обчислення коренів квадратного рівняння для будь-яких цілих та дробових чисел називається:

- А) масовість;
- Б) Скінченність;
- В) результативність;
- Г) універсальність;
- Д) немає правильної відповіді.

13. На етапі кодування ЖЦ ПЗ відбувається:

- А) написання коментарів;
- Б) написання модулів;
- В) реалізація алгоритмів;
- Г) налагодження тексту програми;
- Д) усі відповіді правильні.

14. Для полегшення рефакторингу коду виконують:

- А) застосовують ReSharper;
- Б) створюють коментарі до коду;
- В) пишуть документацію до програми;
- Г) використовують стандарти написання коду;
- Д) усі відповіді правильні.

15. Зворотний польський запис найкраще організувати з використанням такої структури даних:

- А) стек;
- Б) черга;
- В) двосторонній список;
- Г) масив;
- Д) немає правильної відповіді.

16. Який тип структур використовують компілятори для передавання параметрів під час виклику функцій?

- А) стек;
- Б) черга;
- В) двосторонній список;
- Г) масив;
- Д) немає правильної відповіді.

17. Лінійний список, в якому доступний лише останній елемент, називається:

- А) стеком;
- Б) чергою;
- В) деком;
- Г) масивом;
- Д) немає правильної відповіді.

18. Оберіть правильний опис конструкцій відповідно до стандартів написання коду:

- А) int g\_count;
- Б) void MyFunction();
- В) char\* pWord;
- Г) # define ELEMENTCOUNT 5;
- Д) немає правильної відповіді.

19. Для коду оберіть правильні твердження

```
void add()
{
    unsigned int Count;
    for (iCount = 7; iCount > 0; iCount--)
    {
        cout<<iCount;
    }
}
```

- А) цикл буде виконуватись нескінченну кількість разів;
- Б) некоректна назва змінної iCount;
- В) некоректна назва функції;
- Г) неправильне форматування;
- Д) усі відповіді правильні.

20. Властивістю алгоритмів є:

- А) двійковість;
- Б) декретність;
- В) детермінованість;
- Г) дуалістичність;
- Д) немає правильної відповіді.

21. Архітектура ПЗ передбачає формування:

- А) структурного вигляду програми;
- Б) поведінкового вигляду програми;
- В) фізичного вигляду програми;
- Г) логічного вигляду програми;
- Д) усі відповіді правильні.

22. Патерн проектування — це:

- А) фрагмент програмного коду програми;
- Б) план робіт проекту;
- В) схема розподілу компонентів системи за вузлами;
- Г) строгий опис поведінки програми;
- Д) немає правильної відповіді.

23. Недоліками угорської нотації є:

- А) іноді важко визначити, де префікс, а де назва змінної;
- Б) при зміні типу змінної необхідно змінити її назву;
- В) у ПЗ, що має можливість автодокументації коду, змінні будуть сортуватись по префіксу і тяжче буде їх шукати;
- Г) використання префіксів збільшує назву змінної і погіршує іноді читабельність коду;
- Д) усі відповіді правильні.

24. Перевагами використання Resharper є:

- А) дозволяє згенерувати автоматично конструктори, перевизначені методи;
- Б) дає можливість легкої зміни назв змінних;
- В) забезпечує швидкий пошук за проектом;
- Г) дає змогу відформатувати та впорядкувати код;
- Д) усі відповіді правильні.

25. Результатами етапу проєктування є:

- А) зображення компонент ПЗ і взаємозв'язків між ними;
- Б) вибір алгоритмів реалізації деякої функціональності;
- В) вибір шаблонів ПЗ для розв'язання певних підзадач;
- Г) вибір і обґрунтування структур даних для реалізації ПЗ;
- Д) усі відповіді правильні.

26. Проєктування “top-level design” має завдання:

- А) визначити поведінку системи;
- Б) визначити структуру системи;
- В) визначити алгоритми розв'язання задачі;
- Г) розробити графік робіт проекту;
- Д) немає правильної відповіді.

27. Проєктування належить до категорії/групи процесів:

- А) допоміжні;
- Б) інженерні основні;
- В) інженерні неосновні;
- Г) замовник-постачальник;
- Д) немає правильної відповіді.

28. Проєктування “software detailed design” має завданням:

- А) визначити поведінку системи;
- Б) визначити структуру системи;
- В) визначити алгоритми розв'язання задачі;
- Г) розробити графік робіт проекту;
- Д) немає правильної відповіді.

29. Факторинг коду - це:

- А) покращення програмного коду;
- Б) створення програмного коду;
- В) доповнення коду коментарями;
- Г) інтеграція модулів системи;
- Д) немає правильної відповіді.

30. Факторинг коду - це:
- А) створення каркаса програмного коду;
  - Б) розбиття коду на складові;
  - В) об'єднання складових в одну систему;
  - Г) реалізація окремих модулів;
  - Д) немає правильної відповіді.
31. Який етап ЖЦ ПЗ є найпростішим з погляду застосування технологій:
- А) проектування;
  - Б) тестування;
  - В) розроблення;
  - Г) документування;
  - Д) немає правильної відповіді.
32. Архітектура ПЗ передбачає формування:
- А) плану робіт;
  - Б) команди розробників;
  - В) програмного коду;
  - Г) специфікації вимог;
  - Д) немає правильної відповіді.
33. Рефакторинг коду - це:
- А) створення каркаса програмного коду;
  - Б) розбиття коду на складові частини;
  - В) об'єднання складових в одну систему;
  - Г) відлагодження окремих модулів;
  - Д) немає правильної відповіді.
34. Рефакторинг коду - це
- А) покращення програмного коду;
  - Б) створення програмного коду;
  - В) перевірка програмного коду на оптимальність;
  - Г) інтеграція модулів системи;
  - Д) немає правильної відповіді.

35. Структуру програмної системи можна описати:

- А) блок-схемою алгоритму роботи програми;
- Б) інструкцією користувача;
- В) прототипом інтерфейсу користувача;
- Г) складовими модулями;
- Д) усі відповіді правильні.

### **III. Тести до лабораторної роботи № 3**

1. Яке тестування полягає в перевірці коду без виконання програми:

- А) статичне;
- Б) динамічне;
- В) системне;
- Г) бета-тестування;
- Д) немає правильної відповіді.

2. Яке тестування передбачає внесення дрібних помилок у програмний код:

- А) системне;
- Б) навантажувальне;
- В) динамічне;
- Г) мутаційне;
- Д) усі відповіді правильні.

3. Яка складова якості визначає міру використання системних ресурсів:

- А) надійність;
- Б) живучість;
- В) ефективність;
- Г) практичність;
- Д) немає правильної відповіді.

4. Яке тестування спрямоване на перевірку безпеки та зручності використання:

- А) димове;
- Б) нефункціональне;
- В) бета-тестування;
- Г) санітарне;
- Д) немає правильної відповіді.

5. Яке тестування проводять “зовнішні” користувачі:
- А) димове;
  - Б) регресійне;
  - В) бета-тестування;
  - Г) функціональне;
  - Д) немає правильної відповіді.
6. За рівнем тестування розрізняють:
- А) модульне;
  - Б) системне;
  - В) інтеграційне;
  - Г) приймальне;
  - Д) усі відповіді правильної.
7. Тестування окремих найменших частин програми називають:
- А) модульним;
  - Б) “чорної” скриньки;
  - В) санітарним;
  - Г) альфа-тестуванням;
  - Д) усі відповіді правильної.
8. Нехай програма приймає вхідне значення - оцінку у стобалльній шкалі (0-100). Яке значення належатиме до того самого класу еквівалентності, що і значення 53?
- А)-5;
  - Б)-2;
  - В)102;
  - Г) 110;
  - Д) немає правильної відповіді.
9. Нехай програма приймає булівське значення. Скільки і яких є класів еквівалентності?
- А) два правильної і два неправильні;
  - Б) один правильної і один неправильний;
  - В) два правильної і один неправильний;
  - Г) два неправильні;
  - Д) немає правильної відповіді.

10. Якщо розробник щойно взявся за виправлення дефекту, то дефект матиме статус:
- A) New;
  - B) In Progress;
  - C) Deferred;
  - D) Verified;
  - D) немає правильної відповіді.
11. Яке тестування передбачає побудову графу управління?
- A) мутаційне;
  - B) димове;
  - C) бета-тестування;
  - D) “чорної” скриньки;
  - D) немає правильної відповіді.
12. Доступ до вихідного коду програми передбачається у:
- A) структурному тестуванні;
  - B) мутаційному тестуванні;
  - C) тестуванні на основі потоку даних програми;
  - D) тестуванні “скляної” скриньки;
  - D) усі відповіді правильні.
13. Нехай str-рядок символів. Що з переліченого є нульовим прикладом?
- A) 0;
  - B) ‘0’;
  - C) “ ”;
  - D) “000”;
  - D) усі відповіді правильні.
14. Нехай програма приймає натуральне число від 0 до 100. На яких даних перевіряють екстремальні умови?
- A) -5;
  - B) 1;
  - C) 95;
  - D) 103;
  - D) немає правильної відповіді.

15. Усунення синтаксичних помилок - це:

- А) відлагодження;
- Б) верифікація;
- В) валідація;
- Г) тестування;
- Д) немає правильної відповіді.

16. Складовою якості є:

- А) надійність;
- Б) адаптованість;
- В) живучість;
- Г) цілісність;
- Д) усі відповіді правильні.

17. Яке з цих видів тестування проводиться першим:

- А) димове;
- Б) регресійне;
- В) санітарне;
- Г) альфа-тестування;
- Д) усі відповіді правильні.

18. Яке тестування проводиться перед здаванням в експлуатацію:

- А) модульне;
- Б) системне;
- В) інтеграційне;
- Г) приймальне;
- Д) усі відповіді правильні.

19. Нехай у плані будинку передбачено, що балкон виступає всередину, а не назовні, і реальний будинок відповідає плану. Що він не проходить?

- А) димове тестування;
- Б) відлагодження;
- В) верифікацію;
- Г) валідацію;
- Д) усі відповіді правильні.

20. Нехай програма очікує натуральне число від 1 до 5. Яке значення належить тому самому класу еквівалентності, що і 7?

- А) 0;
- Б) 1;
- В) 3;
- Г) 6;
- Д) немає правильної відповіді.

21. Оберіть правильні твердження:

- А) основна задача етапу тестування - пошук помилок;
- Б) тестування - ітераційний процес;
- В) процес тестування варто розпочинати якомога швидше;
- Г) статичне тестування - перегляд написаного коду;
- Д) усі відповіді правильні.

22. Оберіть правильні твердження:

- А) якісне тестування однозначно доводить відсутність помилок;
- Б) про тестування слід думати лише, коли завершений етап кодування;
- В) тестування - це те саме, що відлагодження;
- Г) важливою є не стільки кількість, скільки якість підібраних тестових наборів;
- Д) усі відповіді правильні.

23. Оберіть правильні твердження:

- А) якість - складова частина надійності;
- Б) ймовірність виправлення помилки залежить від того, наскільки добре вона задокументована;
- В) створюючи нову помилку, тестувальник присвоює їй статус fixed;
- Г) процес відлагодження виконується тестувальниками;
- Д) усі відповіді правильні.

24. Яке з видів тестування належить до структурного тестування?

- А) функціональне;
- Б) мутаційне;
- В) димове;
- Г) бета-тестування;
- Д) немає правильної відповіді.

25. Використання даних, значення яких знаходяться за межами допустимої області змін, передбачене у:
- А) перевірці нормальних умов;
  - Б) перевірці у виняткових ситуаціях;
  - В) перевірці екстремальних умов;
  - Г) перевірці граничних умов;
  - Д) немає правильної відповіді.
26. Інтеграційне тестування має на меті:
- А) перевірку правильності роботи окремих компонент системи;
  - Б) перевірку правильності роботи системи загалом;
  - В) перевірку правильності інтерфейсу програми;
  - Г) перевірку правильності обчислень інтегралів;
  - Д) немає правильної відповіді.
27. Тестування належить до категорії/групи процесів:
- А) допоміжні;
  - Б) інженерні основні;
  - В) інженерні неосновні;
  - Г) замовник-постачальник;
  - Д) немає правильної відповіді.
28. Тестування перевіряє:
- А) синтаксис програми;
  - Б) стилістику програми;
  - В) семантику програми;
  - Г) сингуляцію програми;
  - Д) усі відповіді правильні.
29. Тестування перевіряє, чи програма:
- А) компілюється;
  - Б) конвертується;
  - В) конструктується;
  - Г) корелюється;
  - Д) немає правильної відповіді.

30. Тестування й специфікація вимог - це:

- А) незалежні етапи;
- Б) такі, що один етап використовує результати іншого;
- В) такі, що використовують результати один одного;
- Г) інтегровані один в інший;
- Д) немає правильної відповіді.

31. Інтегральне тестування має на меті:

- А) перевірку правильності роботи окремих компонент системи;
- Б) перевірку правильності роботи системи загалом;
- В) перевірку правильності інтерфейсу програми;
- Г) перевірку правильності обчислень інтегралів;
- Д) немає правильної відповіді.

32. Якого тестування не існує:

- А) мутаційного;
- Б) регресійного;
- В) димового;
- Г)автоматичного;
- Д) немає правильної відповіді.

33. Перевірка програми, що опрацьовує список студентів, для випадку, коли відсутній список, є:

- А) перевірка нормальних умов;
- Б) перевірка у виняткових ситуаціях;
- В) перевірка екстремальних умов;
- Г) перевірка граничних умов;
- Д) немає правильної відповіді.

34. Ступінь важливості помилки у звіті може задаватися як:

- А)середній;
- Б)важливий;
- В) об'єктивний;
- Г) основний;
- Д) усі відповіді правильні.

35. Тип звіту у звіті про тестування може задаватися як:

- А) помилка замовника;
- Б) помилка проектування;
- В) помилка тестування;
- Г) помилка експлуатації;
- Д) усі відповіді правильні.

#### IV. Тести до лабораторної роботи № 4

1. Нехай на графічному інтерфейсі користувача потрібно ввести рік (від 1900 до поточного значення). Який з перелічених графічних елементів є найбільш доцільним?

- А) текстове поле;
- Б) спінбокс;
- В) багаторядкове текстове поле;
- Г) група радіокнопок;
- Д) усі відповіді правильні.

2. Як відомо, коли абонент перебуває поза зоною досяжності, останньою фразою, яку чуємо, є пропозиція зателефонувати пізніше. Який принцип реалізується:

- А) близькість до реального світу;
- Б) сповіщення про поточний стан;
- В) управління свободою дій користувача;
- Г) допомога користувачам у розпізнаванні, діагностиці та усуненні помилок;
- Д) усі відповіді правильні.

3. Якщо команди розташовані в порядку “Видалити”, “Відкрити”, “Створити”, “Редагувати”, то порушений принцип:

- А) близькість до реального світу;
- Б) гнуучкість і ефективність;
- В) управління свободою дій користувача;
- Г) запобігання помилкам;
- Д) немає правильної відповіді.

4. Команда Ctrl-Z є прикладом реалізації принципу:

- А) близькість до реального світу;
- Б) розпізнавання, а не згадування;
- В) управління свободою дій користувача;
- Г) цілісність і стандарти;
- Д) немає правильної відповіді.

