

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Запорізький національний технічний університет

МЕТОДИЧНІ ВКАЗІВКИ
ДО САМОСТІЙНОЇ РОБОТИ СТУДЕНТІВ
з дисципліни
“ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ
СИСТЕМ”
для студентів напряму
123 ”Комп'ютерна інженерія”
спеціалізацій (освітніх програм)
”Комп'ютерні системи та мережі”,
”Спеціалізовані комп’ютерні системи”
всіх форм навчання

2017 р.

Методичні вказівки до самостійної роботи студентів з дисципліни "Програмне забезпечення комп'ютерних систем" для студентів напрямку 123 "Комп'ютерна інженерія" спеціалізацій (освітніх програм) "Комп'ютерні системи та мережі", "Спеціалізовані комп'ютерні системи" всіх форм навчання/ Укл.: К.М. Касьян, М.М. Касьян. - Запоріжжя: ЗНТУ, 2017. - 29 с.

Укладачі: К.М. Касьян, доцент, к.т.н.
М.М. Касьян, доцент, к.т.н.

Рецензент М.Ю. Тягунова, доцент, к.т.н.

Відповідальний
за випуск: К.М. Касьян, доцент, к.т.н.

Затверджено
на засіданні кафедри КСМ
Протокол № 4 від 14.11.2017

Затверджено
на засіданні НМК КНТ
Протокол № 4 від 28.11.2017

ЗМІСТ

ВСТУП.....	4
ЗАВДАННЯ № 1. Уніфікована мова моделювання	5
1.1 Предмети в UML.....	5
1.2 Відносини в UML	10
1.3 Діаграми в UML.....	11
1.4 Завдання.....	13
ЗАВДАННЯ №2. Автоматизація конструювання візуальної моделі програмної системи	15
2.1 Загальна характеристика CASE-системи Rational Rose....	15
2.2 Створення діаграми Use Case	17
2.3 Завдання.....	20
ЗАВДАННЯ №3. Тестування програмного забезпечення.....	21
3.1 Тестування «чорного ящика»	21
3.2 Тестування «білого ящика»	21
3.3 Завдання.....	21
ЗАВДАННЯ №4. Супровід програмного забезпечення.....	23
4.1 Процес супроводу і його види.....	23
4.2 Завдання.....	25
ЗАВДАННЯ №5. Реінжиніринг програмного забезпечення	26
5.1 Процес реінжинірингу і його етапи	26
5.2 Завдання.....	28
ЛІТЕРАТУРА.....	29

ВСТУП

Самостійна робота спрямована на засвоєння студентами основних принципів і методів технології підтримки проектування програмних систем та інших стадій їх, життєвого циклу. Вивчаються основні засоби автоматизації проектування ПЗ. Розглядаються основні методи тестування створених програмних систем, а також види та цілі супроводження ПЗ, а також основи процесу реінжинірингу ПЗ та його складові.

У завданнях розглядається створення моделі програмної системи на основі уніфікованої мови моделювання за допомогою CASE-засобу Rational Rose.

ЗАВДАННЯ № 1. УНІФІКОВАНА МОВА МОДЕЛЮВАННЯ

UML — стандартна мова для написання моделей аналізу, проектування і реалізації об'єктно-орієнтованих програмних систем. UML може використовуватися для візуалізації, специфікації, конструювання і документування результатів програмних проектів. UML-це не візуальна мова програмування, але його моделі прямо транслюються в текст на мовах програмування (Java, C++, Visual Basic, Ada 95, Object Pascal) і навіть у таблиці для реляційної БД.

Словник UML утворюють три різновиди будівельних блоків: предмети, відносини, діаграми.

Предмети — це абстракції, що є основними елементами в моделі, відносини зв'язують ці предмети, діаграми групують колекції предметів.

1.1 Предмети в UML

В UML є чотири різновиди предметів:

- структурні предмети;
- предмети поводження;
- предмети, що групують;
- предмети, що пояснюють.

Ці предмети є базовими об'єктно-орієнтованими будівельними блоками. Вони використовуються для написання моделей.

Структурні предмети є іменниками в UML-моделях. Вони представляють статичні частини моделі - понятійні чи фізичні елементи. Перелічимо вісім різновидів структурних предметів.

1. *Клас* — опис безлічі об'єктів, що розділяють однакові властивості, операції, відносини і семантику (зміст). Клас реалізує один чи кілька інтерфейсів. Як показано на рис.1.1, графічно клас відображається у вигляді прямокутника, що звичайно включає секції з ім'ям, властивостями (атрибутами) і операціями.

Человек
имя
возраст
вес
родиться ()
креститься ()
поправиться ()
похудеть ()

Рисунок 1.1 – Класи

2. *Інтерфейс* — набір операцій, що визначають послуги класу чи компонента. Інтерфейс описує поведження елемента, видиме ззовні. Інтерфейс може представляти повні послуги класу чи компонента або частину таких послуг. Інтерфейс визначає набір специфікацій операцій (їхні сигнатури), а не набір реалізацій операцій. Графічно інтерфейс зображується у вигляді кружка з ім'ям, як показано на рис.1.2. Ім'я інтерфейсу звичайно починається з букви "І". Інтерфейс рідко показують самотійно. Звичайно його приєднують до класу чи компоненту, що реалізує інтерфейс.

