**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**Запорізький національний технічний університет**


# МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт
з дисципліни
**“Управління ризиками”**
для студентів спеціальностей
121 “Інженерія програмного забезпечення” та
122 “Комп’ютерні науки та інформаційні технології”
(всіх форм навчання)

**2016**

Методичні вказівки до виконання лабораторних робіт з дисципліни “Управління ризиками” для студентів спеціальностей 121 “Інженерія програмного забезпечення” та 122 “Комп’ютерні науки та інформаційні технології” (всіх форм навчання) / В.М. Льовкін. – Запоріжжя : ЗНТУ, 2016. – 70 с.

Автор: В.М. Льовкін, канд. техн. наук, доцент

Рецензент: В.І. Дубровін, канд. техн. наук, професор

Відповідальний
за випуск: С.О. Субботін, д.т.н., професор

Затверджено
на засіданні кафедри
програмних засобів

Протокол № 1
від “16” серпня 2016 р.

# ЗМІСТ

# ВСТУП

Дане видання призначене для вивчення студентами всіх форм навчання основ управління ризиками та практичного засвоєння вміння використовувати методи і засоби ідентифікації, аналізу та управління ризиками в процесі розроблення програмного забезпечення.

Відповідно до графіка студенти перед виконанням лабораторної роботи повинні ознайомитися з конспектом лекцій та рекомендованою літературою. Дані методичні вказівки містять тільки основні, базові теоретичні відомості, необхідні для виконання лабораторних робіт, тому для виконання лабораторної роботи та при підготовці до її захисту необхідно ознайомитись з конспектом лекцій та опрацювати весь необхідний матеріал, наведений в переліку рекомендованої літературі, використовуючи також статті у інтернет-виданнях та актуальних наукових журналах з інженерії програмного забезпечення.

Для одержання заліку з кожної роботи студент повинен у відповідності зі всіма наведеними вимогами розробити програмне забезпечення та оформити звіт, після чого продемонструвати на комп'ютері розроблене програмне забезпечення з виконанням всіх запропонованих викладачем тестів.

Звіт виконують на білому папері формату А4 ($210 \times 297$ мм). Текст розміщують тільки з однієї сторони листа. Поля сторінки з усіх боків – 20 мм. Аркуші вміщують у канцелярський файл.

Усі завдання повинні виконуватись студентами індивідуально і не містити ознак плагіату як в оформленому звіті так і в розробленому програмному забезпеченні. Під час співбесіди при захисті лабораторної роботи студент повинен виявити знання щодо мети роботи, теоретичного матеріалу, методів виконання кожного етапу роботи, змісту основних розділів звіту з демонстрацією результатів на конкретних прикладах, практичних прийомів використання теоретичного матеріалу в розробленому програмному забезпеченні. Студент повинен вміти обґрунтувати всі прийняті ним проектні рішення і рішення з аналізу і управління ризиками та правильно аналізувати і використовувати на практиці отримані результати. Для базової самоперевірки при підготовці до виконання і захисту роботи студент повинен відповісти на контрольні запитання, наведені наприкінці опису відповідної роботи.

# 1. ЛАБОРАТОРНА РОБОТА № 1 УПРАВЛІННЯ РИЗИКАМИ РОЗРОБЛЕННЯ КОДУ В ПРОЦЕСІ МУЛЬТИПАРАДИГМЕННОГО ПРОГРАМУВАННЯ

## 1.1. Мета роботи

1.1.1 Ознайомитись з основними можливостями, парадигмами, типами даних, синтаксичними особливостями та принципами мови програмування Lua.

1.1.2 Навчитися розробляти програми мовою програмування Lua на основі парадигм структурного та функціонального програмування.

1.1.3 Навчитися ідентифікувати та управляти ризиками в процесі мультипарадигменного програмування.

## 1.2. Короткі теоретичні відомості

### 1.2.1 Середовище та інструментарій розроблення програм мовою Lua

**Lua** – інтерпретовувана мова програмування, розроблена підрозділом Tecgraf Католицького університету Ріо-де-Жанейро. З огляду на можливості та ідеологію мова близька до JavaScript.

Мова програмування Lua використовується зазвичай в проектах, де потрібно вбудовувати швидку та легку скрипкову мову програмування (наприклад, при розробленні ігор, де Lua використовується між ігровим движком та даними для написання сценаріїв взаємодії об’єктів).

Для того щоб запускати програми мовою Lua або виконувати команди в інтерактивному режимі, необхідно інсталювати інтерпретатор.

Для операційних систем сімейства Windows можна використати мультиплатформну систему управління пакетами LuaDist (http://luadist.org/), яка потребує також компілятор C (наприклад, MinGW), CMake та Git (якщо планується інсталювати систему з репозиторію), інстальованих у системі. Для того щоб розгорнути систему, необхідно виконати команди (*_install\bin* – директорія, де знаходяться файли системи):

CMD

```
cd _install\bin
luadist C:\my_lua install lua luasocket md5
```

Можна просто скопіювати завантажений дистрибутив та перейшовши в теку /bin запустити файл lua.exe інтерпретатора або передати йому на виконання файл з текстом програми наступною командою:

CMD

```
lua myf.lua
```

Завантажити безпосередньо окремо сам інтерпретатор можна за посиланням http://www.dcc.ufrj.br/~fabiom/lua/lua52_win32.zip.

Під час запуску інтерпретатора в інтерактивному режимі підтримуються наступні налаштування для запуску lua:

а) -i – розпочати інтерактивну сесію після запуску програми (lua –i prog);

б) -e – дозволяє вводити код прямо в командний рядок;

в) -l – завантажити бібліотеку;

У деяких дистрибутивах Linux бібліотеку Lua встановлено за замовчанням, у інших її можна інсталювати за допомогою відповідного пакету.

В якості IDE для розроблення програм мовою Lua можна використовувати середовище Lua Development Tools на базі середовища Eclipse. Даний інструмент дозволяє розробникам використовувати засоби середовища Eclipse для роботи над проектами Lua: згортання коду, підсвічування синтаксису і семантики, автодоповнення коду, зневадження тощо.

Запустивши Lua development Tools IDE та обравши робочу директорію, необхідно створити новий проект Lua за допомогою меню File → New → Lua Project. Після цього з’явиться вікно (рис. 1.1), де можна задати назву проекту та обрати середовище виконання.

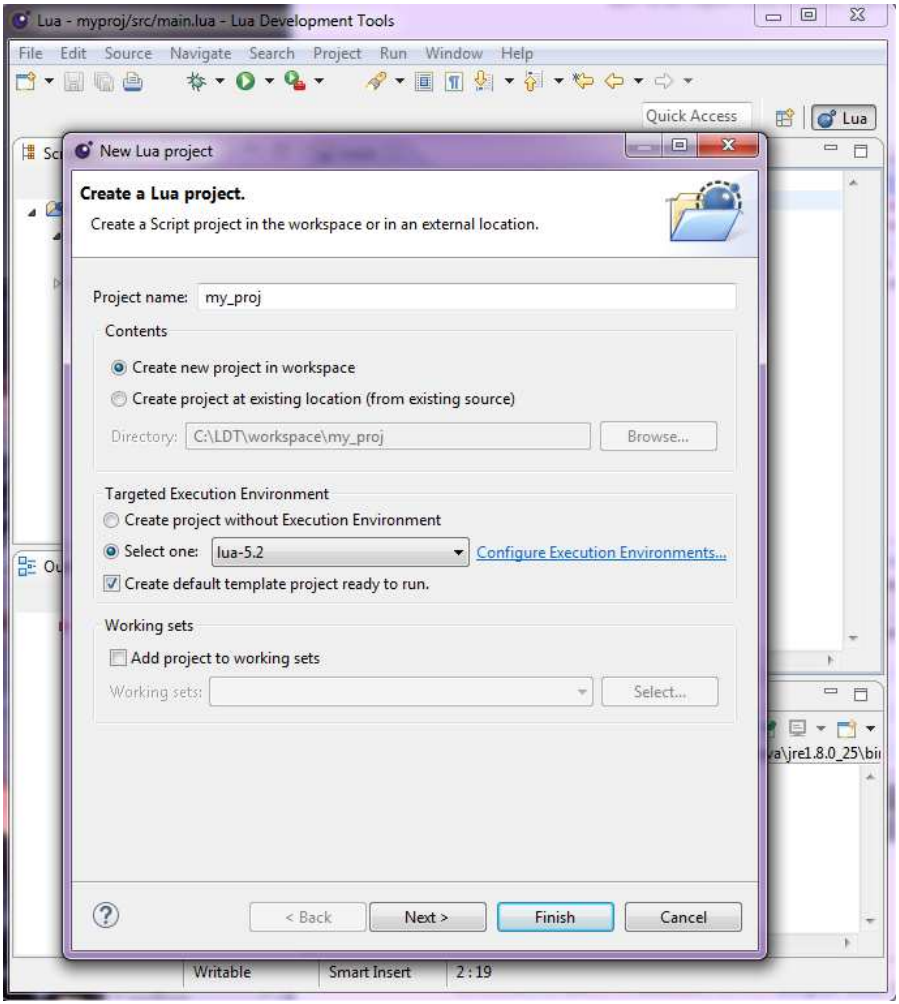



Рисунок 1.1 – Створення нового проекту Lua у Lua Development Tools

У вікні, що відкриється після створення проекту, (рис. 1.2) можна отримати доступ до браузера скриптів Script Explorer, де представлені всі проекти з робочої області. Кожний такий проект має теку /src, де розташовані файли проекту. У правій частині вікна відображається редактор вихідного коду з текстом обраного з браузеру скриптів файлу проекту, звідки його можна редагувати

необхідним чином.

У нижній частині вікна за закладками відображаються помилки, знайдені в проекті, документація проекту, консоль, куди виводяться результати виконання проекту зокрема.

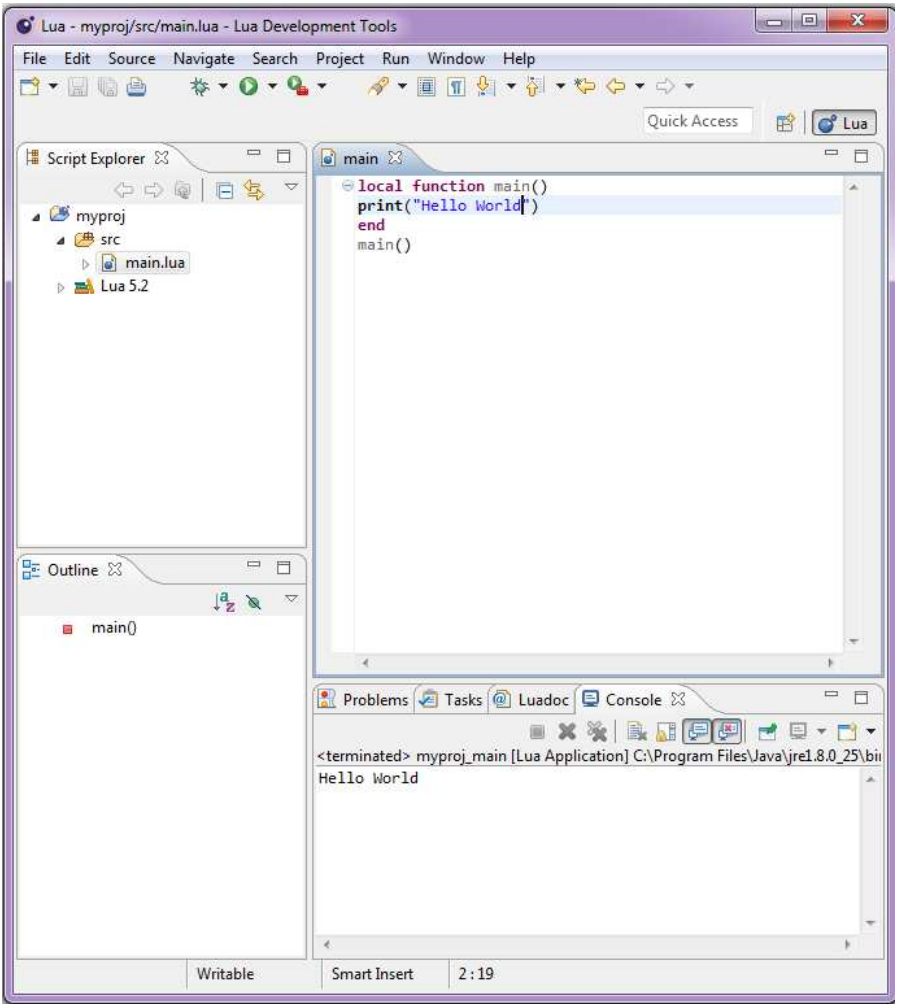



Рисунок 1.2 – Розроблення та виконання скрипту в Lua Development Tools

Для додавання нових файлів у проект можна скористатися меню File → New → Lua File (або DocLua File для документування).

### 1.2.2 Основи мови програмування Lua

У мові програмування Lua глобальні змінні не потрібно оголошувати, а можна одразу використовувати (спочатку такі змінні мають значення nil). Вбудованих типів змінних не існує, будь-які змінні можуть містити значення будь-якого типу.

Для оголошення локальних змінних використовується оператор local:

```Lua
local i = 1
```

Базові типи мови програмування Lua:
– nil;
– логічний тип boolean;
– числовий тип number;
– рядковий тип string;
– userdata;
– function;
– thread;
– табличний тип table.

Таблиці імплементують асоціативні масиви – масиви, які можуть індексуватися не тільки за номерами, але й за рядками або іншими значеннями, допустимими в Lua.

Приклад використання таблиць, які за своєю суттю є об’єктами:

```Lua
a = {}
k = "x"
a[k] = 10
a[20] = "great"
```

```
-- Lua
print(a["x"])
k = 20
print(a[k])
a["x"] = a["x"] + 1
print(a["x"])
```

Таблиця в Lua мають широке коло застосування і використовуються також для збереження структур даних. Вони можуть використовуватися в якості масивів, матриць і багатовимірних масивів, зв'язних переліків, черг і відповідних структур з двоканальною системою, множин і мультимножин, рядкових буферів, графів.

Lua використовує наступні арифметичні операції: бінарні '+' (додавання), '-' (віднімання), '*' (множення), '/' (ділення), '^' (піднесення до ступеня), '%' (ділення за модулем) та унарна '-' (заперечення). Всі дані дії виконуються над дійсними числами.

Використовуються наступні операції порівняння: <, >, <=, >=, ==, ~=.

Логічні оператори: and, or, not.

Для управління використовуються умовний набір операторів if then else, цикли for, while та repeat–until, оператори контролю break, return та goto.

Управляюча конструкція if у загальному вигляді має наступний синтаксис:

```
Lua
if <умова> then
<набір операторів 1>
else <набір операторів 2>
end
```

Традиційний числовий цикл for має наступний синтаксис:

**Lua**

```
for var = exp1, exp2, exp3 do
<тіло циклу>
end
```

Значення exp1 та exp2 задають межі інтервалу, exp3 – крок циклу.

Узагальнена форма циклу for має наступний синтаксис:

**Lua**

```
for <var-list> in <exp-list> do
<body>
end
```

Цикл while має наступний синтаксис:

**Lua**

```
while <умова>
<тіло циклу>
end
```

Цикл з постумовою repeat визначається за допомогою наступного синтаксису:

**Lua**

```
repeat
<тіло циклу>
until <умова>
```

Приклад програми, що визначає функцію для обчислення факторіалу числа:

Lua

```
function fact (n)
  if n == 0 then
    return 1
  else
    return n * fact(n-1)
  end
end
print("enter a number:")
a = io.read("*n")
print(fact(a))
```

### 1.2.3 Стандартні бібліотеки мови програмування Lua

Підключення стандартних бібліотек та модулів в програмах мовою Lua відбувається наступним чином:

Lua

```
local m = require "math"
```

Стандартний набір математичних функцій міститься в бібліотеці math: тригонометричні функції (sin, cos, tan, asin, acos), піднесення до ступеня і логарифмування (exp, log, log10), функції округлення (floor, ceil), max, min, функції генерації псевдовипадкових чисел (random, randomseed), значення pi і huge, що є найбільшим числом, функції deg I rad для перетворення градусів і радіанів.