5. Нехай є список прізвищ, відсортованих за алфавітом, і першим у списку є прізвище “Андрієнко”. Користувач змінив у цьому списку прізвище “Микитен” на “Микитин” і відправив форму на сервер, після чого виділенім стало прізвище “Андрієнко”. Який принцип порушений?

- А) цілісність і стандарти;
- Б) розпізнавання, а не згадування;
- В) допомага користувачам в розпізнаванні, діагностиці та усуненні помилок;
- Г) гнучкість і ефективність використання;
- Д) немає правильної відповіді.

6. Якщо успішність або неуспішність операцій показано людськими емоціями, це приклад реалізації принципу:

- А) близькість до реального світу;
- Б) розпізнавання, а не згадування;
- В) управління свободою дій користувача;
- Г) цілісність і стандарти;
- Д) немає правильної відповіді.

7. Якщо на веб-сторінці країни Європи показані випадним списком, це приклад реалізації принципу:

- А) близькість до реального світу;
- Б) розпізнавання, а не згадування;
- В) управління свободою дій користувача;
- Г) допомага користувачам в розпізнаванні, діагностиці та усуненні помилок;
- Д) немає правильної відповіді.

8. За допомогою якого графічного елемента найкраще ввести довільне число від 0 до 9999?

- А) текстового поля, в яке неможливо ввести не-цифри;
- Б) спінбокса;
- В) випадного списку;
- Г) простого текстового поля;
- Д) усі відповіді правильні.

9. Пісочний годинник асоціюється з принципом:

- А) допомога користувачам у розпізнаванні, діагностиці та усуненні помилок;
- Б) мінімалістичний і ефективний дизайн;
- В) сповіщення про поточний стан;
- Г) запобігання помилок;
- Д) усі відповіді правильні.

10. Інструментом для побудови прототипу графічного інтерфейсу користувача є:

- А) олівець і папір;
- Б) Axure;
- В) Balsamic;
- Г) MS PowerPoint;
- Д) усі відповіді правильні.

11. Відомо, що Word дає змогу переглянути останні відкриті файли. Який принцип реалізовано?

- А) ефективний і мінімально необхідний дизайн;
- Б) сповіщення про поточний стан;
- В) близькість до реального світу;
- Г) розпізнавання, а не згадування;
- Д) усі відповіді правильні.

12. Наявність режиму статистики у програмі “Калькулятор” є прикладом реалізації принципу:

- А) естетичний і мінімально необхідний дизайн;
- Б) сповіщення про поточний стан;

- В) гнучкість і ефективність використання;
- Г) розпізнавання, а не згадування;
- Д) усі відповіді правильні.

13. Інструкція користувача до програми для роботи зі списком студентів з лабораторної роботи до дисципліни “Основи програмування” пишеться у стилі:

- А) довідника;
- Б) посібника;
- В) енциклопедії;
- Г) новели;
- Д) усі відповіді правильні.

14. Відсутність інформаційного шуму є змістом принципу:

- А) близькість до реального світу;
- Б) розпізнавання, а не згадування;
- В) управління свободою дій користувача;
- Г) цілісність і стандарти;
- Д) немає правильної відповіді.

15. Нехай натиск піктограми зі скріпкою веде до відкриття довідки про програму. Який принцип порушений?

- А) гнучкість і ефективність використання;
- Б) запобігання помилок;
- В) управління свободою дій користувача;
- Г) естетичний і мінімально необхідний дизайн;
- Д) немає правильної відповіді.

16. Наявність діалогу “Налаштування” у складному програмному продукті є прикладом реалізації принципу:

- А) близькість до реального світу;
- Б) розпізнавання, а не згадування;
- В) управління свободою дій користувача;
- Г) гнучкість і ефективність використання;
- Д) немає правильної відповіді.

17. Використання піктограм, які нагадують дорожні знаки, і кольорів світлофору для привертання уваги користувача є прикладом реалізації принципу:

- А) близькість до реального світу;
- Б) сповіщення про поточний стан;
- В) управління свободою дій користувача;
- Г) гнучкість і ефективність використання;
- Д) немає правильної відповіді.

18. Користувач додає елемент у список. Як правильно показати успішне додавання?

- А) відобразити повідомлення “Новий елемент успішно доданий”;
- Б) показати оновлений список з щойно доданим елементом у ньому;
- В) показати оновлений список з виділенням новим елементом у ньому і повідомлення “Новий елемент успішно доданий”;
- Г) нічого не змінювати на екрані;
- Д) немає правильної відповіді.

19. Можливість виконати одну і ту саму дію декількома способами є змістом принципу:

- А) естетичний і мінімально необхідний дизайн;
- Б) розпізнавання, а не згадування;
- В) гнучкість і ефективність використання;
- Г) допомога і документація;
- Д) усі відповіді правильні.

20. Нехай користувач заповнює форму, що складається з двох сторінок, і на кожній сторінці потрібно ввести дату народження. Який принцип порушений?

- А) близькість до реального світу;
- Б) сповіщення про поточний стан;
- В) запобігання помилок;
- Г) естетичний і мінімально необхідний дизайн;
- Д) усі відповіді правильні.

21. Яка піктограма використовується для повідомлення-попередження:

- А) знак питання;

- Б) знак оклику;
- В) знак стоп;
- Г) решітка;
- Д) немає правильної відповіді.

22. Яка піктограма використовується для повідомлення про помилку:

- А) знак питання;
- Б) знак оклику;
- В) знак стоп;
- Г) решітка;
- Д) немає правильної відповіді.

23. Повідомлення “***confirmation message***” необхідне для:

- А) попередження про можливу аварійну ситуацію;
- Б) підтвердження дій;
- В) вибору варіанта рішення;
- Г) інформування користувача;
- Д) немає правильної відповіді.

24. Повідомлення “***notification message***” необхідне для:

- А) попередження про можливу аварійну ситуацію;
- Б) підтвердження дій;
- В) вибору варіанту рішення;
- Г) інформування користувача;
- Д) немає правильної відповіді.

25. Яку піктограму використовують для прикріплених файлів?

- А) конверт;
- Б) знак додавання;
- В) скріпка;
- Г) цвях;
- Д) немає правильної відповіді.

26. Етап супроводу ПЗ передбачає:

- А) виправлення знайдених дефектів;
- Б) додавання нового функціоналу;

- В) реорганізацію програмного коду;
- Г) використання програми для виявлення недоліків роботи програми;
- Д) усі відповіді правильні.

27. Супровід належить до категорії/групи процесів:

- А) допоміжні;
- Б) інженерні основні;
- В) інженерні неосновні;
- Г) замовник-постачальник;
- Д) немає правильної відповіді.

28. Зручність користування програмою - це:

- А) основна функціональна вимога до будь-якого ПЗ;
- Б) складова супроводжуваності ПЗ;
- В) складова ефективності роботи програми;
- Г) складова якості ПЗ;
- Д) усі відповіді правильні.

29. Контекстне меню - це:

- А) верхній рядок меню у програмі;
- Б) підменю, що є складовим головного меню;
- В) меню, що викликається до активного об'єкта;
- Г) меню довідкової системи;
- Д) немає правильної відповіді.

30. Три крапки в кінці назви пункту меню означає:

- А) незавершеність у назві;
- Б) неактивність цього пункту;
- В) появу діалогового меню після виклику;
- Г) появу підменю після виклику;
- Д) немає правильної відповіді.

31. Запис пункту меню з накресленням Bold означає:

- А) неможливість виконання цього пункту;
- Б) обов'язковість виконання цього пункту;
- В) появу діалогового меню після виклику;

Г) появу підменю після виклику;

Д) немає правильної відповіді.

32. Запис пункту меню сірим кольором означає:

А) появу діалогового меню після виклику;

Б) незавершеність у назві;

В) неможливість виконання цього пункту;

Г) появу підменю після виклику;

Д) немає правильної відповіді.

33. Рядок статусу необхідний для:

А) естетичного вигляду;

Б) виконання статусних команд;

В) інтерактивності;

Г) інформативності;

Д) немає правильної відповіді.

34. Ярлик до програми - це:

А) те саме, що піктограма;

Б) спеціальне меню;

В) окремий пункт меню;

Г) спеціальна папка;

Д) немає правильної відповіді.

35. Якісна характеристика ПЗ супроводжуваність означає, що:

А) програму можна зручно використовувати;

Б) у програмі немає дефектів;

В) програму можна легко модифікувати;

Г) програму можна використовувати на різних платформах;

Д) усі відповіді правильні.

# **САМОСТІЙНА РОБОТА**

## **у межах розділу “Життєвий цикл програмного забезпечення”**

### **Загальні положення**

Самостійна робота студента є обов'язковою складовою навчального процесу.

Положенням про організацію навчального процесу у вищих навчальних закладах України передбачено, що навчальний час, відведений для самостійної роботи студентів, визначається робочим навчальним планом спеціальності і повинен становити не менше ніж 1/3 й не більше ніж 2/3 від загального обсягу навчального часу, відведеного студенту для вивчення конкретної навчальної дисципліни. Самостійна робота студента є основним засобом засвоєння ним навчального матеріалу в час, вільний від обов'язкових навчальних занять.

Завданням самостійної роботи студентів є:

- навчити студентів самостійно працювати з інформаційними джерелами (літературою, програмним забезпеченням, інтернет-ресурсами);
- творчо сприймати навчальний матеріал і осмислювати його;
- сформувати навички щоденної самостійної роботи з метою одержання та узагальнення знань, умінь і навичок.

Для опрацювання розділу “Життєвий цикл програмного забезпечення” рекомендують самостійну роботу в обсязі 60 годин:

- вивчення теоретичного матеріалу розділу - 28 год.
- підготовка до лабораторних робіт - 18 год.
- виконання практичних завдань за темами самостійної роботи - 14 год.

Теми самостійної роботи для виконання практичних завдань пов'язані з основами роботи з типовим ПЗ масового використання. Метою виконання цих робіт є повторення й поглиблення знань, удосконалення вмінь, отриманих у межах шкільного курсу інформатики.

Контролюють самостійну роботу за допомогою опитування під час захисту лабораторних робіт, виконання яких вимагає застосування знань, умінь і навичок, набутих після виконання самостійної роботи. Для виконання лабораторних робіт із розділу студентам необхідні знання й вміння роботи в середовищі ОС Windows, з офісними програмами для організації роботи, створення й опрацювання електронної документації.

## **Послідовність виконання практичного завдання самостійної роботи**

1. Ознайомитися з темою й короткими теоретичними відомостями.
2. Відшукати й відібрати необхідні інформаційні джерела за темою роботи.
3. Опрацювати відібрані інформаційні джерела.
4. Використати необхідне програмне забезпечення для виконання завдань.
5. Підготувати відповіді на питання з відповідного переліку.

### **Рекомендовані обсяги годин для виконання практичних завдань самостійної роботи**

Перелік тем самостійної роботи	Години	Номери лабораторних робіт, що ґрунтуються на самостійній роботі
Тема 1	4	1,2, 3, 4
Тема 2	2	1,2, 3, 4
Тема 3	6	1, 2, 3, 4
Тема 4	6	2, 3, 4
Разом	18	

# Самостійна робота № 1

## ТЕМА. ОСНОВИ РОБОТИ В ОС WINDOWS

### Короткі теоретичні відомості

Призначення операційних систем - надати користувачеві засоби управління обчислювальними ресурсами комп'ютера й інформацією. Операційні системи (ОС) можуть відрізнятися особливостями реалізації внутрішніх алгоритмів керування основними ресурсами комп'ютера (процесорами, пам'ятю, пристроями), особливостями використаних методів проектування, типами апаратних платформ, областями використання та багатьма іншими властивостями.



Рис. 1. Схематичне зображення складових ОС

Архітектура сучасних ОС передбачає існування ядра ОС та її оболонки. Роль оболонки - забезпечити інтерфейс користувача, тобто зручні засоби зв'язку користувача зі системою.

Передісторією системи Windows є виконання функцій графічного інтерфейсу системи DOS (Disk Operating System - дискова операційна система). Вбудованою оболонкою в MS-DOS є командний процесор.

Компонентами ядра сучасних ОС є набір драйверів пристройів, програма управління пам'ятю, програма управління файлами, планувальник і диспетчер.

Для комп'ютерів використовують різні за технологією ОС: **UNIX-системи** (AIX, HP-UX, IRIX, Mac OS X, SCO OpenServer, Solaris, Tru64, z/OS); **POSIX або UNIX-подібні** (FreeBSD, OpenBSD, NetBSD, OpenSolaris, BeleniX, Nexenta, Linux) й **Windows**.

Найпоширеніші операційні системи для мобільних платформ: Android OS, Apple iOS, Symbian OS, Windows Mobile, RIM BlackBerry, PalmOS.

Сьогодні серед найпопулярніших ОС для кінцевих користувачів комп'ютерів є сімейство продуктів Microsoft Windows, які займають основну частку ринку операційних систем. Компанія Microsoft почала роботу над операційною системою з графічним інтерфейсом у 1981 році. Першу версію Windows випущено в 1985 році. Вона мала значно менші можливості, ніж Mac

OS, і до випуску в 1990-му Windows 3.0 була не дуже відомою. Ситуація почала змінюватися з виходом версій 3.1 і 3.11, а вдала маркетингова кампанія в середині 90-х років привела до різкого зростання популярності Windows 95 і подальших продуктів цього сімейства як для корпоративних робочих станцій, так і для домашніх комп'ютерів.

**WINDOWS** - це багатозадачна ОС з графічним інтерфейсом користувача. Простий і зручний інтерфейс системи забезпечує гармонійне спілкування користувача з комп'ютером. У системі підтримується складна високорівнева графіка - графічний інтерфейс пристрій (GDI). Характерна наявність стандартного набору об'єктів користувачького інтерфейсу, таких як вікна, меню, піктограми - інтерфейс прикладних програм (API).

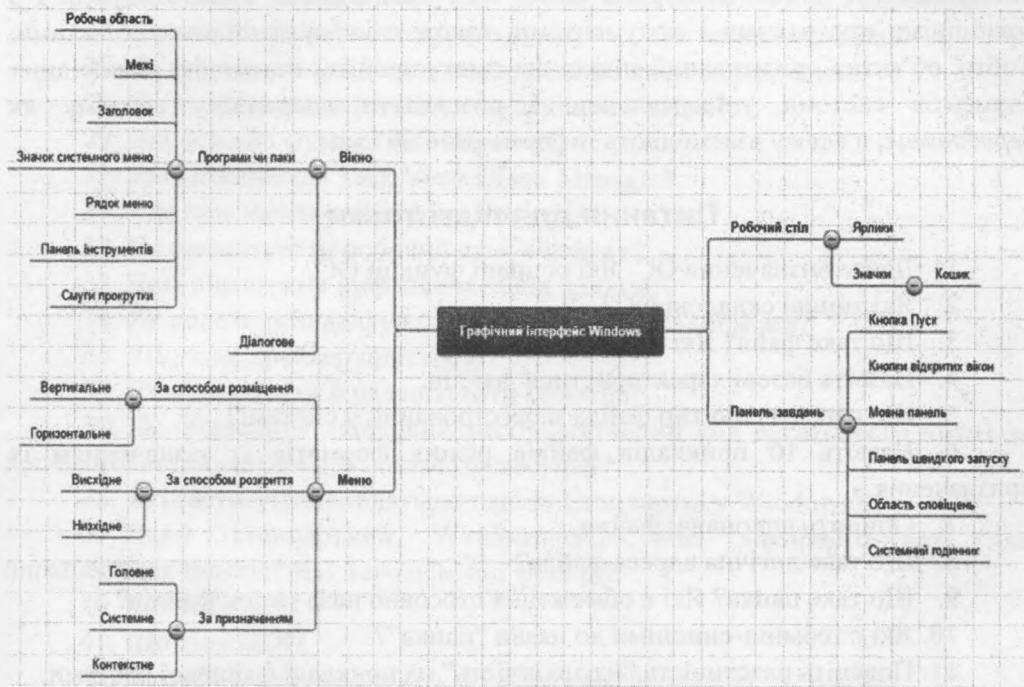


Рис. 2. Типові складові графічного інтерфейсу користувача ОС Windows

Система має широкий спектр можливостей, що дозволяє налагодити інтерфейс так, як подобається користувачу. Головна особливість інтерфейсу - наявність панелі задач, на якій розміщено кнопку **Start**. Переміщення

об'єктів виконується просто й зручно за принципом Drag and Drop - “перетягни і покинь”.

