МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний економічний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп'ютерних наук

КОСТУР Тарас Михайлович

Програмний засіб для підтримки вибору CASE-засобів для проектування систем управління підприємствами/ Software for selection support of CASE tools for design the enterprise management systems

напрям підготовки: 6.050103 - Програмна інженерія
фахове спрямування - Програмне забезпечення систем

Бакалаврська дипломна робота

Виконав студент групи ПЗС-41
Т. М. Костур

_____

Науковий керівник:
викладач ЛИСЕНКО О.О.

_____

Бакалаврську дипломну роботу
допущено до захисту:

"___" _____ 20___ р.

Завідувач кафедри
_____ **А. В. Пукас**

ЗМІСТ

ВСТУП

CASE-технології (Computer Aided System Engineering) - це технології автоматизованої розробки систем (програмного забезпечення), які забезпечують за допомогою призначеного для цих цілей інструментарію (CASE-засобів) комплексну підтримку розробки або підтримки окремих стадій життєвого циклу складних програмних систем - їх специфікацію, проектування, реалізацію, тестування, супровід і розширення. CASE-технології базуються на певних методологіях проектування систем управління, а також включають в себе набір інструментальних заобів, що дозволяють в наочній формі моделювати предметну область, аналізувати цю модель на всіх етапах розробки і супроводу систем і розробляти програми відповідно до потреб користувачів [1-3].

Початковими етапами процесів життєвого циклу систем управління є стадії аналізу вимог і проектування (конструювання) структури і взаємозв'язків, що розробляється. Це найбільш трудомісткі і погано формаліловані етапи розробки систем управління, в процесі яких CASE-засоби забезпечують підвищення якості прийнятих технічних рішень і підготовку якісної проектної документації.

При розробці будь-якої системи управління виникає завдання вибору інструменту проектування, який відповідав би всім вимогам проектувальників і дозволяв грамотно і ефективно вирішувати поставлені завдання. Сучасний ринок програмних засобів налічує близько 300 видів різних CASE-засобів, найбільш потужні з яких так чи інакше використовуються практично всіма провідними західними та українськими фірмами, і багато компаній стикаються з проблемою вибору підходящого програмного рішення. Вибір ефективних і адекватних методів, які використовуються при аналізі, проектуванні, розробці або впровадженні систем управління, являє собою складну і відповідальну задачу. Розробка програмної системи, яка дозволить спростити процес

прийняття рішень стосовно вибору CASE-засобів для для проектування систем управління підприємствами є дуже актуальною задачею, що потребує вирішення.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ПІДТРИМКИ ПРОЦЕСІВ ВИБОРУ CASE-ЗАСОБІВ

1.1. Методології та методи аналізу та проектування

Одними з визначальних чинників при виборі тих чи інших інструментальних засобів є методології і методи проектування, на яких базуються відповідні CASE-засоби. Розглянемо основні методології і методи, що застосовуються при аналізі, проектуванні, розробці та впровадженні систем управління. Дамо визначення основним використовуваним далі поняттям.

Методологія - вчення про структуру, логічну організацію, методи і засоби діяльності. Метод - спосіб досягнення будь-якої мети, вирішення конкретного завдання; сукупність прийомів або операцій практичного або теоретичного освоєння (пізнання) дійсності. Модель (інформатика) - це система, дослідження якої служить засобом для отримання інформації про іншу систему. Система - множина елементів, що знаходяться у відношеннях і зв'язках один з одним, утворюють певну цілісність, єдність.

Основні методології моделювання, які використовуються в даний час на етапах аналізу, розробки та впровадження систем управління представлені на рисунку 1.1. Їх можна розділити на дві великі групи: структурні і об'єктно-орієнтовані [1].