Побітові операції містяться в бібліотеці bit32 і включають побітові операції і, або, ні, виключне або (band, bor, bnot, bxor), операції зсуву і циклічного зсуву (lshift, rshift, arshift (арифметичний

зсув), lrotate, rrotate). Бібліотека оперує з беззнаковими цілими значеннями.

Бібліотека table включає допоміжні функції для маніпулювання з таблицями як з масивами:

– table.insert (array, position, element) – вставляє елемент у задану позицію масиву;

– table.remove – вилучає елемент за заданою позицією з масиву (без заданої позиції вилучає останній елемент);

– table.sort (array, function) – виконує сортування за допомогою заданої функції, яка повертає істинне значення, якщо перший аргумент повинен розташовуватися першим у відсортованому масиві;

– table.concat – отримує перелік рядків та повертає результат конкатенації всіх даних рядків (другий аргумент може задавати розділювач).

Бібліотека string містить наступні рядкові функції:

– string.upper(s) – перетворює регістр символів у рядку на верхній;

– string.lower(s) – перетворює регістр символів у рядку на нижній;

– string.len(s) – повертає довжину рядка;

– string.rep(s,n) – повертає рядок, повторений задану кількість разів;

– string.sub(s,i,j) – видобуває підрядок з даного рядка, обмежений заданими за позиціями символами;

– string.char і string.byte – перетворюють символи у їх числове представлення і навпаки;

– string.format – виконує форматування рядка;

– string.find – виконує пошук у рядку за заданим шаблоном і повертає індекси початку і закінчення знайденого спів падіння з шаблоном;

– string.match – виконує пошук у рядку за заданим шаблоном і повертає частину рядка, яка відповідає шаблону;

– string.gsub (str, pattern, replacement) – замінює всі входження шаблону в заданий рядок str на заданий рядок replacement (таблицю або функцію);

– string.gmatch – повертає функцію, яка літерує за всіма входженнями заданого шаблону в рядок.

Приклад використання шаблонів для пошуку:

**Lua**

```
date = "Today is 17/7/1990"
d = string.match(date, "%d+/%d+/%d+")
print(d) --> 17/7/1990
```

Класи символів, які можуть використовуватися в шаблонах:
– . – всі символи;
– %a – літери;
– %c – управляючі символи;
– %d – цифри;
– %g – друковані символи окрім пробілів;
– %l – літери в нижньому регістрі;
– %p – знаки пунктуації;
– %s – символ пробілу;
– %u – літери у верхньому регістрі;
– %w – алфавітно-цифровий символ;
– %x – шістнадцятирічні цифри.

Спеціальні символи, які можуть використовуватися в шаблонах:
– + – одне або більше повторень;
– * – 0 або більше повторень;
– - – 0 або більше лінивих повторень;
– ? – 0 або 1 входження;
– % – вихід зі спеціальних символів;
– [] – групування символів;
– ^ – виключення заданої групи символів (якщо починає шаблон, то співпадіння знаходяться тільки на початку рядку);
– $ – якщо шаблон завершується даним символом, то співпадіння будуть знаходитись в кінці рядка.

Спеціальні символи, круглі дужки, використовуються для розбиття результатів пошуку. Наприклад:

**Lua**

```
pair = "name = Anna"
key, value = string.match(pair, "(%a+)%s*=%s*(%a+)")
print(key, value) --> name Anna
```

Стандартна бібліотека введення/виведення даних io пропонує дві моделі: просту та повну.

Проста модель використовує два поточні файли, які є стандартними вхідним stdin та вихідним stdout потоками, за допомогою наступних функцій:

– io.input(filename) – функція, яка визначає джерело вхідного потоку;

– io.output(filename) – функція, яка визначає файл, в який записуються всі результати команд виведення;

– io.write – виводить задані рядки у поточний файл виведення;

– io.read – зчитує рядки з поточного файлу і має наступні аргументи:

а) “*a” – зчитує файл повністю;

б) “*l” – зчитує наступний рядок (без символу нового рядка);

в) “*L” – зчитує наступний рядок (з символом нового рядка);

г) “*n” – зчитує число;

д) num – зчитує рядок з num символів.

Повна модель використовує дескриптори і є еквівалентною до моделі роботи з файлами в C.

Для відкриття файлу використовується функція io.open, в якій задається ім’я файлу і режим доступу та яка повертає номер і повідомлення про помилку в разі її виявлення. Після застосування даної функції створюється дескриптор, який має наступні методи: read, write, close.

Бібліотека введення/виведення пропонує наступні C-потоки: io.stdin, io.stdout, io.stderr.

Функція tmpfile() повертає дескриптор на тимчасовий файл, flush – виконує всі незавершені виведення в файл. Метод setvbuf встановлює режим буферизації потоку.

Метод seek(whence,offset) може повертати та встановлювати поточну позицію в файлі. Параметр whence може приймати значення “set” (зсув визначається з початку файлу), “cur”, ‘end”. Відповідно без параметрів метод повертає поточну позицію в файлі.

### 1.3. Завдання на лабораторну роботу

1.3.1. Ознайомитись з теоретичними відомостями, необхідними для виконання роботи.

1.3.2. Розробити програмне забезпечення у відповідності з індивідуальним завданням, узгодженим з викладачем.

1.3.3. Розробити програму у відповідності з індивідуальним завданням, узгодженим з викладачем, застосовуючи тільки парадигму функціонального програмування (використовувати тільки виклики функцій, визначення анонімних функцій на основі викликів інших функцій, рекурсію; заборонено використовувати цикли, умовні оператори, оператори присвоювання, оператори контролю окрім return).

1.3.4. Розв’язати завдання з пункту 1.3.3, застосовуючи парадигму імперативного програмування. Передбачити, що дані, оброблення яких виконується за завданням, містяться у файлі. Розробити функцію, яка зчитує дані з файлу і представляє їх у вигляді, необхідному для виконання завдання, та застосувати її.

1.3.5. Проаналізувати ризики, які виникли в процесі реалізації програм на основі функціональної й імперативної парадигм і в загальному випадку в процесі мультипарадигменного програмування, та які можуть виникнути в процесі експлуатації відповідних реалізацій програмного забезпечення. Розглянути як можна більше проблем та відповідних ризиків.

1.3.6. Виконати тестування розробленого програмного забезпечення.

1.3.7. Оформити звіт.

1.3.8. Відповісти на контрольні запитання.

### 1.4. Зміст звіту

1.4.1. Мета роботи.

1.4.2. Завдання до роботи.

1.4.3. Тексти розробленого програмного забезпечення.

1.4.4. Результати тестування: вхідні дані та результати роботи програми.

1.4.5. Перелік ризиків з описом та відповідних їм засобів управління.

1.4.6. Висновки, що відображають особисто отримані результати виконання роботи, їх критичний аналіз та порівняння парадигм програмування.

## 1.5. Контрольні запитання

1.5.1. Для яких задач можна використовувати мову програмування Lua?

1.5.2. Що таке динамічна типізація?

1.5.3. Які інструменти можуть використовуватись для програмування мовою Lua?

1.5.4. Які базові типи мови програмування Lua?

1.5.5. Які типи мови програмування Lua є незмінними?

1.5.6. Які особливості функцій в мові програмування Lua?

1.5.7. Яким чином можуть використовуватися таблиці?

1.5.8. Які парадигми програмування підтримує мова Lua?

1.5.9. У чому полягає парадигма функціонального програмування?

1.5.10. У чому полягає механізм сміттяра?

# 2. ЛАБОРАТОРНА РОБОТА № 2
# ПРОТОТИПНЕ ПРОГРАМУВАННЯ

## 2.1. Мета роботи

Ознайомитись з основами прототипного програмування, навчитися використовувати його для розроблення програмного забезпечення, інтегруючи в процес управління ризиками.

## 2.2. Короткі теоретичні відомості

### 2.2.1 Прототипування як метод управління ризиками

У програмній інженерії **прототипування** означає створення часткових програмно-апаратних реалізацій моделей програмного забезпечення.

Програмні прототипи конструюються для візуалізації системи або її частини для замовників з метою отримання їх відгуку. Прототип представляє демонстраційну систему – “нашвидкуруч та грубо” зроблену робочу модель рішення, яка надає графічний користувацький інтерфейс та моделює поведінку системи з ініціювання користувачем різноманітних подій.

Складність та варіативність сучасних графічних користувацьких інтерфейсів робить прототипування обов’язковим елементом розроблення програмного забезпечення. Прототипи дозволяють оцінювати корисність та реалізованість системи до початку її розроблення.

У загальному випадку прототип – доволі ефективний спосіб виявлення вимог, які важко отримати від замовника за допомогою інших засобів.

Якщо ризик проекту пов’язаний із застосуванням певного технологічного підходу, тобто певна технологія не застосовувалася у такій задачі раніше і невідомо, чи забезпечить вона належні виробничі характеристики, то варто створити архітектурний прототип, який демонструє можливість використання цієї технології.

Якщо ж основний ризик в проекті становить користувацький інтерфейс і основною проблемою є уточнення вимог користувача, потрібен прототип вимог.

Існує два основні різновиди прототипів:

– одноразовий прототип;
– еволюційний прототип.

Використання прототипів під час моделювання та розроблення програмних систем базується на низці переваг, які надає прототипування. Водночас прототипування як модель життєвого циклу ПЗ нерідко зазнає критики, обґрунтованої певними недоліками. Переваги й недоліки прототипування наведено нижче.

Переваги прототипування:
– мінімізація часових і фінансових витрат;
– вдосконалене залучення кінцевого користувача до процесу створенн;
– навчання користувача.

Недоліки прототипування:
– недостатній аналіз;
– непорозуміння з користувачем, який не розуміє різниці між прототипом та завершеною системою;
– неправильне використання часу розробником;
– зацикленість на прототипі.

### 2.2.2 Прототипне програмування в Lua

Дана мова використовує безкласову парадигму об’єктно-орієнтованого програмування, де кожен об’єкт є деяким прототипом. Нові екземпляри можуть бути утворені шляхом зміни властивостей існуючих екземплярів. Такий стиль програмування називається прототип-орієнтованим.

Для того щоб створити функцію, яка будете виконувати роль методу об’єкту:

Lua

```
Lib = {}
Lib.foo = function (x,y) return x + y end
Lib.goo = function (x,y) return x - y end
```

Для створення конструкторів:

Lua

```
Lib = {
foo = function (x,y) return x + y end,
goo = function (x,y) return x - y end
}
```

Для того щоб визначити операції таблиці:

Lua

```
Account = {balance = 0}
function Account.withdraw (v)
Account.balance = Account.balance - v
end
```

Для того щоб використати даний метод для інших об’єктів:

Lua

```
a2 = {balance=0, withdraw = Account.withdraw}
...
a2.withdraw(a2, 260.00)
```

Для того щоб визначити метод, в якому можна буде отримати доступ до даних об’єкта за допомогою покажчика self, необхідно використати символ двокрапки:

Lua

```
function Account:withdraw (v)
self.balance = self.balance - v
end
```

Виклик такого методу відбувається за допомогою аналогічного прийому:

Lua

```
a:withdraw(100.00)
```

Прототип-орієнтоване програмування – стиль об'єктно-орієнтованого програмування, за якого відсутнє поняття класу, а повторне використання (наслідування) відбувається шляхом клонування існуючого екземпляру об'єкту, який виступає в ролі прототипу.

Якщо є два об'єкти a і b, то для того щоб перетворити b на прототип a, необхідно скористатися наступним механізмом:

Lua

```
setmetatable(a, {__index = b})
```

Для забезпечення захисту даних кожний об'єкт представляється у вигляді двох таблиць: одна – для станів, інша – для операцій або інтерфейсу.

Стандартна бібліотека функцій операційної системи os включає функції для маніпуляції з файлами, визначення поточної дати і часу тощо. Інші функції для роботи з операційною системою можна підключити за допомогою додаткових бібліотек, наприклад:

– luasocket – надає функції підтримки мережі;

– LuaFileSystem – надає функції для базових маніпуляцій з директоріями та файловими атрибутами;

– iup – кросплатформна бібліотека інструментів графічного інтерфейсу користувача.

Основні функції станбібліотеки os:

– time – повертає поточну дату і час у вигляді числа (якщо передати у якості параметра таблицю з датою і часом (параметри year, month, day, hour, min, sec), то повертає задані значення у вигляді

числа);

– date – зворотна до time, перетворює числове значення на рядкове, що представляє дату і час у заданому форматі (перший параметр функції);

– clock – повертає кількість секунд часу центрального процесору;

– exit – перериває виконання програми;

– getenv – повертає значення змінних середовища;

– execute – запускає системні команди;

– setlocale – встановлює поточну місцеву специфіку.

## 2.3. Завдання на лабораторну роботу

2.3.1. Ознайомитись з теоретичними відомостями, необхідними для виконання роботи.

2.3.2. Визначити функціональні вимоги до програмного забезпечення у відповідності з індивідуальним завданням, узгодженим з викладачем.

2.3.3. Визначити систему об’єктів, які можуть використовуватися в заданій предметній області для реалізації індивідуального завдання. У процесі проектування системи дотримуватися виконання вимоги її оптимальності для подальшої її модифікації у процесі еволюційного прототипування.

2.3.4. Розробити прототип комп’ютерної гри за допомогою мови програмування Lua у відповідності з індивідуальним завданням та спроектованою системою об’єктів на основі прототип-орієнтованого програмування.

2.3.5. Оформити звіт.

2.3.6. Відповісти на контрольні запитання.

## 2.4. Зміст звіту

2.4.1 Мета роботи.

2.4.2 Концепція комп’ютерної гри.

2.4.3 Функціональні вимоги до програмного забезпечення.

2.4.4 Система об’єктів.

2.4.5 Текст програми.

2.4.6 Результати роботи програми.

2.4.7 Висновки, що містять відповіді на контрольні запитання, а також відображають результати виконання роботи та їх критичний аналіз.

## 2.5. Контрольні запитання

2.5.1 У чому полягає прототипне програмування?

2.5.2 Чим відрізняється прототипне програмування від об’єктно-орієнтованого?

2.5.3 Чи можна змінювати прототип в процесі роботи програми?

2.5.4 Для чого використовуються програмні прототипи?

2.5.5 Які види програмних прототипів існують?

2.5.6 Яким чином пов’язане прототипування з ризиками програмного забезпечення?

2.5.7 Які переваги протипування?

2.5.8 Які недоліки прототипування?

2.5.9 Коли варто застосовувати прототипування?

2.5.10 Чи може бути інтегровано протипування в інші моделі життєвого циклу програмного забезпечення?

# 3. ЛАБОРАТОРНА РОБОТА № 3
# РИЗИКИ ПОВТОРНОГО ВИКОРИСТАННЯ КОДУ

## 3.1. Мета роботи

Визначити основні ризики, які виникають у процесі розроблення програмного забезпечення на основі повторного використання коду, навчитися розробляти ігрові додатки на основі використання існуючих движків та приймати рішення з управління даними ризиками.

## 3.2. Короткі теоретичні відомості

Повторне використання коду в процесі розробки програмного забезпечення пов'язано з ризиками. Ризики можуть виникати зокрема через те, що на вивчення коду може не відводитись достатньо часу, а також через те, що відсутнє розуміння області застосування коду. Для абсолютного виконання всіх поставлених завдань за повторного використання коду необхідно провести детальний аналіз коду, виділити ключові для повторного використання частини.

**Ігровий движок** – це центральний програмний компонент комп'ютерних ігрових додатків з графікою, що обробляється в реальному часі.

