МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

МЕТОДИЧНІ ВКАЗІВКИ
ЩОДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ
З НАВЧАЛЬНОЇ ДИСЦИПЛІНИ
«КЕРУВАННЯ РОЗРОБКОЮ ПРОГРАМНИХ СИСТЕМ»
ДЛЯ СТУДЕНТІВ ДЕННОЇ ФОРМИ НАВЧАННЯ
ЗІ СПЕЦІАЛЬНОСТІ
123 – «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ»
ОСВІТНЬОГО СТУПЕНЯ «МАГІСТР»
ЧАСТИНА ІІ

КРЕМЕНЧУК 2018

Методичні вказівки щодо виконання лабораторних робіт з навчальної дисципліни «Керування розробкою програмних систем» для студентів денної форми навчання зі спеціальності 123 – «Комп'ютерна інженерія» освітнього ступеня «Магістр». Частина ІІ

Укладачі: д. т. н., проф. М. І. Гученко,
к. т. н. П. П. Костенко,
асист. Н. Л. Сохін
Рецензент к. т. н., доц. О. Г. Славко

Кафедра комп'ютерних та інформаційних систем

Затверджено методичною радою Кременчуцького національного університету імені Михайла Остроградського
Протокол № _____ від _____
Голова методичної ради _____ проф. В. В. Костін

ЗМІСТ

ВСТУП

Предметом навчальної дисципліни «Керування розробкою програмних систем» є методи, призначені для створення проектних основ, моделей та алгоритмів, опису майбутнього проекту, документації, призначеної для розробки програмних комплексів. Під час виконання лабораторних робіт закріплюються знання, отримані на лекціях, де студенти вивчають основні поняття, структуру, вимоги до створення і моделювання комп'ютерних проектів. Студенти вивчають методи відслідковування помилок на етапі створення проекту та методи тестування готових програмних комплексів.

Виконання кожної лабораторної роботи передбачає ознайомлення студента з лабораторним практикумом та його теоретичну підготовку з відповідних розділів дисципліни. До кожного розділу наведено перелік питань для контролю. Об'єкт автоматизації для дослідження визначається викладачами індивідуально. На захист виконаної студентом лабораторної роботи оформлюється окремий звіт, за потреби демонструється електронний документ з виконаним завданням. Підсумкові оцінки, отримані студентом за виконання лабораторних робіт і відповіді на контрольні запитання, враховуються під час обчислення модульних оцінок та семестрової підсумкової оцінки з навчальної дисципліни.

Послідовне виконання лабораторних робіт формує цілісне документальне представлення, що є необхідним для створення автоматизованої системи керування, починаючи з визначення самого проекту, опису робіт, надання моделі проекту, алгоритму виконання програми, розробки програмного модуля, методів відслідковування помилок.

Під час виконання лабораторних робіт студент отримує знання, що розвинуть аналітичні здібності, нададуть змогу правильно оформляти проектну документацію, виступати керівником проекту з впровадження автоматизованих систем. Отримані знання забезпечать достатню кваліфікацію студента як спеціаліста в сфері аналітики і керівника проектів.

1 ПЕРЕЛІК ЛАБОРАТОРНИХ РОБІТ

Лабораторна робота № 4

Тема. Структурний підхід до програмування. Стадія «Налагодження програмного засобу та розробка програмної документації»

Мета: навчитися розробляти і оформляти документацію, необхідну для супроводження та експлуатації програмного засобу.

Короткі теоретичні відомості

Комплексне налагодження програмного засобу

Під час комплексного налагодження тестується ПЗ в цілому, причому тести готуються щодо кожного з документів ПЗ. Тестування цих документів здійснюється, як правило, в порядку, зворотному їх розробці. Виняток становить лише тестування документації із застосування, яка розробляється за зовнішнім описом паралельно з розробкою текстів програм – це тестування краще проводити після завершення тестування зовнішнього опису. Тестування під час комплексного налагодження являє собою використання ПЗ з конкретними даними, які можуть виникнути у користувача (зокрема, всі тести готуються у формі, розрахованій на користувача), але можуть бути і у вигляді моделей. Наприклад, деякі недоступні під час комплексного налагодження пристрої введення і виведення можуть бути замінені їх програмними імітаторами.

Тестування архітектури ПЗ. Метою тестування є пошук невідповідності між описом архітектури та сукупністю програм ПЗ. До моменту початку тестування архітектури ПЗ має бути закінчено автономне налагодження кожної підсистеми. Помилки реалізації архітектури можуть бути пов’язані, насамперед, з взаємодією цих підсистем, зокрема, з реалізацією архітектурних функцій (якщо вони є). Тому необхідно перевірити всі шляхи взаємодії між підсистемами ПЗ. Для цього необхідно протестувати всі ланцюжки виконання підсистем без повторного входження останніх. Якщо задана архітектура ПЗ

являє собою сукупність малих систем з виділених підсистем, то кількість таких ланцюжків буде цілком визначеною.

Тестування зовнішніх функцій. Метою тестування є пошук розбіжностей між функціональною специфікацією і сукупністю програм ПЗ. Незважаючи на те, що всі ці програми автономно вже налагоджені, зазначені розбіжності можуть бути, наприклад, через невідповідність внутрішніх специфікацій програм і їх модулів (на підставі яких проводилося автономне тестування) функціональної специфікації ПЗ. Як правило, тестування зовнішніх функцій проводиться так само, як і тестування модулів, аналогічно методу чорного ящика.

Тестування якості ПЗ. Метою тестування є пошук порушень вимог якості, сформульованих у специфікації якості ПЗ. Це найбільш важкий і найменш вивчений вид тестування. Ясно лише, що далеко не кожний параметр якості ПЗ може бути випробуваний тестуванням. Завершеність ПЗ перевіряється вже під час тестування зовнішніх функцій. На цьому етапі тестування цього параметру якості може бути продовжено, якщо потрібно отримати якусь вірогідну оцінку ступеня надійності ПЗ. Однак методика такого тестування ще вимагає своєї розробки. Можуть тестуватися такі параметри якості, як точність, стійкість, захищеність, часова ефективність, якоюсь мірою – ефективність щодо пам’яті, ефективність щодо пристроїв, розширюваність і, частково, незалежність від пристроїв. Кожен з цих видів тестування має свою специфіку і заслуговує окремого розгляду. Легкість застосування ПЗ оцінюється під час тестування документації із застосування ПЗ.