Структурні методології базуються на декомпозиції, що моделюється та автоматизує функції: система розбивається на функціональні підсистеми, які в свою чергу діляться на функції, що підрозділяються на завдання і так далі. При цьому система зберігає цілісне уявлення, в якому всі компоненти взаємопов'язані. Найбільш відомими структурними методологіями є: структурний аналіз/структурне проектування (SA/SD), комплексна автоматизація виробничих них процесів (IDEF), архітектура інтегрованих інформаційних систем (ARIS), методології фірм розробників (ORACLE, BAAN)

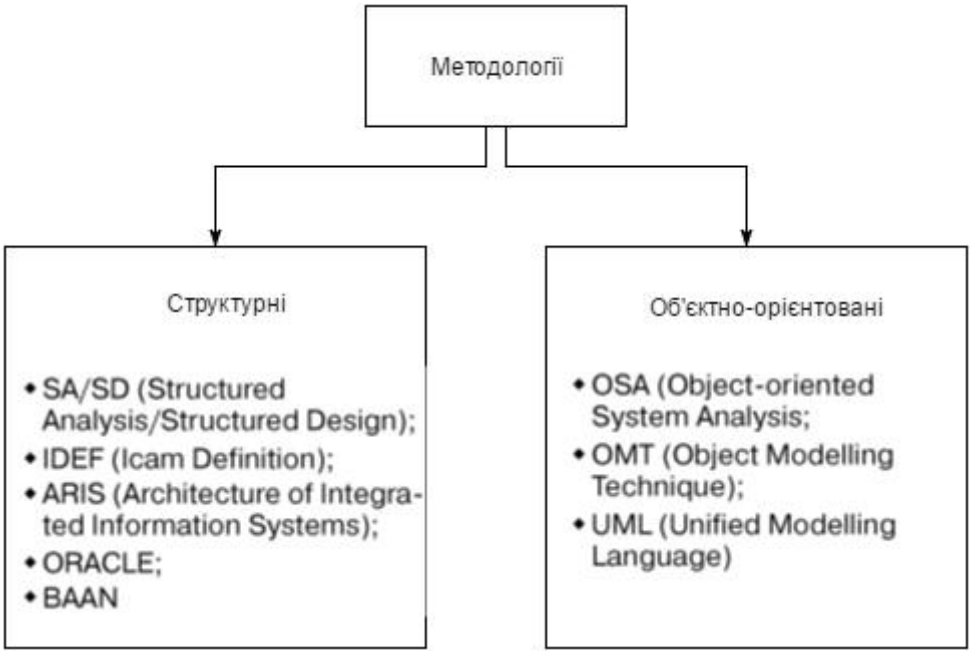

Рис. 1.1. Види методологій

Об'єктно-орієнтовані методології основані на представленні системи у вигляді сукупності об'єктів, кожен з яких є реалізацією певного типу, використовує механізми пересилання повідомлень і класи, організовані в ієрархію наслідування. Найбільш відомими об'єктними методологіями є: об'єктно-орієнтований системний аналіз (OSA), технологія об'єктного моделювання (OMT), уніфікована мова моделювання (UML).

Кожна з методологій включає в себе набір методів графічного моделювання аспектів предметної області (видів моделей, діаграм, нотацій). На рисунку 1.2 наведено класифікацію основних методів моделювання, які використовуються при аналізі, проектуванні, розробці та впровадженні систем управління [7].

Функціональні моделі - моделі, орієнтовані на функції і представляють собою структуру зображення функцій системи або середовища, інформації і об'єктів, що зв'язують ці функції. Функціональні моделі застосовують при аналізі вимог до системи, при проектуванні нової системи, а також для аналізу бізнес-процесів при прийнятті рішень про реконструкцію (реінжиніринг) системи управління.

Моделі потоків даних - моделі графічного структурного аналізу, що описують зовнішні по відношенню до системи джерела і приймачі даних, процеси, потоки даних і сховища даних, до яких здійснюється доступ. Моделі потоків даних використовуються на етапі проектування систем управління.

Моделі бізнес-процесів - моделі видів діяльності організації, що включають опис ділових об'єктів (процесів, бізнес-функцій, потоків робіт, підрозділів, посад, ресурсів, ролей, інформаційних систем, носіїв інформації і т.д.) і відображення зв'язків між ними. Відзначимо, що моделі бізнес-процесів є динамічними моделями, так як описують послідовність (і іноді умови) виконання бізнес-процесів, на відміну від інших груп структурних моделей (рисунок 1.2), описуючих виключно статику системи. Моделі бізнес-процесів використовуються при описі та реінжинірингу видів діяльності компанії, на етапах аналізу і визначення вимог до системи, а також при впровадженні систем управління на підприємствах.