Windows є об'єктно-орієнтованою операційною системою, тобто все, з чим операє система, трактується як інформаційний об'єкт. Він має певні властивості, тобто адекватно відображає об'єкти реального світу. Виконання програм здійснюється звичним для людини способом як взаємодія об'єктів.

Інформаційними об'єктами є прикладні програми Windows (Application), які працюють під управлінням Windows. За допомогою прикладних програм створюються документи, які теж є інформаційними об'єктами Windows. Апаратні засоби також трактуються як інформаційні об'єкти, оскільки взаємодія ОС з програмними і апаратними засобами відбувається на загальних засадах, універсальним чином. Програми самої ОС, взаємодіючи одна з однією і з зовнішніми програмами і документами, також є інформаційними об'єктами. Тобто, об'єктно-орієнтований підхід дає змогу зробити взаємодію різномірних елементів системи універсальною і розглядати операційну систему як **середовище**, в якому взаємодіють інформаційні об'єкти.

## Питання для підготовки

1. Дайте визначення ОС. Які основні функції ОС?
2. Які типові складові ОС?
3. Що таке файл? Які є типи файлів?
4. Назвіть базові характеристики файлів.
5. Що означає, що тип файла зареєстрований у системі?
6. Назвіть 10 прикладів файлів різних форматів із зазначенням їх призначення.
7. Опишіть виконавчі файли.
8. Що таке логічна адреса файла?
9. Що таке папка? Які є обмеження стосовно назв папок/файлів?
10. Які є терміни-синоніми до назви “папка”?
11. Поясніть властивість “ієрархічність” на прикладі файлової системи.
12. Що таке логічний диск комп'ютера? Що таке фізичний диск комп'ютера?
13. Які є папки спеціального призначення, що використовуються ОС Windows?
14. Що є складовими інтерфейсу ОС Windows?

15. Що таке ярлик? Як його створити на робочому столі?
16. Як налаштувати панель задач? Продемонструйте.
17. З яких ділянок складається панель задач і яке їх призначення?
18. Що таке вікно і меню?
19. Які складові вікна?
20. Які є типи вікон?
21. Які є типи меню?
22. Що таке панель інструментів?
23. Як знайти найбільший/найновіший файл у заданій папці серед великої кількості інших файлів?
24. Які є режими відображення вмістимого папки?
25. Як знайти файл на диску з ім'ям, що складається з п'яти символів?
26. Як знайти файл на диску, що був створений минулого місяця?
27. Як групувати файли для операцій копіювання/видалення?
28. Як впорядкувати розміщення декількох вікон?
29. Які основні функції File Explorer?
30. Які можливості Task View і Task Manager?
31. Яке призначення Windows Settings?
32. Як налаштувати робочий стіл Windows?
33. Чим піктограма відрізняється від ярлика?
34. Як додати розкладку клавіатури для польської мови?
35. Що таке драйвер пристрою?
36. Як підключити й налаштувати принтер?
37. Як обчислити двійковий/шістнадцятковий код десяткового числа за допомогою стандартного Windows-застосунку?
38. Які застосунки входять до пакета Стандартні у Windows?
39. Який стандартний Windows-застосунок використовують для опрацювання текстів? Які його основні функції?
40. Що таке API?
41. Що таке GDI?
42. Чому Windows вважають об'єктно-орієнтованою системою?
43. У чому полягає принцип Drag and Drop?
44. Назвіть основні версії ОС Windows?
45. Які є інші, крім Windows, ОС?
46. Проаналізуйте схожість і відмінність інтерфейсів користувача ОС Windows і ОС для мобільного телефону.

47. Яка історія розвитку ОС сімейства Windows.
48. Яка передісторія створення ОС Windows?
49. Як реалізовано багатозадачність ОС Windows?
50. Як реалізовано багатокористувацький режим в ОС Windows?
51. Які, на вашу думку, недоліки інтерфейсу ОС Windows.
52. Порівняйте відмінності інтерфейсу ОС Windows найновішої й попередніх версій.
53. Назвіть популярні інтернет-оглядачі, що використовуються під Windows. Обґрунтуйте власний вибір програми такої категорії.
54. Що таке Cloud-технології?
55. Як хмарні технології використовуються у роботі Windows-застосунків?
56. Які програми називають файловими менеджерами? Наведіть приклади.
57. Продемонструйте операції з файлами й папками в одному з обраних файлових менеджерів.
58. Які переваги й недоліки використання файлових менеджерів?

## Самостійна робота № 2

### ТЕМА 2. ОФІСНИЙ ПАКЕТ ОС WINDOWS

#### Короткі теоретичні відомості

Серед розмаїття програмного забезпечення персональних комп'ютерів існує поняття “Офісний пакет” (англ. Office suite), що означає набір застосунків, призначених для роботи з електронною документацією. Компоненти офісних пакетів розповсюджуються, як правило, разом, мають одинаковий типовий інтерфейс і добре розвинену схему взаємодії. Переважно офісний пакет містить такий набір компонент: текстовий процесор, табличний редактор, програма для створення презентацій, система управління базами даних, редактор формул, графічний редактор та ін.

Microsoft Office - це найпопулярніший пакет офісних програм, який дає зможу вирішувати безліч завдань з широкого спектра найрізноманітніших областей.

Microsoft Office - офісний пакет, створений корпорацією Microsoft для операційних систем Microsoft Windows, Apple Mac OS X і Apple iOS (на iPad). До цього пакета входить програмне забезпечення для роботи з різними типами документів: текстами, електронними таблицями, презентаціями, базами даних, схемами тощо.

Microsoft Office поставляють у декількох редакціях, відмінності між якими у складі пакета є відповідно в ціні. Найповніша з редакцій містить такі програмні продукти:

Назва	Призначення	Короткий опис
1	2	3
<b>MS-Word</b>	текстовий процесор	Доступний під Windows і Mac OS X. Дає змогу готувати документи різної складності. Продукт займає провідне місце на ринку текстових процесорів, і його формати використовуються як стандарт у документообігу більшості підприємств. Головні конкуренти - OpenOffice.org Writer, StarOffice Writer, Corel WordPerfect і Apple Pages (тільки на платформі Mac OS).
<b>MS-Excel</b>	табличний процесор	Підтримує всі необхідні функції для створення електронних таблиць будь-якої складності, засоби для графічного представлення даних й відповідних обчислень. Займає провідне положення на ринку. Головні конкуренти - OpenOffice.org Calc, StarOffice, Gnumeric і Corel Quattro Pro.

**Продовження таблиці**

1	2	3
<b>MS-Outlook</b>	персональний комунікатор	До складу Outlook входять: календар, планувальник завдань, записи, менеджер електронної пошти, адресна книга. Підтримується спільна мережева робота. Головні конкуренти поштового клієнта - Mozilla Thunderbird/SeaMonkey, Eudora Mail, The Bat!. Головні конкуренти диспетчера персональних даних - Mozilla, Lotus Organizer i Novell Evolution. Доступний під Windows. Еквівалент для Apple Mac OS X - Microsoft Entourage.
<b>MS-PowerPoint</b>	застосунок для підготовки презентацій	Доступний під Microsoft Windows і Apple Mac OS X. Можливості програми визначаються величезною кількістю користувачьких шаблонів, анімаций, смарт-фігур, зручною організацією робочого простору. Головні конкуренти - OpenOffice.org Impress, Corel WordPerfect і Apple Keynote.
<b>MS-Access</b>	система управління базами даних	Має широкий спектр функцій, зокрема зв'язані запити, сортування за різними полями, зв'язок із зовнішніми таблицями і базами даних. Завдяки вбудованій мові VBA в самому Access можна писати підпрограми, що працюють з базами даних. Дає змогу створювати не лише класичні бази даних, а й зручні веб-програми для роботи з базами даних, які полегшують ведення бізнесу. Дані автоматично зберігаються в базі даних SQL, тому вони надійно захищені, а програми можна легко використовувати спільно з колегами.
<b>MS-InfoPath</b>	застосунок збирання даних і управління	Інструмент для створення форм і збирання даних, який допомагає організаціям оптимізувати бізнес-процеси. Програму призначено для досвідчених бізнес-користувачів і розробників. Не пишучи програмний код, користувачі можуть створювати складні електронні форми, щоб швидко й ефективно збирати інформацію. Розробники можуть створювати просунуті форми для відомих і корпоративних бізнес-процесів, а також складені програми та послідовності робочих процесів.
<b>MS-Communicator</b>	організація всеобщого спілкування між людьми	Забезпечує можливість спілкування за допомогою простого обміну миттевими повідомленнями, а також проведення голосової і відеоботесіди. Допомагає користувачам значно збільшити продуктивність роботи, використовувати застосунки спільно з колегами, які мають доступ до Інтернету, будь-де.
<b>MS-Publisher</b>	застосунок для підготовки публікацій	Видавнича система містить бібліотеку з сотень шаблонів оформлення і публікацій, зокрема інформаційні бюллетені, брошури, рекламні листівки, листівки, веб-вузли, формати поштових повідомлень і багато іншого.

1	2	3
<i><b>MS-Visio</b></i>	застосунок для роботи з бізнес-діаграмами і технічними діаграмами	Універсальний креслярський засіб, який може бути корисний кожному, кому потрібно створити діаграму для формалізації і передавання інформації про процеси, інфраструктуру і застосунки. Дає змогу перетворювати концепції і звичайні бізнес-дані на діаграми.
<i><b>MS-Project</b></i>	управління проектами	Система управління проектами та портфелями проектів містить інтегровані засоби планування, відстежування й контролю проектів. Створює розклади критичного шляху, які можуть бути складені з урахуванням використовуваних ресурсів. Ланцюжок візуалізується в діаграмі Ганта.
<i><b>MS-Query</b></i>	перегляд і відбір інформації з баз даних	Можна отримати дані з корпоративних баз даних і файлів, тому дані, які потрібно проаналізувати в Excel, не потрібно вводити повторно. Також можна автоматично оновлювати звіти та зведення Excel із вихідної бази даних під час кожного введення в неї нової інформації.
<i><b>MS-OneNote</b></i>	застосунок для запису заміток і управління ними	Допомагає організувати особисту інформацію. Найзручнішим є використання програми на планшетному комп'ютері, де є можливість рукописного введення тексту і додавання нотаток. Також можна встановити на пристрої з: Android, Mac, Windows Phone тощо; доступна також веб-версія.
<i><b>MS-Groove</b></i>	застосунок для підтримки спільної роботи.	Рішення для спільної роботи, що забезпечує динамічну й ефективну взаємодію розподілених робочих груп із співробітників, які працюють у різних організаціях і в віддаленому або автономному режимі. Використання робочих областей Groove заощаджує час, збільшує продуктивність і підвищує якість колективної роботи.
<i><b>MS-SharePoint Designer</b></i>	інструмент для побудови застосунків	Настільний застосунок, який дає змогу робити дизайн сторінок SharePoint за допомогою HTML, CSS, XML/XSLT, ASP.NET, налаштовувати бібліотеки і списки, робочі процеси, інтеграцію з зовнішніми системами. Програма для веб-дизайну, заміна для Microsoft Office.
<i><b>MS-Picture Manager</b></i>	робота з малюнками	За допомогою деяких вбудованих функцій можна не просто переглядати, але і редагувати малюнки і фотографії. Можна кадрувати малюнок, тобто відсікти непотрібне, повернути або розгорнути його, полегшити вагу картинки, прибрати ефект “червоних очей”, відрядагувати яскравість і контрастність.
<i><b>MS-Diagnostics</b></i>	діагностика і відновлення пошкоджених застосунків Microsoft Office	Добірка утиліт, призначених для надання адміністраторам допомоги в швидкому відновленні критично важливих робочих станцій під управлінням Microsoft Windows в ситуаціях, коли перевстановлення операційної системи не має сенсу або неможливе.

Microsoft додала інструменти, загальні для всієї лінійки Office:  
о інтелектуальний пошук;  
о контекстний пошук;  
о рукописне введення формул;  
о спільне редагування файлів та інтеграцію з OneDrive.

Microsoft Office містить лінгвістичні засоби для роботи з текстами понад 40 мовами, зокрема українською. Модуль підтримки української мови на замовлення Microsoft розробила українська компанія ProLing Ltd.

Дляожної мови в Microsoft Office передбачено чотири лінгвістичних інструменти: перевірка орфографії (spelling), розставлення переносів (hyphenation), словник синонімів (thesaurus) та перевірка граматики (grammar).

Некомерційне користування програмними продуктами Microsoft дозволяє програма DreamSpark. Це програма компанії Microsoft з підтримки технічної освіти наданням доступу до програмного забезпечення Microsoft для навчальних, викладацьких та дослідницьких цілей.

Тривалий час практично синонімом поняття офісний пакет був Microsoft Office, його успіх у споживачів значною мірою забезпечив і загальний успіх операційної системи Windows. Однак існували альтернативні пропозиції офісних пакетів, наприклад, WordPerfect (його сучасний наступник Corel WordPerfect Office), IBM Lotus SmartSuite чи Ability Office. На платформі Mac компанія Apple випустила свій власний пакет iWork. Співтовариство відкритого програмного забезпечення GNU/Linux розробляє такі офісні проекти, як Gnome Office та Koffice. Відкрита платформа і формат OpenDocument підтримується такими вільними пакетами, як OpenOffice.org та IBM Lotus Symphony, доступними на багатьох операційних системах. Розвиток служб Інтернету привів до нової хвилі повноцінних офісних пакетів, які є переважно онлайновими веб-службами, такими як Zoho Office Suite чи Google Docs. Сьогодні стають популярними мобільні офісні пакети: Polaris Office, Kingsoft Office, Quickoffice і мобільні версії десктопних пакетів.

## Завдання

1. Ознайомитися з основними складовими Microsoft Office.
2. Охарактеризувати спільні властивості інтерфейсу користувача програм пакета Microsoft Office.