Тестування документації по застосуванню ПЗ. Метою тестування є пошук неузгодженості документації по застосуванню і сукупністю програм ПЗ, а також виявлення незручностей, що виникають при застосуванні ПЗ. Цей етап безпосередньо передує підключенню користувача до завершення розробки ПЗ (тестуванню визначення вимог до ПЗ та атестації ПЗ), тому досить важливо розробникам спочатку самим скористатися ПЗ так, як це буде робити користувач. Всі тести на цьому етапі готуються виключно на підставі тільки

документації із застосування ПЗ. Перш за все, мають тестуватися можливості ПЗ, як це робилося під час тестування зовнішніх функцій, але тільки на підставі документації із застосування. Мають бути протестовані всі неясні місця в документації, а також всі приклади, використані в документації. Далі тестуються найбільш важкі випадки застосування ПЗ з метою виявити порушення вимог відносно легкості застосування ПЗ.

Тестування визначення вимог до ПЗ. Метою тестування є з'ясування, якою мірою ПЗ не відповідає висунутим вимогам до нього. Особливістю цього виду тестування є те, що його здійснює організація-покупець або організація-користувач ПЗ, як один із шляхів подолання бар'єру між розробником і користувачем. Зазвичай це тестування проводиться за допомогою контрольних завдань – типових завдань, для яких відомий результат рішення у тих випадках, коли розроблюване ПЗ має прийти на зміну іншої версії ПЗ, яка вирішує хоча б частину завдань розроблюваного ПЗ. Тестування проводиться шляхом вирішення загальних завдань за допомогою як старого, так і нового ПЗ (з наступним зіставленням отриманих результатів). Іноді, як форми такого тестування, використовують *дослідну* експлуатацію ПЗ – обмежене застосування нового ПЗ з аналізом використання результатів у практичній діяльності. По суті, цей вид тестування багато в чому перетинається з випробуванням ПЗ під час його атестації, але виконується до атестації, а іноді й замість атестації.

Документування програмних засобів

Документація, що створюється і використовується в процесі розробки програмних засобів

Під час розробки ПЗ створюється і використовується великий обсяг різноманітної документації. Вона необхідна як:

- засіб передачі інформації між розробниками ПЗ;
- засіб керування розробленням ПЗ;
- засіб передачі користувачам інформації, необхідної для застосування і

супроводу ПЗ. На створення цієї документації припадає велика частка вартості ПЗ.

Цю документацію можна розбити на дві групи:

– документи керування розробленням ПЗ;

– документи, що входять до складу ПЗ.

Документи керування розробкою ПЗ (*software process documentation*) керують і протоколюють процеси розробки і супроводу ПЗ, забезпечуючи зв'язки в середині колективу розробників ПЗ і між колективом розробників і менеджерами ПЗ (*software managers*) – особами, які керують розробленням ПЗ. Ці документи можуть бути наступних типів:

– *плани, оцінки, розклади* (ці документи створюються менеджерами для прогнозування і управління процесами розроблення і супроводу ПЗ);

– *звіти* (про використання ресурсів у процесі розроблення; створюються менеджерами);

– *стандарти* (ці документи вказують розробникам, яким принципам, правилам, угодам вони мають слідувати в процесі розроблення ПЗ. Ці стандарти можуть бути як міжнародними або національними, так і спеціально створеними для організації, в якій ведеться розроблення ПЗ);

– *робочі документи* (це основні технічні документи, що забезпечують зв'язок між розробниками. В них фіксуються ідеї та проблеми, що виникають у процесі розробки, опис використовуваних стратегій і підходів, а також робочі (тимчасові) версії документів, які мають увійти до ПЗ);

– *замітки і листування* (ці документи фіксують різні деталі взаємодії між менеджерами та розробниками).

Документи, що входять до складу ПЗ (*software product documentation*), описують програми ПЗ як з точки зору їх застосування користувачами, так і з точки зору їх розробників та супровідників (відповідно до призначення ПЗ). Тут слід зазначити, що ці документи будуть використовуватися не тільки на стадії експлуатації ПЗ (в її фазах застосування і супроводу), але і на стадії

розробки для керування процесом розробки (разом з робочими документами). Вони мають бути перевірені (протестовані) на відповідність програмам ПЗ. Ці документи утворюють два комплекти з різним призначенням:

- користувацька документація ПЗ (К-документація);
- документація із супроводження ПЗ (С-документація).

Користувацька документація програмних засобів.

Користувацька документація ПЗ (user documentation) пояснює користувачам, як вони повинні діяти, щоб застосувати розроблене ПЗ. Вона необхідна, якщо ПЗ припускає будь-яку взаємодію з користувачами. До такої документації належать документи, якими повинен керуватися користувач під час *інсталяції* ПЗ, *застосування* ПЗ для вирішення своїх завдань і під час *управління* ПЗ (наприклад, коли розроблене ПЗ буде взаємодіяти з іншими системами). Ці документи частково стосуються питань супроводу ПЗ, але не стосуються питань, пов'язаних з модифікацією програм.

Під час створення користувача документації необхідно розрізняти дві категорії користувачів ПЗ: ординарних користувачів ПЗ та адміністраторів ПЗ.

Ординарний користувач ПЗ (e*nd–user)* використовує ПЗ для вирішення своїх завдань (в своїй предметній галузі). Це може бути інженер, який проектує технічний пристрій, або касир, який продає залізничні квитки за допомогою ПЗ. Він може і не знати багатьох деталей роботи комп'ютера або принципів програмування.

Адміністратор ПЗ (system administrator) управляє використанням ПЗ ординарними користувачами і здійснює супровід ПЗ, не пов'язаний з модифікацією програм. Наприклад, він може регулювати права доступу до ПЗ між ординарними користувачами, підтримувати зв'язок з постачальниками ПЗ або виконувати певні дії, щоб підтримувати ПЗ в робочому стані, якщо воно включене, як частина, в іншу систему.