3. *Кооперація* (співробітництво) визначає взаємодію і є сукупністю ролей і інших елементів, що працюють разом для забезпечення колективного поведження більш складного, чим проста сума всіх елементів. Таким чином, кооперації мають як структурний, так і поведінковий вимір. Конкретний клас може брати участь у декількох коопераціях. Ці кооперації представляють реалізацію патернів (зразків), що формують систему. Як показано на рис.1.3, графічно кооперація зображується як пунктирний еліпс, у який вписується її ім'я.



Учеба

Рисунок 1.2 – Інтерфейси

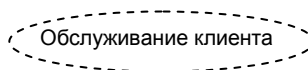


Рисунок 1.3 – Кооперації

4. *Актор* — набір погоджених ролей, що можуть грати користувачі при взаємодії з системою (її елементами Use Case). Кожна роль вимагає від системи визначеного поводження. Як показано на рис.1.4, актор зображується як дровотий чоловічок з ім'ям.

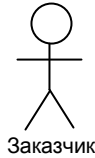


Рисунок 1.4 – Актори

5. *Елемент Use Case* (Прецедент) — опис послідовності дій (чи декількох послідовностей), виконуваних системою в інтересах окремого актора і виробляючих видимий для актора результат. У моделі елемент Use Case застосовується для структурування предметів поводження. Елемент Use Case реалізується кооперацією. Як показано на рис.1.5, елемент Use Case зображується як еліпс, у який вписується його ім'я.

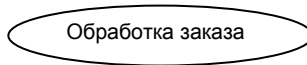


Рисунок 1.5 – Елементи Use Case

6. *Активний клас* — клас, чиї об'єкти мають один чи кілька процесів (чи потоків) і тому можуть ініціювати керуючу діяльність. Активний клас схожий на звичайний клас за винятком того, що його об'єкти діють одночасно з об'єктами інших класів. Як показано на рис.10.6, активний клас зображується як стовщений прямокутник, що звичайно включає ім'я, властивості (атрибути) і операції.

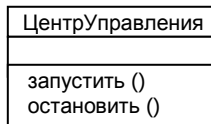


Рисунок 1.6 – Активні класи

7. *Компонент* — фізична і замінна частина системи, що відповідає набору інтерфейсів і забезпечує реалізацію цього набору інтерфейсів. У систему включаються як компоненти, що є результатами процесу розробки (файли вихідного коду), так і різні різновиди використовуваних компонентів (COM+- компоненти, Java Beans). Звичайно компонент-це фізичне упакування різних логічних елементів (класів, інтерфейсів і кооперацій). Як показано на рис.1.7, компонент зображується як прямокутник з вкладками, що звичайно включає ім'я.

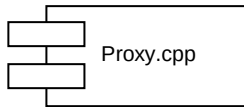


Рисунок 1.7 – Компоненти

8. *Вузол* — фізичний елемент, що існує в період роботи системи і представляє ресурс, що звичайно має пам'ять і можливості обробки. У вузлі розміщується набір компонентів, що може переміщатися від вузла до вузла. Як показано на рис.1.8, вузол зображується як куб з ім'ям.



Рисунок 1.8 – Вузли

Предмети поводження — динамічні частини UML-моделей. Вони є дієсловами моделей, представленням поводження в часі і просторі. Існують два основні різновиди предметів поводження.

1. *Взаємодія* — поводження, що укладає в собі набір повідомлень, якими обмінюється набір об'єктів у конкретному контексті для досягнення визначеної мети. Взаємодія може визначати динаміку як сукупності об'єктів, так і окремої операції. Елементами взаємодії є повідомлення, послідовність дій (поводження, викликуване повідомленням) і зв'язку (з'єднання між об'єктами). Як показано на рис.1.9, повідомлення зображується у вигляді спрямованої лінії з ім'ям її операції.



Рисунок 1.9 – Повідомлення

2. *Кінцевий автомат* — поведження, що визначає послідовність станів об'єкта чи взаємодії, виконуваний в ході його існування у відповідь на події (із урахуванням обов'язків по цих подіях). За допомогою кінцевого автомата може визначатися поведження індивідуального класу чи кооперації класів. Елементами кінцевого автомата є стани, переходи (від стану до стану), події (предмети, що викликають переходи) і дії (реакції на перехід). Як показано на рис.1.10, стан зображується як закруглений прямокутник, що звичайно включає його ім'я і його підстани (якщо вони є).

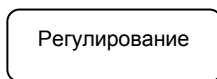


Рисунок 1.10 – Стани

Ці два елементи – взаємодії і кінцевих автоматів – є базисними предметами поведження, що можуть включатися в UML-моделі. Семантично ці елементи асоціюються з різними структурними елементами (насамперед із класами, коопераціями й об'єктами).

Групуючи предмети — організаційні частини UML-моделей. Це шухляди, по яких може бути розкладена модель. Передбачено один різновид групуючого предмета - пакет.

Пакет — загальний механізм для розподілу елементів по групах. У пакет можуть міститися структурні предмети, предмети поведження і навіть інші угруповання предметів. На відміну від компонента (який існує в період виконання), пакет — чисто концептуальне поняття. Це означає, що пакет існує тільки в період розробки. Як показано на рис.1.11, пакет зображується як папка з закладкою, на якій позначене його ім'я і, іноді, його зміст.

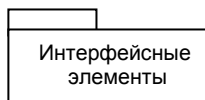


Рисунок 1.11 – Пакети

Пояснюючі предмети – частини UML-моделей, що роз'яснюють. Вони є зауваженнями, які можна застосувати для опису, пояснення і коментування будь-якого елемента моделі. Передбачено один різновид пояснюючого предмета-примітка.