Движок зазвичай має наступні підсистеми:

– графічна підсистема;

– підсистема вводу;

– звукова підсистема;

– системне ядро.

Існує багато ігрових движків, які можуть бути застосовані в процесі розробки мобільних ігор або браузерних ігор. Однак вибір має бути достатньо обґрунтованим, з урахуванням всіх ризиків та виконаним після достатньо детального аналізу.

**Corona SDK** – це кросплатформний фреймворк для розроблення мобільних додатків під Android, iOS. В якості мови програмування використовується мова Lua. Corona SDK є 2d-движком для створення ігор, хоча може застосовуватися і для розроблення деяких бізнес-додатків. Архітектура движка наведена на рис. 3.1.

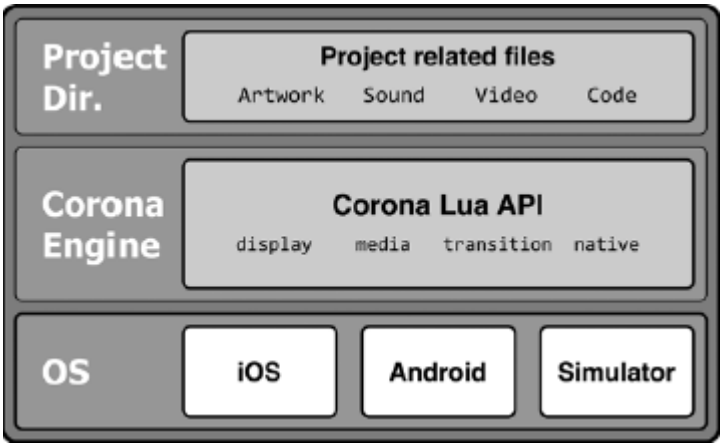



Рисунок 3.1 – Архітектура Corona SDK

Движок Corona складається з базових бібліотек Lua та інших бібліотек, відповідальних за графіку, аудіо, відео тощо, зокрема:

– analytics – всі функції, що стосуються використання аналітики;

– audio – всі функції, що стосуються аудіо;

– crypto – всі функції, що стосуються криптографії;

– display – всі функції, що стосуються відображення елементів;

– facebook – всі функції, що стосуються Facebook;

– graphics – всі функції, що стосуються графіки та допомагають відображати елементи;

– lfs – бібліотека файлової системи Lua для роботи з файлами та теками;

– media – медіа-функції для відображення аудіо та відео;

– native – функції для роботи з функціональністю пристроїв;

– network – функції, що стосуються мережевої комунікації та комунікації даних;

– physics – бібліотеки фізики (Box2D);

– socket – бібліотека для комунікації сокетів TCP та UDP;

– sqlite3 – функції для доступу до баз даних SQLite3;

– storyboard – функції, що стосуються розкадрування;

– timer – функції для роботи з таймером;

– widget – функції для створення віджетів.

Кожна бібліотека містить відповідний набір функцій. Розглянемо деякі з них.

Бібліотека display зокрема містить наступні функції:

– display.newCircle( xCenter, yCenter, radius ) – створює коло з заданими координатами центру та радіусом;

– display.newImage( [parent,] filename [,baseDir] [,x,y] [,isFullResolution]) – відтворює зображення на екрані з заданого файлу;

– display.newLine( [parentGroup,] x1, y1, x2, y2 [, x3, y3, ... ] ) – малює лінії з однієї точки в іншу;

– display.newRect( x, y, width, height ) – створює прямокутний об’єкт;

– display.newText( options ) – створює текстовий об’єкт;

– display.remove( object ) – вилучає об’єкт або групу;

– display.newGroup() – створює групу, в яку можна добавляти або вилучати об’єкти.

Приклад використання даних функцій:

**Corona SDK**

```
local square = display.newRect( myGroup, 0, 0, 100, 100 )  --
red square is at the bottom
square:setFillColor( 1, 0, 0 )
local circle = display.newCircle( myGroup, 80, 120, 50 )  --
green circle is in the middle
circle:setFillColor( 0, 1, 0 )
local rect = display.newRect( myGroup, 0, 0, 120, 80 )  --blue
rectangle is at the top
rect:setFillColor( 0, 0, 1 )
```

Бібліотека audio зокрема містить наступні функції:

– audio.loadSound( audiofileName [, baseDir ] ) – завантажує файл безпосередньо у пам’ять та повертає посилання на аудо-дані;

– audio.setMaxVolume( volume, options ) – встановлює максимальний рівень гучності;

– audio.setMinVolume( volume, options ) – встановлює мінімальний рівень гучності;

– audio.setVolume( volume [, options ] ) – встановлює рівень гучності;

– audio.getVolume( { channel=c } ) – повертає рівень гучності;

– audio.loadStream( audioFileName [, baseDir ] ) – відкриває файл для читання в потоковому режимі;

– audio.pause( [channel] ) – встановлює на паузу відтворення в аудіо-каналі;

– audio.play( audioHandle [, options ] ) – розпочинає відтворення в аудіо-каналі;

– audio.stop( [channel] ) – зупиняє відтворення в аудіо-каналі;

Приклад:

**Corona SDK**

```
local soundTable = {
   chimeSound = audio.loadSound( "chime.wav" ),
   bellSound = audio.loadSound( "bell.wav" ),
   buzzSound = audio.loadSound( "buzz.aac" ),
   clickSound = audio.loadSound( "click.aac" )
}
```

Бібліотека media зокрема містить наступні функції:

– media.playVideo( path [, baseSource ], showControls, listener ) – відтворює відео за вказаним шляхом;

– media.playSound( soundfile [, baseDir] [, onComplete] ) – відтворює звук;

– media.stopSound() – закінчує відтворення звуку;

– media.pauseSound() – зупиняє відтворення звуку;

– media.show( mediaSource, listener ) – відкриває платформнозалежний інтерфейс до камери пристрою або фото бібліотеки;

– media.newRecording( [path] ) – створює об’єкт для аудіо запису;

Бібліотека native зокрема містить наступні функції:

– native.showPopup( name, options ) – відображує вспливаюче вікно, стандартне для вказаного сервісу;

– native.getFontNames() – повертає масив доступних шрифтів на пристрої;

– native.showAlert( title, message [, { buttonLabels } [, listener] ] ) – відображає попереджуюче вікно з однією або декількома кнопками, використовуючи стандартне для даного пристрою попереджуюче вікно;

– native.cancelAlert( alert ) – програмно відхиляє попереджуюче вікно;

– native.requestExit() – закриває вікно додатку, не перериваючи процес;

– native.newTextBox( centerX, centerY, width, height ) – створює багаторядкове текстове поле;

– native.newFont( name [, size] ) – створює об’єкт шрифт, який можна використовувати, щоб визначити шрифти у вбудованих текстових полях;

– native.setKeyboardFocus( textField ) – встановлює фокус на текстовому полі та відображає або приховує клавіатуру.

Бібліотека physics зокрема містить наступні функції:

– physics.addBody( object, [bodyType,] [params] ) – перетворює об’єкт на фізичний об’єкт, специфічні властивості якого симулюються;

– physics.removeBody( object ) – вилучає створену фізичні імітацію об’єкта без вилучення самого об’єкта;

– physics.setGravity( gx, gy ) – імітує гравітацію;

– physics.start( noSleep ) – розпочинає імітацію фізики об’єктів.

Всі об’єкти, намальовані на екрані за допомогою бібліотеки display, є display object, що зокрема мають наступні властивості: object.x, object.y, object.width (ширина), object.height (висота), object.rotation (обертання), object.isVisible (чи видимий об’єкт на екрані) та методи, які виконують повертання object:rotate(), масштабування object:scale(), відправлення візуально на задній фон серед елементів групи object:toBack(),на передній фон object:toFront().

Після інсталяції Corona SDK в системі з’явиться симулятор (рис. 3.2), з головного вікна якого можна створити проект, відкрити існуючий та отримати доступ до прикладів проектів і документації.

Після створення проекту буде створено теку, що містить деякий набір файлів звуків і зображень, головний файл проекту main.lua, конфігураційний файл config.lua, де задається розмір екрану.

Для того щоб створювати проекти для пристроїв під керуванням мобільної операційної системи Android, необхідно встановити Java SDK. Пункт меню Build for Android дозволить виконати відповідну зборку. В процесі зборки необхідно визначити зокрема тип магазину додатків, який буде використовуватися (або не буде). У результаті буде створено файл *.apk, який можна інсталювати на пристрої під управлінням операційної системи Android.

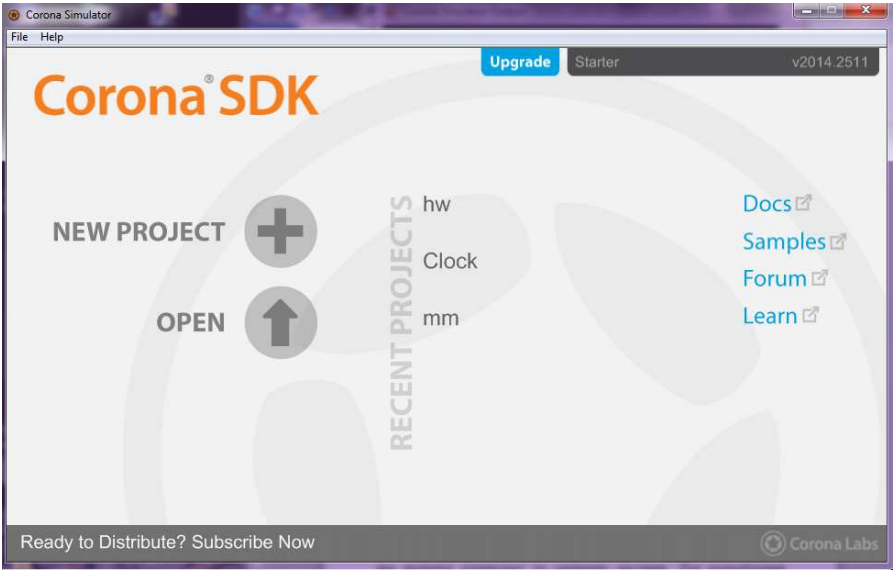



Рисунок 3.2 – Вікно симулятора Corona

Для Corona існують також деякі IDE, зокрема Corona ® Cider, яке включає розширюваний зневаджувач, провідник, автокомпіляцію, точки відновлення в реальному часі, підтримку розділу екрану, попередній перегляд тощо.

Розглянемо інші движки, які також можуть використовуватися для розроблення ігрових додатків.

**Gideros Studio** – це кросплатформене середовище розробки ігор для iOS та Android. В якості мови програмування використовується Lua. Працює під управлінням операційних систем Windows, Mac OS, Linux. Графічні можливості ґрунтуються на OpenGL, OpenAL, Box2D. Включає у свій склад Studio (IDE для розробки), інструмент Player, який надає можливість запускати проект для відлагодження на власному персональному комп'ютері, TexturePacker, що дає можливість упакування різноманітних текстур в атласи, FontCreator, що дозволяє створювати растрові шрифти для додатку з *.ttf,*.otf,*.ttc. Архітектура даного движка наведена на рис. 3.3.

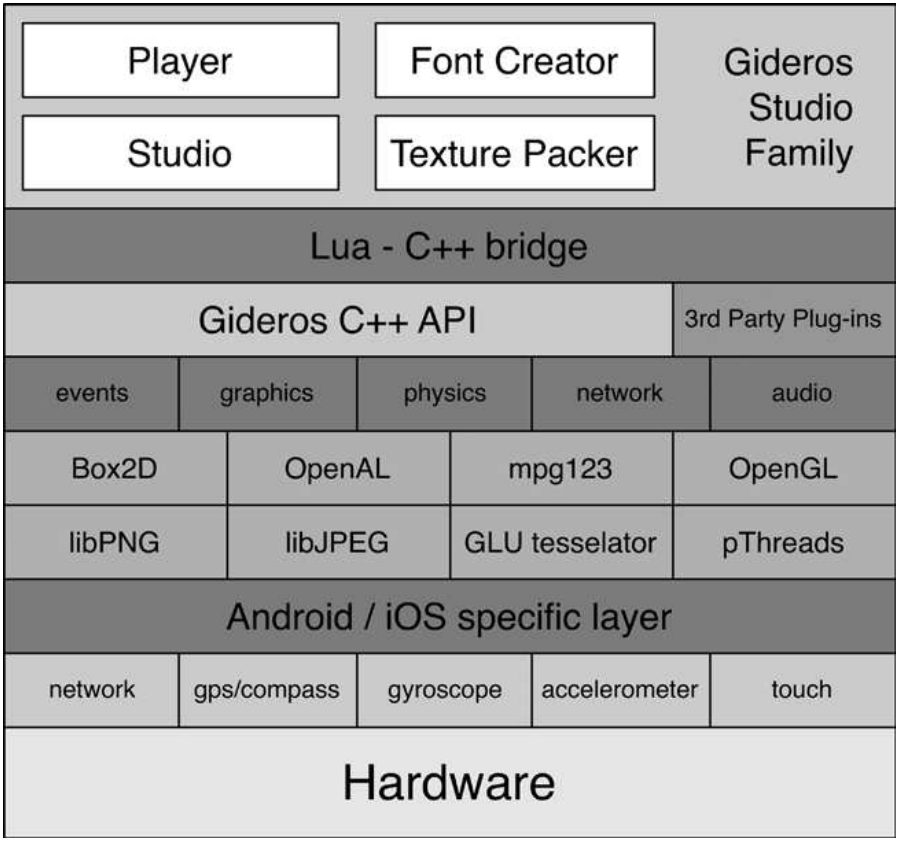



Рисунок 3.3 – Архітектура Gideros Studio

**Unity** – це багатоплатформний інструмент для розробки дво- і тривімирних додатків та ігор під операційні системи Windows, OS X, Android, iOS, Linux, а також деякі гральні консолі та web.

Використовує мови програмування C#, JavaScript, Boo. Скриптова система ігрового движка виконана на Mono (вільний відкритий проект з реалізації .NET Framework).

Увесь процес розробки відбувається у відповідній IDE, яке включає редактор сцен, редактор ігрових об'єктів та редактор скриптів. Сервер ресурсів Unity – це повнофункціональне рішення для контролю версій для всіх ігрових скриптів і ресурсів, що управляється базою даних PostgreSQL.

**Moai** – не тільки ігровий движок, але й повноцінна платформа, яка включає клієнтську та серверну частини. Для розроблення додатків (як клієнтської, так і северної частини) використовується мова Lua. Має відкритий код, за рахунок чого його можна розширювати необхідним чином. Повністю безкоштовний. Дозволяє розробляти 2D та 3D ігри для iOS, Android, Windows, Mac OS X та Chrome.

**Cocos2d-x** – це кросплатформний ігровий 2D движок, створений як копія cocos2d для iPhone. Cocos2d використовує Object C та може використовуватися тільки для iOS. Використовує мови програмування C++, C#, JavaScript та Lua. Підтримує мобільні платформи iOS та Android.

**LÖVE** (рис. 3.4) – це фреймворк з відкритим кодом, який використовує Lua та OpenGL. Доступний на операційних системах Windows, Mac та Unix. Дозволяє створити десктопні розширення для додатків, які можуть комунікувати з мобільними додатками.

На відміну від Cocos2d-x, в якому всі об'єкти знаходяться в шарах, а шари знаходяться в сценах, у Love2D немає ні шарів, ні сцен. За допомогою шарів можна створювати гнучкі додатки більш простими методами. У Cocos2d-x все представлено об'єктами і, щоб якийсь елемент з'явився на екрані, потрібно додати його на сцену, а у Love2d все представлено змінними, тобто кожний кадр потрібно малювати те, що повинно з'явитись.