Подієві моделі - це моделі, в яких функціонування системи представляється у вигляді набору станів об'єктів і переліку подій (функцій) системи. Подієві моделі використовуються на етапах аналізу вимог і проектування поведінки динамічних об'єктів (елементів) системи.

Інформаційні моделі - моделі даних конкретної предметної області або її об'єктів, вони відображають структуру даних проектованих систем. Інформаційні моделі розробляється на етапах проектування і реінжинірингу систем управління.

Ієрархічні моделі - моделі представлення системи у вигляді дерева (ієрархічної) структури, що складається з об'єктів різних рівнів. З поміж ієрархічних моделей для організацій і підрозділів, які описують організаційні структури, дерева цілей, виконуваних функцій, продуктів і послуг, що надаються і т. д. Ієрархічні моделі використовують в управлінському консалтингу при реінжинірингу систем управління.

Об'єктні статичні моделі - моделі, які не відображають динаміку системи, тобто зміни, проходять з плином часу. Основними статичними моделями на

етапах об'єктно-орієнтованого аналізу і проектування систем служать діаграми, що представляють класи, об'єкти або компоненти системи і їх відносини між собою: асоціації, узагальнення і різні види залежностей.

Об'єктні динамічні моделі - моделі, які описують зміну (динаміку) функцій (параметрів, станів об'єктів) системи. Об’єктні динамічні моделі використовуються на етапах об'єктно-орієнтованого аналізу і проектування і відображають процес зміни станів реальної або проектованої системи, показують відмінності між станами об'єктів, послідовність зміни станів і послідовність виконання функцій системи протягом часу.