3. Продемонструвати вміння роботи в п'яти програмах (на вибір) з пакета Microsoft Office.
4. Продемонструйте взаємодію цих програм пакета Microsoft Office. Що означає інтегрованість пакета програм?
5. Як *Visual Basic for Applications* (VBA) використовується в програмах з пакета Microsoft Office?
6. Порівняйте можливості найновішої й попередньої версій Microsoft Office. Яка основна відмінність між ними?
7. Які існують офісні пакети для мобільних платформ? Порівняйте їхні можливості відносно пакета Microsoft Office.
8. Які можливості некомерційного використання Microsoft Office?
9. Що таке утиліта? Утиліти якого призначення надаються Microsoft Office?
10. Назвіть редакції пакета Microsoft Office. У чому полягають їх відмінності?
11. Проаналізуйте он-лайн аналоги пакета Microsoft Office. Яка принципова відмінність?
12. Охарактеризуйте лінгвістичні компоненти пакета Microsoft Office.
13. Що таке база даних (БД)? Які є типи баз даних?
14. Який стандартний Windows-застосунок з офісного пакета є системою управління базами даних (СУБД)? Який файл зберігає БД, створену в цьому застосунку?
15. Які основні функції СУБД?
16. Які є відомі СУБД?
17. Наведіть приклад використання СУБД для обраної предметної області.
18. Що таке електронна таблиця?
19. Який стандартний Windows-застосунок з офісного пакета призначений для роботи з електронними таблицями? Який файл зберігає електронну таблицю, створену в цьому застосунку?
20. Що таке електронна таблиця?
21. За допомогою чого виконуються обчислення в електронних таблицях?
22. За допомогою чого створюються діаграми й графіки для електронних таблиць?

23. Наведіть приклад використання електронних таблиць для обраної предметної області.
24. Який стандартний Windows-застосунок з офісного пакета призначений для роботи з електронними презентаціями? Який файл зберігає електронну презентацію, створену в цьому застосунку?
25. Що таке шаблон презентації? Які існують шаблони?
26. Що таке макет презентації? Які існують макети?
27. Які є режими демонстрації презентації?
28. Як здійснити звуковий супровід презентації?
29. Наведіть приклад використання електронних презентацій для обраної предметної області.
30. Назвіть аналогічні програмні засоби для створення електронних презентацій.
31. За допомогою якого Windows-застосунку з офісного пакета можна створити макет буклету про спеціальність? Який файл зберігає електронний макет публікації, створений у цьому застосунку?
32. У якому застосунку офісного пакета Windows можна відредагувати фотографію?

## **Самостійна робота № 3**

### **ТЕМА. ПРИЗНАЧЕННЯ Й ОСНОВНІ МОЖЛИВОСТІ ТЕКСТОВОГО ПРОЦЕСОРА MS-WORD**

#### **Короткі теоретичні відомості**

Текстовий процесор - система обробки тексту, призначена для створення, редагування та форматування простих і комплексних текстових документів.

Одним із найзручніших і універсальних текстових редакторів є текстовий процесор Microsoft Word. Документи, які створюються за допомогою Word, зберігаються у файлі з розширенням \*.docx/doc. Зазвичай текстовий процесор відрізняється від текстового редактора додатковими можливостями роботи з документами. Зокрема це робота з графічними елементами, розширений набір засобів для форматування й художнього представлення текстів.

Існує декілька версій Microsoft Word для Windows: кожна наступна версія, як правило, сумісна з попередніми версіями і має додаткові можливості.

Microsoft Word має потужні засоби для адаптації до вимог конкретного користувача - як початківця, так і професіонала. За допомогою цих засобів можна змінити зовнішній вигляд екрана редактора, параметри редагування, перегляду, збереження і друку документів. Ці засоби реалізуються командами меню. Команда Сервіс/ Параметри виводить на екран вікно діалогу для параметрів і налаштувань.

Текстовий процесор Word дає змогу працювати одночасно з кількома документами. Кожен з документів розміщується в окремому вікні. Для зручності опрацювання декількох файлів використовуються команди впорядкування вікон. З документами можна проводити стандартну обробку даних: набір тексту, редагування, форматування та інше, обмінюватися даними між документами та використовувати дані з інших програм, що входять до пакета Microsoft Office.

Основна мета використання текстового процесора - це створення належно оформленого текстового документа. Текстовий процесор Microsoft Word має дуже широкий спектр можливостей для створення, оформлення та опрацювання документів. До його основних функцій належать:

- організація введення й редагування тексту за допомогою клавіатури та збереження його в пам'яті;

- форматування тексту (оформлення вигляду тексту, зміна його параметрів: розміру, гарнітури шрифтів, накреслення, вирівнювання, кольорових характеристик);
- автоматичне структурування текстів (використання стилей, шаблонів, посилань, маркованих списків);
- опрацювання декількох документів одночасно;
- рецензування текстів (використання приміток, управління виправленнями, статистика);
- попередній перегляд перед друком та друкування документів чи його складових;
- перевірка правопису;
- використання готових графічних зображень у тексті;
- створення рисунків за допомогою стандартних фігур;
- побудова наукових формул, діаграм, схем;
- використання таблиць у тексті;
- використання макросів у документах та ін.

Добре розроблена система допомоги з. прикладами є важливою компонентою, призначеною для детального вивчення можливостей MS-Word.

### **Питання для підготовки**

1. Яка відмінність між текстовим редактором і текстовим процесором? Наведіть приклади.
2. Яке призначення ділянок рядка статусу?
3. Як налаштовувати панель швидкого доступу до команд?
4. Що таке буфер обміну? Коли його використовують?
5. Як дізнатися, зі скількох слів складається документ? Дайте декілька варіантів відповіді.
6. Як дізнатися, зі скількох рядків, абзаців, знаків складається документ?
7. Які є режими перегляду документа?
8. Що таке гарнітура тексту?
9. Які є накреслення тексту?
10. Що таке форматування тексту? Як змінювати параметри форматування?

11. Як встановити абзац у тексті? Що таке післяабзацний відступ?
12. Що таке міжрядковий інтервал? Як його встановлювати?
13. Як вирівнювати текст у документі?
14. Як пронумерувати сторінки? Які є формати нумерування?
15. Що таке колонтитули? Як їх створювати і змінювати?
16. Які можна використовувати параметри кольору для тексту?
17. Як встановлювати розрив сторінки? Навіщо його використовувати?
18. Як створюється фігурний текст?
19. Як змінити написання слова на аналогічне, але зі всіх великих букв, без перенабирання?
  20. Як набрати знак нескінченності, градуса, менше й дорівнює?
  21. Що таке стиль тексту? Як його використовують?
  22. Як розпочинати абзац великою фігурною буквою?
  23. Як упорядковувати списки в тексті? Які маркери для цього використовують?
    24. Які використовують параметри для відображення списків?
    25. Як встановлюють параметри сторінки?
    26. Яка є орієнтація сторінок?
    27. Як перейти на сторінку заданого номеру?
    28. Які є зручні способи гортання документів?
    29. Що таке багатошпалтовий документ?
    30. Що таке автотекст? Як з ним працювати?
    31. Як відсортувати список прізвищ за алфавітом?
    32. Як замінити по всьому тексту одне слово на інше?
    33. Як сформувати автоматично зміст документа?
    34. Як створити примітки до фрагментів тексту?
    35. Як створити закладки до тексту?
    36. За допомогою якого інструменту створюють математичні формули у MS-Word? Наведіть приклад.
    37. Як створюють таблиці у MS-Word? Наведіть приклад.
    38. Як відсортувати рядки таблиці? Які використовуються критерії сортування?
    39. Як обчислити суму чисел, заданих у стовпці таблиці?
    40. Як можна створити найпростіший малюнок у MS-Word? До якого типу графіки (векторна, раstrova) він належить?

41. Як можна побудувати графік за введеними значеннями у MS-Word?
42. Які типи діаграм можна використовувати у MS-Word?
43. Як можна захистити документ від змін?
44. Як вставити в текст фото? Як змінити ширину/висоту зображення?
45. Які є способи обтікання тексту для вставленого в документ фото?
46. Які є засоби для створення, використання типового титульного аркуша документа? Оформіть титульний аркуш для звітів.
47. Наберіть оголошення про захід “Весна Політехніки”, використовуючи можливості MS-Word для ілюстративного й художнього представлення тексту.
48. Створіть діаграму для представлення своєї успішності минулого семестру.
49. Як можна створити блок-схему алгоритму у MS-Word? Наведіть приклад.
50. Налаштуйте автоматичну зміну неправильно написаних слів при перевірці правопису.
51. Які програми входять у стандартну поставку Microsoft Office?
52. Назвіть аналогічні програмні засоби MS-Word.
53. Чим MS-Word принципово відрізняється від Wordpad та Notepad?
54. Яке призначення формату RTF?
55. Охарактеризуйте формат файлів TXT (простий текст).
56. Як роздрукувати документ повністю, окрім сторінки у MS-Word?
57. Як створювати і використовувати макроси для роботи у MS-Word?
58. У яких форматах можна зберігати документ, створений у MS-Word?

## **Самостійна робота № 4**

### **ТЕМА. ПРИЗНАЧЕННЯ Й ОСНОВНІ МОЖЛИВОСТІ РЕДАКТОРА ДІЛОВОЇ ГРАФІКИ MS-VISIO**

#### **Короткі теоретичні відомості**

Одним із найпоширеніших інструментів для створення різноманітних схем, малюнків, креслень, мережних моделей, графіків, діаграм є редактор ділової графіки Microsoft Visio (MS-Visio).

Visio є одним зі стандартних Windows-застосунків, що входять до пакета Microsoft Office (від 2000 року). Тому технологія роботи, зміст меню, склад панелей інструментів, призначення основних “гарячих клавіш” аналогічні.

Цей редактор має зручний і простий в освоєнні інтерфейс, містить широкий набір ефективних інструментів для роботи з векторними зображеннями, широкі можливості включення об'єктів Visio в електронні документи, створені за допомогою інших застосунків.

Документи, які створюються за допомогою Visio, зберігаються у файлі з розширенням \*.vsdx/ vsd.

Visio дозволяє зберегти єдину технологію в підході до підготовки наочної документації: плакатів, креслень, діаграм, схем для пояснівальної записки курсової або дипломної роботи, проекту, статей, посібників, дисертацій.

Один раз накреслену схему можна з мінімальними витратами часу масштабувати й вивести на паперовий носій у вигляді плаката або вставити в будь-який електронний документ.

У сам документ Visio також можна вставити будь-який матеріал, створений в інших стандартних застосунках (текст, малюнки, фотографії, формули, діаграми, схеми), а потім поєднувати їх у єдину схему.

Однією з особливостей Visio є наявність множини дуже корисних і зручних надбудов, які забезпечують, наприклад, доступ до організаційних діаграм, побудову звичайних і тривимірних графіків. Але найпривабливішою є можливість розробляти свої власні бібліотеки з графічними фігурами або використовувати вбудовану колекцію бібліотек Visio.

Розроблення або налаштування користувальських фігур може передбачати безпосереднє малювання, модифікацію готових фігур, обертання їх, роботу з

об'єднаними або згрупованими фігурами, зміну кольорових схем, додавання тексту і т.д. Крім того, можна керувати параметрами фігур у режимі електронної таблиці, що, на відміну від ручного редагування, дозволяє досягти високої точності отриманого зображення.

Базою можливості з високим ступенем швидкості створювати схеми є вбудована в редактор бібліотека трафаретів (*англ. Stencils*), орієнтована на різноманітні застосування. Трафарет - це набір зв'язаних між собою за змістом фігур, створених для подальшого їх використання під час роботи з іншими документами. З бібліотеки користувач вибирає потрібний трафарет, переносить із нього у свій документ необхідні блоки, а потім розмножує їх, масштабує, переміщує, з'єднує лініями й стрілками, описує написами тощо.

У редакторі Visio є засоби створення авторських трафаретів, шаблонів цілих документів, що дає можливість користувачеві розширювати бібліотеку відповідно до своїх потреб, створювати свої власні бібліотеки.

Схеми, діаграми, малюнки, створені в редакторі Visio, ніколи “не розвалюються” під час перенесення в інші електронні документи завдяки наявності в ньому таких автоматизованих процедур, як “склеювання” й груповання.

## Питання для підготовки

1. Якого призначення рисунки можна створити у Visio?
2. Що таке фігура у Visio? Які є можливості зі створення/використання фігур?
3. Як змінити розмір фігури?
4. Як можна захистити фігуру від змін?
5. Які є параметри для редагування фігури?
6. Які є лінії для з'єднання фігур?
7. Які є властивості ліній? Як їх змінювати?
8. Як відредактувати стрілки на лініях?
9. Що таке маркери фігури? Які є типи маркерів і як їх використовувати?
10. Як заповнити фігуру? Які є способи заливки?
11. Як “зліпити” фігури в один рисунок, щоб не відокремлювалися частини?
12. Чим групування фігур відрізняється від їх об'єднання ?

13. Які є варіанти об'єднання фігур?
  14. Як вирівняти розміщення декількох фігур?
  15. Як домогтися точності в розташуванні об'єктів відносно аркуша й один до одного?
  16. Як долучитися до окремої фігури з комбінації, об'єднаних фігур, для зміни її параметрів?
  17. Що таке трафарет у редакторі Visio? Назвіть приклад.
  18. Як створити свій трафарет?
  19. Що таке шаблон у редакторі Visio? Назвіть приклад.
  20. Що таке бібліотека трафаретів у редакторі Visio? Назвіть популярні компоненти цієї бібліотеки.
  21. Що таке стиль у Visio?
  22. Яке розширення у файлів, що містять шаблони?
  23. Яке розширення у файлів, що містять трафарети?
  24. Які формати графічних файлів підтримує редактор Visio?
  25. З яких частин може складатися Visio-документ?
  26. Які є можливості роботи з текстом у редакторі Visio?
  27. Які є варіанти перегляду у редакторі Visio-документа?
  28. З яким типом графіки (векторна/раstrova) працює редактор Visio?
- Відповідь обґрунтуйте.
29. Створіть план квартири, у якій проживаєте.
  30. Побудуйте організаційну діаграму для навчального закладу.
  31. Які є готові конструкції для схем “призначення - програмне забезпечення”?
  32. Які є засоби у Visio для створення високоточних креслень?
  33. Які є засоби у Visio для створення графіків? Які типи графіків можна побудувати?
  34. Які є засоби у Visio для створення високоточних креслень?
  35. Які є засоби у Visio для налаштування користувачького інтерфейсу?
  36. Назвіть аналогічні програмні засоби редактору Visio.
  37. Проаналізуйте найновішу й попередні версії програми Visio. Яка принципова відмінність між ними?
  38. Яка передісторія програми MS-Visio?