Склад документації користувача залежить від аудиторій користувачів, на які орієнтоване розроблене ПЗ, і від режиму використання документів. Під *аудиторією* тут розуміється контингент користувачів ПЗ, у якого є необхідність

у певній документації користувача ПЗ. Вдалий користувацький документ суттєво залежить від точного визначення аудиторії, для якої він призначений. Користувацька документація має містити інформацію, необхідну для кожної аудиторії. Під *режимом використання* документа розуміється спосіб, що визначає, як використовується цей документ. Звичайно користувачеві досить великих програмних систем потрібні або документи для вивчення ПЗ (використання у вигляді *інструкції),* або для уточнення деякої інформації (використання у вигляді *довідника).*

Тому можна вважати типовим такий склад користувацької документації для достатньо великих ПЗ.

– *Загальний функціональний опис ПЗ*. Дає коротку характеристику функціональних можливостей ПЗ. Призначено для користувачів, які повинні вирішити, наскільки необхідно їм дане ПЗ. (ГОСТ 19.401-78. Єдина система програмної документації. Текст програми, вимоги до змісту та оформлення).

– *Керівництво з інсталяції ПЗ*. Призначено для адміністраторів ПЗ. Воно має детально вказувати, як встановлювати системи в конкретному середовищі, зокрема, має містити опис комп'ютерно-зчитуваного носія, на якому поставляється ПЗ, файли, що являють собою ПЗ, і вимоги до мінімальної конфігурації апаратури.

– *Інструкція із застосування ПЗ*. Призначена для ординарних користувачів. Містить необхідну інформацію щодо застосування ПЗ, організовану у формі зручній для її вивчення.

– *Довідник із застосування ПЗ*. Призначений для ординарних користувачів. Містить необхідну інформацію щодо застосування ПЗ, організовану у формі, зручній для вибіркового пошуку окремих деталей.

– *Керівництво з керування ПЗ*. Призначено для адміністраторів ПЗ. Воно має описувати повідомлення, що генеруються, коли ПЗ взаємодіє з іншими системами, і як повинен реагувати адміністратор на ці повідомлення. Крім того, якщо ПЗ використовує системну апаратуру, цей документ може

пояснювати, як супроводжувати цю апаратуру.

Розроблення документації користувача починається відразу після створення зовнішнього опису (ГОСТ 2.119-73. Єдина система конструкторської документації. Ескізний проект. ГОСТ 2.120-73. Єдина система конструкторської документації. Технічний проект).

Якість цієї документації може істотно визначати успіх ПЗ. Вона має бути досить проста і зручна для користувача (в іншому випадку це ПЗ, взагалі, не варто було створювати). Тому, хоча чорнові варіанти (начерки) користувача документів створюються основними розробниками ПЗ, до створення їх остаточних варіантів часто залучаються професійні технічні письменники. Крім того, для забезпечення якості документації користувача розроблено низку стандартів, в яких пропонується порядок розробки цієї документації, формулюються вимоги до кожного виду користувача документів і визначаються їх структура та зміст.

Документація із супроводження програмних засобів.

Документація із супроводження ПЗ (system documentation) описує ПЗ з точки зору її розроблення. Ця документація необхідна, якщо ПЗ припускає вивчення того, як воно влаштоване (сконструйоване), і модернізацію його програм. Як уже зазначалося, супровід – це триваюче розроблення. Тому в разі потреби модернізації ПЗ до цієї роботи залучається спеціальна команда розробників-супровідників. Цій команді доведеться мати справу з такою ж документацією, яка визначала діяльність команди первісних (основних) розробників ПЗ, – з тією лише різницею, що ця документація для команди розробників-супровідників буде, як правило, чужою (вона створювалася іншою командою). Щоб зрозуміти будову і процес розроблення модернізованого ПЗ, команда розробників-супровідників повинна вивчити цю документацію, а потім внести в неї необхідні зміни, повторюючи значною мірою технологічні процеси, за допомогою яких створювалося первинне ПЗ.

Документацію по супроводженню ПЗ можна розбити на дві групи:

– документація, яка визначає будову програм і структур даних ПЗ і технологію їх розроблення;

– документацію, що допомагає вносити зміни в ПЗ.

Документація першої групи містить підсумкові документи кожного технологічного етапу розроблення ПЗ. Вона включає такі документи.

1. Зовнішній опис ПЗ (Requirements document) ГОСТ 19.202-78. Єдина система програмної документації. Специфікація, вимоги до змісту та оформлення.

2. Опис архітектури ПЗ (description of the system architecture), включаючи зовнішню специфікацію кожної її програми (підсистеми). ГОСТ 19.003-80 Схеми алгоритмів і програм. Позначення умовні графічні.

3. Для кожної програми ПЗ опис її модульної структури, включаючи зовнішню специфікацію кожного включеного в неї модуля.

4. Для кожного модуля його специфікація і опис його будови (design description).

5. Тексти модулів вибраною мовою програмування (program source code listings) ГОСТ 19.401-78. Єдина система програмної документації. Текст програми, вимоги до змісту та оформлення.

6. Документи встановлення достовірності ПЗ (validation documents), які описують, як встановлювалася достовірність кожної програми ПЗ і як інформація про встановлення достовірності пов’язувалася з вимогами до ПЗ.

Документи встановлення достовірності ПЗ включають, перш за все, документацію з тестування (схема тестування і опис комплекту тестів), але можуть включати і результати інших видів перевірки ПЗ, наприклад, докази властивостей програм. Для забезпечення прийнятної якості цієї документації корисно слідувати загальноприйнятим рекомендаціям і стандартам.

Документація другої групи містить *Керівництво із супроводження ПЗ* (system maintenance guide), яке описує особливості реалізації ПЗ (зокрема, труднощі, які довелося долати) і як враховані можливості розвитку ПЗ в його будові (конструкції). У ньому також фіксується, які частини ПЗ є апаратно і

програмно залежними.

Загальна проблема супроводу ПЗ – забезпечити, щоб всі його операції йшли послідовно (залишалися узгодженими), коли ПЗ змінюється. Щоб цьому зарадити, зв’язки і залежності між документами і їх частинами мають бути відображені в керівництві з супроводження, і зафіксовані в базі даних керування конфігурацією.

Порядок виконання роботи

1. Налагодити програмний засіб, розроблений у попередній лабораторній роботі.
2. Розробити та оформити програмну документацію.

Зміст звіту

1. Назва та мета роботи.
2. Оформлена документація на налагоджений програмний засіб.
3. Письмові відповіді на контрольні запитання.