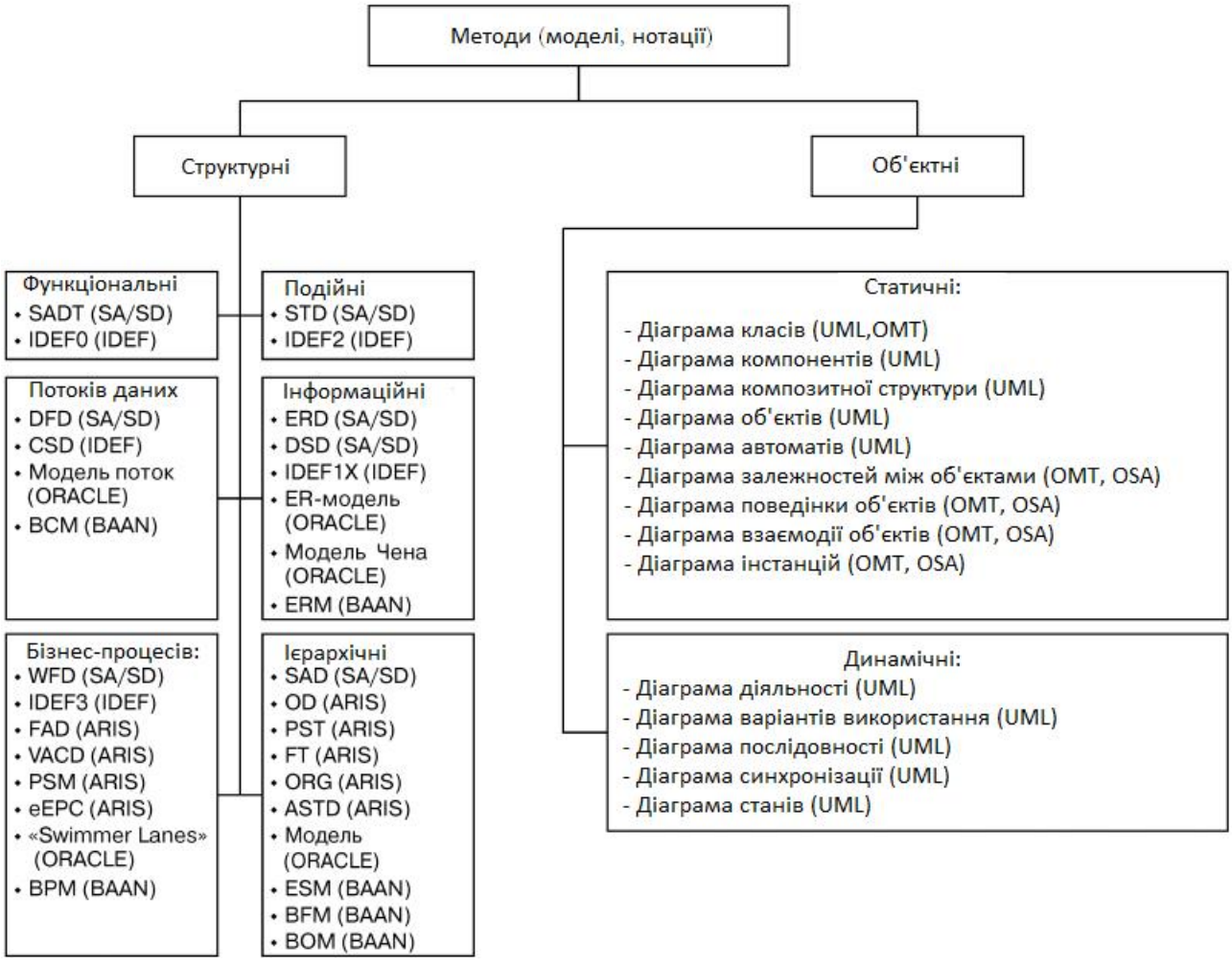

Рис. 1.2. Класифікація методів моделювання

1.2. Методи оцінки якості програмного забезпечення

Програмне забезпечення (ПЗ) є специфічним об’єктом: воно настільки багатофункціональне, що навіть схематично може бути описано тільки дуже великим числом критеріїв якості (як правило, не менше двохсот). Це не дозволяє застосовувати сучасні методи оцінки якості інших видів продукції до ПЗ в незмінному вигляді. Оцінки якості ПЗ сформовані в державні та міжнародні стандарти. Оцінка якості ПЗ являє собою сукупність операцій, які включають вибір номенклатури показників якості, визначення значень цих показників і порівняння їх з базовими значеннями. Основні існуючі методи оцінки якості ПЗ можна розділити на три групи (рисунок 1.3).