## СПИСОК ЛІТЕРАТУРИ

1. Alan Mark Davis. Just Enough Requirements Management: Where Software Development Meets Marketing. – Dorset House, 2005.
2. Budiu R. Memory Recognition and Recall in User Interfaces. – [Електронний ресурс]. – Режим доступу: <https://www.nngroup.com/articles/recognition-and-recall/>.
3. Design applications for the Windows desktop. – [Електронний ресурс]. – Режим доступу: <https://developer.microsoft.com/en-us/windows/desktop/design>.
4. Google C++ Style Guide. [Електронний ресурс]. – Режим доступу: <https://google.github.io/styleguide/cppguide.html>
5. Graham D. Foundations of software testing. ISTQB certification / E. Veenendaal, I. Evans, R. Black. – Intl. Thomson Business Pr, 2007. – 243 p.
6. Guide to the Software Engineering Body of Knowledge: Edition – SWEBOKv3.0. IEEE, 2005. – [Електронний ресурс]. – Режим доступу: <http://www4.ncsu.edu/~tjmenzie/cs510/pdf/SWEBOKv3.pdf>
7. ISO/IEC 12207: 1995. Information technology - Software life cycle processes. Информационные технологии. – Процессы жизненного цикла программного обеспечения.
8. ISO/IEC 9126, Information Technology. – Software quality characteristics and metrics (Part 1-4).
9. ISO/IEC TR 15504, Information Technology. – Software Process Assessment (Part 1-9).
10. Laubheimer P. Preventing User Errors: Avoiding Unconscious Slips. – [Електронний ресурс]. – Режим доступу: <https://www.nngroup.com/articles/slips/>.
11. Lauesen S. User Interface Design: A Software Engineering Perspective. Addison Wesley, 2004. – 624 pp.
12. Molich R., Nielsen J. Improving a human-computer dialogue // Communications of the ACM 33. – 1990. – N 3. – P. 338–348.
13. Nielsen J., Molich R. Heuristic evaluation of user interfaces // Proceedings of ACM CHI'90 Conf. (Seattle, WA, 1-5 April). – 1990. – P. 249–256.
14. Nilsen J. 10 Usability Heuristics for User Interface Design. – [Електронний ресурс]. – Режим доступу: <https://www.nngroup.com/articles/ten-usability-heuristics/>.
15. Resharper C++. Visual Studio Extension for C++ Developers. [Електронний ресурс]. – Режим доступу: <https://www.jetbrains.com/resharper-cpp/#>.
16. SWEBOK. Основные области знаний / А. Островский Физико-технический учебно-научный центр НАН Украины [Електронний ресурс]. – Режим доступу: <http://softandware.org.ua/wp-content/uploads/2014/11/base-areas.pdf>
17. Van Veendabl E. Standard glossary of term used in Software testing / E. Van Veendabl. – ISTQB. – 2007. – Vol. 1,2.

18. Англо-український тлумачний словник з обчислювальної техніки і програмування / ред. О. Р. Микитюк. – Львів: СП “БаK”, 1995. – 304 с.
19. Англо-український тлумачний словник з обчислювальної техніки, Інтернету, програмування. – К.: СофтПрес, 2006. – 823 с.
20. Андон П. И. Основы качества программных систем / П. И. Андон, Г. И. Коваль, Т. М. Коротун, Е. М. Лаврищева, В. Ю. Суслов. – К.: Академперіодика, 2007. – 860 с.
21. Ахо Альфред. Структуры данных и алгоритмы / Альфред Ахо, Джон Хопкрофт, Джеффри Ульман. – М.: Издательский дом “Вильямс”, 2003. – 384 с.
22. Бабенко Л. П. Основи програмної інженерії: навч. посіб. / Л. П. Бабенко, К. М. Лавріщева. – К.: Знання, 2001. – 269 с.
23. Бахтизин В. В. Технология разработки программного обеспечения: учеб. пособ. / В. В. Бахтизин, Л. А. Глухова. – Минск: БГУИР, 2010. – 267 с.
24. Бевз О. М. Проектування програмних засобів систем управління: навч. посіб. Ч. 1. Основи об’єктно-орієнтованого проектування / О. М. Бевз, В. М. Папінов, Ю. А. Скидан; Він. нац. техн. ун-т. – Вінниця, 2010. – 124 с.
25. Бейзер Б. Тестирование чёрного ящика. Технологии функционального тестирования программного обеспечения и систем / Б. Бейзер – СПб.: Питер, 2004. – 320 с.
26. Білас О. Якість програмного забезпечення та тестування / О. Білас. – Львів: Вид-во Львівської політехніки, 2011. – 214 с.
27. Браунси К. Основные концепции структур данных и реализация в C++ / К. Браунси. – М.: Изд. дом “Вильямс”, 2002. – 320 с.
28. Брукшир Дж. Г. Введение в компьютерные науки / Дж. Гленн Брукшир. – М.: Изд. дом “Вильямс”, 2001. – 688 с.
29. Вакалюк Т. А. Технології тестування програм: навч. посіб. / Т. А. Вакалюк. – Житомир, Вид-во ЖДУ, 2013. – 96 с.
30. Введение в программную инженерию и управление жизненным циклом программного обеспечения = Guide to Software Engineering Base of Knowledge (SWEBOK): пер. с англ. С. Орлик [Електронний ресурс]. – Режим доступу: sorlik.blogspot.com/
31. Вигерс К. И. Разработка требований к программному обеспечению / Вигерс К. И.; пер. с англ. – М.: Издательско-торговый дом “Русская редакция”, 2004. – 576 с.
32. Гейтс Б. Бизнес со скоростью мысли / Билл Гейтс. – М.: Эксмо, 2003. – 480 с.
33. Глинський Я. М. Практикум з інформатики / Я. М. Глинський. – Тернопіль: Підручники і посібники, 2014. – 304 с.
34. Глинський Я. М. Linux – практикум з інформатики / Я. М. Глинський, В. А. Ряжська. – Львів: СПД Глинський, 2004. – 248 с.

35. Глушаков С. В. Персональный компьютер / С. В. Глушаков, А. С. Сурядный. – Харьков: Фолио, 2004. – 500 с.
36. Гудлиф П. Ремесло программиста. Практика написания хорошего кода / Гудлиф П.; пер. с англ. – СПб.: Символ Плюс, 2009. – 704 с.
37. Довідка Word [Електронний ресурс]. – Режим доступу: <https://support.office.com/uk-ua/word>
38. Довідка з програми Visio [Електронний ресурс]. – Режим доступу: <https://support.office.com/uk-ua/article/Довідка-з-програми-Visio>.
39. Канер С. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений / Сэм Канер, Джек Фолк, Енг Нгуен. – К.: Диасофт, 2001. – 544 с.
40. Коберн А. Сучасні методи опису функціональних вимог до систем / Коберн А. – М.: Лорі, 2002.
41. Коротєєва Т. О. Алгоритми і структури даних: навч. посіб. / Т. О. Коротєєва. – Львів: Вид-во Львівської політехніки, 2014. – 280 с.
42. Кулямин В. В. Формализация требований на практике / В. В. Кулямин, Н. В. Пакулин, О. Л. Петренко, А. А. Сортов, А. В. Хорошилов. – Препринт ИСП РАН 2005. – 50 с.
43. Кулямин В. В. Технология программирования. Компонентный подход / В. В. Кулямин. – М.: Интернет-университет информационных технологий; БИНОМ. Лаборатория знаний, 2007. – 463 с.
44. Лавріщева К. М. Методы программирования. Теория, инженерия, практика / К. М. Лавріщева. – К.: Наукова думка, 2006. – 451с.
45. Лавріщева К.М. Программа инженерия / К. М. Лавріщева. – К.: Видавничий дім “Академперіодика” НАН України, 2008. – 319 с.
46. Левус Є. В. Основи інформаційних технологій: конспект лекцій для студентів Інституту комп’ютерних наук та інформаційних технологій / Є. В. Левус. – Львів: Видавництво Нац. ун-ту “Львівська політехніка”, 2006. – 60 с.
47. Левус Є. В. Життєвий цикл ПЗ: тестування: методичні вказівки до виконання лабораторної роботи із дисципліни “Основи програмної інженерії” для студентів базового напряму “Програмна інженерія” / укл.: Є. В. Левус, Т. А. Марусенкова. – Львів: Вид-во Львівської політехніки, 2016. – 26 с.
48. Леффінгуелл Д. Принципы работы с требованиями для программного обеспечения / Д. Леффінгуелл, Д. Уідріг. – М.: Вільямс, 2002.
49. Липаев В. В. Программная инженерия. Методологические основы / В. В. Липаев. – М.: ТЕИС, 2006. – 608 с.
50. Макконнелл С. Совершенный код. Мастер-класс / пер. с англ. – М.: Издательско-торговый дом “Русская Редакция”; СПб.: Питер, 2007. – 896 с.

51. Орлов С. А. Технологии разработки программного обеспечения / С. А. Орлов, Б. Я. Цилькер. – 4-е изд. – СПб.: Питер, 2012. – 608 с.
52. Офісні пакети [Електронний ресурс]. – Режим доступу: [http://programy.com.ua/ua/office\\_suite](http://programy.com.ua/ua/office_suite).
53. Павич Н. Я. Людино-машинна взаємодія: конспект лекцій для студентів Інституту комп’ютерних наук та інформаційних технологій базового напряму 6.050103 Програмна інженерія / Н. Я. Павич. – Львів: Видавництво Львівської політехніки, 2013. – 100 с.
54. Поморова О. В. Проектування інтерфейсів користувача: навч. посібник / О. В. Поморова, Т. О. Говорущенко. – Хмельницький: ХНУ, 2011. – 206 с.
55. Савин Р. Тестирование DOT COM, или пособие по жесткому обращению с багами в интернет-стартапах / Савин Р. – М.: Дело, 2007. – 312 с.
56. Сидоров М. О. Вступ до програмної інженерії: конспект лекцій / М. О. Сидоров. – К.: НАУ, 2009. – 130 с.
57. Скотт Б. Проектирование веб-интерфейсов / Б. Скотт, Т. Нейл. – М.: Символ-плюс, 2010. – 352 с.
58. Соммервилл И. Инженерия программного обеспечения / Иан Соммервилл. – М.: Вильямс, 2002. – 624 с.
59. Тетерук И. Тестирование программного обеспечения – основные понятия и определения [Електронний ресурс] / И. Тетерук, А. Булат. – Портал знань – Режим доступу: <http://www.znannya.org/?view=software-testing>
60. Тидвелл Д. Разработка пользовательских интерфейсов / Д. Тидвелл. – СПб.: Питер, 2008. – 416 с.
61. Шатовська Т. Б. Аналіз вимог до інформаційних систем: лабораторний практикум [Електронний ресурс]. – Режим доступу: <http://hire2.hzmk.com.ua/courses/AVPZ/003/content/content1.html>.

## Додатки

### Додаток А

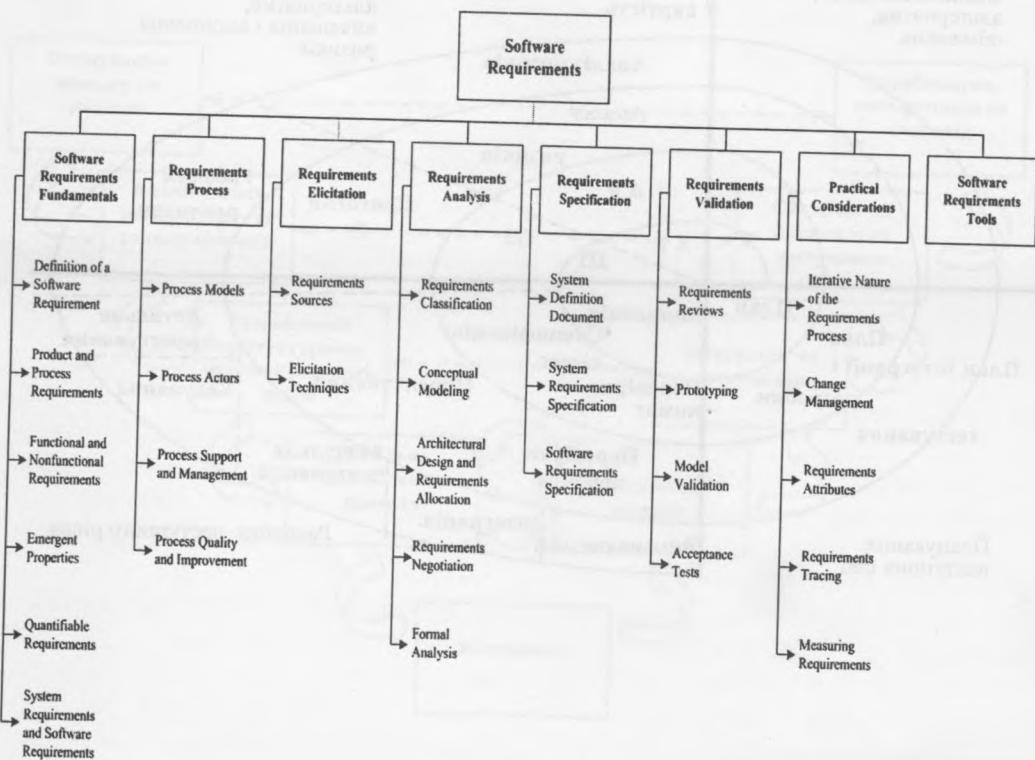
Схема V-подібної моделі життєвого циклу ПЗ



### Схема спіральної моделі життєвого циклу ПЗ



## Опис області знань Software Requirements (Програмні Вимоги) згідно із SWEBOK



## Приклад сценарію використання

### *Замовлення літератури в читацькому залі бібліотеки*

***Головний сценарій* (успішний):**

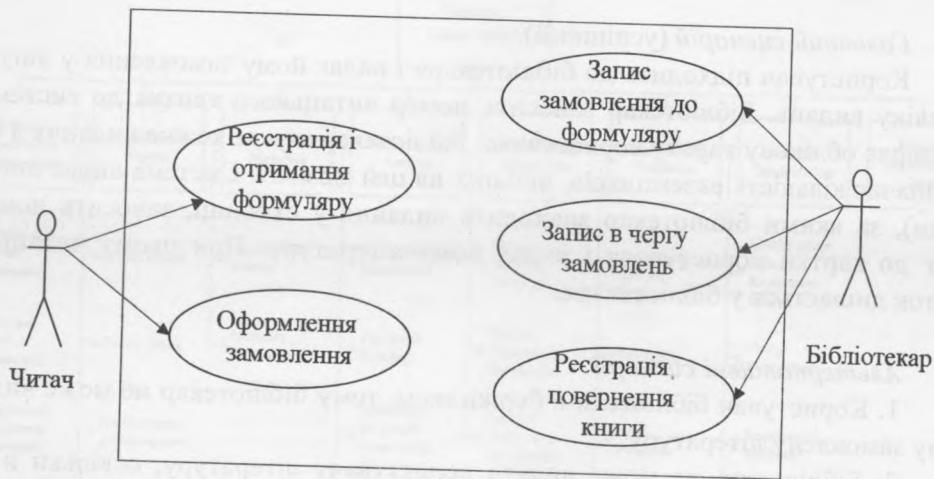
Користувач підходить до бібліотекаря і надає йому замовлення у вигляді переліку видань. Бібліотекар заносить номер читацького квитка до системи і перевіряє облікову картку користувача. Бібліотекар шукає кожне видання в базі і визначає кількість екземплярів, вільних на цей момент. Система видає шифри (коди), за якими бібліотекар знаходить видання у сховищі, заносить номери книг до картки користувача і видає йому літературу. При цьому читацький квиток лишається у бібліотекаря.