Контрольні питання

1. Для чого потрібне документування програмного засобу?
2. Вимоги до документації на програмний засіб.
3. Призначення і зміст керівництва системного програміста.
4. Призначення і зміст керівництва користувача.

Література: [1, 2].

Лабораторна робота № 5

Тема. Структурний підхід до програмування. Стадія «Тестування програмного засобу»

Мета: ознайомитися з методами тестування програмного засобу та практичною реалізацією тестів.

Теоретичні відомості

Види тестування

Тестування програмного забезпечення включає в себе цілий комплекс дій,

аналогічних послідовності процесів розробки програмного забезпечення. До нього належать:

- постановка завдання для тесту;
- проектування тесту;
- написання тестів;
- тестування тестів;
- виконання тестів;
- вивчення результатів тестування.

Найбільш важливим є проектування тестів. Існують різні підходи до проектування тестів.

Перший полягає в тому, що тести проектуються на базі зовнішніх специфікацій програм і модулів або специфікацій сполучення модуля з іншими модулями, програма у такому разі розглядається як «чорний ящик».

Суть тесту полягає в тому, щоб перевірити, чи відповідає програма зовнішнім специфікаціям. При цьому зміст модуля не має значення. Такий підхід отримав назву – стратегія *«чорного ящика»*.

Другий підхід – стратегія *«білого ящика»,* заснований на аналізі логіки програми. Такий підход до тестування полягає у перевірці кожного шляху, кожної гілки алгоритму. При цьому зовнішня специфікація до уваги не береться.

Жоден з цих підходів не є оптимальним. Реалізація тестування методом «чорного ящика» зводиться до перевірки всіх можливих комбінацій вхідних даних. Неможливо протестувати програму, подаючи на вхід нескінченну безліч значень, тому обмежуються певним набором даних. При цьому виходять з максимальної віддачі тесту порівняно з витратами на його створення. Вона вимірюється імовірністю того, що тест виявить помилки, якщо вони є в програмі. Витрати вимірюються часом і вартістю підготовки, виконання та перевірки результатів тесту.

Тестування методом «білого ящика» також не дає 100 % гарантії того, що

модуль не містить помилок. Навіть якщо припустити, що виконані тести для всіх гілок алгоритму, не можна з повною впевненістю стверджувати, що програма відповідає її специфікаціям. Наприклад, якщо було потрібно написати програму для обчислення кубічного кореня, а програма фактично обчислює корінь квадратний, то реалізація буде абсолютно неправильною, навіть якщо перевірити всі шляхи.

Друга проблема – відсутність шляху. Якщо програма реалізує специфікації не повністю (наприклад, відсутня така спеціалізована функція, як перевірка на від'ємне значення вхідних даних програми обчислення квадратного кореня), ніяке тестування існуючих шляхів не виявить такої помилки. І, нарешті, проблема залежності результатів тестування від вхідних даних. Одні дані будуть давати правильний результат, а інші ні. Наприклад, якщо для визначення рівності трьох чисел програмується вираз вигляду *if (A+B+C)/3=A,* то воно буде вірним не для всіх значень A, B і C (помилка виникла в тому випадку, коли з двох значень B або C одне є більшим, а інше на стільки ж меншим за A). Якщо концентрувати увагу тільки на тестуванні шляхів, немає гарантії, що ця помилка буде виявлена.

Отже повне тестування програми неможливе, тобто ніяке тестування не гарантує повну відсутність помилок в програмі. Тому необхідно проектувати тести так, щоб збільшити імовірність виявлення помилки в програмі.

Стратегія « білого ящика»

Існують такі методи тестування за принципом «білого ящика»:

- покриття операторів;
- покриття рішень;
- покриття умов;
- покриття рішень / умов;
- комбінаторне покриття умов.

Метод покриття операторів

Метою цього методу тестування є виконання кожного оператора

програми хоча б один раз.

Метод покриття рішень (покриття переходів)

Згідно з методом покриття рішень кожний напрям переходу має бути реалізовано, принаймі, один раз. Цей метод включає в себе критерій покриття операторів, оскільки під час виконання всіх напрямків переходів виконаються всі оператори, що знаходяться на цих напрямках.

Метод покриття умов

Цей метод може дати кращі результати порівняно з попередніми. Відповідно до методу покриття умов записується кількість тестів, достатня для того, щоб всі можливі результати кожної умови в рішенні виконувалися принаймі один раз.

Метод покриття рішень/умов

Критерій покриття рішень/умов вимагає такого достатнього набору тестів, щоб всі можливі результати кожної умови виконувалися принаймні один раз, всі результати кожного рішення виконувалися принаймні один раз і, крім того, кожній точці входу передавалося керування принаймі один раз.

Недоліки методу:

- не завжди можна перевірити всі умови;
- неможливо перевірити умови, які приховані іншими умовами;
- недостатня чутливість до помилок в логічних виразах.

Метод комбінаторного покриття умов

Критерій комбінаторного покриття умов задовольняє також і критеріям покриття рішень, покриття умов і покриття рішень / умов.

Цей метод вимагає створення такої кількості тестів, щоб всі можливі комбінації результатів умови в кожному рішенні виконувалися принаймні один раз.

Порядок виконання роботи

1. Для тестування функції введення даних у програмному засобі, розробленому у л/р № 3, спроектувати тести обраними методами (за принципом

«білої скриньки») та провести тестування.

2. Спроектувати тести для перевірки будь-якої іншої функції на вибір студента.

3. Оформити результати тестування.

Зміст звіту

1. Назва та мета роботи.

2. Опис розроблених тестів та результатів тестування.

3. Висновки щодо тестування програмного засобу.

4. Письмові відповіді на контрольні запитання.

Контрольні питання

1. Охарактеризуйте етап реалізації та тестування програмного продукту.
2. Назвіть види тестування.
3. Назвіть критерії вибору тестів.
4. Перелічіть властивості тестів.
5. Наведіть критерії надійності програм.
6. У чому полягає оцінювання надійності програм?

Література: [1, 2].

ЛАБОРАТОРНА РОБОТА № 6

Тема. Моделювання проекту за допомогою пакета Rational Rose (Технологія RUP)

Мета: навчитися будувати основні діаграми UML для розробленого програмного продукту в об’єктно-орієнтованому середовищі пакета Rational Rose (технологія RUP).