Примітка — символ для відображення обмежень і зауважень, що приєднуються до елемента чи сукупності елементів. Як показано на рис.1.12 примітка зображується у вигляді прямокутника з загнутим кутом, у який вписується текстовий чи графічний коментар.

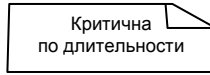


Рисунок 1.12 – Примітка

1.2 Відносини в UML

В UML є чотири різновиди відносин:

- залежність;
- асоціація;
- узагальнення;
- реалізація.

Ці відносини є базовими будівельними блоками відносин. Вони використовуються при написанні моделей.

1. *Залежність* — семантичне відношення між двома предметами, у якому зміна в одному предметі (незалежному предметі) може впливати на семантику іншого предмета (залежного предмета). Як показано на рис.1.13, залежність зображується у вигляді пунктирної лінії, що можливо спрямована на незалежний предмет і іноді має мітку.

----->

Рисунок 1.13 – Залежності

2. *Асоціація* — структурне відношення, що описує набір зв'язків, що є з'єднанням між об'єктами. Агрегація-це спеціальний різновид асоціації, що представляє структурне відношення між цілим і його частинами. Як показано на рис.1.14, асоціація зображується у вигляді суцільної лінії, що можливо спрямована та іноді має мітку, та часто включає інші "прикраси", такі як потужність і імена ролей.

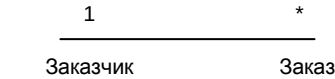


Рисунок 1.14 – Асоціації

3. *Узагальнення* — відношення спеціалізації/узагальнення, у якому об'єкти спеціалізованого елемента (нащадка, дитини) можуть замінювати об'єкти узагальненого елемента (предка, батька). Інакше кажучи, нащадок розділяє структуру і поведінку батька. Як показано на рис.1.15, узагальнення зображується у вигляді суцільної стрілки з порожнім наконечником, що вказує на батька.

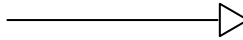


Рисунок 1.15 – Узагальнення

4. *Реалізація* – семантичне відношення між класифікаторами, де один класифікатор визначає контракт, що інший класифікатор зобов'язується виконувати (до класифікаторів відносять класи, інтерфейси, компоненти, елементи Use Case, кооперації). Відносини реалізації застосовують у двох випадках: між інтерфейсами і класами (чи компонентами), що реалізують їх; між елементами Use Case і коопераціями, що реалізують їх. Як показано на рис.1.16, реалізація зображується як щось середнє між узагальненням і залежністю.

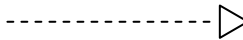


Рисунок 1.16 – Реалізації

1.3 Діаграми в UML

Діаграма — графічне представлення множини елементів, найбільше часто зображується як зв'язний граф з вершин (предметів) і дуг (відносин). Діаграми рисуються для візуалізації системи з різних точок зору, потім вони відображаються в систему. Звичайно діаграма дає не повне представлення елементів, що складають систему. Хоча той самий елемент може з'являтися у всіх діаграмах, на практиці він з'являється тільки в деяких діаграмах. Теоретично діаграма може міс-

тити будь-яку комбінацію предметів і відносин, на практиці обмежуються малою кількістю комбінацій, що відповідають п'ятьом представленням архітектури ПС. З цієї причини UML включає дев'ять видів діаграм:

- діаграми класів;
- діаграми об'єктів;
- діаграми Use Case (діаграми прецедентів);
- діаграми послідовності;
- діаграми співробітництва (кооперації);
- діаграми схем станів;
- діаграми діяльності;
- компонентні діаграми;
- діаграми розміщення (розгортання).

Діаграма класів показує набір класів, інтерфейсів, співробітництв і відносин. При моделюванні об'єктно - орієнтованих систем діаграми класів використовуються найбільш часто. Діаграми класів забезпечують статичне проектне представлення системи. Діаграми класів, що включають активні класи, забезпечують статичне представлення процесів системи.

Діаграма об'єктів показує набір об'єктів і їхні відносини. Діаграма об'єктів представляє статичний "моментальний знімок" з екземплярів предметів, що знаходяться в діаграмах класів. Як і діаграми класів, ці діаграми забезпечують статичне проектне представлення чи статичне представлення процесів системи (але з погляду реальних чи фото типових випадків).

Діаграма Use Case (діаграма прецедентів) показує набір елементів Use Case, акторів і їхніх відносин. За допомогою діаграм Use Case для системи створюється статичне представлення Use Case. Ці діаграми особливо важливі при організації і моделюванні поведінки системи, завданні вимог замовника до системи.

Діаграми послідовності і діаграми співробітництва-це різновиди діаграм взаємодії.

Діаграма взаємодії показує взаємодію, що включає набір об'єктів і їхніх відносин, а, також повідомлення, що пересилаються між об'єктами. Діаграми взаємодії забезпечують динамічне представлення системи.

Діаграма послідовності — це діаграма взаємодії, що виділяє упорядкування повідомлень за часом.

Діаграма співробітництва (діаграма кооперації) — це діаграма взаємодії, що виділяє структурну організацію об'єктів, що посилають і приймають повідомлення. Діаграми послідовності і діаграми співробітництва ізоморфні, що означає, що одну діаграму можна трансформувати в іншу діаграму.