***Альтернативні сценарії:***

1. Користувач бібліотеки є боржником, тому бібліотекар не може видати йому замовлену літературу.
2. Бібліотекар не може видати користувачу літературу, оскільки в цей момент немає вільних екземплярів у сховищі.
3. Читацький квиток користувача є недійсним. Бібліотекар вилучає його.
4. Технічний збій роботи системи. Видається повідомлення “немає зв’язку із сервером баз даних”. Бібліотекар викликає адміністратора системи.

## Приклад діаграми варіантів використання

### *Інформаційна система “Бібліотека”*



### Коментар

Інформаційна система бібліотеки передбачає два типи користувачів з розділеною функціональністю.

## Складові специфікації вимог до програмного забезпечення (стандарт IEEE/ANSI 830-1993)

### **Передмова**

- Особи, для яких розробляють документ;
- Попередні версії та зміни, внесені в ПЗ
- Обґрунтування нової версії

### **Вступ**

- Потреби в створенні системи
- Системні функції та пояснення роботи із іншими системами
- Пояснення, який ефект для організації після впровадження системи

**Глосарій** - опис технічних термінів документа

### **Вимоги користувачів**

- Опис сервісів (функцій), які надаються користувачу;
- Нефункціональні системні вимоги;
- Стандарти на ПЗ та на процес створення;

### **Системна архітектура**

- Високорівневе представлення архітектури із розподілом функцій за компонентами системи із виділенням існуючих компонент

**Системні вимоги:** Функціональні та нефункціональні

### **Системні моделі**

- Показують взаємодію між компонентами та зовнішнім середовищем
- Типи: моделі потоків даних; об'єктні; моделі даних

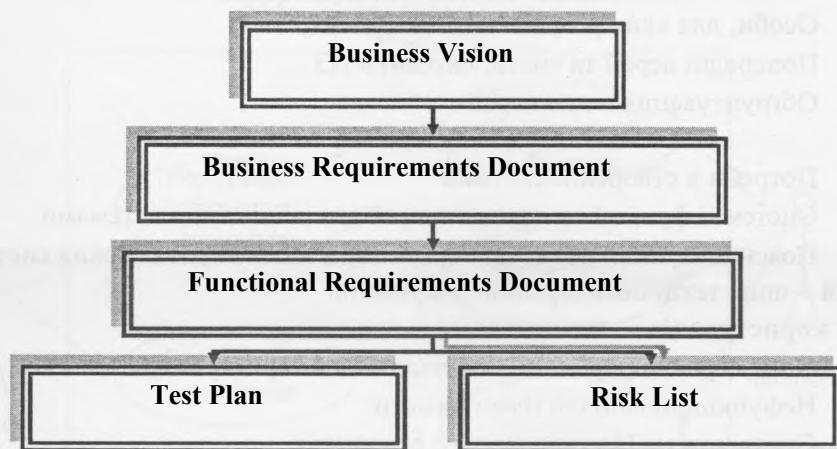
### **Еволюція системи**

- Припущення, на яких основана система
- Зміна апаратних засобів, необхідна для реалізації системи
- Зміна системи відповідно до потреб користувачів у майбутньому

### **Додатки**

- Опис апаратних засобів (мінімальна та максимальна конфігурація) та баз даних (логічна структура, із якою працюватиме ПЗ), із якими працюватиме система

**Приклад ієрархії документів, які створюються на етапі аналізу  
й визначення вимог для масштабного програмного проекту**



**Коментар.**

**Business Vision** - концепція;

**Business Requirements Document (BRD)** - документ з бізнес-вимогами;

**Functional Requirements Document (FRD)** - документ з функціональними вимогами;

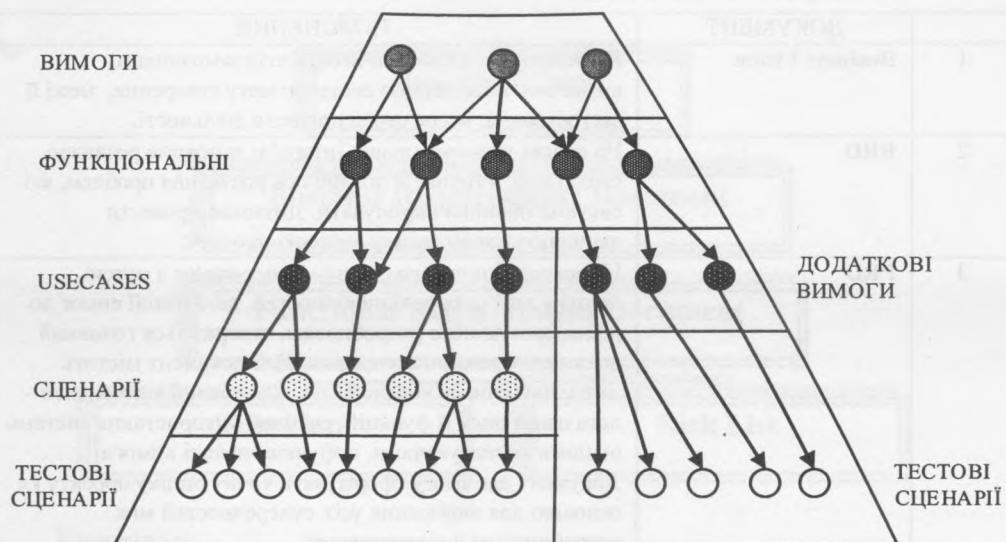
**Test Plan** - план тестування;

**Risk List** - список ризиків.

**Таблиця ЖД.1****Список артефактів, отриманих на етапі аналізу й визначення вимог**

ДОКУМЕНТ		ПОЯСНЕННЯ
1	Business Vision	На основі попереднього інтер'ю із замовником визначають концепцію системи: мету створення, межі її застосування, місце серед процесів діяльності.
2	BRD	На основі уточнювальних інтерв'ю замовник розвиває свою ідею. Уточняє й поглиблює розуміння проблем, які система повинна вирішувати. Дозволяє провести першопочаткову оцінку вартості проекту.
3	FRD	На основі ретельного аналізу бізнес-вимог з метою пошуку логічних невідповідностей, деталізації вимог до рівня, зрозумілого розробникам, створюється головний документ - технічне завдання. Цей документ містить загальний список компонентів, для кожної компоненти - детальний опис її функцій, сценарії використання системи різними користувачами, нефункціональні вимоги. Документ дає змогу сформувати точну оцінку проекту і є основою для вирішення усіх суперечностей між розробниками й замовниками.
4	Test Plan	Документ необхідний для команди тестувальників і складається на основі FRD та інших супутніх документів (приклади дизайну, протоколи взаємодії зі сервером тощо). Мета документа: визначити: що, як і коли тестуватиметься. Разом з планом тестування формують тестові сценарії.
5	Risk List	Спеціальний документ зі списком ризиків, які можуть вплинути на розроблення системи в задані терміни. Також вказуються способи реагування на ті чи інші ризики. Список ризиків - це відповіді на питання “А що буде, якщо...?”

## Піраміда вимог

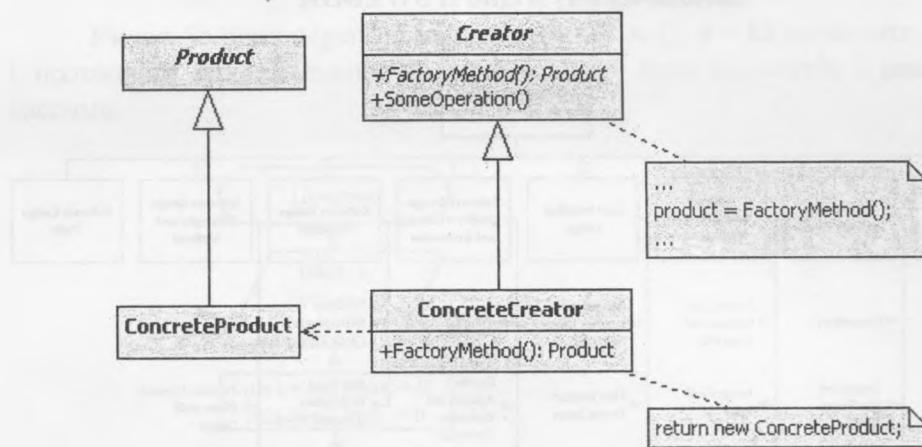


### Пояснення

Вимоги до програмної системи утворюють ієрархію:

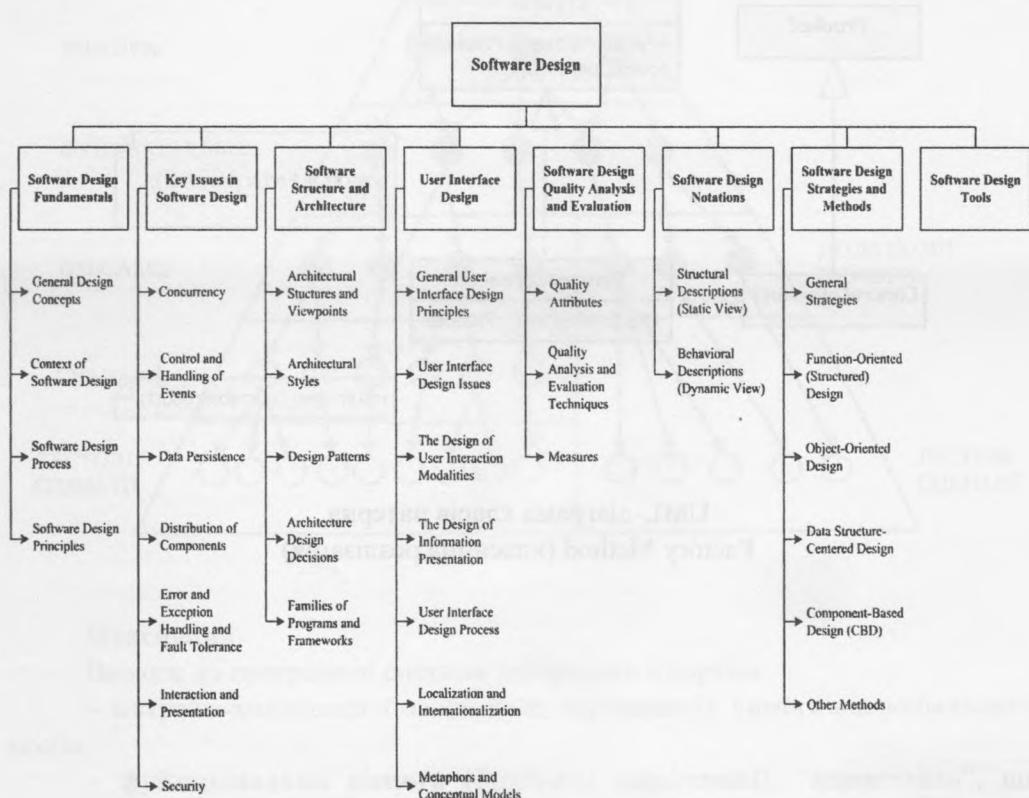
- **потреби** зацікавленої особи (user requirement): вимога від зацікавленої особи;
- **функціональна вимога** (functional requirement): “користність”, що надається системою, зазвичай формулюється бізнес-аналітиком; призначення цієї вимоги - задоволити потреби замовника;
- **сценарій використання** (use case): опис поведінки системи в термінах послідовності дій;
- **додаткова вимога** (in-functional requirement): інша вимога (зазвичай нефункціональна), яка не може бути охоплена сценаріями використання;
- **сценарій** (алгоритм, scenario): визначена послідовність дій; певний шлях за сценарієм використання;
- **тестовий сценарій** (test cases): специфікація тестових початкових даних, умов виконання й очікуваних результатів;

## Приклад патерна Factory Method (фабричний метод)



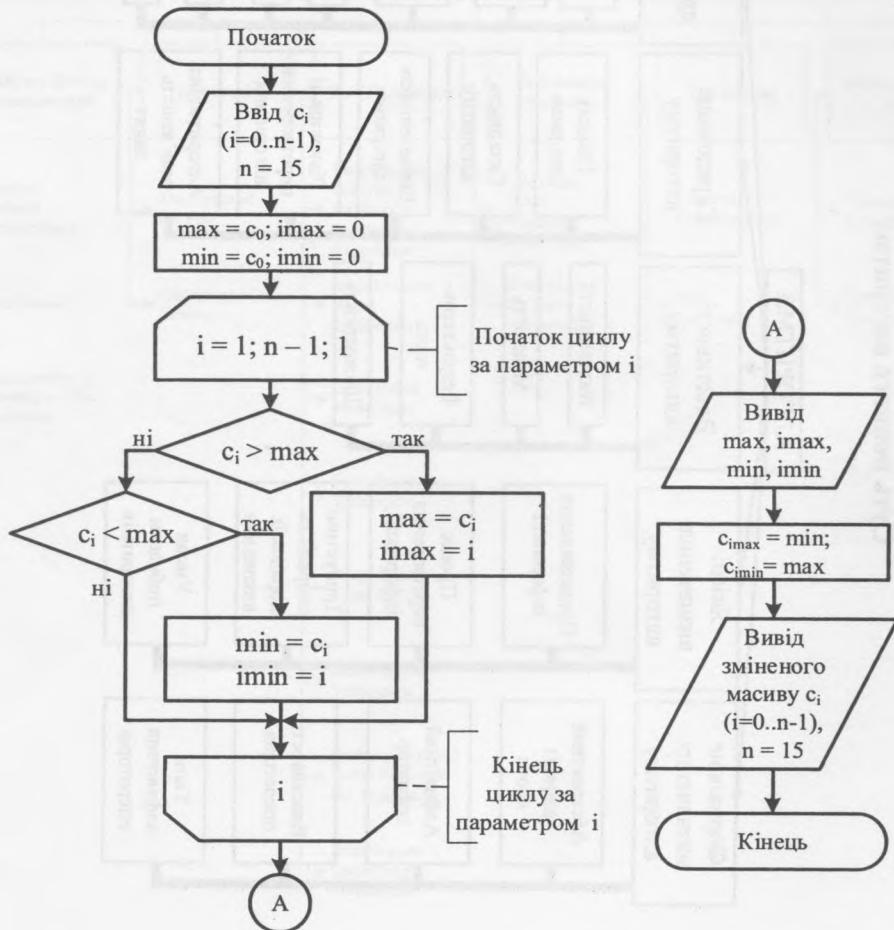
UML-діаграма класів патерна  
Factory Method (класична реалізація)