Короткі теоретичні відомості

1. Основні відомості про роботу в середовищі Rational Rose

Елементи екрану

П’ять основних елементів інтерфейсу Rose – це браузер, вікно документації, панелі інструментів, вікно діаграми і журнал (log). Їх

призначення полягає в таому:

– браузер (browser) – використовується для швидкої навігації по моделі;

– вікно документації (documentation window) – застосовується для роботи з текстовим описом елементів моделі;

– панелі інструментів (toolbars) – застосовуються для швидкого доступу до найбільш поширених команд;

– вікно діаграми (diagram window) – використовується для перегляду і редагування однієї або декількох діаграм UML;

– журнал (log) – застосовується для перегляду помилок і звітів про результати виконання різних команд.

На рис. 2 та 3 показані різні частини інтерфейсу Rational Rose.

У верхній частині екрану знаходиться панель інструментів – Tool Bar.

Зліва – вікно Browser для швидкого доступу до діаграм. Це вікно дозволяє швидко переміщатись по дереву діаграм, перетягувати діаграми «мишею» та змінювати структуру моделі.

Під вікном Browser знаходиться вікно Documentation. У цьому вікні з’являється опис, введений розробником для активного на цей момент елемента.

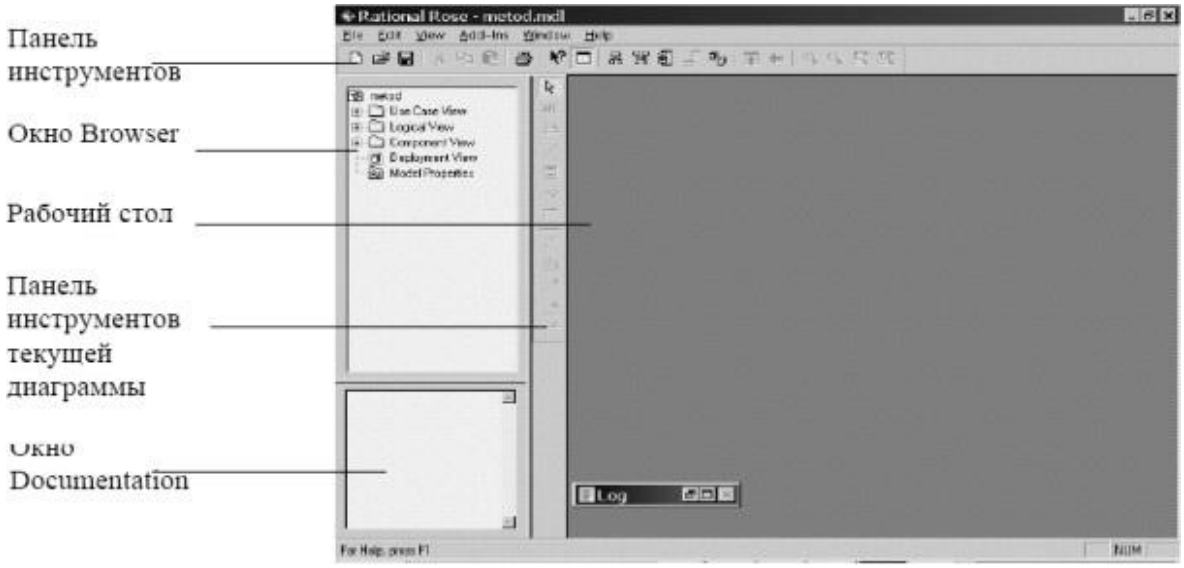

Рисунок 2 – Головне вікно «Rational Rose»