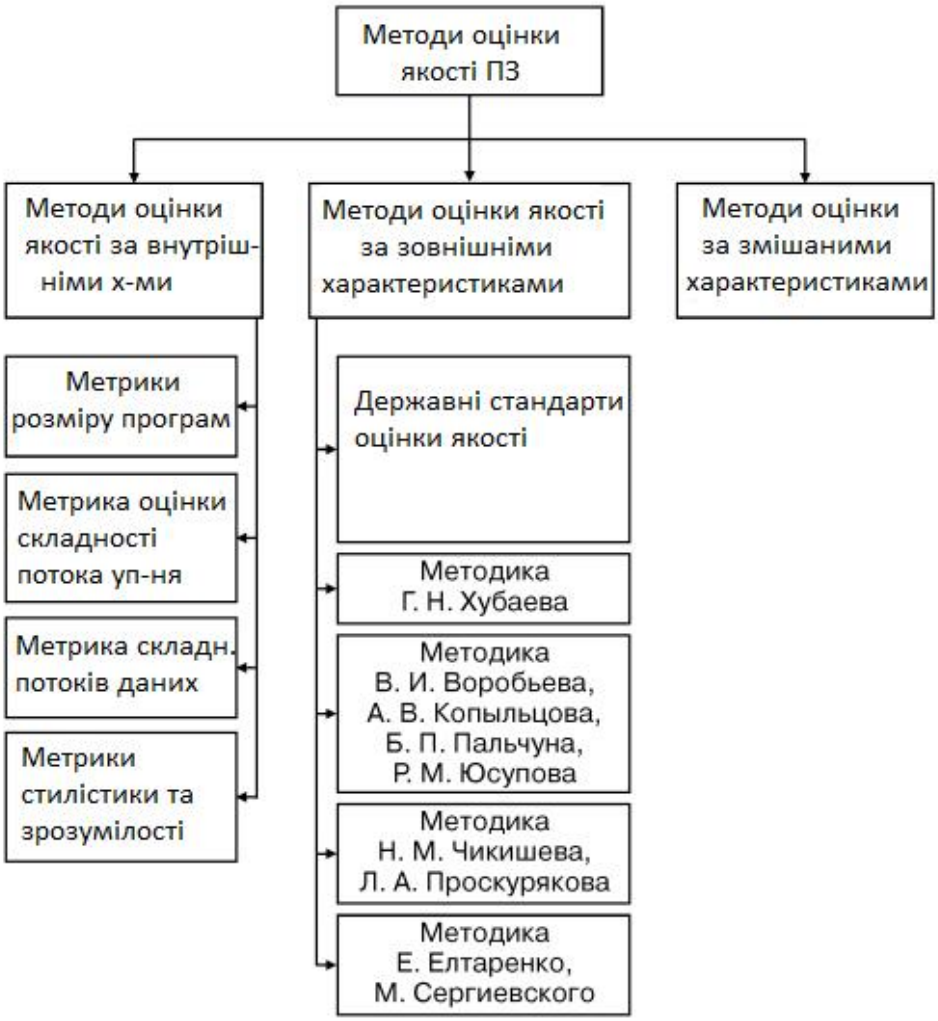

Рис. 1.3. Класифікація методів оцінки якості програмного забезпечення

До першої групи належать методи, призначені для оцінки якості ПЗ по внутрішнім характеристикам, таким технічними показниками програм, як їх складність, обчислювальна складність, структурна доцільність і т. д. До найбільш поширених методів даної групи можна віднести метрики розміру програм, метрики Холстеда, метрики Альбрехта, цикломатичну міру складності програми Маккейба, метрики стилістики програм Ван Тассел, Конаковського та ін. До другої групи належать методи оцінки якості ПЗ за зовнішніми характеристиками, спрямовані на оцінку функціональних можливостей ПЗ без обліку його внутрішньої реалізації.

В цілому, методи оцінки якості за внутрішніми характеристикам дозволяють отримати досить точні оцінки складності програм, однак їх практично неможливо використовувати стороннім експертам через відсутність доступу до вихідних текстів програм. Що стосується методів оцінки якості ПЗ по зовнішніх і змішаних характеристиках, то вони, як правило, спрямовані або на оцінку вузькоспеціалізованих програмних продуктів, або навпаки – на оцінку занадто узагальнених показників якості.

Проте, в порівнянні з методами оцінки по внутрішнім характеристикам, вони пропонують більш універсальний підхід до оцінки якості, дозволяючи розглядати програмні продукти у вигляді «чорного ящика »на будь-якому етапі його розробки. У загальному вигляді етапи оцінки якості ПЗ, характерні практично для всіх методів другої і третьої груп, можуть бути представлені наступною послідовністю:

1. Складання системи характеристик якості програмних продуктів. Як правило, ця система має вигляд ієрархічної структури і може включати як внутрішні, так і зовнішні характеристики якості ПЗ.

2. Визначення значень вагових коефіцієнтів (важливості) характеристик якості із залученням думок експертів.

3. Оцінка значень одиничних показників якості. Інформація про їх значеннях може бути отримана за результатами випробувань програмних

продуктів, а якщо це неможливо або затруднено, то за результатами експертного опитування.

4. Нормування значень одиничних показників якості.

5. Обчислення інтегральної оцінки якості програмних продуктів шляхом зваженої згортки значень його одиничних показників якості.

Рис. 1.4. Метрики програмного коду

1.3. Оцінка та вибір CASE-засобів

Процес застосування CASE-засобів складається з наступних етапів (рисунок 1.5):

- визначення потреб у CASE-засобах;
- оцінка і вибір CASE-засобів;
- виконання пілотного проекту;
- практичне впровадження CASE-засобів.

Модель процесу оцінки і вибору, що представлена на рисунку 1.6, описує найбільш загальну ситуацію оцінки і вибору, а також показує залежність між ними. Як можна побачити, оцінка та вибір можуть виконуватися незалежно

один від одного або разом, кожен з цих процесів вимагає застосування певних критеріїв.

Процес оцінки і вибору може переслідувати декілька цілей, включаючи одну або більше з таких:

- оцінка декількох CASE-засобів і вибір одного або більше з них;
- оцінка одного або більше CASE-засобів і збереження результатів для подальшого використання;
- вибір одного або більше CASE-засобів з використанням результатів попередніх оцінок.

Рис. 1.5. Етапи визначення необхідності у CASE-засобах

Як видно з рисунка 1.6, вхідний інформацією для процесу оцінки є:

- визначення призначених для користувача потреб;
- мета і обмеження проекту;
- дані про доступні CASE-засоби;
- список критеріїв, які використовуються в процесі оцінки.