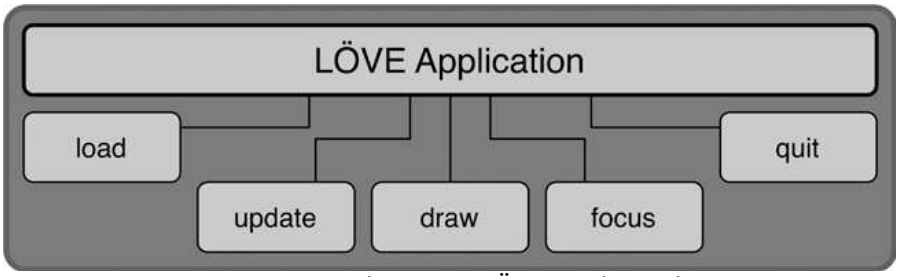



Рисунок 3.4 – Базова архітектура LÖVE та функцій зворотного виклику

### 3.3. Завдання на лабораторну роботу

3.3.1. Ознайомитись з теоретичними відомостями, необхідними для виконання роботи.

3.3.2. Обрати движок, який дозволяє реалізувати концепцію та функціональність комп'ютерної гри відповідно до індивідуального завдання з лабораторної роботи № 2.

3.3.3. Розробити програмне забезпечення, яке реалізує всі функціональні вимоги, визначені індивідуальним завданням. Програмне забезпечення обов'язково має відповідати вимогам безпеки даних та програмного коду.

3.3.4. Визначити ризики, які було ідентифіковано і реалізовано в процесі розробки програмного забезпечення та ризики, які можуть реалізуватись під час подальших етапів життєвого циклу програмного забезпечення.

3.3.5. Визначити рішення, прийняті для управління ризиками.

3.3.6. Оформити звіт.

3.3.7. Відповісти на контрольні запитання.

### 3.4. Зміст звіту

3.4.1. Мета роботи.

3.4.2. Завдання на роботу.

3.4.3. Опис та обґрунтування вибору движка.

3.4.4. Текст програми.

3.4.5. Інтерфейс роботи з програмою в декількох режимах.

3.4.6. Результати роботи програми.

3.4.7. Таблиця, що містить назву ризику, короткий опис, його ступінь впливу та відповідні прийняті рішення.

3.4.8. Таблиця, що містить назву ризику, короткий опис, ситуація, в якій його може бути реалізовано, та перелік рішень, які можуть бути прийняті.

3.4.9. Висновки, що містять відповіді на контрольні запитання, а також відображають результати виконання роботи та їх критичний аналіз.

### 3.5. Контрольні запитання

3.5.1. Що таке ігровий движок?

3.5.2. Для чого використовуються ігрові движки?

3.5.3. Які частини комп'ютерної гри зазвичай дозволяє контролювати ігровий движок?

3.5.4. Які ризики характерні для повторного використання коду?

3.5.5. Які ігрові движки ви знаєте і які особливості вони мають?

3.5.6. Яка архітектура обраного ігрового движка?

3.5.7. Які бібліотеки має обраний движок?

3.5.8. Які функції надає обраний движок?

3.5.9. Чи варто використовувати код повторно?

3.5.10. Які найголовніші фактори успішного повторного використання програмного коду?

# 4. ЛАБОРАТОРНА РОБОТА № 4
# РОЗРОБЛЕННЯ КОМПОНЕНТНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЯК МЕТОД УПРАВЛІННЯ РИЗИКАМИ

## 4.1. Мета роботи

4.1.1 Ознайомитись з принципами компонентно-орієнтованого програмування.

4.1.2 Навчитися використовувати компонентні технології в якості методу управління ризиками в процесі розроблення програмного забезпечення.

## 4.2. Короткі теоретичні відомості

Розроблення та використання компонентів є основою компонентно-орієнтованого програмування. Дана парадигма програмування реалізується в багатьох технологіях, зокрема в мові програмування Java за допомогою технології JavaBeans, відомі також розповсюджені рішення CORBA, COM, SOAP. Платформа .NET реалізує компонентно-орієнтований підхід, забезпечуючи його використання для підтримуваних мов програмування.

У загальному випадку **компонентом** .NET є наступний приклад створення класу .NET:

C#

```
public class MyClass
{
   public string GetMessage()
   {
      return "Hello";
   }
}
```

Компоненти .NET можуть належати збиранням на основі EXE (збирання застосувань) або DLL (бібліотечні збирання).

Для того щоб створити нове бібліотечне збирання C# у Visual Studio, необхідно скористатися меню File → New → Project... Після того як на екрані з'явиться діалогове вікно New Project (рис. 4.1), необхідно обрати Visual C# в якості типу проекту, після чого обрати Windows, а у шаблонах – Class Library. Необхідно також задати розташування проекту та ввести ім'я (наприклад, MyClassLib).

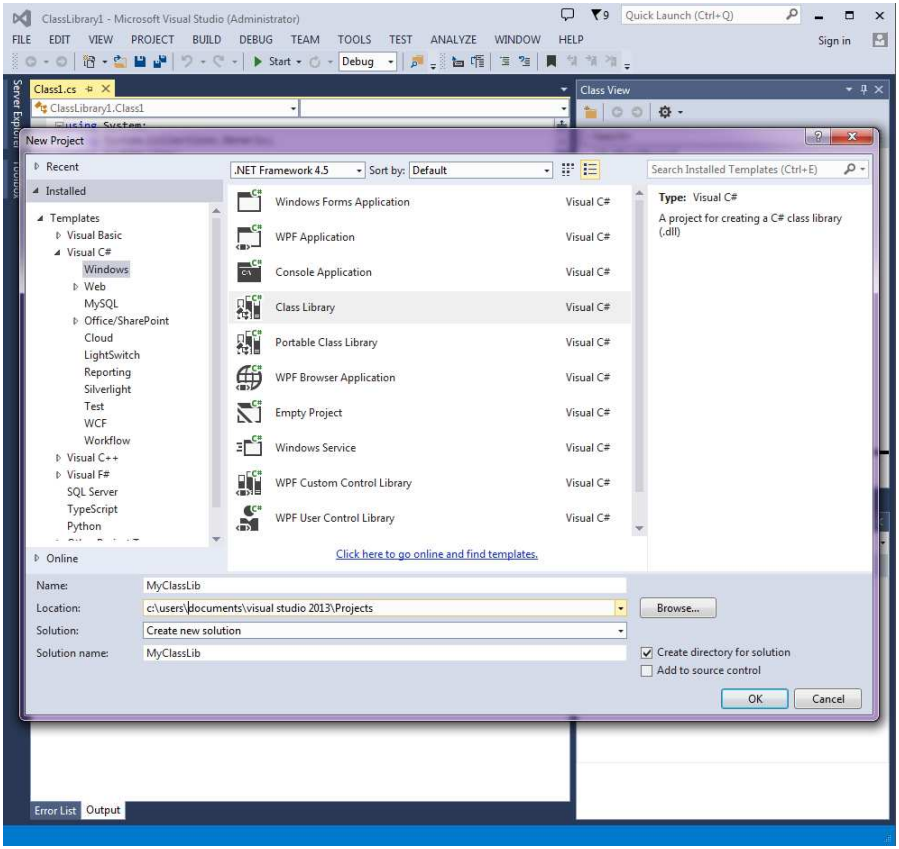



Рисунок 4.1 – Створення проекту Class Library у Microsoft Visual Studio

Після створення проекту за допомогою меню можна додати новий клас, інтерфейс, компонентний клас тощо. Для цього

призначений пункт меню Project → Add Class, який виводить на екран діалогове вікно Add New Item, зображене на рис. 4.2.

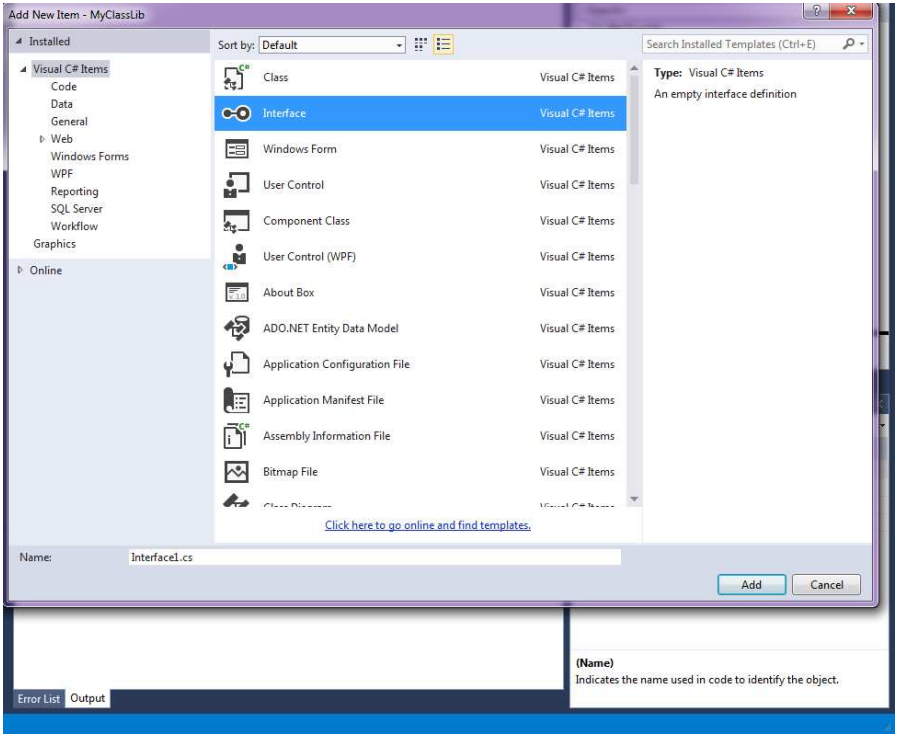



Рисунок 4.2 – Додавання нового класу в проект

Для того щоб використати розроблений компонент у клієнтському застосуванні, необхідно виконати наступні дії:

а) створити новий проект Windows Application Form;

б) додати посилання на створене раніше збирання, обравши в меню Project → Add Reference…, після чого на екрані з'явиться діалогове вікно Reference Manager, у якому необхідно обрати створене збирання (обрати варіант Browse та вказати шлях до збирання);

в) додати директиву using для простору імен AssemblyDemo namespace;

г) використати в тексті програми розроблений компонент як звичайний клас, наприклад:

**C#**

```
using System;
using System.Windows.Forms;
using AssemblyDemo;

  partial class ClientForm : Form
  {
    void OnClicked(object sender,EventArgs e)
    {
      MyClass obj = new MyClass(  );
      string message = obj.GetMessage(  );
      MessageBox.Show(message);
    }
  }
```

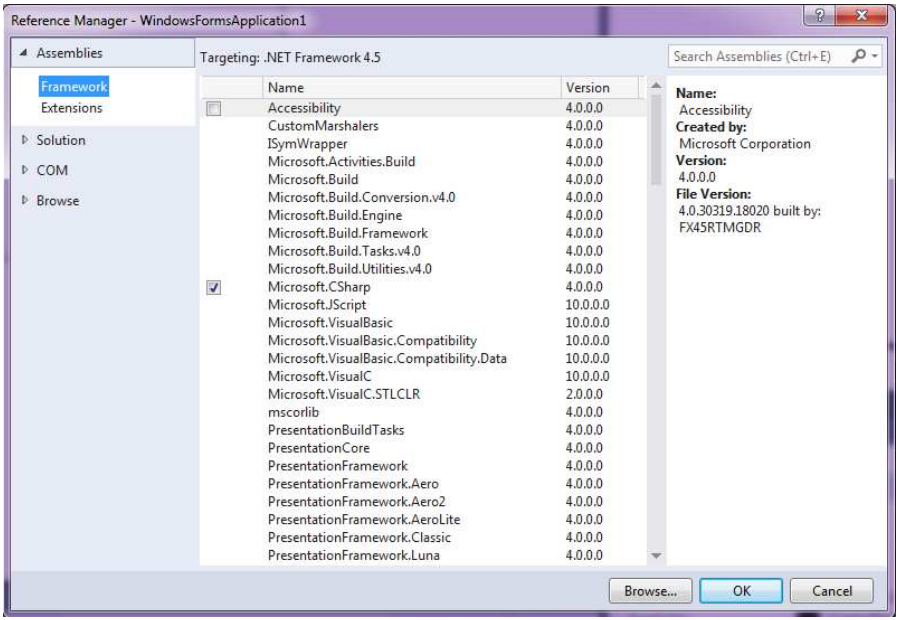



Рисунок 4.3 – Додавання збирання до клієнтського проекту

### 4.3. Завдання на лабораторну роботу

4.3.1. Ознайомитись з теоретичними відомостями, необхідними для виконання роботи.

4.3.2. Розробити програмне забезпечення у відповідності з індивідуальним завданням, узгодженим з викладачем. Програмне забезпечення обов'язково має відповідати вимогам безпеки даних та програмного коду.

4.3.3. Визначити ризики, які було ідентифіковано і реалізовано в процесі розробки програмного забезпечення і ризики, які можуть реалізуватись під час подальших етапів життєвого циклу програмного забезпечення, та відповідні їм засоби управління ризиками.

4.3.4. Оформити звіт.

4.3.5. Відповісти на контрольні запитання.

### 4.4. Зміст звіту

4.4.1. Мета роботи.

4.4.2. Завдання на роботу.

4.4.3. Архітектура програмного забезпечення.

4.4.4. Текст розробленого програмного забезпечення.

4.4.5. Опис ризиків та засобів управління ними.

4.4.6. Результати тестування: вхідні дані та результати роботи програми.

4.4.7. Висновки, що містять відповіді на контрольні запитання, а також відображають результати виконання роботи, їх критичний аналіз та аналіз досліджених засобів управління ризиками.

### 4.5. Контрольні запитання

4.5.1 Що таке компонент?

4.5.2 З чого складається архітектура програмного забезпечення, розробленого на основі компонентів?

4.5.3 Які технології розроблення компонентного програмного забезпечення існують?

4.5.4 Що таке фреймворк .NET?

4.5.5 Для чого необхідні інтерфейси?

4.5.6 З яких етапів складається розроблення програмного забезпечення на основі компонентів мовою програмування C# на платформі .NET?

4.5.7 Яким чином застосування компонентно-орієнтованої парадигми програмування впливає на ризики розроблення програмного забезпечення?

4.5.8 Яким чином компоненти .NET розташовуються в пам’яті?

4.5.9 Що таке та яким чином працює CLR?

4.5.10 У чому полягає модель безпеки .NET?

# 5. ЛАБОРАТОРНА РОБОТА № 5
# УПРАВЛІННЯ РИЗИКАМИ ЯКОСТІ ПРОГРАМНОГО КОДУ

## 5.1. Мета роботи

5.1.1 Ознайомитись з основними засобами, які можуть використовуватися для управління ризиками якості програмного коду.

5.1.2 Навчитися застосовувати на практиці прийоми рефакторингу та оптимізації коду.

## 5.2. Короткі теоретичні відомості

### 5.2.1 Основні прийоми рефакторинга та оптимізації коду

**Рефакторинг** – підхід до розроблення програмного забезпечення з удосконаленням коду шляхом послідовності семантично коректних перетворень, зміна внутрішньої структури програмного забезпечення для полегшення розуміння коду і здешевлення модифікації.

Підстави для проведення рефакторингу:

– код дублюється;

– метод занадто довгий;

– цикл занадто довгий або рівень вкладеності тіла циклу занадто великий;

– клас має погану зв’язність;

– інтерфейс класу не формує узгоджену абстракцію;

– метод має занадто багато параметрів;

– окремі частини класу змінюються незалежно від інших частин;

– під час зміни програми потрібно паралельно змінювати декілька класів;

– доводиться паралельно змінювати декілька ієрархій наслідування;

– споріднені дані, які використовуються разом, не організовані в клас;

– назва класу чи методу має ім'я, яке недостатньо точно відповідає змісту;

– занадто поширене використання глобальних змінних;

– складний код пояснюється за допомогою коментарів;

– підклас використовує тільки деяку частину методів батьківського

класу;

– об’єкт-посередник нічого не виконує;

– дані–члени класу відкриті.