Діаграма схем станів показує кінцевий автомат, представляє стани, переходи, події і дії. Діаграми схем станів забезпечують динамічне представлення системи. Вони особливо важливі при моделюванні поведінки інтерфейсу, чи класу кооперації. Ці діаграми виділяють таке поведінки об'єкта, що керується подіями, що особливо корисно при моделюванні реактивних систем.

Діаграма діяльності — спеціальний різновид діаграми схем станів, що показує потік від дії до дії усередині системи. Діаграми діяльності забезпечують динамічне представлення системи. Вони особливо важливі при моделюванні функціональності системи і виділяють потік керування між об'єктами.

Компонентна діаграма показує організацію набору компонентів і залежності між компонентами. Компонентні діаграми забезпечують статичне представлення реалізації системи. Вони зв'язані з діаграмами класів у тому змісті, що в компонент звичайно відображається один чи кілька класів, інтерфейсів або кооперацій.

Діаграма розміщення (діаграма розгортання) показує конфігурацію обробних вузлів періоду виконання, а також компоненти, що живуть у них. Діаграми розміщення забезпечують статичне представлення розміщення системи. Вони зв'язані із компонентними діаграмами в тому змісті, що вузол звичайно включає один чи кілька компонентів.

1.4 Завдання

Підготувати відповіді на питання [3 с.190-199]:

1. Скільки поколінь мов візуального моделювання ви знаєте?
2. Назвіть чисельність мов візуального моделювання 2-го покоління.
3. Яка необхідність привела до створення мови візуального моделювання третього покоління?

4. Поясніть призначення UML.
5. Які будівельні блоки утворюють словник UML? Охарактеризуйте їх.
6. Які різновиди предметів UML ви знаєте? Їхнє призначення?
7. Перелічите відомі вам різновиди структурних предметів UML.
8. Перелічите відомі вам різновиди предметів поводження UML.
9. Перелічите відомі вам групуючі предмети UML.
10. Перелічите відомі вам пояснюючі предмети UML.
11. Які різновиди відносин передбачені в UML? Охарактеризуйте кожне відношення.
12. Дайте характеристику діаграми класів.
13. Дайте характеристику діаграми об'єктів.
14. Охарактеризуйте діаграму Use Case.
15. Охарактеризуйте діаграми взаємодії.
16. Дайте характеристику діаграми послідовності.
17. Дайте характеристику діаграми співробітництва.
18. Охарактеризуйте діаграму схем станів.
19. Охарактеризуйте діаграму діяльності.
20. Дайте характеристику компонентної діаграми.
21. Охарактеризуйте діаграму розміщення.

ЗАВДАННЯ №2. АВТОМАТИЗАЦІЯ КОНСТРУЮВАННЯ ВІЗУАЛЬНОЇ МОДЕЛІ ПРОГРАМНОЇ СИСТЕМИ

В сучасних умовах створення складних програмних додатків неможливо без використання систем автоматизованого конструювання ПЗ (CASE-систем). CASE-системи істотно скорочують терміни і витрати розробки, роблячи допомогу інженеру в проведенні рутинних операцій, полегшуючи його роботу на самих різних етапах життєвого циклу розробки. Найбільш відомою об'єктно-орієнтованою CASE-системою є Rational Rose.

2.1 Загальна характеристика CASE-системи Rational Rose

Rational Rose — це CASE-система для візуального моделювання об'єктно-орієнтованих програмних продуктів. Візуальне моделювання-процес графічного опису розроблювального програмного забезпечення. Екран середовища Rational Rose показаний на рис.2.1.

В його складі виділимо шість елементів: рядок інструментів, панель "інструменти діаграми", вікно діаграми, браузер, вікно специфікації, вікно документації.



Рисунок 2.1 – Екран середи Rational Rose

Як показано на рис.2.2, кнопки рядка інструментів дозволяють виконувати стандартні і спеціальні дії.

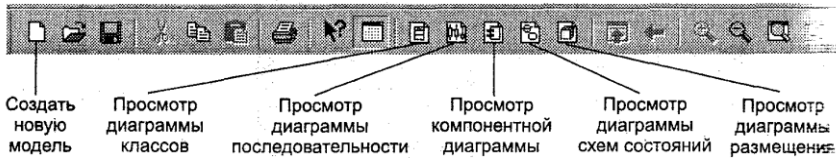


Рисунок 2.2 – Кнопки рядка інструментів Rational Rose

Зміст панелі інструментів діаграми міняється в залежності від активної діаграми. Вікно активної діаграми має синій рядок заголовка (рис.2.3).

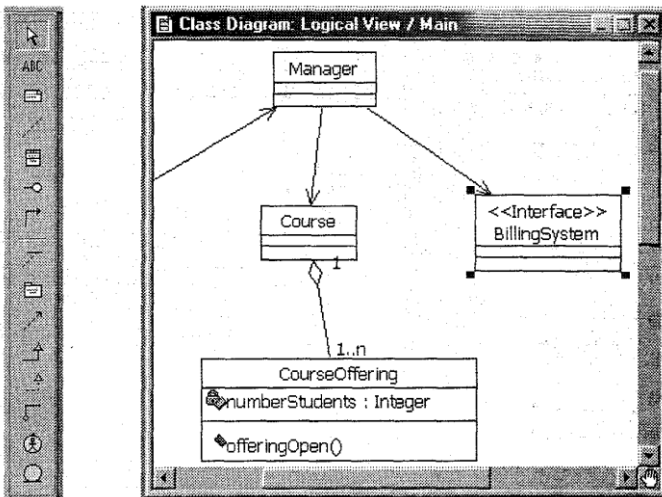


Рисунок 2.3 – Панель інструментів і вікно активної діаграми

У вікні діаграми можна створювати, відображати і змінювати діаграму мовою UML.