Результати оцінки можуть включати результати попередніх оцінок. При цьому не слід забувати, що набір критеріїв, що використовувалися при

попередній оцінці, повинен бути сумісним з поточним набором. Конкретний варіант реалізації процесу (оцінка і вибір, оцінка для майбутнього вибору або вибір, заснований на попередніх оцінках) визначається перерахованими вище цілями.

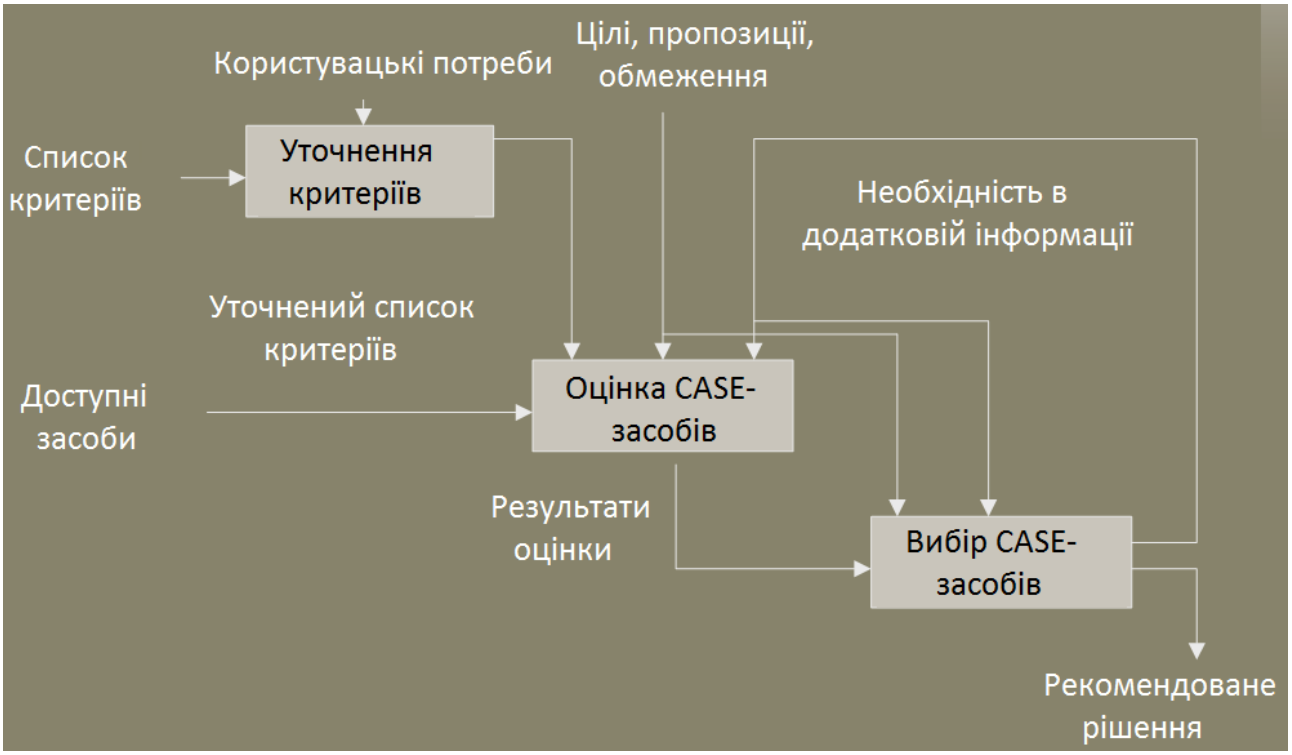

Рис. 1.6. Процес вибору CASE-засобів

Елементи процесу включають:

- мету, припущення та обмеження, які можуть уточнюватися в ході процесу;
- потреби користувачів, що відображають кількісні та якісні вимоги користувачів до CASE-засобів;
- критерії, що визначають набір параметрів, відповідно до яких проводиться оцінка і прийняття рішення про вибір;
- формалізовані результати оцінок одного або більше засобів;
- рекомендоване рішення (зазвичай або рішення про вибір, або подальшу оцінка).