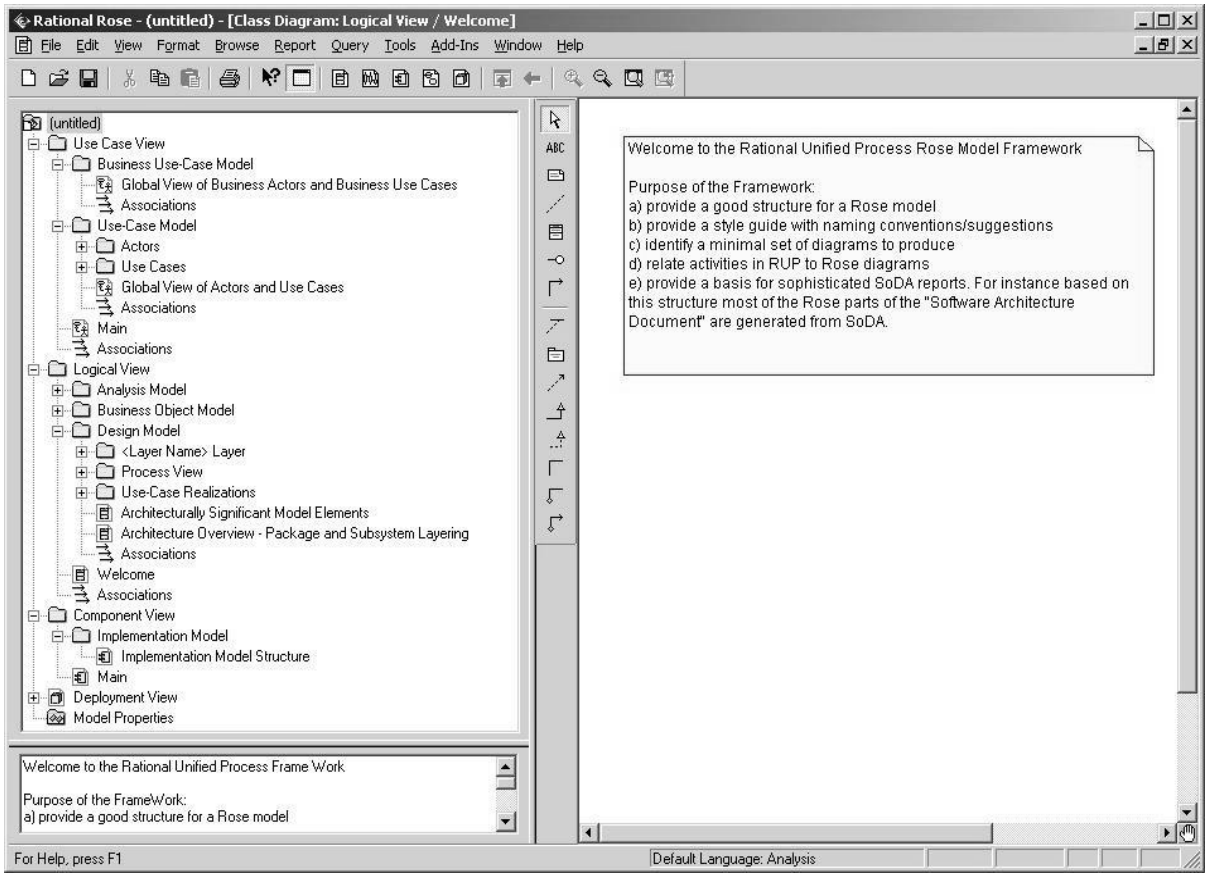

Рисунок 3 – Інтерфейс Rational Rose

У правій частині екрану знаходяться діаграми, відкриті в даний момент. Ця частина називається Робочим столом Rational Rose.

Між вікном Browser та Diagram знаходиться панель інструментів поточної діаграми, яка змінюється залежно від обраної діаграми.

Знизу робочого столу знаходиться згорнуте вікно Log (протокол). У ньому фіксуються всі дії, виконані над діаграмою.

Браузер

Браузер – це ієрархічна структура, що дозволяє здійснювати навігацію по моделі. Все, що додається до неї – дійові особи, варіанти використання, класи, компоненти – буде показано у вікні браузера.

За допомогою браузера можна:

– додавати до моделі елементи (діючі особи, варіанти використання, класи, компоненти, діаграми та ін.);

– переглядати наявні елементи моделі;
– переглядати наявні зв’язки між елементами моделі;
– переміщати елементи моделі;
– перейменовувати ці елементи;
– додавати елементи моделі до діаграми;
– зв’язувати елемент з файлом або адресою інтернет;
– групувати елементи в пакети;
– працювати з деталізованою специфікацією елемента;
– відкривати діаграму.

Браузер підтримує чотири вистави (view): подання варіантів використання, компонентів, розміщення і логічне уявлення.

Браузер організований у деревовидному стилі. Кожен елемент моделі може містити інші елементи, що знаходяться нижче його в ієрархії. Знак «-» біля елемента означає, що його гілка повністю розкрита, знак «+» – що його гілка згорнута.

Вікно документації

За його допомогою можна документувати елементи моделі Rational Rose. Наприклад, можна зробити короткий опис кожної дійової особи. Під час документування класу все, що буде написано у вікні документації, з’явиться потім як коментар у згенерованому коді, що позбавляє від необхідності згодом вносити ці коментарі вручну. Документація буде виводитися також у звітах, що створюються в середовищі Rose.

Панелі інструментів

Панелі інструментів Rose забезпечують швидкий доступ до найбільш поширених команд. У цьому середовищі існує два типи панелей інструментів: стандартна панель і панель діаграми. Стандартну панель видно завжди, її кнопки відповідають командам, які можуть використовуватися для роботи з будь-якою діаграмою. Панель діаграми своя для кожного типу діаграм UML.

Всі панелі інструментів можуть бути змінені і налаштовані користувачем.

Для цього в меню Tools> Options слід вибрати вкладку Toolbars.

Вікно діаграми

У вікні діаграми видно, як виглядає одна або кілька діаграм UML моделі. Під час внесення в елементи діаграми змін Rose автоматично оновить браузер. Аналогічно, під час внесення змін до елемента за допомогою браузера Rose автоматично оновить відповідні діаграми. Це допомагає підтримувати модель в несуперечливому стані.

Журнал

Принаймні роботи над вашою моделлю певна інформація буде спрямовуватися у вікно журналу. Наприклад, туди поміщаються повідомлення про помилки, що виникають під час генерації коду.

Не існує способу закрити журнал зовсім, але його вікно може бути мінімізовано.

Чотири подання моделі Rational Rose

У моделі Rose підтримується чотири вистави (views) – подання варіантів використання, логічне подання, подання компонентів і уявлення розміщення. Кожне з них призначено для своїх цілей.

Подання варіантів використання

Це подання містить модель бізнес-процесів і модель варіантів використання.

Логічне подання

Логічне подання концентрується на тому, як система буде реалізовувати поведінку, описану у варіантах використання. Воно дає докладну картину складових частин системи і описує взаємодію цих частин. Логічне подання включає в основному класи та діаграми класів. За їх допомогою конструюється детальний проект створюваної системи.

Логічне подання містить:

– класи;

– діаграми класів (як правило, для опису системи використовується кілька діаграм класів, кожна з яких відображає деяку підмножину всіх класів

системи);

– діаграми взаємодії (застосовуються для відображення об’єктів, що беруть участь у одному потоці подій варіанту використання);

– діаграми станів;

– пакети, які є групами взаємопов’язаних класів.

Подання компонентів містить:

– компоненти, які є фізичними модулями коду;

– діаграми компонентів;

– пакети, які є групами пов’язаних компонентів.

Подання розміщення

Останнє подання Rose – це подання розміщення. Воно відповідає фізичному розміщенню системи, яке може відрізнятися від її логічної архітектури.

В подання розміщення входять:

– процеси, що є потоками (threads), виконуваними у відведеній для них області пам’яті;

– процесори, що включають будь-які комп’ютери, здатні обробляти дані (будь-який процес виконується на одному або декількох процесорах);

– пристрої, тобто будь-яка апаратура, не здатна обробляти дані (до таких пристроїв належать, наприклад, термінали введення-виведення та принтери);

– діаграма розміщення.

Параметри налаштування відображення

У Rose є можливість налаштувати діаграми класів так, щоб:

– показувати всі атрибути та операції;

– сховати операції;

– сховати атрибути;

– показувати тільки деякі атрибути або операції;

– показувати операції разом з їх повними сигнатурами або тільки їх імена;

– показувати чи не показувати видимість атрибутів і операцій;

– показувати чи не показувати стереотипи атрибутів і операцій.

Значення кожного параметра за замовчуванням можна задати за допомогою вікна, що відкривається при виборі пункту меню Tools-> Options.

У даного класу на діаграмі можна:

– показати всі атрибути;

– сховати всі атрибути;

– показати тільки вибрані атрибути.