Браузер Rational Rose є інструментом ієрархічної навігації, що дозволяє переглядати назви і піктограми, що відображають діаграми і елементи візуальної моделі.

Вікно специфікації дозволяє задавати характеристики елемента діаграми. У полі Documentation цього вікна вводиться словесний опис даного елемента. Цей же опис можна вводити у Вікно документації RationalRose (коли даний елемент виділений у діаграмі).

Як приклад роботи з Rational Rose розглянемо побудову моделі університетської системи для реєстрації навчальних курсів (класичний приклад компанії IBM Rational).

Ця система використовується:

- професором — для завдання курсу, що викладається;
- студентом — для вибору курсу, що вивчається;
- реєстратором — для формування навчального плану і розкладу;
- обліковою системою — для визначення грошових витрат.

2.2 Створення діаграми Use Case

Моделювання проблеми реєстрації курсів почнемо зі створення діаграми Use Case. Цей тип діаграми представляється акторами, елементами Use Case і відносинами між ними. Відкриємо головну діаграму Use Case.

1. У вікні браузера клацнемо позначку + ліворуч від пакета Use Case View.

2. Для відкриття діаграми виконаємо подвійний щиглик по значку Main.

Перший крок побудови цієї діаграми складається у визначенні акторів, що фіксують ролі зовнішніх об'єктів, взаємодіючих із системою. У нашій проблемній області можна виділити 4 актора — Student (Студент), Professor (Професор), Registrar (Реєстратор) и Billing System (Облікова система) (рис.2.4).

1. На панелі інструментів клацніть по значку актора.

2. Для додавання актора в діаграму клацніть у потрібному місці діаграми.

3. Поки актор залишається виділеним, введіть ім'я Student (Студент).

4. Повторіть попередні кроки для введення трьох інших акторів (Professor, Registrar и Billing System — Професор, Регістратор, Облікова система).

Далі для кожного актора потрібно визначити відповідні елементи Use Case. Елемент Use Case представляє визначену частину функціональності, забезпечувану системою. Ви можете ідентифікувати елементи Use Case шляхом розгляду кожного актора і його взаємодії із системою. В нашій моделі актор Student хоче реєструватися на курси (Register for Courses). Актор Billing System отримує інформацію про реєстрацію. Актор Professor хоче запросити список курсу (Request a Course Roster). Нарешті, актор Registrar повинний керувати навчальним планом (Manage Curriculum) (рис.2.5).

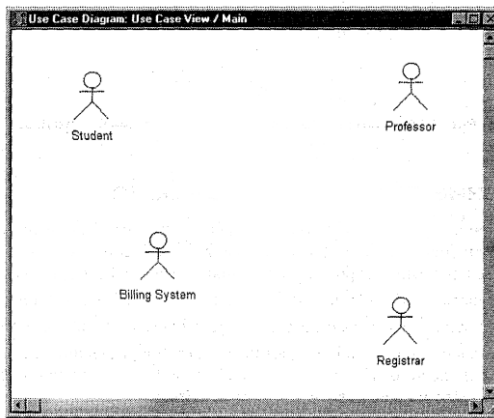


Рисунок 2.4 – Чотири актори

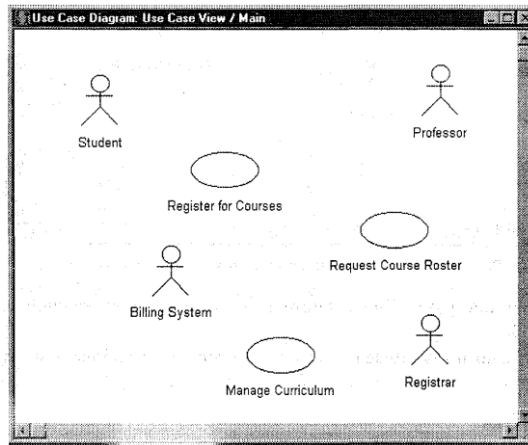


Рисунок 2.5 – Елементи Use Case для акторів

1. На панелі інструментів клацніть по значку елемента Use Case.
2. Для додавання елемента Use Case у діаграму клацніть у потрібному місці діаграми.
3. Поки елемент Use Case залишається виділеним, введіть ім'я Register for Courses.
4. Повторіть попередні кроки для введення інших елементів Use Case (Request Course Roster, Manage Curriculum).

Далі між акторами й елементами Use Case рисуються відносини. Щоб показати напрямок взаємодії (хто ініціює взаємодію), використовуються односпрямовані стрілки (uni-directional arrows). У системі реєстрації курсів актор Student ініціює елемент Use Case Register for Courses, що, у свою чергу, взаємодіє з актором Billing System. Актор Professor ініціює елемент Use Case Request Course Roster. Актор Registrar ініціює елемент Use Case Manage Curriculum (рис.2.6).