Напрямки проведення рефакторингу можуть включати наступні:

– рефакторинг даних;

– рефакторинг на рівні окремих операторів;

– рефакторинг на рівні окремих методів;

– рефакторинг реалізації класів;

– рефакторинг інтерфейсів класів;

– рефакторинг на рівні системи.

### 5.2.2 Засоби рефакторингу в сучасних IDE

Рефакторинг коду в Microsoft Visual Studio можна виконати за допомогою пункту контекстного меню Refactor (рис. 5.1). Для кожного з доступних засобів рефакторингу запропоновано також відповідну комбінацію клавіш, представлену на рис. 5.1.

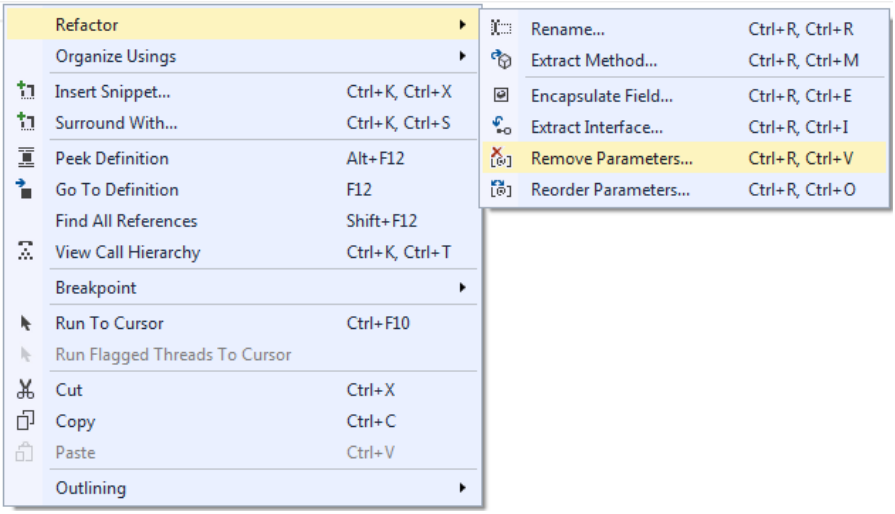



Рисунок 5.1 – Меню рефакторингу в Microsoft Visual Studio 2013

Засоби рефакторингу коду в Microsoft Visual Studio включають:

– Extract Method (Витягання методу) – дозволяє визначати новий

метод на основі обраних операторів програмного коду;

– Encapsulate Field (Інкапсуляція поля) – дозволяє перетворювати загальнодоступне поле в приватне, інкапсульовану в форму властивість C#;

– Extract Interface (Витягання інтерфейсу) – дозволяє визначати новий тип інтерфейсу на основі набору існуючих членів типу;

– Reorder Parameters (Переупорядкування параметрів) – дозволяє змінити порядок слідування аргументів у члені класу;

– Remove Parameters (Вилучення параметрів) – дозволяє вилучати деякий аргумент з поточного переліку параметрів;

– Rename (Перейменування) – дозволяє перейменовувати використовуваний в коді метод, поле, локальну змінну, простір імен, властивості та типи у всьому проекті.

Для витягання методу необхідно в існуючій програмі виділити відповідний фрагмент коду та обрати з контекстного меню відповідний пункт меню, в результаті чого на екрані з'явиться діалогове вікно, представлене на рис. 5.2.

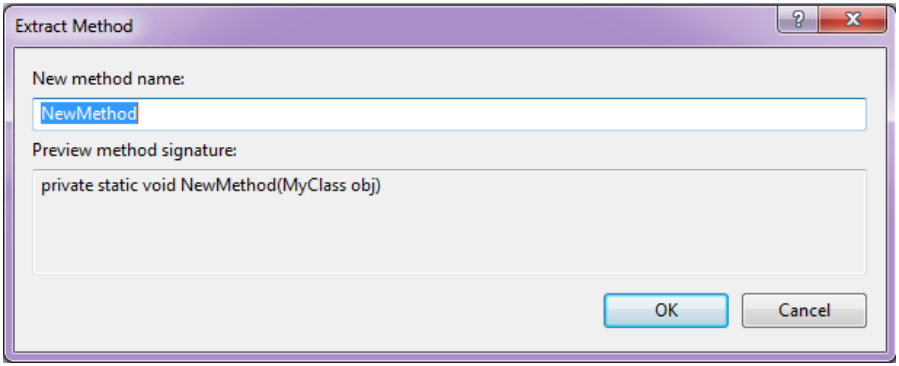



Рисунок 5.2 – Вікно витягання методу в Visual Studio 2013

Для того щоб використати рефакторинг за допомогою перейменування, необхідно встановити курсор на відповідний ідентифікатор та викликати аналогічно попередньому способу діалогове вікно перейменування (рис. 5.3).

Для інкапсуляції поля необхідно встановити курсор на рядок оголошення поля та обрати відповідний пункт контекстного меню. У результаті на екрані з'явиться діалогове вікно, зображене на рис. 5.4.

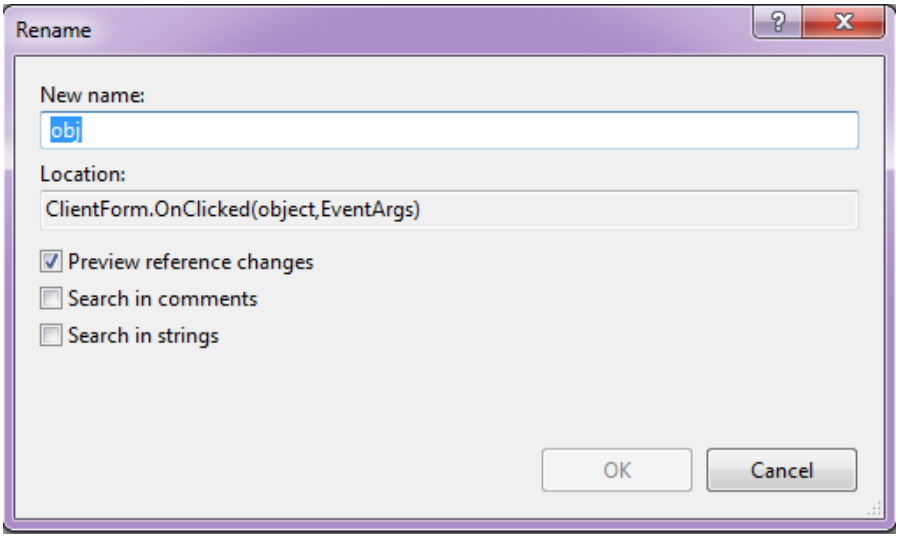



Рисунок 5.3 – Вікно перейменування в Visual Studio 2013

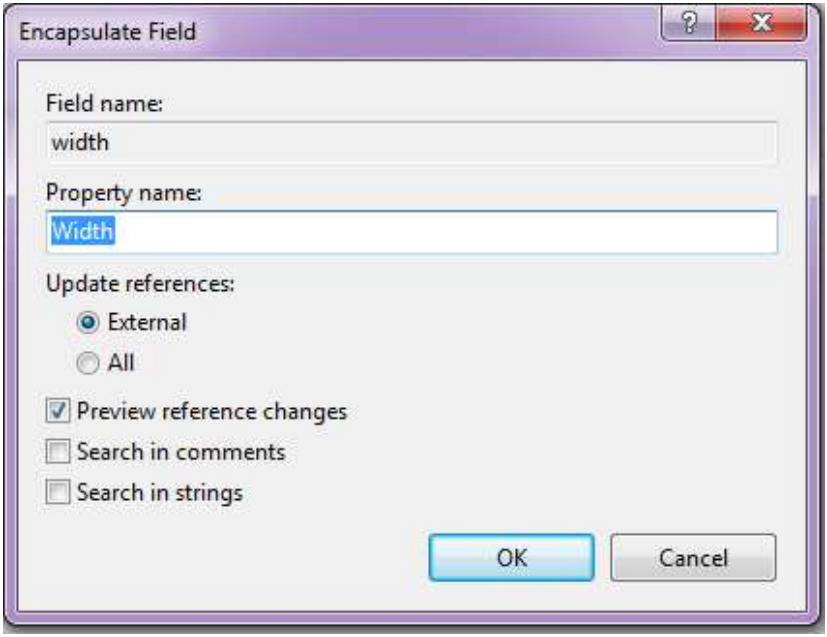



Рисунок 5.4 – Вікно інкапсуляції поля в Visual Studio 2013

Для витягання інтерфейсу необхідно розташувати курсор на методі класу та обрати пункт контекстного меню Refactor → Extract Interface. Після цього на екрані з'явиться діалогове вікно витягання інтерфейсу (рис. 5.5).

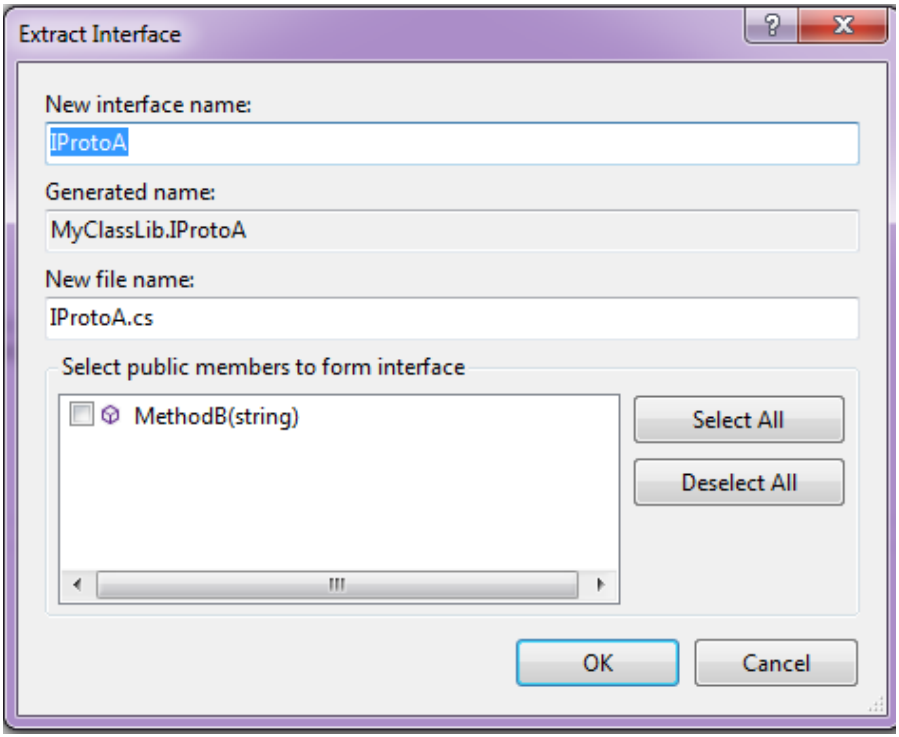



Рисунок 5.5 – Вікно витягання інтерфейсу в Visual Studio 2013

На рис. 5.6 наведено приклад діалогового вікна вилучення параметрів з методів, індексаторів або делегатів.

На рис. 5.7 наведено приклад діалогового вікна пере упорядкування параметрів, яке дозволяє змінити порядок слідування параметрів методів, індексаторів та делегатів і застосувати ці зміни у всіх викликах даного методу тощо.

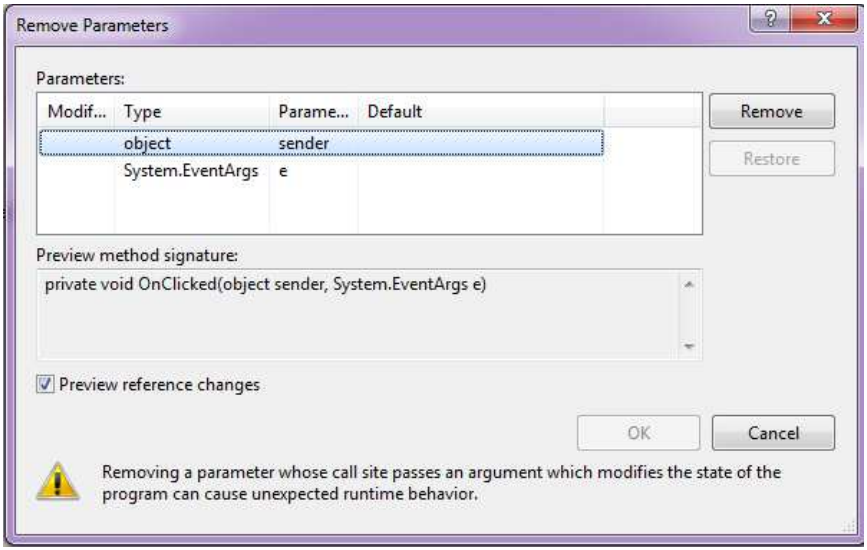



Рисунок 5.6 – Вікно вилучення параметрів у Visual Studio 2013

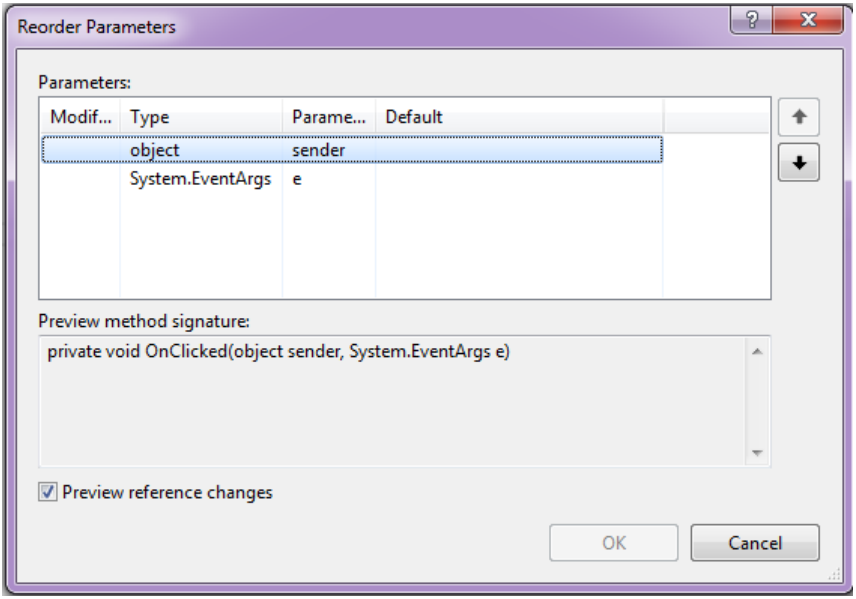



Рисунок 5.7 – Діалогове вікно пере упорядкування параметрів в Microsoft Visual Studio 2013

Засоби рефакторингу в IDE Eclipse дозволяють виконувати наступні дії:

– Rename… – перейменовує обраний елемент та виправляє всі посилання на нього (і в інших файлах, якщо можливо);

– Move… – переміщує обраний елемент та виправляє всі посилання на нього (і в інших файлах, якщо можливо);

– Change Method Signature… – змінює імена, типи, порядок параметрів та оновлює всі посилання на відповідний метод;

– Extract Method… – створює новий метод, що містить вибрані оператори, та замінює вибір посиланням на новий метод;

– Extract Local Variable… – створює нову змінну, присвоюючи їй значення вибраного виразу та замінює вибір посиланням на нову змінну;

– Extract Constant… – створює статичне поле з вибраного виразу і замінює посилання на поле, опційно перезаписуючи інші входження даного виразу;

– Inline… – вбудовуваний метод, змінна або константи;

– Convert Local Variable to Field… – перетворює локальну змінну на поле;

– Convert Anonymous Class to Nested… – перетворює анонімний внутрішній клас на клас-член;

– Move Type to New File… – створює новий файл для типу вибраного члена, оновлюючи всі посилання;

– Extract Superclass… – витягує загальний суперклас з множини типів класів одного рівня;

– Extract Interface… – створює новий інтерфейс з множиною методів та імплементує інтерфейс для вибраного класу;

– Use Supertype Where Possible… – заміняє всі входження типу одним з його супертипів;

– Push Down… – переміщує множину методів та полів з класу в підкласи;

– Pull Up… – переміщує поле або метод у суперклас;

– Extract Class… – замінює множину полів новим об’єктом-контейнером;

– Introduce Parameter Object… – замінює множину параметрів новим класом та оновлює всі виклики методу для передачі даної сутності нового класу в якості значення представленому параметру;

– Introduce Indirection… – створює статичний непрямий метод;

– Introduce Factory… – створює новий фабричний метод, який буде викликати обраний конструктор та повертати створений об’єкт;

– Introduce Parameter… – замінює вираз посиланням на новий параметр методу та оновлює всі виклики методу для передачі виразу в якості значення параметру;

– Encapsulate Field… – запінює всі посилання на поле методам set/get;

– Generalize Declared Type… – дозволяє користувачу вибрати супертип для поточного типу посилання;

– Create Script… – створює скрипт рефакторингу;

– Apply Script… – застосовує скрипт рефакторингу до проектів у робочому просторі;

– History… – дозволяє переглядати історію виконання рефакторингу в даному робочому просторі.