2. Моделювання проекту за допомогою пакета Rational Rose

Діаграма прецедентів (use case diagram)

Будь-які (в тому числі і програмні) системи проектуються з урахуванням того, що в процесі своєї роботи вони будуть використовуватися людьми та/або взаємодіяти з іншими системами. Сутності, з якими взаємодіє система в процесі своєї роботи, називаються акторами. Графічно актор зображується або «чоловічком» (рис. 4), або символом класу з відповідним стереотипом. Обидві форми подання мають один і той же зміст і можуть використовуватися в діаграмах. «Стереотипована» форма частіше застосовується для відображення системних акторів або у випадках, коли актор має властивості і їх потрібно відобразити (рис. 5).

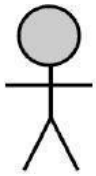

Рисунок 4 – Зображення сутності в системі

Рисунок 5 – Зображення стереотипованої форми актора

Прецедент (use case) – опис множини послідовних подій (включаючи варіанти), виконуваних системою, які призводять до результату, що спостерігає актор. Прецедент являє поведінку сутності, описуючи взаємодію між актором і системою. Прецедент не показує, «як» досягається певний результат, а тільки «що» саме виконується.

Прецеденти позначаються дуже просто – у вигляді еліпса, в середині якого зазначено його назву (рис. 6). Прецеденти і актори з'єднуються за допомогою ліній. Часто на одному з кінців лінії зображують стрілку, причому спрямована вона до того, у кого запитують сервіс, іншими словами, чиїми послугами користуються. Це просте пояснення ілюструє розуміння прецедентів як сервісів.

Рисунок 6 – Зображення прецеденту

Діаграми прецедентів належть до тієї групи діаграм, які представляють динамічні або поведінкові аспекти системи. Це засіб для досягнення взаєморозуміння між розробниками, експертами та кінцевими користувачами продукту. Такі діаграми дуже прості для розуміння і можуть сприйматися і, що важливо, обговорюватися людьми, які не є фахівцями в галузі розробки ПЗ.

Можна виділити такі цілі створення діаграм прецедентів:

– визначення кордону і контексту модельованої предметної області на ранніх етапах проектування;

– формування загальних вимог до поведінки проектованої системи;

– розробка концептуальної моделі системи для її подальшої деталізації;

– підготовка документації для взаємодії із замовниками та користувачами системи.

Діаграма класів (class diagram)

Клас (class) – категорія речей, які мають спільні атрибути і операції.

Класи – це будівельні блоки будь-якої об’єктно-орієнтованої системи. Вони являють собою опис сукупності об’єктів із загальними атрибутами, операціями, відносинами і семантикою. Під час проектуванні об’єктно-орієнтованих систем діаграми класів обов’язкові. Класи використовуються в процесі аналізу предметної області для складання словника предметної області, що розробляється. Це можуть бути як абстрактні поняття предметної області, так і класи, на які спирається розробка та які описують програмні або апаратні сутності.

Діаграма класів – це набір статичних, декларативних елементів моделі. Діаграми класів можуть застосовуватися і в разі прямого проектування, тобто в процесі розробки нової системи, і в разі зворотного проектування – описі існуючих і використовуваних систем. Інформація з діаграми класів безпосередньо відображається у вихідний код додатка – у більшості існуючих інструментів UML-моделювання можлива кодогенерацію для певної мови програмування (зазвичай Java або C++). Отже діаграма класів – кінцевий результат проектування та відправна точка процесу розробки.

Клас на діаграмі зображується у вигляді прямокутника, розділеного горизонтальними лініями на три частини. У першій частині вказується назва класу. Як правило, ім’я класу складається з одного, максимум двох слів. Друга частина містить перелік атрибутів класу, які характеризують той чи інший об’єкт цього класу в моделі предметної області. Третя частина містить перелік операцій, що відображають його поведінку в моделі предметної області (рис. 7).

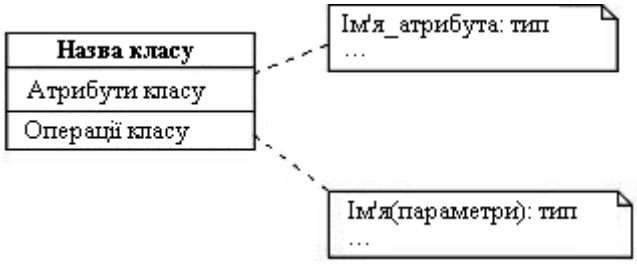

Рисунок 7 – Вигляд діаграми класів

Діаграма об’єктів (object diagram)

Об’єкт (object) – це:

– конкретна матеріалізація абстракції;

– сутність з добре визначеними межами, в якій вміщено стан і поведінку;

– екземпляр класу (вірніше, класифікатора – актор, клас або інтерфейс). Об’єкт унікально ідентифікується значеннями атрибутів, що визначають його стан в даний момент часу.

Об’єкт, як і клас, позначається прямокутником, але його ім’я підкреслюється. Під словом ім’я тут ми розуміємо назву об’єкта та назву його класу, розділені двокрапкою. Для вказівки значень атрибутів об’єкта в його позначенні може бути передбачена спеціальна секція. Ще один нюанс полягає в тому, що об’єкт може бути анонімним: це потрібно в тому випадку, якщо в даний момент не важливо, який саме об’єкт цього класу бере участь у взаємодії.

Діаграми об’єктів показують множину об’єктів – екземплярів класів (зображених на діаграмі класів) і відносин між ними в деякий момент часу. Тобто діаграма об’єктів – це свого роду знімок системи у певний момент часу, що показує множину об’єктів, їх стан і відносини між ними в цей момент.

Отже діаграми об’єктів є статичним видом системи з точки зору проектування і процесів, будучи підгрунтям для сценаріїв, що описуються діаграмами взаємодії.

Іншими словами, діаграма об’єктів використовується для пояснення і деталізації діаграм взаємодії, наприклад, діаграм послідовностей.

Діаграма послідовностей (sequence diagram)

Діаграма послідовностей відображає взаємодію об’єктів в динаміці.

У UML взаємодія об’єктів розуміється як обмін інформацією між ними. При цьому інформація набуває вигляду повідомлень. Крім того, що повідомлення несе якусь інформацію, воно також впливає на одержувача. Як бачимо, в цьому плані UML повністю відповідає основним принципам ООП,

відповідно до яких інформаційна взаємодія між об’єктами зводиться до відправки і прийому повідомлень.