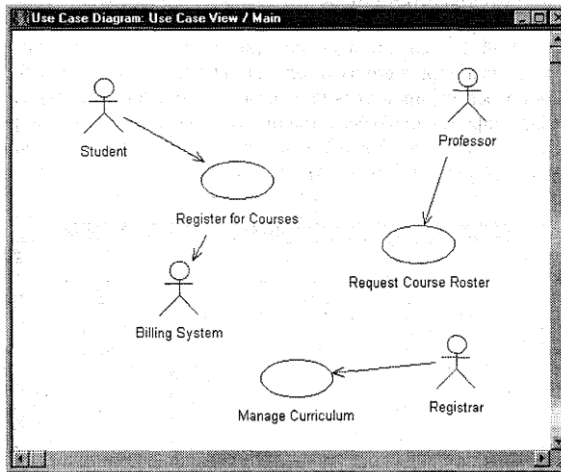


Рисунок 2.6 – Відносини між акторами і елементами Use Case

1. На панелі інструментів клацніть позначку односпрямованої асоціації (стрілці).
2. Клацніть по актору Student і перетягніть лінію на елемент Use Case Register for Courses.
3. На панелі інструментів клацніть позначку односпрямованої асоціації (стрілці).
4. Клацніть по елементу Use Case Register for Courses і перетягніть лінію на актора Billing System.
5. Повторіть попередні кроки для введення інших відносин (від актора Processor, к елементу Use Case Request Course Roster і від актора Registrar к елементу Use Case Manage Curriculum).

2.3 Завдання

Для розглянутої системи створити:

1. Діаграму послідовності [3 с.393-399],
2. Діаграму класів [3 с.399-409],
3. Компонентну діаграму [3 с.409-411],
4. Генерувати програмний код мовою C++ [3 с.411-420].

ЗАВДАННЯ №3. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Тестування-процес виконання програми з метою виявлення помилок. Кроки процесу задаються тестами.

Кожен тест визначає:

- свій набір вихідних даних і умов для запуску програми;
- набір очікуваних результатів роботи програми.

Існують два принципи тестування програми:

- функціональне тестування (тестування «чорного ящика»);
- структурне тестування (тестування «білого ящика»).

3.1 Тестування «чорного ящика»

Відомі: функції програми.

Досліджується: робота кожної функції на всій області визначення.

Найбільш популярні способи тестування «чорного ящика»:

- спосіб розбивки по еквівалентності,
- спосіб аналізу граничних значень,
- спосіб діаграм причин-наслідків.

3.2 Тестування «білого ящика»

Відомі: внутрішня структура програми.

Досліджується: внутрішні елементи програми і зв'язки між ними.

Найбільш популярні способи тестування «білого ящика»:

- спосіб тестування базового путі,
- способи тестування умов,
- спосіб тестування потоків даних,
- тестування циклів.

3.3 Завдання

Підготувати відповіді на питання [3 с.116-148]:

1. Визначте поняття тестування.

2. Що таке тест? Поясніть зміст процесу тестування.
3. Які принципи тестування ви знаєте? У чому їхня відмінність друг від друга?
4. У чому складається зміст тестування «чорного ящика», які його особливості?
5. У чому складається зміст тестування «білого ящика», які його особливості?
6. Дайте характеристику способу тестування базового шляху.
7. Поясніть кроки способу тестування базового шляху.
8. Поясніть достоїнства, недоліки й область застосування способу тестування базового шляху.
9. Дайте загальну характеристику способів тестування умов.
10. Поясніть зміст способу тестування гілок і операторів відносин. Які він має обмеження?
11. Поясніть зміст способу тестування потоків даних.
12. Поясніть особливості тестування циклів.
13. Поясніть зміст способу розбивки по еквівалентності.
14. Як вибирається тестовий варіант при тестуванні способом розбивки по еквівалентності?
15. Поясніть зміст способу аналізу граничних значень.
16. У чому суть способу діаграм причин-наслідків?
17. Дайте загальну характеристику графа причинно-наслідкових зв'язків.
18. Які функції використовуються в графі причин і наслідків?
19. Які обмеження використовуються в графі причин і наслідків?
20. Поясніть кроки способу діаграм причин-наслідків.
21. Яку структуру має таблиця рішень у способі діаграм причин-наслідків?
22. Як таблиця рішень перетворюється в тестові варіанти?

ЗАВДАННЯ №4. СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Супровід-це звичайний процес зміни системи після її постачання замовнику. Ці зміни можуть бути як елементарно простими (виправлення помилок програмування), так і більш серйозними, зв'язаними з коректуванням окремих недоробок або приведенням у відповідність з новими вимогами.

4.1 Процес супроводу і його види

Існує три види супроводу системи:

1. Супровід з метою виправлення помилок.
2. Супровід з метою адаптації ПЗ до специфічних умов експлуатації.
3. Супровід з метою зміни функціональних можливостей системи.

Ключові фактори, що визначають вартість розробки і супроводу:

1. Стабільність команди розроблювачів.
2. Відповідальність відповідно до контракту.
3. Кваліфікація фахівців
4. Вік і структура програми.

На самому загальному рівні будь-які процеси супроводу мають загальні етапи, а саме: аналіз змін, планування версій, реалізація нової версії системи і постачання системи замовнику.

Процес супроводу починається при наявності достатньої кількості запитів на зміни від користувачів, менеджерів чи покупців. Далі оцінюються можливі зміни для того, щоб визначити рівень модернізації системи, а також вартість впровадження цих змін. Якщо приймається рішення про модернізацію системи, починається етап планування нової версії системи. Під час планування аналізується можливість реалізації всіх необхідних змін, будь то виправлення помилок, адаптація чи розширення функціональних можливостей системи. Тільки після цього приймається остаточне рішення проте, які саме зміни будуть внесені в систему. Коли зміни реалізовані, виходить чергова версія системи (рис.4.1).