## Опис області знань Software Design (Проектування програмного забезпечення) згідно із SWEBOK

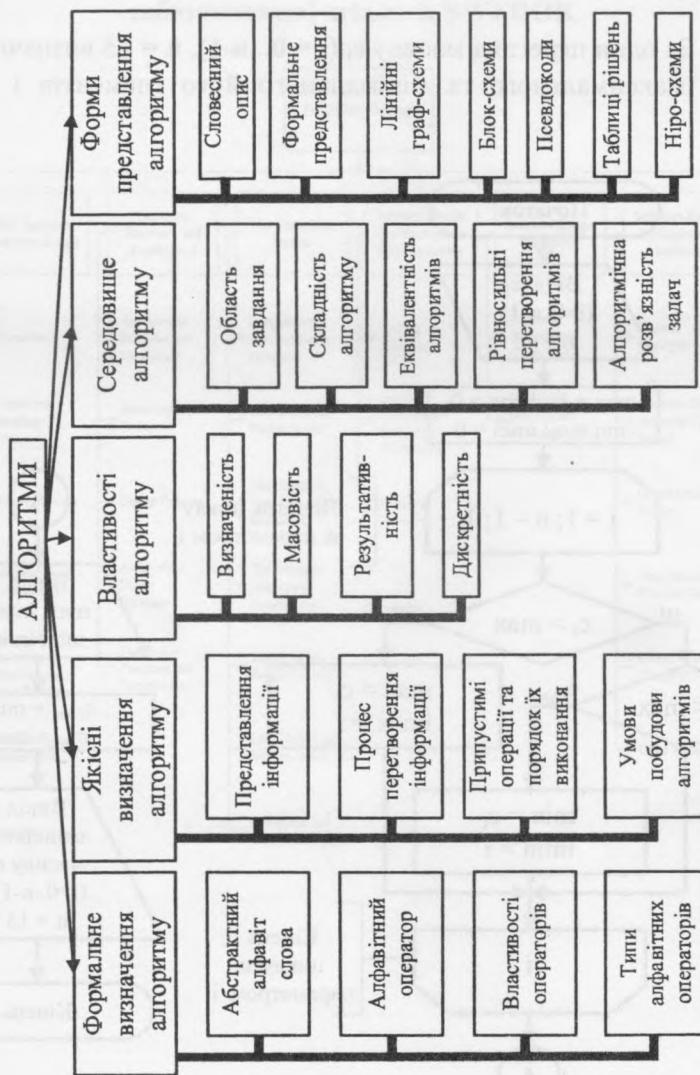


### Приклад побудови блок-схеми алгоритму

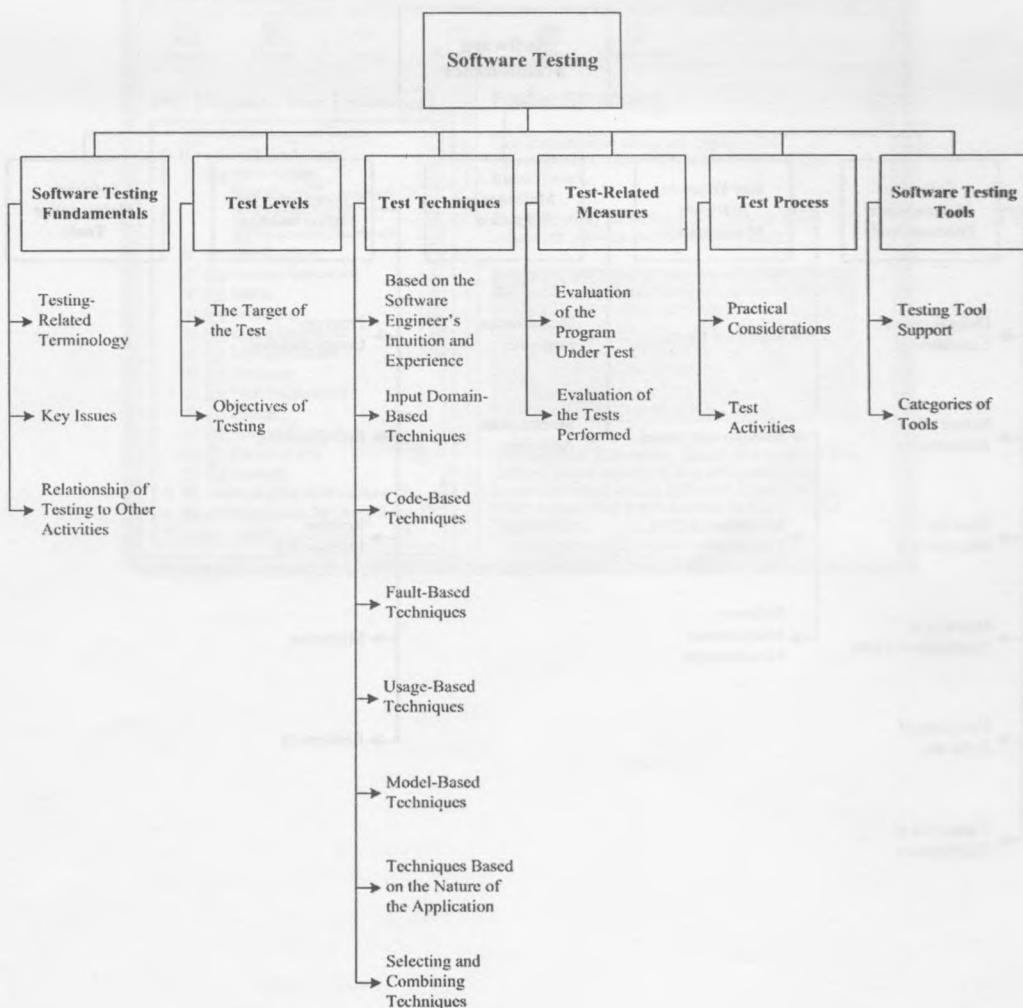
**Умова.** За один перегляд масиву  $C_i$  ( $i = 0..n-1$ ),  $n = 15$  визначити значення і положення максимального та мінімального його елементів і поміняти їх місцями.



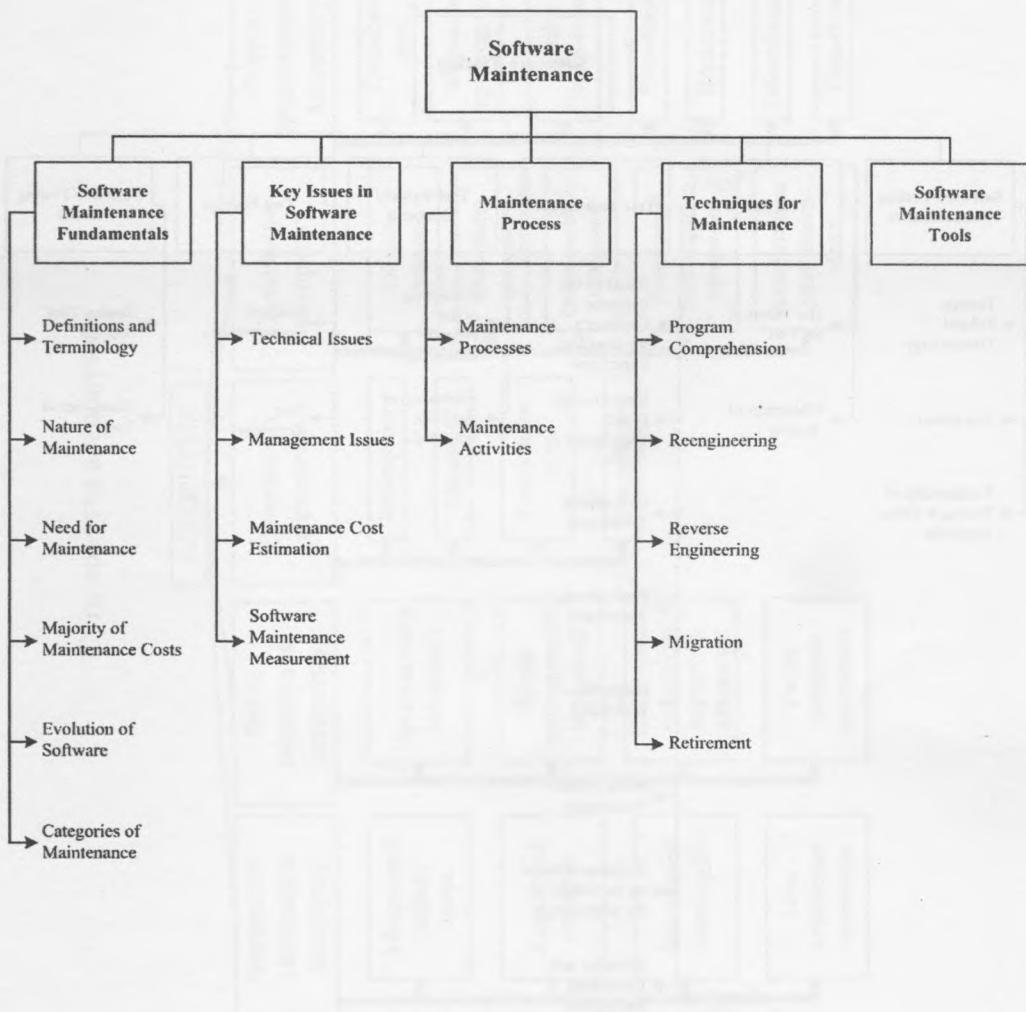
**Суть поняття алгоритму**



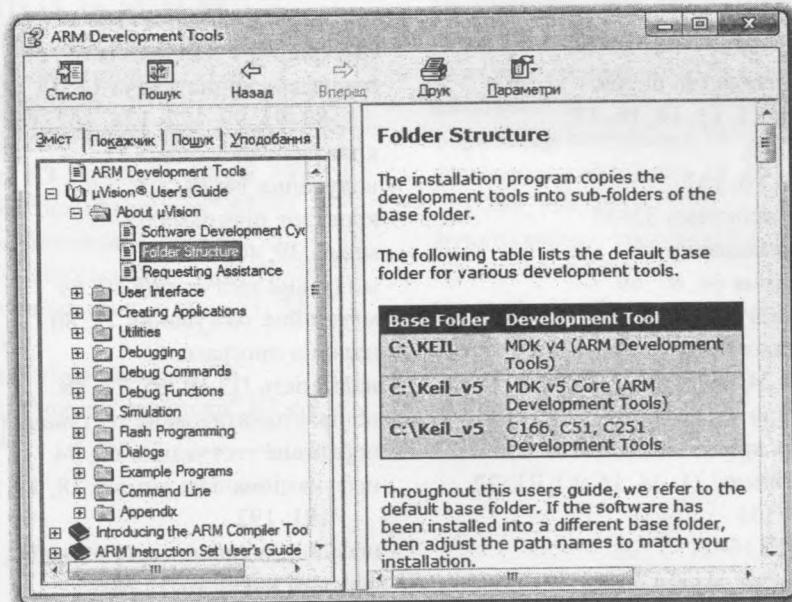
## Опис області знань Software Testing (Тестування програмного забезпечення) згідно із SWEBOK



## Опис області знань Software Maintenance (Супровід програмного забезпечення) згідно із SWEBOK



## Приклад діалогового вікна довідкової системи



## Термінологічний показчик (укр.)

- автоматизоване тестування, 68, 69  
адаптованість ПЗ 65  
алгоритм 35-39, 43, 44, 78  
альфа-тестування 66, 67, 69  
аналіз вимог 11, 13, 14, 16, 17,  
22-24, 150  
артефакт 15, 16, 191  
архітектура програми 33-35  
архітектурне подання 35  
бета-тестування 66, 67, 69  
блок-схема алгоритму 36, 37, 77, 78, 195  
валідація вимог 18  
верифікація 24, 69, 70  
відлагодження 45, 63, 64  
видобування вимог 17  
визначення вимог 11, 14, 16, 17, 21, 22,  
24, 150, 151  
вимоги до ПЗ 16-21  
високорівневий дизайн 34  
встановлення процесу 13  
границі випробування 73  
граф управління 77, 79, 80  
детальне проектування 34  
дефект 68, 74-76, 87, 88  
діаграма прецедентів 21, 188  
димовий тест 66, 67, 69, 71  
динамічний аналіз коду 65  
експлуатація ПЗ 14, 86, 87  
етап життєвого циклу ПЗ 13-15, 23,  
33, 63  
ефективність ПЗ 12, 65, 88  
живучість ПЗ 65  
життєвий цикл ПЗ 13-15, 23, 64, 184, 185  
зв'язний список 42, 43  
зручність інтерфейсу користувача 89,  
90, 124  
інженерія програмного  
забезпечення 11-13  
інтеграційне тестування 67, 69  
інтерфейс користувача 12, 14, 16, 19, 34,  
89-91, 97, 122-124, 162, 163, 198  
клас еквівалентності 71  
кодування 14, 44, 50  
коректна програма 65  
масив 39, 40, 195  
модульне тестування 67, 69  
мутаційне тестування 77, 80  
надійна програма 65  
надійність ПЗ 19, 65, 71, 88  
напівавтоматизоване тестування 68  
негативне тестування 69, 74  
нефункціональна вимога 18, 19, 22, 189,  
191, 192  
нефункціональне тестування 66, 69  
офісний пакет 167, 170  
патерн проектування 33, 34, 193  
переносимість ПЗ 19, 88  
позитивне тестування 74  
практичність ПЗ 65, 89  
програмне забезпечення 11  
продуктивність ПЗ 19, 44, 86-88  
проектування ПЗ 13, 14, 33-35, 38,  
57, 194  
процес 13-15  
приймальне тестування 67, 68  
регресійне тестування 66, 67  
рефакторинг коду 49, 53, 57  
ручне тестування 68  
санітарне тестування 66, 67  
системне тестування 67, 68  
специфікація вимог 13, 14, 16, 21, 23, 24  
65, 67, 70, 122, 189

- статичний аналіз коду 53, 64
- стек 41, 42
- структурні дані 35, 38, 39
- структурне тестування 67, 77
- супровід ПЗ 13, 14, 63, 86-88, 124
- сценарій 187, 192
- сценарій використання 21, 123, 187, 191, 192
- тестовий сценарій 191, 192
- тестування ПЗ 14, 24, 63, 64, 66, 197
- тестування “білої скриньки” 67, 77
- тестування виняткових ситуацій 72-74, 76
- тестування екстремальних умов 72-74, 76
- тестування нормальних умов 71, 72, 74, 76
- тестування “сірої скриньки” 67
- тестування “чорної скриньки” 67, 69, 70, 74
- тестування на основі потоку даних програми 77
- тестування на основі потоку керування програми 77
- технічне завдання 21-24, 191
- управління вимогами 17
- фаза життєвого циклу ПЗ 13, 44, 86
- факторинг 53
- формальне тестування 75
- функціональна вимога 18, 19, 22, 35, 189, 190, 192
- функціональне тестування 66, 67, 69, 70-72, 74, 76
- черга 40, 41, 43
- шаблон
  - проектування 33, 34
- цілісність ПЗ 65
- якість ПЗ 12, 65, 88, 89

# **Термінологічний показчик (англ.)**

- API, Application Programming Interface 163
- Acceptance Testing 68
- black box \_Testing 67
- Bugtracking 74
- Code Review 64
- Coding Conventions 45
- Confirmation Message 116
- Design Pattern 33
- Efficiency 88
- Enhancing 87
- Error Message 116
- GDI, Graphics Device Interface 163
- FAQs (Frequently Asked Questions 95
- FIFO - first in, first out 40
- FILO - first in, last out 41
- Fixing 87
- Functional or Behavioral Requirements 18
- Functionality 88
- GUI, Graphical user interface 90
- Header File 48
- High Fidelity 122, 123
- Integration Testing 67
- Low Fidelity 122, 123
- Maintainability 88
- Maintenance 86, 198
- Notification Message 116
- Office suite 167
- Portability 88
- Reliability 88
- Requirements Analysis 17, 186
- Requirements Elicitation 17, 186
- Requirements Validation 18, 186
- Software Architecture 33
- Software Detailed Design 34
- Software Quality 88
- Software Requirements Management 17
- Software Requirements Specification 16
- Software Testing 64, 197
- Stencils 178
- Top-level Design 34
- Unit Testing 67
- Usability 88, 90
- VBA, Visual Basic Application 168
- Verification 70
- View 35
- Warning Message 116
- white box Testing 67