Процес оцінки і/або вибору може бути початий тільки тоді, коли особа, група або організація повністю визначила для себе конкретні потреби і

формалізувала їх у вигляді кількісних і якісних вимог в заданій предметній області. Термін "призначені для користувача вимоги" далі означає саме такі формалізовані вимоги.

Користувач повинен визначити конкретний порядок дій і прийняття рішень з будь-якими необхідними ітераціями. Наприклад, процес може бути представлений у вигляді дерева рішень з його послідовним обходом і вибором підмножин кандидатів для більш детальної оцінки. Опис послідовності дій має визначати потік даних між ними.

Визначення переліку критеріїв засноване на призначених для користувача вимогах і включає:

- вибір критеріїв для використання з наведеного далі переліку;
- визначення додаткових критеріїв;
- визначення області використання кожного критерію (оцінка, вибір або обидва процеси);
- визначення однієї або більше метрик для кожного критерію оцінки;
- призначення ваги кожному критерію при виборі;
- процес оцінки;

Метою процесу оцінки є визначення функціональності і якості CASE-засобів для подальшого вибору. Оцінка виконується відповідно до конкретних критеріїв, її результати включають як об'єктивні, так і суб'єктивні дані по кожному засобу.

Процес оцінки включає наступні дії:

- формулювання завдання оцінки, включаючи інформацію про мету і масштаби оцінки;
- визначення критеріїв оцінки, що випливає з визначення завдання;
- визначення засобів-кандидатів шляхом перегляду списку кандидатів і аналізу інформації про конкретні засоби;
- оцінка засобів-кандидатів в контексті обраних критеріїв. Необхідні для цього дані можуть бути отримані шляхом аналізу самих засобів і їх документації, опитування користувачів, роботи з демо-версіями, виконання

тестових прикладів, експериментального застосування засобів і аналізу результатів попередніх оцінок;

- підготовка звіту за результатами оцінки.

Одним з найважливіших критеріїв в процесі оцінки може бути потенційна можливість інтеграції кожного з засобів-кандидатів з іншими засобами, які вже знаходяться в експлуатації або плановаными до використання в даній організації.

Масштаб оцінки повинен встановлювати необхідний рівень деталізації, необхідні ресурси і ступінь придатності її результатів. Наприклад, оцінка повинна виконуватися для набору з одного або більше конкретних CASE-засобів; CASE-засоби, що підтримують один або більше конкретних процесів створення та супроводження ПЗ або CASE-засобів, що підтримують один або більше проектів або типів проектів.

Список CASE-засобів - можливих кандидатів формується з різних джерел: оглядів ринку ПЗ, інформації постачальників, оглядів CASE-засобів і інших подібних публікацій.

Наступним кроком є отримання інформації про CASE-засоби або отримання їх самих або і те, і інше. Ця інформація може складатися з оцінок незалежних експертів, повідомлень і звітів постачальників CASE-засобів, результатів демонстрації можливостей CASE-засобів з боку постачальників та інформації, отриманої безпосередньо від реальних користувачів. Самі CASE-засоби можуть бути отримані шляхом придбання, у вигляді оціночної копії або іншими способами.

Оцінка і накопичення відповідних даних може виконуватися такими способами:

- аналіз CASE-засобів і документації постачальника;
- опитування реальних користувачів;
- аналіз результатів проектів, які використовували дані CASE-засоби;
- перегляд демонстрацій і опитування демонстраторів;
- виконання тестових прикладів;

- застосування CASE-засобів в пілотних проектах;
- аналіз будь-яких доступних результатів попередніх оцінок.

Існують як об'єктивні, так і суб'єктивні критерії. Результати оцінки відповідно до конкретного критерієм можуть бути двійковими, знаходитися в деякому числовому діапазоні, являти собою просто числове значення або мати будь-яку іншу форму.

Для об'єктивних критеріїв оцінка повинна виконуватися шляхом відтворення процедури, щоб будь-який інший фахівець, що виконує оцінку, міг отримати такі ж результати. Якщо використовуються тестові приклади, їх набір повинен бути заздалегідь визначений, уніфікований і документований.

За суб'єктивними критеріями CASE-засіб повинен оцінюватися групою фахівців, що використовують одні й ті ж критерії. Необхідний рівень досвіду фахівців або груп повинен бути заздалегідь визначений.