В ідеалі процес модернізації повинний привести до зміни системної специфікації, архітектури і програмної реалізації. Нові вимоги повинні відбивати зміни, внесені в систему. Введення в систему нових компонентів вимагає її перепроєктування, після чого необхідно повторне тестування системи.

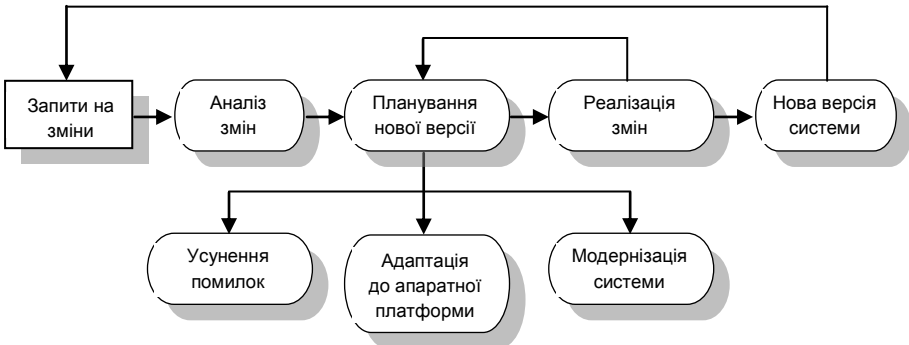


Рисунок 4.1 - Схема процесу супроводу

Іноді в екстрених випадках потрібно швидке внесення змін. У таких випадках швидка реалізація змін має велику важливість, чим чітке слідування формальностям процесу модернізації системи. Замість того щоб змінювати вимоги чи структуру системи, краще швидко внести корективи в програмний код. Однак цей підхід небезпечний тим, що вимоги, системна архітектура і програмний код поступово втрачають цілісність.

Ще одна проблема термінових змін системи полягає в тому, що через дефіцит часу з двох можливих рішень буде прийнято не найкраще (в аспекті збереження структури системи), а те, яке можна швидше й ефективніше реалізувати. В ідеалі після термінової корекції коду системи запит на зміни повинний усе ще залишатися в силі. Тому після ретельного аналізу змін можна скасувати внесені зміни і прийняти більш оптимальне рішення для модернізації системи. Однак на практиці така можливість використовується вкрай рідко, найчастіше в силу суб'єктивних причин.

4.2 Завдання

Підготувати відповіді на питання [5 с.552-567]:

1. Поясніть, чому в "молодих" організаціях—розроблювачах постійно виникають запити на зміну програмного забезпечення. Чому витрати на ці зміни неминуче зростають?

2. Поясніть, на чому засновані закони Лемана. При яких умовах ці закони не будуть діяти?

3. Опишіть труднощі, що виникають при визначенні складності супроводу. Чому для оцінювання майбутнього процесу супроводу рекомендується застосовувати різні показники складності систем?

4. Допустимо, ви менеджер по програмному забезпеченню компанії, що спеціалізується на розробці програм для нафтовидобувної промисловості, і вам дали завдання визначити фактори, що впливають на супровід систем, створених компанією. Опишіть, як ви організуєте аналіз процесу супроводу для того, щоб знайти систему показників супроводу, що підходить для вашої компанії.

5. Поясніть, чому перетворення в сервер наслідуваної системи, реалізованої на мейнфреймі, є лише короткостроковим рішенням проблеми еволюції системної архітектури.

6. Два великих міжнародних банки з різними базами даних клієнтів вирішили об'єднатися і надати доступ до об'єднаної бази даних із усіх відділень банків. Надайте найбільш коректне рішення для зміни ПЗ обох банків, обґрунтуйте свій вибір і опишіть процес його реалізації.

7. Чи буде, на вашу думку, розроблювач ПЗ, спираючись на свій професійний обов'язок, створювати програмний код, зручний для супроводу, у тому випадку, якщо це не входить у список вимог замовника ПЗ?

ЗАВДАННЯ №5. РЕІНЖИНІРИНГ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Реінжиніринг — це повторна реалізація наслідуваної системи з метою підвищення зручності її експлуатації і супроводу.

5.1 Процес реінжинірингу і його етапи

У порівнянні з більш радикальними підходами до удосконалювання систем реінжиніринг має дві переваги:

- *Зниження ризиків.*
- *Зниження витрат.*

Основне розходження між реінжинірингом і новою розробкою системи зв'язано зі стартовою крапкою початку роботи над системою. При реінжинірингу замість написання системної специфікації "з нуля" стара система є основою для розробки специфікації нової системи. Це розходження проілюстроване на рис.5.1. Традиційна розробка починається з етапу створення системної специфікації, за якою іде проектування і реалізація нової системи. Реінжиніринг ґрунтується на існуючій системі, що розроблювачами вивчається і перетворюється в нову.