## ЗМІСТ

ПЕРЕДМОВА.....	3
МЕТОДИЧНІ РЕКОМЕНДАЦІЇ.....	4
Загальна інформація.....	4
Компетентності, які забезпечуються вивченням розділу.....	5
“Життєвий цикл програмного забезпечення”.....	5
Місце навчальної дисципліни “Вступ до інженерії програмного забезпечення” серед інших дисциплін навчального плану спеціальності “Інженерія програмного забезпечення”.....	6
Вимоги до виконання лабораторних робіт.....	7
Рекомендована послідовність виконання лабораторної роботи.....	8
Рекомендована система оцінювання.....	9
Розподіл годин за видами занять.....	9
Перелік скорочень.....	10
<b>ТЕМА 1. АНАЛІЗ ТА ВИЗНАЧЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....</b>	<b>11</b>
1.1. Означення життєвого циклу програмного забезпечення.....	11
1.2. Аналіз та специфікація вимог.....	16
1.3. Технічне завдання - документування аналізу та визначення вимог.....	22
Контрольні питання для самоперевірки.....	25
<b>ЛАБОРАТОРНА РОБОТА №1.....</b>	<b>27</b>
Завдання.....	27
Варіанти завдань до теоретичних відомостей у звіті.....	31
Вимоги до звіту.....	32
<b>ТЕМА 2. ПРОЕКТУВАННЯ Й РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....</b>	<b>33</b>
2.1. Означення етапу проектування.....	33
2.2. Алгоритми - поведінкова складова результатів проектування.....	35
2.3. Структури даних - структурна складова результатів проектування.....	38
2.4. Зміст етапу реалізації (розроблення, кодування).....	44
2.5. Правила оформлення коду на C++/C.....	45
2.6. Застосунок для підвищення якості коду.....	53
Контрольні питання для самоперевірки.....	58
<b>ЛАБОРАТОРНА РОБОТА №2.....</b>	<b>60</b>
Завдання.....	60
Варіанти завдань до теоретичних відомостей у звіті.....	61
Вимоги до звіту.....	62

<b>ТЕМА 3. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ</b>	63
3.1. Визначення та роль тестування.....	63
3.2. Функціональне тестування.....	70
3.3. Документування результатів тестування.....	74
3.4. Структурне тестування.....	77
Контрольні питання для самоперевірки	81
<b>ЛАБОРАТОРНА РОБОТА №3.</b>	83
Завдання.....	83
Варіанти завдань до теоретичних відомостей у звіті.....	84
Вимоги до звіту.....	85
<b>ТЕМА 4. ЗАДАЧА СУПРОВОДУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ</b>	86
4.1. Основні означення етапу супроводу.....	86
4.2. Зручність використання програми.....	89
4.3. Зміст принципів побудови графічного інтерфейсу.....	92
4.4. Типова структура інструкції користувача.....	113
4.5. Інтерфейсні правила побудови повідомлень.....	116
4.6. Програмні продукти для розроблення графічного інтерфейсу.....	122
Контрольні питання для самоперевірки.....	125
<b>ЛАБОРАТОРНА РОБОТА №4.</b>	127
Завдання.....	127
Варіанти завдань до теоретичних відомостей у звіті.....	129
Вимоги до звіту.....	129
<b>ТЕСТОВІ ЗАВДАННЯ ДЛЯ САМОПЕРЕВІРКИ ЗНАНЬ ЗА ТЕМАМИ ЛАБОРАТОРНИХ РОБІТ</b>	130
I. Тести до лабораторної роботи № 1.....	130
II. Тести до лабораторної роботи № 2.....	137
III. Тести до лабораторної роботи № 3.....	145
IV. Тести до лабораторної роботи № 4.....	152
Самостійна робота у межах розділу “Життєвий цикл програмного забезпечення”.....	160
Загальні положення.....	160
Самостійна робота № 1.....	162
Питання для підготовки.....	164
Самостійна робота № 2.....	167
Завдання.....	170
Самостійна робота № 3.....	173
Питання для підготовки.....	174
Самостійна робота № 4.....	177

Питання для підготовки.....	178
<b>СПИСОК ЛІТЕРАТУРИ.....</b>	<b>180</b>
Додаток А.....	184
Додаток Б.....	185
Додаток В.....	186
Додаток Г.....	187
Додаток Д.....	188
Додаток Е.....	189
Додаток Ж.....	190
Додаток З.....	192
Додаток 1.....	193
Додаток К.....	194
Додаток Л.....	195
Додаток М.....	196
Додаток Н.....	197
Додаток П.....	198
Додаток Р.....	199
Термінологічний покажчик (укр.).....	200
Термінологічний покажчик (англ.).....	202

# Книги для навчання і роботи!



Білущак Т. М.

## МЕНЕДЖМЕНТ АРХІВНОЇ ДІЯЛЬНОСТІ

Навчальний посібник. - Львів: Видавництво Львівської

політехніки, 2017. - 240 с. Формат 145x200 мм.

М'яка обкладинка. (Серія "Інформація. Комунація.

Документація". Випуск 12).

ISBN 978-617-607-495-3 (серія)

ISBN 978-966-941-045-0 (випуск 12)

Навчальний посібник призначений для забезпечення навчального процесу з дисципліни "Менеджмент бібліотечних та архівних установ". У книзі зібрано матеріали, які можна використовувати на всіх етапах навчального процесу: на лекційних, практичних-лабораторних заняттях та під час контрольного заходу.

Пропонований навчальний посібник можуть використовувати студенти вищих навчальних закладів — майбутні фахівці у галузі архівної, інформаційної, бібліотечної та музеїної справи, документно-комунаційної діяльності.

Тимовчак-Максимець О. Ю. та ін.

## ПОШУКОВІ ТЕХНОЛОГІЇ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

Навчальний посібник / О. Ю. Тимовчак-Максимець,

А. М. Пелещин, О. В. Марковець, Н. О. Думанський. -

Львів : Видавництво Львівської політехніки, 2017. - 124 с.

Формат 145x200 мм. М'яка обкладинка. (Серія "Інформація.

Комунація. Документація"; випуск 11).

ISBN 978-617-607-495-3 (серія)

ISBN 978-966-941-036-8 (вип. 11)

Викладено основні поняття та види інформаційного пошуку, інформаційні потреби та їхній вплив на інформаційний пошук, розглянуто інформаційно-пошукові мови та лінгвістичне забезпечення інформаційного пошуку, охарактеризовано види пошукових систем WWW та їх функції, що широко використовують у практиці для створення пошукових запитів.

Для студентів вищих навчальних закладів - майбутніх фахівців у галузі інформаційно-пошукової діяльності, а також студентів та викладачів гуманітарних напрямів підготовки

Басюк Т. М. та ін.

## ОНТОЛОГІЧНИЙ ІНЖИНІРИНГ

Навчальний посібник / Т. М. Басюк, Д. Г. Досин,  
В. В. Литвин. - Львів: Видавництво Львівської політехніки,  
2017. - 224 с. Формат 170\*240 мм. Тверда опраva.  
ISBN 978-966-941-031-3

Наведено дослідницькі результати, отримані в новій галузі науки, пов'язаної з побудовою і застосуванням онтології. Висвітлено різні погляди на поняття онтології, що використовується в сучасних інформаційних технологіях, визначено цей термін, а також розглянуто відомі класифікації. Наведено загальну характеристику методів побудови онтологій, описано основні мови подання онтологій і найістотніші онтологічні ресурси. Розглянуто проблеми, що супроводжують створення онтологій, та наведено можливі шляхи їхнього вирішення.

Для студентів вищих навчальних закладів, які хочуть попідпити свої знання в галузі побудови та експлуатації онтологій, а також для користувачів комп'ютерів та спеціалістів з інформаційних технологій.

Ткаченко Р. О. та ін.

## НЕЙРОМЕРЕЖЕВІ ЗАСОБИ ШТУЧНОГО ІНТЕЛЕКТУ

Навчальний посібник / Р. О. Ткаченко, П. Р. Ткаченко,  
І. В. Ізонін. - Львів: Видавництво Львівської політехніки,  
2017. - 208 с. Формат 170x240 мм. Тверда опраva.  
ISBN 978-966-941-011-5

Призначено для студентів спеціальності “Видавництво та поліграфія”, а також для спеціальності “Інформаційні управлюючі системи та технології” та інших, за якими вивчають нейромережеві засоби штучного інтелекту.



**Видавництво Львівської політехніки**

вул. Ф. Копесси, 4, кorp. 23A, м. Львів, 79013  
тел. +380 32 2582146, факс +380 32 2582136, <http://vlp.com.ua>, [vmp@vlp.com.ua](mailto:vmp@vlp.com.ua)



# НАВЧАЛЬНЕ ВИДАННЯ

**Левус Євгенія Василівна  
Марусенкова Тетяна Анатоліївна  
Нитребич Оксана Олександрівна**

## ЖИТТЄВИЙ ЦІКЛ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Навчальний посібник

Редактор *Ольга Дорошенко*

Коректор *Олеся Косик*

Технічний редактор *Лілія Саламін*

Комп'ютерне верстання *Олени Катачиної*

Художник-дизайнер *Анна Христонько*

Здано у видавництво 17.01.2017. Підписано до друку 13.11.2017.

Формат 70x90/16. Папір офсетний. Друк офсетний.

Умови, друг. арк. 15,8. Обл.-вид. арк. 8,6.

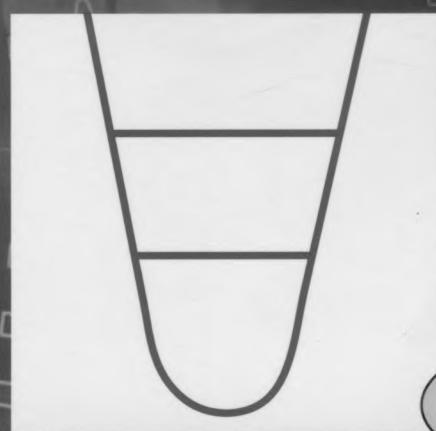
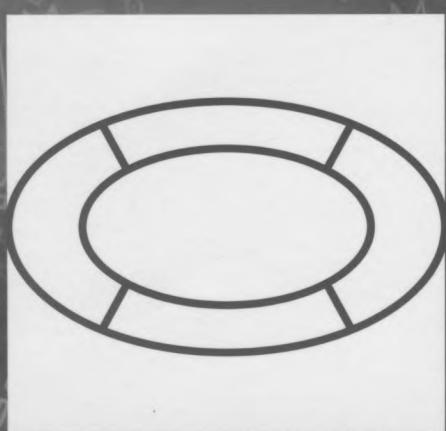
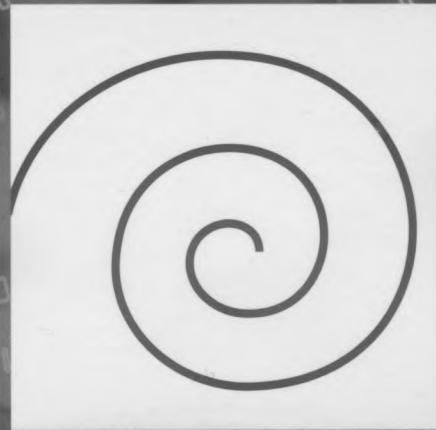
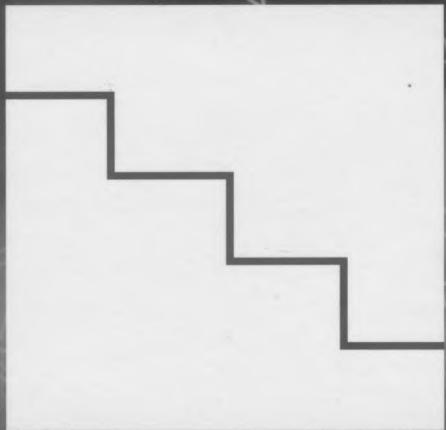
Наклад 150 прим. Зам. 170038.

**Видавець і виготовник: Видавництво Львівської політехніки**  
*Свідоцтво суб'єкта видавничої справи ДК № 4459 від 27.12.2012 р.*

бул. Ф. Колесси, 4, Львів, 79013  
тел. +380 32 2582146, факс+380 32 2582136  
vlp.com.ua, ел. пошта: vmr@vlp.com.ua

Є. В. Левус, Т. А. Марусенкова, О. О. Нитребич

# ЖИТТЄВИЙ ЦИКЛ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ





### ЛЕВУС Євгенія Василівна

кандидат технічних наук, доцент кафедри програмного забезпечення.

Закінчила з відзнакою факультет прикладної математики Львівського державного університету імені Івана Франка та магістратуру за програмою «Комп'ютерні системи проектування» Державного університету «Львівська політехніка». Досвід викладацької роботи понад 15 років. є автором понад 65 наукових і навчально-методичних публікацій. Керівник освітньої програми рівня «бакалавр» спеціальності «Інженерія програмного забезпечення» у Національному університеті «Львівська політехніка». Читає лекції з дисциплін «Вступ до інженерії програмного забезпечення», «Комп'ютерна графіка», «Декларативне програмування».

*Наукові інтереси:* якість програмного забезпечення, математичне та програмне забезпечення систем автоматизованого проектування складних об'єктів.



### МАРУСЕНКОВА Тетяна Анатоліївна

кандидат технічних наук, старший викладач кафедри програмного забезпечення.

Закінчила з відзнакою факультет комп'ютерних наук та інформаційних технологій Національного університету «Львівська політехніка». Автор близько 50 наукових і методичних публікацій. Читає лекції з курсу «Програмування мікроконтролерів» і практичні та лабораторні заняття з дисциплін «Методи та засоби наукових досліджень в інженерії програмного забезпечення», «Основи програмування», «Об'єктно-орієнтоване програмування», «Вступ до інженерії програмного забезпечення». Є постійним виконавцем науково-дослідних робіт у межах співпраці між Львівською політехнікою та італійською компанією Dinamica Generale, що спеціалізується на інноваційних вбудованих системах у сільському господарстві.

*Наукові інтереси:* вбудовані системи, математичне моделювання.



### НИТРЕБИЧ Оксана Олександровна

кандидат технічних наук, асистент кафедри програмного забезпечення.

Закінчила з відзнакою факультет прикладної математики Львівського державного університету імені Івана Франка. Автор близько 20 наукових та науково-методичних публікацій. Читає лекції з дисциплін «Інновації та підприємництво в програмній інженерії», «Управління якістю програмного забезпечення», веде практичні та лабораторні заняття з курсів «Дослідження операцій», «Комп'ютерна дискретна математика», «Комп'ютерна графіка», «Конструювання програмного забезпечення», «Вступ до інженерії програмного забезпечення».

Викладацьку роботу на кафедрі О. О. Нитребич успішно поєднує з роботою на фірмі EPAM Systems на посаді інженера з автоматизації тестування.

*Наукові інтереси:* тестування, надійність програмного забезпечення.

ISBN 978-966-941-098-6



9 789669 410986 >