### 5.3. Завдання на лабораторну роботу

5.3.1. Ознайомитись з теоретичними відомостями, необхідними для виконання роботи.

5.3.2. Використовуючи розроблені у попередніх лабораторних роботах програмні проекти, визначити наявні ризики неякісного коду.

5.3.3. Виконати рефакторинг та оптимізацію коду за допомогою засобів IDE та окремо застосовуючи всі інші розглянуті на лекціях засоби.

5.3.4. Оформити звіт.

5.3.5. Відповісти на контрольні запитання.

### 5.4. Зміст звіту

5.4.1. Мета роботи.

5.4.2. Текст розробленого програмного забезпечення з коментарями щодо впроваджених засобів управління якістю програмного коду.

5.4.3. Опис ризиків та прийнятих рішень з управління ними.

5.4.4. Результати тестування: вхідні дані та результати роботи програми.

5.4.5. Висновки, що містять відповіді на контрольні запитання, а також відображають результати виконання роботи та їх критичний аналіз.

## 5.5. Контрольні запитання

5.5.1. Що таке рефакторинг?
5.5.2. Які підстави проведення рефакторингу?
5.5.3. Які засоби рефакторингу надаються в Microsoft Visual Studio?
5.5.4. Які засоби рефакторингу коду надаються в Eclipse?
5.5.5. Яким чином можна оптимізувати код?
5.5.6. Яким чином засоби рефакторингу впливають на ризики програмного забезпечення?

# 6. ЛАБОРАТОРНА РОБОТА № 6
# ВИКОРИСТАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АНАЛІЗУ РИЗИКІВ

## 6.1. Мета роботи

6.1.1 Ознайомитись з засобами сучасних програмних систем аналізу ризиків.

6.1.2 Навчитися використовувати на практиці сучасні програмні системи аналізу ризиків для управління ризиками проектів.

## 6.2. Короткі теоретичні відомості

Oracle Primavera Risk Analysis – програмний інструментар, який використовується для аналізу невизначеності та ризиків протягом всього життєвого циклу проекту.

Робота з проектами у Primavera Risk Analysis починається з визначення плану проекту, яке може виконуватися як шляхом відкриття існуючого, так і шляхом створення нового:

– створення нового плану;
– введення завдань та етапів;
– додавання підсумкових завдань;
– додавання логіки для визначення відношень завдань;
– створення та зв’язування ресурсів;
– вивчення витрат проекту;
– визначення та розподілення ресурсів за потребами;
– відображення діаграми Гантта;
– збереження плану.

У результаті план проекту буде виглядати подібно до рис. 6.1. Детальніше кожний етап даного процесу описано в документації на програму.

Процес аналізування ризиків проекту в Primavera Risk Analysis складається з наступних кроків:

– створення або використання готового плану;
– додавання даних про ризик;
– виконання аналізу ризиків.

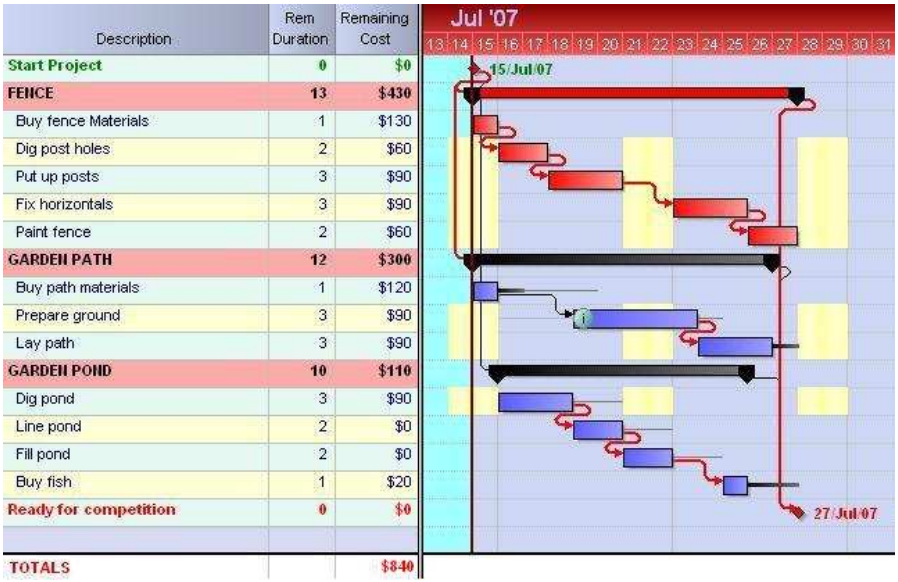



Рисунок 6.1 – План проекту

Інформація про кожний проект відображається у вигляді діаграми Гантта та даних за кожним етапом виконання проекту. Окрім безпосередньо переліку етапів, для кожного з них задаються дата початку, завершення, мінімальна, максимальна та очікувана тривалість (формують трикутний розподіл, рис. 6.2), загальні витрати, невизначеність тривалості та рівень ризику тривалості, пов'язані ресурси.

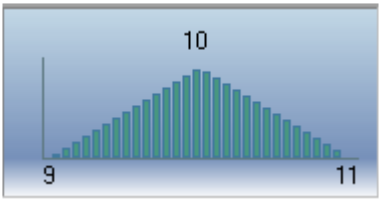



Рисунок 6.2 – Розподіл тривалості виконання завдання

Для введення даних про ризики можуть використовуватися діаграма Гантта та безпосередньо план проекту. Для цього потрібно

внести інформацію про тривалість виконання завдань проекту і проекту загалом, ресурси проекту і окремо кожного завдання (в тмоу числі грошові, які визначають витрати).

Для того щоб розпочати аналіз ризиків призначений пункт меню програми Risk → Run Risk Analysis, після чого у вікні, що відкриється, (рис. 6.3) необхідно задати кількість ітерацій, варіант демонстрації результату (графік розподілу Distribution Graph, різновид стовпчастої діаграми Tornado Graph, діаграма розсіювання Scatter Plot, ймовірнісний грошовий потік Probabilistic Cash Flow) та натиснути кнопку Analyze.

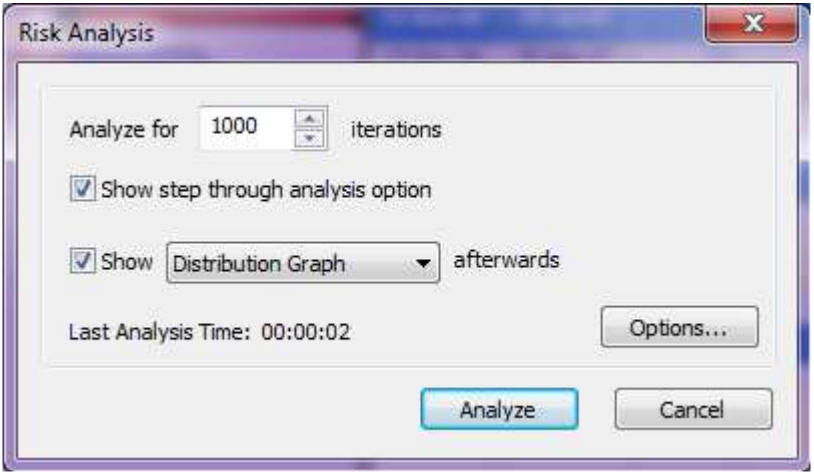



Рисунок 6.3 – Налаштування аналізу ризиків

Для налаштування додаткових параметрів, зокрема обчислення чутливості тривалості й витрат, необхідно натиснути кнопку Options. На рис. 6.4 наведено вікно налаштування параметрів чутливості завдань. У даному вікні для тривалості й витрат можна визначити, для яких завдань та відносно яких завдань даний показник визначати, а також метод обчислення.

Після того, як буде натиснуто кнопку Analyze, на екрані з’явиться вікно з кнопками Step, Go, Complete та Cancel, за допомогою яких можна керувати процесом. Кожна ітерація змінює діаграму Гантта, обираючу випадкову тривалість кожного завдання.

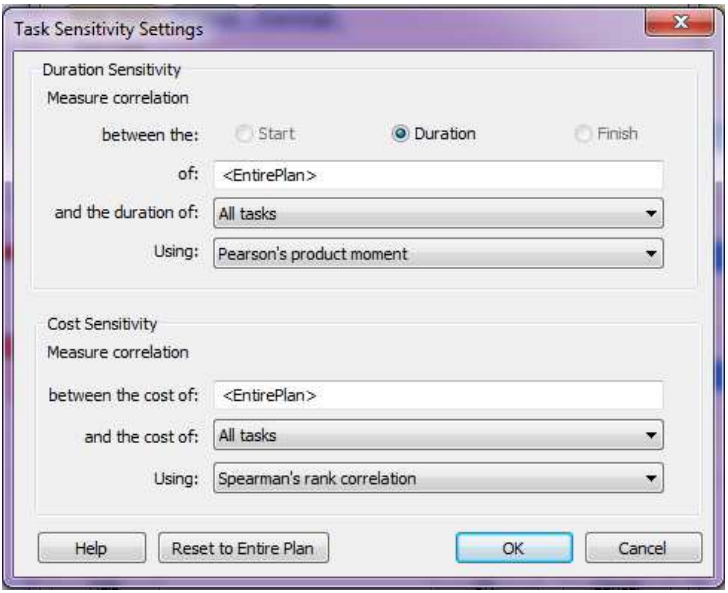



Рисунок 6.4 – Налаштування чутливості завдань

Приклад відображення результату аналізу ризиків у вигляді графіку розподілу представлено на рис. 6.5. Графік разом зі статистикою представляє інформацію про тривалість виконання, дату початку, завершення, витрати на виконання кожного завдання, кожного ресурсу та всього проекту загалом в залежності від результатів кожної окремої ітерації. Велика кількість налаштувань дозволяє визначити, яку саме інформацію необхідно відображати та яким чином (наприклад, можна відобразити статистичні показники: середнє, моду, медіану, максимум, мінімум).

Пункт меню Format → Highlighters дозволяє продемонструвати процентне співвідношення можливості досягнення визначеної дати, витрат, тривалості тощо.

Індекс критичності (Criticality Index, частка знаходження завдання на критичному шляху) дозволяє встановити завдання, які ймовірно можуть викликати затримки проекту. Виконуючи моніторинг таких завдань, можна знизити ймовірність затримки проекту. Відобразити даний показник можна за допомогою меню Format → Columns, після чого у вікні Columns необхідно обрати Risk

Outputs → Criticality Index. З даного вікна можна обрати і інші показники, які необхідно відобразити.

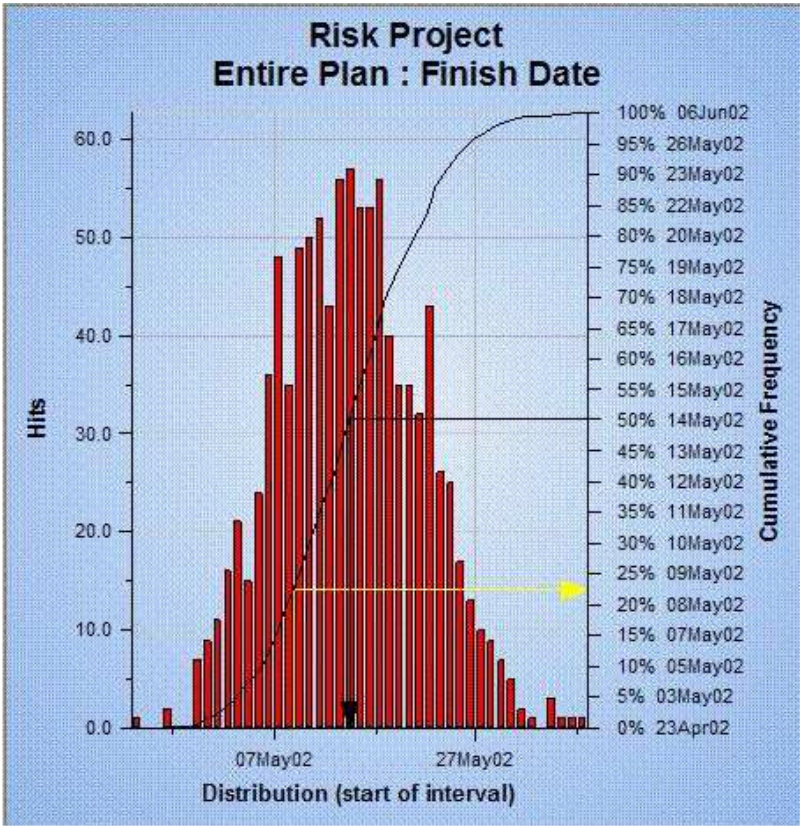



Рисунок 6.5 – Графік розподілу за результатами аналізу ризиків проекту

Після виконання аналізу ризиків отримати доступ до всіх сформованих діаграм можна за допомогою пункту меню Reports.

Приклад стовпчастої діаграми наведено на рис. 6.6. Дана діаграма дозволяє користувачам визначити ключові напрямки ризиків та події, які можуть призвести до ризиків.

На рис. 6.7 наведено приклад відображення результатів аналізу ризиків у вигляді діаграми розсіювання.

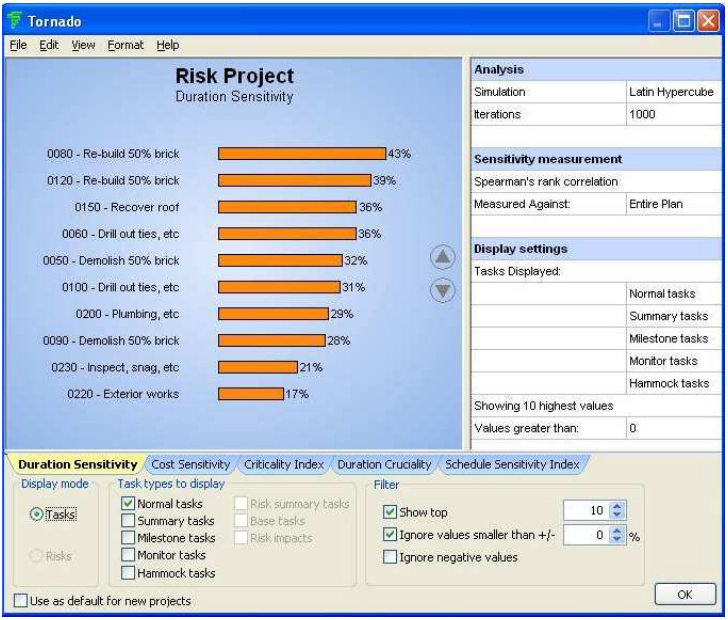



Рисунок 6.6 – Стовпчаста діаграма за результатами аналізу ризиків

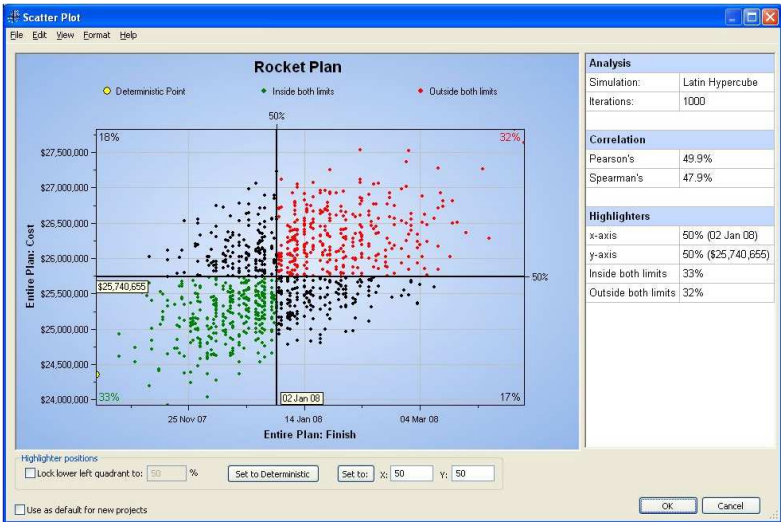



Рисунок 6.7 – Діаграма розсіювання за результатами аналізу ризиків

Дана діаграма може використовуватися для визначення комбінованої імовірності досягнення заданого бюджету та строків виконання та виконання «що-якщо» аналізу, інтерактивно змінюючи пороги витрат і тривалості виконання, щоб визначити ймовірність успіху.