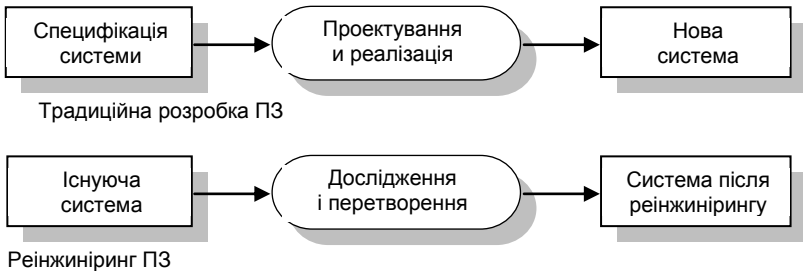


Рисунок 5.1 - Традиційна розробка і реінжиніринг ПЗ

Основні етапи процесу реінжинірингу:

- *Переклад вихідного коду.* Конвертування програми зі старої мови програмування на сучасну версію тієї ж мови або на іншу мову;

- *Аналіз програм.* Документування структури і функціональних можливостей програм на основі їхнього аналізу;
- *Модифікація структури програм.* Аналізується і модифікується керуюча структура програм з метою зробити їх більш простими і зрозумілими;
- *Розбиття на модулі.* Взаємозалежні частини програм групуються в модулі; там, де можливо, усувається надмірність. У деяких випадках змінюється структура системи;
- *Зміна системних даних.* Дані, з якими працює програма, змінюються для того, щоб відповідати нововведенням.

При реінжинірингу програм не обов'язково проходити всі стадії. Наприклад, не завжди потрібно переводити вихідний код, якщо мова програмування, на якій написана програма, усе ще підтримується розроблювачем компілятора. Якщо реінжиніринг проводиться за допомогою автоматизованих засобів, то не обов'язково відновлювати документацію на програму. Зміна системних даних необхідна, якщо в результаті реінжинірингу змінюється їхня структура. Однак реструктуризація даних у процесі реінжинірингу потрібна завжди.

Крім обсягу виконуваних робіт, є й інші фактори, що обумовлюють вартість реінжинірингу.

- *Якість програмного забезпечення, що піддається реінжинірингу.* Чим нижче якість програм і їхньої документації (якщо вона є в наявності), тим вище вартість реінжинірингу.

- *Наявність засобів підтримки процесу реінжинірингу.* Звичайно реінжиніринг економічно вигідний, якщо застосовуються CASE-засоби для автоматизованого внесення змін у програми.

- *Обсяг необхідного перетворення даних.* Вартість процесу реінжинірингу зросте при збільшенні обсягу даних, що перетворюються.

- *Наявність необхідних фахівців.* Якщо персонал, що займається супроводом системи, не може виконати реінжиніринг, це також може стати причиною підвищення вартості процесу.

Основним недоліком реінжинірингу прийнято вважати те, що з його допомогою систему можна поліпшити тільки до визначеного ступеня. Наприклад, за допомогою реінжинірингу не можливо функціонально-орієнтовану систему зробити об'єктно-орієнтованою. Основні архітектурні зміни чи повну реструктуризацію програм не можливо виконати автоматично, що також збільшує вартість реінжинірингу.

гу. І, не зважаючи на те, що реінжиніринг допоможе поліпшити супровід системи, все рівно вона буде набагато гірше в супроводі, чим нова, створена за допомогою сучасних методів інженерії ПЗ.

5.2 Завдання

Підготувати відповіді на питання [5 с.570-583]:

1. В яких випадках необхідно замінити старе ПЗ новим замість його реінжинірингу?

2. Порівняйте керуючі структури (цикли й умовні оператори) у двох будь-яких відомих вам мовах програмування. Коротко опишіть процес перекладу керуючих структур з однієї мови на іншу.

3. Напишіть ряд указівок по визначенню модулів у неструктурованій програмі.

4. Які труднощі виникнуть при перенесенні даних із СУБД одного типу в СУБД іншого типу (наприклад, з ієрархічної бази даних у реляційну чи з реляційної в об'єктно-орієнтовану)?

5. Поясніть, чому не можливо відновити системну специфікацію шляхом автоматичного аналізу вихідного коду системи.

6. Приведіть приклади для опису проблем, зв'язаних з порушенням даних при їхньому чищенні.

7. Проблема 2000 року (коли дати представлені за допомогою двох цифр) стала однією з основних причин для багатьох організацій внести корективи в супровід програм. Як це вплинуло на процес зміни даних?

ЛІТЕРАТУРА

1. Орлов, С. А. Программная инженерия: технологии разработки программного обеспечения. Учебник для вузов / С. А. Орлов - 5-е изд., обновл. и доп. – СПб.: Питер, 2016. - 640 с.
2. Лаврищева, Е.М. Software Engineering компьютерных систем. Парадигмы, технологии и CASE- средства программирования/ Е. М. Лаврищева. – К.: Наук. думка, 2013. – 283 с.
3. Орлов С.А. Технологии разработки программного обеспечения. Разработка сложных программных систем: Учеб. пособие.- 2-е изд.- СПб.: Питер, 2003.- 480с: ил.
4. Трофимов С.А. CASE-технологии: практическая работа в Rational Rose. Изд. 2-е. – М.: Бином-Пресс, 2002. – 288 с.: ил.
5. Соммервил Иан. Инженерия программного обеспечения, 6-е издание.: Пер. с англ. – М.: Издательский дом «Вильямс», 2002.- 624 с.: ил.
6. Методичні вказівки до виконання лабораторних робіт з дисципліни "програмне забезпечення комп'ютерних систем" для студентів напряму 123 "Комп'ютерна інженерія" спеціалізацій (освітніх програм) "Комп'ютерні системи та мережі", "Спеціалізовані комп'ютерні системи" усіх форм навчання/ Укл.: К.М. Касьян, М.М. Касьян.- Запоріжжя: ЗНТУ, 2017. – 70 с.