Діаграма послідовностей належить до діаграм взаємодії UML, що описує поведінкові аспекти системи, але розглядає взаємодію об’єктів у часі. Іншими словами, діаграма послідовностей відображає часові особливості передачі і прийому повідомлень об’єктами.

Можна сказати, що щось подібне робить і діаграма прецедентів.

Діаграми послідовностей можна (і потрібно) використовувати для уточнення діаграм прецедентів, більш детального опису логіки сценаріїв використання. Це хороший засіб документування проекту з точки зору сценаріїв використання. Діаграми послідовностей зазвичай містять об’єкти, які взаємодіють у рамках сценарію, повідомлення, якими вони обмінюються, і які повертають результати, що пов’язані з повідомленнями. Втім, часто результати, що повертаються, позначають лише в тому випадку, якщо це не очевидно з контексту.

На діаграмі послідовностей використовуються такі позначення:

– об’єкти позначаються прямокутниками з підкресленими іменами (щоб відрізнити їх від класів);

– повідомлення (виклики методів) позначаються лініями зі стрілками;

– результати, що повертаються, позначаються пунктирними лініями зі стрілками.

Прямокутники на вертикальних лініях під кожним з об’єктів показують «час життя» (фокус) об’єктів. Досить часто їх не зображують на діаграмі, все це залежить від індивідуального стилю проектування.

Порядок виконання роботи

1. Ознайомитися з основними елементами інтерфейсу Rational Rose.
2. Побудувати діаграми прецедентів, класів, послідовностей і діяльності для програмного продукту, розробленого в попередніх лабораторних роботах.
3. Описати сценарій (flow of events) для обраного прецеденту (за вибором

студента).

Зміст звіту

1. Назва та мета роботи.
2. Опис виконаних дій у довільній формі.
3. Скріншоти розроблених діаграм.
4. Опис сценарію обраного прецеденту.
5. Письмові відповіді на контрольні запитання.

Література: [2, 4–6].

Контрольні питання

1. Для чого використовують діаграму прецедентів?
2. Поняття «актор».
3. Поняття «прецедент».
4. Що таке сценарій прецеденту?
5. Опишіть шаблон сценарію прецеденту.
6. Дайте визначення класу та об’єкта.
7. Для чого використовують діаграми класів і об’єктів?

2 КРИТЕРІЇ ОЦІНЮВАННЯ ЗНАНЬ СТУДЕНТІВ

У 11-му семестрі студенти виконують 6 лабораторних робіт. Загальна кількість балів, яку отримують студенти за виконання та захист лабораторних робіт, становить 18 балів (максимально по 3 бали на кожну лабораторну роботу).

Шкала оцінювання знань студентів: національна та ECTS

<table>
<tr><td rowspan="2">Сума балів за всі види навчальної діяльності</td><td rowspan="2">Оцінка ECTS</td><td colspan="2">Оцінка за національною шкалою</td></tr>
<tr><td>Для іспиту, курсового проекту (роботи), практики</td><td>Для заліку</td></tr>
<tr><td>90–100</td><td>A</td><td>Відмінно</td><td rowspan="5">Зараховано</td></tr>
<tr><td>82–89</td><td>B</td><td rowspan="2">Добре</td></tr>
<tr><td>74–81</td><td>C</td></tr>
<tr><td>64–73</td><td>D</td><td rowspan="2">Задовільно</td></tr>
<tr><td>60–63</td><td>E</td></tr>
<tr><td>35–59</td><td>FX</td><td>Незадовільно з можливістю повторного складання</td><td>Не зараховано з можливістю повторного складання</td></tr>
<tr><td>0–34</td><td>F</td><td>Незадовільно з обов’язковим повторним вивченням навчальної дисципліни</td><td>Не зараховано з обов’язковим повторним вивченням навчальної дисципліни</td></tr>
</table>

СПИСОК ЛІТЕРАТУРИ

1. Гагарина Л. Г. Технология разработки программного обеспечения : учебное пособие / Л. Г. Гагарина, Е. В. Кокорева, Б. Д. Виснадул. – М. : ИД «ФОРУМ», 2008. – 400 с.

2. Иванова Г. С. Технология программирования : учебник / Г. С. Иванова. – М. : Издательство МГТУ им. Н. Э. Баумана, 2002. – 243 с.

3. Андон Ф. И. Основы инженерии качества программных систем / Ф. И. Андон. – Издательский Дом «Академпериодика», 2007. – 673 с.

4. Вендров А. М. Проектирование программного обеспечения экономических информационных систем : учебник / А. М. Вендров. – М. : Финансы и статистика, 2005. – 544 с.

5. Вендров А. М. Проектирование программного обеспечения экономических информационных систем / А. М. Вендров. – М. : Финансы и статистика, 2003. – 352 с.

6. Мацяшек Л. Анализ и проектирование информационных систем с помощью UML 2.0 / Л. Мацяшек. – К. : Вильямс, 2008. – 816 с.

7. Литвинов В. А. Технологія створення програмних продуктів: лабораторний практикум для студентів напряму підготовки 6.050101 «Комп’ютерні науки» денної та заочної форм навчання / В. А. Литвинов, М. В. Гладка, О. А. Хлобистова. – К. : НУХТ, 2014. – 86 с.

8. ДСТУ 4302–2004. Інформаційні технології. Настанови щодо документування комп’ютерних програм.

Методичні вказівки щодо виконання лабораторних робіт з навчальної дисципліни «Керування розробкою програмних систем» для студентів денної форми навчання зі спеціальності 123 – «Комп'ютерна інженерія» освітнього ступеня «Магістр». Частина ІІ

Укладачі: д. т. н., проф. М. І. Гученко,
к. т. н. П. П. Костенко,
асист. Н. Л. Сохін

Відповідальний за випуск зав. кафедри «Комп'ютерні та інформаційні системи» проф. А. В. Луговой

Підп. до др. _____. Формат 60×84 1/16. Папір тип. Друк ризографія.
Ум. друк. арк. ____. Наклад _____ прим. Зам. №_____. Безкоштовно.

Видавничий відділ
Кременчуцького національного університету
імені Михайла Остроградського
вул. Першотравнева, 20, м. Кременчук, 39600