На рис. 6.8 наведено приклад побудови ймовірнісного грошового потоку для заданого проекту, побудованого за результатами аналізу ризиків. Даний потік дозволяє працювати з невизначеністю, яка полягає в тому, коли гроші будуть витрачатися та поступати.

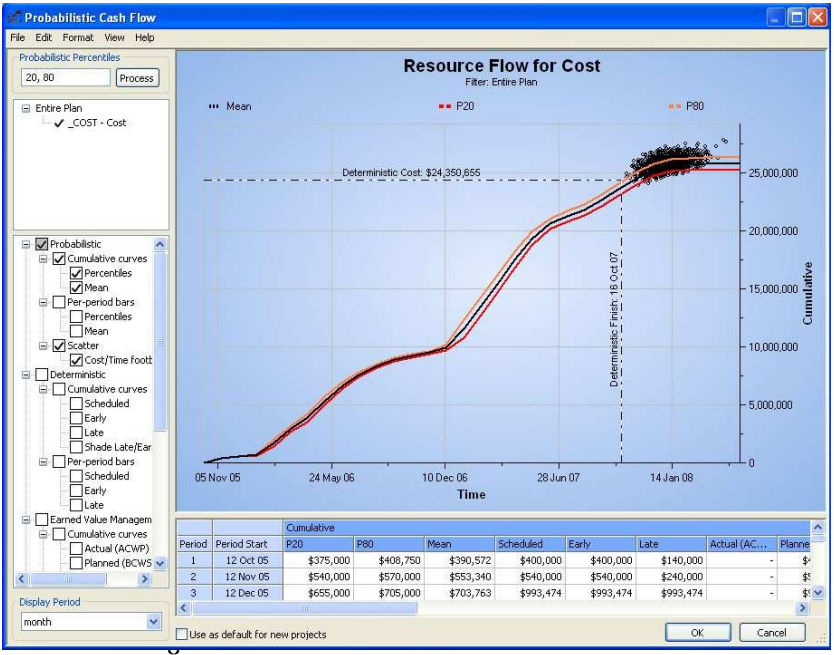



Рисунок 6.8 – Ймовірнісний грошовий потік за результатами аналізу ризиків

Для проведення моделювання аналізу ризиків можна також створити власну модель (закладка Develop Risk Model у лівій частині екрану) шляхом визначення завдань, які будуть брати участь у даному процесі, форми розподілу та його характеристик (рис. 6.9).


Рисунок 6.9 – Визначення моделі ризику

Сформувати загальний звіт за результатами аналізу ризиків можна за допомогою меню Reports → Summary Risk Report, після чого у вікні, що відкриється, (рис. 6.10) необхідно визначити вхідні та вихідні дані, які ввійдуть у звіт.

Для того щоб виконати моделювання впливу ризику на розклад проекту, використовується Risk Register, доступне з меню через Risk → Register. У даному реєстраторі ризиків представлені кількісні та якісні ризики (рис. 6.11), які можуть впливати на проект.

Для того щоб визначити новий ризик у Risk Register, необхідно встановити курсор на наступний, порожній, рядок та заповнити його відповідним чином, визначаючи в частині Pre-Mitigation ID ризику, тип (можливість або загроза), назву, ймовірність настання, вплив на розклад, на тривалість і на виконання результатів, з чого формується оцінка.

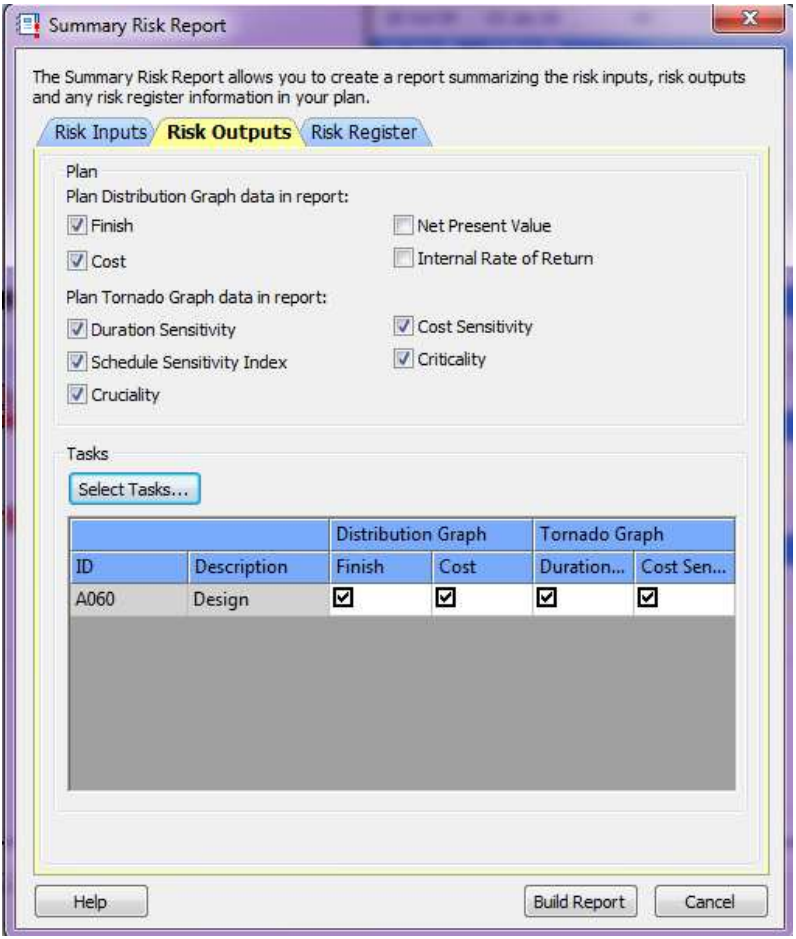
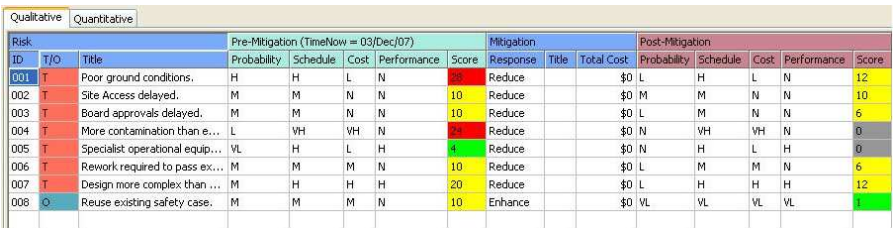



Рисунок 6.10 – Формування звіту за результатами аналізу ризиків



Qualitative | Quantitative

| Risk | | | Pre-Mitigation (TimeNow = 03/Dec/07) | | | | | Mitigation | | | Post-Mitigation | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ID | T/O | Title | Probability | Schedule | Cost | Performance | Score | Response | Title | Total Cost | Probability | Schedule | Cost | Performance | Score |
| 001 | T | Poor ground conditions. | H | H | L | N | 20 | Reduce | | $0 | L | H | L | N | 12 |
| 002 | T | Site Access delayed. | M | M | N | N | 10 | Reduce | | $0 | M | M | N | N | 10 |
| 003 | T | Board approvals delayed. | M | M | N | N | 10 | Reduce | | $0 | L | M | N | N | 6 |
| 004 | T | More contamination than e... | L | VH | VH | N | 24 | Reduce | | $0 | N | VH | VH | N | 0 |
| 005 | T | Specialist operational equip... | VL | H | L | H | 4 | Reduce | | $0 | N | H | L | H | 0 |
| 006 | T | Rework required to pass ex... | M | M | M | N | 10 | Reduce | | $0 | L | M | M | N | 6 |
| 007 | T | Design more complex than ... | M | H | H | H | 20 | Reduce | | $0 | L | H | H | H | 12 |
| 008 | O | Reuse existing safety case. | M | M | M | N | 10 | Enhance | | $0 | VL | VL | VL | VL | 1 |

Рисунок 6.11 – Визначення якісних ризиків проекту

Окрім того для ризику можна також визначити причину, вплив, опис, власника, статус, дату початку і закінчення, керованість, фінансовий рівень ризику.

У частині Mitigation необхідно визначити дії, які можуть виконуватися у випадку реалізації даного ризику: реакція (прийняти, зменшити тощо), назву такої дії, загальні витрати. Окрім того потрібно визначити відповідні зазначені спершу показники (ймовірність, вплив на розклад, витрати, результати та результуючу оцінку) для випадку, якщо такі дії будуть реалізовані, заповнюючи дані показники в частині Post-mitigation.

На закладці Waterfall Chart можна переглянути, яким чином змінюється оцінка ризику за часовим інтервалом.

У меню Reports є три пункти, які дозволяють відповідно переглянути матрицю ризиків (Risk Matrix), оцінки ризиків (Risk Scoring) та загальний звіт (Report Manager) за ризиками. На рис. 6.12 наведено приклад матриці ризиків, яка відображає розташування кожного ризику на діаграмі Імовірність/Вплив, ґрунтуючись на їх імовірності та впливі з найвищим значенням.


Рисунок 6.12 – Матриця ризиків

Кількісні ризики визначаються відповідно на закладці Quantitative вікна Risk Register (рис. 6.13), звідки можна задати ID, тип

ризику, його назву, ймовірність настання та завдання, на які він має вплив. Для кожного завдання можна визначити функцію розподілу значень, мінімальний, максимальний, найбільш імовірний вплив на розклад, а також відповідні показники для витрат.

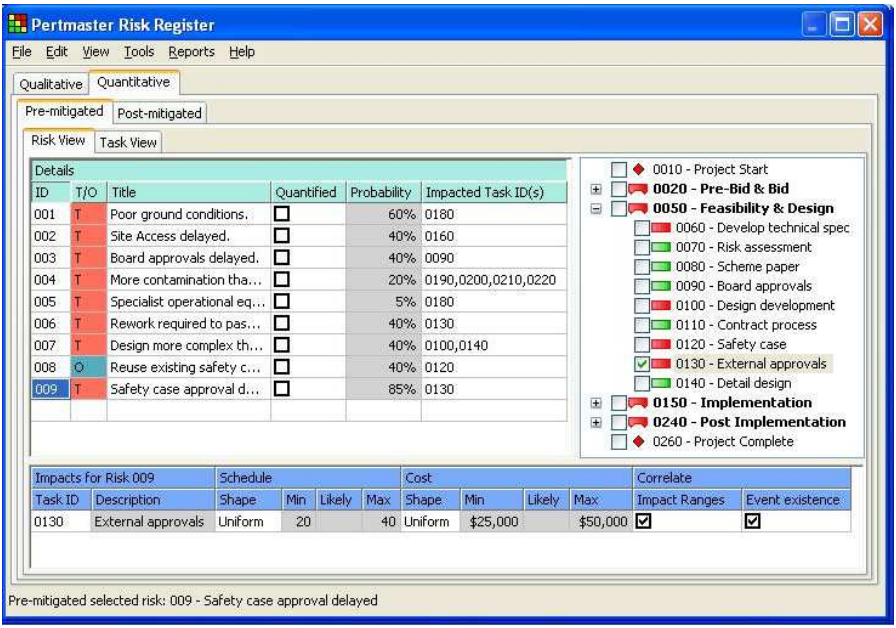



Рисунок 6.13 – Визначення кількісних ризиків та їх впливу на завдання проекту

На рис. 6.14 наведено приклад оцінювання ризиків, який можна виконати за допомогою меню Edit → Risk Scoring.

Після того, як виконано зв'язування ризиків з завданнями в проекті, може бути побудовано плани ризиків до реагування (Pre-mitigated) та після (Post-mitigated). Для того щоб побудувати такий план, необхідно обрати в меню Risk → Risk Register → Tools → Build Impacted Risk Plan(s), де обрати всі необхідні характеристики.

Після цього необхідно обрати відповідний створений план з меню Window та розпочати аналіз ризиків.

Для порівняння планів до реагування та після може використовуватися Distribution Analyzer (рис. 6.15).

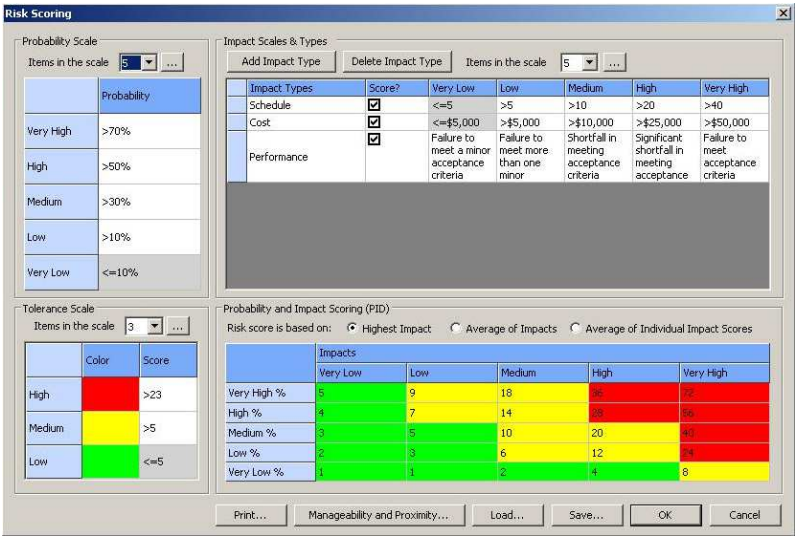



Рисунок 6.14 – Оцінювання ризиків

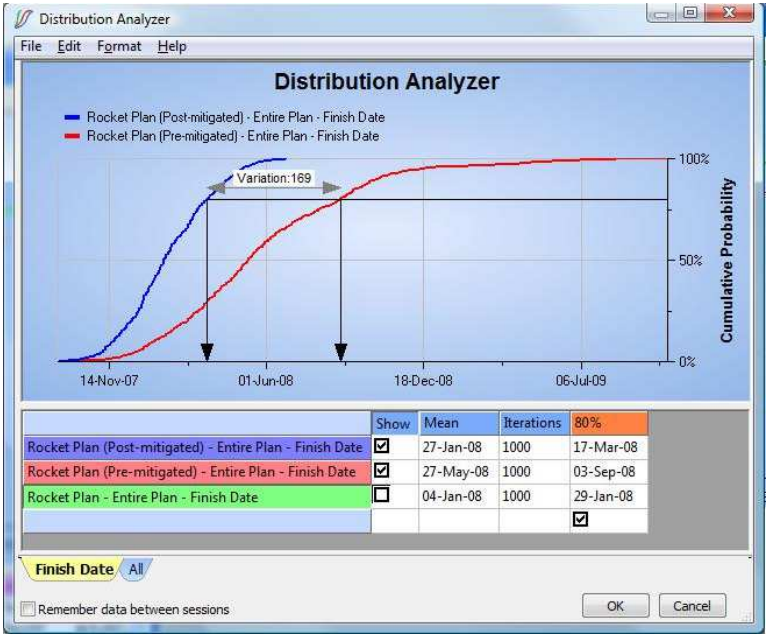



Рисунок 6.15 – Аналіз розподілу

### 6.3. Завдання на лабораторну роботу

6.3.1. Ознайомитись з теоретичними відомостями, необхідними для виконання роботи.

6.3.2. Дослідити можливості програми Oracle Primavera Risk Analysis.

6.3.3. Використовуючи дані файлу проекту, отриманого за індивідуальним завданням у викладача, виконати аналіз ризиків, ґрунтуючись на виконанні наступних етапів:

– проаналізувати план проекту;

– внести дані про розподіл даних у плані проекту;

– виконати аналіз ризиків проекту;

– ідентифікувати ризики проекту (обов'язково проаналізувати та додати до проекту самостійно визначені ризики);

– виконати аналіз ризиків проекту;

– порівняти результати отримані на різних етапах аналізу ризиків та виділити вплив окремих ризиків (обов'язково критичних) на результати проекту.

6.3.4. Оформити звіт, включаючи в нього власний звіт за результатами аналізу ризиків.

6.3.5. Відповісти на контрольні запитання.

### 6.4. Зміст звіту

6.4.1. Мета роботи.

6.4.2. Опис проекту та основні його дані.

6.4.3. Аналіз ризиків проекту зі всіма необхідними графіками, звітами, показниками та виводами за отриманими результатами з зазначенням методів управління ризиками.

6.4.4. Висновки, що містять відповіді на контрольні запитання, а також відображають результати виконання роботи та їх критичний аналіз.

### 6.5. Контрольні запитання

6.5.1. Який інструментар надає Primavera Risk Analysis?

6.5.2. Що дозволяє виконувати аналіз ризиків?

6.5.3. Які засоби управління ризиками доступні в Primavera Risk Analysis?

6.5.4. У якому вигляді можуть відображатися результати аналізу ризиків?

6.5.5. Які ризики можна віднести до якісних?

6.5.6. Які ризики можна віднести до кількісних?

6.5.7. Що таке матриця ризиків?

6.5.8. Яким чином зареєстровані ризики впливають на аналіз ризиків у Primavera Risk Analysis?

6.5.9. За допомогою яких показників можна проаналізувати ступінь ризику проекту?

6.5.10. За допомогою яких методів виконується моделювання ризиків у Primavera Risk Analysis?

# 7. ЛАБОРАТОРНА РОБОТА № 7
# ЯКІСНИЙ ТА КІЛЬКІСНИЙ АНАЛІЗ РИЗИКІВ

## 7.1. Мета роботи

7.1.1 Ознайомитись з методами якісного та кількісного аналізу ризиків.

7.1.2 Розробити програмну реалізацію кількісного аналізу ризиків на основі імітаційного моделювання.

## 7.2. Короткі теоретичні відомості

Процес кількісного аналізу ризику за допомогою методів імітаційного моделювання можна розділити на наступні кроки.

Перший крок – формування моделі проекту.

Другий крок – визначення ключових аргументів, застосовуючи зокрема метод аналізу чутливості щодо низки чинників, які входять у модель, але до уваги не приймається те, наскільки ймовірними є ті чи інші випадкові коливання аргументу ризику. Для того щоб дані, одержані в процесі аналізу чутливості, мали сенс, у тест аналізу повинна бути введена концепція впливу невизначеності, пов'язаної з чинниками, що аналізуються, а також можливості використати цей аналіз для вибору чинників підвищеного ризику. Наприклад, може бути визначено, що відхилення у ціні купівлі певного виду устаткування на початку здійснення проекту має значний вплив на чистий інтегрований дисконтований дохід від проекту. Однак імовірність будь-якого, навіть досить незначного відхилення цього чинника може бути дуже малою, якщо, наприклад, постачальник згідно з умовами контракту зобов'язаний здійснити поставки за фіксованою ціною. Отже ризик, пов'язаний з цим аргументом, незначний. Даний чинник вилучається з подальшого аналізу. Тобто для подальшого аналізу ризику залишаються лише ті чинники, які не є детермінованими, а еластичність відповідної функції за даним аргументом значна.

Третій крок полягає в тому, щоб визначити можливі інтервали відхилень прогнозованих значень параметрів від очікуваних. На цьому

етапі доречно використовувати математичні (статистичні) оцінки якості прогнозів.

Четвертий крок – визначення розподілу ймовірності випадкових значень аргументів. Здійснюється паралельно з третім кроком. Підготовка даних та оцінка розподілу ймовірності для відібраних чинників ризику включає як встановлення множини їх можливих значень, так і ймовірностей (вагових величин). На практиці визначення цих величин є ітераційним процесом, інтервали значень відповідних чинників уточнюються, враховуючи конкретний профіль розподілу ймовірності, і навпаки.

П’ятий крок призначений для виявлення взаємозалежності, яка на практиці може існувати між ключовими аргументами. У наборі ризикових чинників такі залежності зустрічаються досить часто. Наприклад, рівень собівартості значною мірою зумовлює величину ціни реалізації. Рівень ціни на певний товар, як правило, має обернене співвідношення з обсягом його продажу.

Ігнорування кореляції може призвести до неправильних результатів у аналізі ризику, тому важливо переконатися в наявності чи відсутності таких взаємозв’язків і, де це необхідно, ввести у моделюванні обмеження, що знизили б до раціонального рівня ймовірність вироблення сценаріїв, які порушують вплив кореляції. Фактично наявність кореляційного зв’язку обмежує випадковий вибір значень корельованих випадкових змінних. Цей вибір стає зумовленим як межами відповідних характеристик, так і напрямом (прямо чи обернено пропорційним) зв’язку.

Для адекватнішого відображення багатосторонніх реальних взаємозв’язків між явищами застосовуються економетричні моделі та методи.

Шостий крок полягає в генерації випадкових сценаріїв, які ґрунтуються на системі прийнятих гіпотез щодо чинників ризику згідно з обраною моделлю на першому кроці. Після того, як усі гіпотези були ретельно досліджені і побудовано відповідні залежності, необхідно послідовно здійснювати обчислення згідно з обраною на першому кроці моделлю, доки не буде одержана репрезентативна вибірка з нескінченої множини можливих значень ключових аргументів, враховуючи накладені на них обмеження. Для цього, як свідчить досвід, достатньо, щоб вибірка була одержана в результаті

здійснення 200–500 обчислень, серія яких здійснюється за методом Монте-Карло.

Сьомий крок. Після серії обчислень можна одержати розподіл частот для результуючого показника (ефективності, чистої теперішньої вартості проекту, норми доходу тощо). Отримані результати вимагають їх інтерпретації. Коли обчислено сподіване значення результуючого показника (наприклад, чиста приведена вартість (ЧПВ) або норма доходу) проекту, то рішення щодо прийняття чи відхилення даного проекту залежить від того, який знак має ця величина. Якщо він додатний, то це є необхідною, але не достатньою умовою, щоб даний проект прийняти. Якщо знак відповідного показника від'ємний, то такий проект слід відхилити. Остаточне рішення виявляється об'єктивно-суб'єктивним, тобто значною мірою залежить від того, як СПР ставиться до ризику.

Беручи до уваги традиційну поведінку СПР, може бути розглянуто кілька можливих спрощених ситуацій. У кожній ситуації подається як інтегральна (F(x)), так і диференційна (f(x)) функції певного закону розподілу ймовірностей випадкової величини економічного показника (наприклад, ЧПВ), яка характеризує доцільність інвестування в один проект або вибору між альтернативними проектами.

З аналізу ситуацій можна вивести кілька правил щодо того, який з альтернативних проектів є сенс прийняти, беручи до уваги ризик.

### 7.3. Завдання на лабораторну роботу

7.3.1. Ознайомитись з теоретичними відомостями, необхідними для виконання роботи.

7.3.2. Виконати програмну реалізацію імітаційного моделювання на основі методу Монте Карло для кількісного аналізу ризику.

7.3.3. Виконати кількісний аналіз ризиків на основі методу Монте Карло для проекту, визначеного індивідуальним завданням з лабораторної роботи № 7.

7.3.4. Оформити звіт.

7.3.5. Відповісти на контрольні запитання.

### 7.4. Зміст звіту

7.4.1. Мета роботи.
7.4.2. Завдання.
7.4.3. Текст розробленого програмного забезпечення.
7.4.4. Інтерфейс роботи з програмою в декількох режимах.
7.4.5. Результати тестування: вхідні дані та результати роботи програми.
7.4.6. Висновки, що містять відповіді на контрольні запитання, а також відображають результати виконання роботи та їх критичний аналіз.

### 7.5. Контрольні запитання

7.5.1. Яким чином здійснюється якісний аналіз ризику?
7.5.2. Які розрізняються методи кількісного аналізу ризику?
7.5.3. З яких кроків складається імітаційне моделювання?
7.5.4. В чому полягає узагальнений підхід до кількісної оцінки ризику?
7.5.5. Які математичні теорії слід застосовувати для формалізації невизначеної інформації й вимірювання ризику?

## ЛІТЕРАТУРА

1. Дубровін, В.І. Прийняття рішень у процесі управління ризиками проектів : навчальний посібник [Текст] / В. І. Дубровін, В. М. Льовкін. – Запоріжжя : ЗНТУ, 2012. – 196 с.

2. Хохлов, Н.В. Управление риском : Учеб. пособие для вузов [Текст] / Н.В. Хохлов. – М. : ЮНИТИ-ДАНА, 2001. – 239 с.

3. Шапкин, А.С. Теория риска и моделирование рисковых ситуаций : Учебник [Текст] / А.С. Шапкин, В.А. Шапкин. – М. : Издательско-торговая корпорация «Дашков и Ко», 2005. – 880 с.

4. Згуровський, М.З. Основи системного аналізу [Текст] / М.З. Згуровський, Н.Д. Панкратова. – К. : Видавнича група BHV, 2007. – 544 с. : іл..

5. DeMarco, T. Waltzing with Bears : Managing Risk on Software Projects [Текст] / Tom DeMarco, Timothy Lister. – New York : Dorset House Publishing, 2003. – 208 p.

6. Varma, J. Learn Lua for iOS Game Development [Текст] / jayant Varma. – Apress, 2012. – 426 p.

7. Ierusalimschy, R. Programming in Lua [Текст] / Roberto Ierusalimschy. – 3th ed. – Rio de Janeiro, 2013. – 348 p.

8. Jung, K. Beginning Lua programming [Текст] / Kurt Jung and Aaron Brown. – Indianapolis : Wiley Publishing, Inc., 2007. – 646 p.

9. Gutschmidt, T. Game Programming with Python, Lua, and Ruby [Текст] / Tom Gutschmidt. – Premier Press, 2003. – 472 p.

10. Flanagan, N. Corona SDK [Текст] / Nevin Flanagan. – Birmingham-Mumbai : Packt Publishing, 2013. – 321 p.

11. Corona Docs – API [Електронний ресурс] / Режим доступу : http://docs.coronalabs.com/api/index.html

12. Wang, J. A. W. Component-Oriented Programming [Текст] / Andy JuAn Wang, Kai Qian. – New Jersey : John Wiley & Sons, Inc., 2005. – 336 p.

13. Frohlich, P. H. Component-Oriented Programming Languages : Why, What, and How : A dissertation ubmitted in partial satisfaction of the requirements for the degree of doctor of philisiphy in Information and Computer Science [Текст] / Peter Hans Frohlich. – University of California. – Irvine, 2003. – 151 p.

14. Нортроп, Т. Основы разработки приложений на платформе Microsoft . NET Framework. Учебный курс Microsoft ; пер. с англ. [Текст] / Тони Нортроп, шон Уилдермьюс, Билл Райан. – М. : Русская редакция, СПб. : Питер, 2007. – 864 с. : ил.

15. Lowy, J. Programming .NET Components, Second Edition [Текст] / Juval Lowy. – O'Reilly Media, Inc., 2005. – 646 p.

16. Троэлсен, Э. Язык программирования C# 2010 и платформа .NET 4.0, 5-е изд. ; Пер. с англ. [Текст] / Эндрю Троэлсен. – М. : ООО «И.Д. Вильямс», 2011. – 1392 с. : ил. – Парал. тит. англ.

17. Макконнелл, С. Совершенный код. Мастер-класс [Текст] / Стив Макконнелл ; пер. с англ. – М. : Издательско-торговый дома «Русская редакция» ; СПб. : Питер, 2005. – 896 с.

18. Мартин, Р. Чистый код : создание, анализ и рефакторинг. Библиотека программиста [Текст] / Р. Мартин. – СПб. : Питер, 2010. – 464 с. : ил.

19. Управление риском [Текст] / В.А. Владимиров, Ю.Л. Воробьев, С.С. Салов и др. – М. : Наука, 2000. – 431с.

20. Шафер, Д. Управление программными проектами : достижение оптимального качества при минимуме затрат [Текст] / Дональд Шафер ; пер. с англ. – М. : Издательский дом «Вильямс», 2003. – 1136 с. : ил.

21. Вітлінський, В.В. Аналіз, оцінка і моделювання економічного ризику [Текст] / В.В. Вітлінський. – Київ : ДЕМІУР, 1996. – 212 с.

22. Вітлінський, В. В. Економічний ризик і методи його вимірювання : Підручник [Текст] / В.В. Вітлінський, С.І. Наконечний, О.Д. Шарапов. – К : ІЗМН, 1996. – 338 с.

23. Мазур, И.И. Управление проектами : Учебное пособие [Текст] / И.И. Мазур, В.Д. Шапиро, Н.Г. Ольдерогге ; под общ. ред. И.И. Мазура. – 5-е изд., перераб. – М. : Омега-Л., 2009. – 960 с.

24. Мартин, П. Управление проектами [Текст] / П. Мартин, К. Тейт ; пер. с англ. – СПб. : Питер, 2006. – 224 с. : ил.

25. Управление инновационными проектами : Учеб. пособие [Текст] / В.Л. Попов, Н.Д. Кремлев, В.С. Ковшов и др. ; под ред. проф. В.Л. Попова. – М. : ИНФРА-М, 2009. – 336 с.

26. Шапкин, А.С. Экономические и финансовые риски. Оценка, управление, портфель инвестиций : Монография [Текст] /

А.С. Шапкин. – М. : Издательско-торговая корпорация «Дашков и Ко», 2003. – 544 с. : ил.

27. Уткин, Э.А. Управление рисками предприятия : Учебно-практическое пособие [Текст] / Э.А. Уткин, Д.А. Фролов. – М. : ТЕИС, 2003. – 247 с.

28. Управление инновационными проектами : Учеб. пособие [Текст] / В.Л. Попов, Н.Д. Кремлев, В.С. Ковшов и др. ; под ред. проф. В.Л. Попова. – М. : ИНФРА-М, 2009. – 336 с.

29. Дубровін, В.І. Оцінювання ризиків інвестиційного портфеля [Текст] / В.І. Дубровін, В.М. Льовкін // Радіоелектроніка. Інформатика. Управління. – 2010. – №1 (22). – С. 51-55.

30. Арчибальд, Р. Управление высокотехнологичными программами и проектами [Текст] / Рассел Д. Арчибальд ; пер. с англ. Мамонтова Е.В. ; под ред. Баженова А.Д., Арефьева А.О. – 3-е изд., перераб. и доп. – М. : Компания АйТи, 2004. – 472 с., ил.