Результати оцінки повинні бути стандартним чином задокументовані (для полегшення подальшого використання) і, при необхідності, затверджені.

Звіт за результатами оцінки повинен містити наступну інформацію: вступ; загальний огляд процесу і перелік основних результатів; передумови. мета оцінки і бажані результати, період часу, протягом якого виконувалася оцінка, визначення ролей і відповідного досвіду фахівців, які виконували оцінку; підхід до оцінки. Опис загального підходу, включаючи отримані CASE-засоби, інформацію, визначальну контекст і масштаб оцінки, а також будь-які припущення і обмеження;

Інформація про CASE-засоби. Вона повинна включати наступне: 1) найменування CASE-засобу; 2) версію CASE-засобу; 3) дані про постачальника, включаючи контактну адресу та телефон; 4) конфігурацію технічних засобів; 5) вартісні дані; 6) опис CASE-засобу, що включає підтримувані даними засобом процеси створення і супроводу ПЗ, програмне середовище CASE-засобу (зокрема, підтримувані мови програмування, операційні системи, сумісність з базами даних), функції CASE-засобу, вхідні/вихідні дані і область застосування;

Етапи оцінки. Конкретні дії, що виконуються в процесі оцінки, повинні бути описані зі ступенем деталізації, необхідної як для розуміння масштабу і глибини оцінки, так і для її повторення при необхідності;

Конкретні результати. Результати оцінки повинні бути представлені в термінах критеріїв оцінки. У тих випадках, коли звіт охоплює цілий ряд CASE-засобів або результати даної оцінки будуть зіставлятися з результатами інших оцінок, необхідно звернути особливу увагу на формат представлення результатів, що сприяє такому порівнянню. Суб'єктивні результати повинні бути відділені від об'єктивних і повинні супроводжуватися необхідними поясненнями;

Виявлені факти та висновки; Формулювання завдання оцінки і уточнений список критеріїв.

Процес вибору. Процеси оцінки і вибору тісно взаємопов'язані між собою. За результатами оцінки мети вибору і/або критерії вибору і їх ваги можуть зажадати модифікації. У таких випадках може знадобитися повторна оцінка. Коли аналізуються остаточні результати оцінки і до них застосовуються критерії вибору, може бути рекомендовано придбання CASE-засобу або набору CASE-засобів. Альтернативою може бути відсутність адекватних CASE-засобів, в цьому випадку рекомендується розробити новий CASE-засіб, модифікувати існуючий або відмовитися від впровадження.

Процес вибору тісно взаємозалежний з процесом оцінки і включає наступні дії:

- формулювання завдань вибору, включаючи цілі, припущення і обмеження;

- виконання всіх необхідних дій по вибору, включаючи визначення і ранжування критеріїв, визначення засобів-кандидатів, збір необхідних даних та застосування ранжируваних критеріїв до результатів оцінки для визначення засобів з найкращими показниками. Для багатьох користувачів важливим критерієм вибору є інтегрованість CASE-засобу з існуючою середовищем;

виконання необхідної кількості ітерацій з тим, щоб вибрати (або відкинути) засоби, які мають схожі показники; підготовка звіту за результатами вибору.

В процесі вибору можливе одержання двох результатів:

- рекомендацій по вибору конкретного CASE-засобу;
- запиту на отримання додаткової інформації до процесу оцінки.

Масштаб вибору повинен встановлювати необхідний рівень деталізації, необхідні ресурси, графік і очікувані результати. Існує ряд параметрів, які можуть бути використані для визначення масштабу, включаючи:

- використання попереднього відбору (наприклад, відбір тільки засобів, що працюють на конкретній платформі);
- використання раніше отриманих результатів оцінки, результатів оцінки з зовнішніх джерел або комбінації того й іншого;

У тому випадку, якщо попередні оцінки виконувалися з використанням різних наборів критеріїв або виконувалися з використанням конкретних критеріїв, але різними способами, результати оцінок повинні бути представлені в узгодженій формі. Після завершення цього етапу оцінка кожного CASE-засобу повинна бути представлена в рамках єдиного набору критеріїв і повинна бути безпосередньо порівнянна з іншими оцінками.

Алгоритми, які зазвичай використовуються для вибору, можуть бути засновані на масштабі або ранзі. Алгоритми, засновані на масштабі, обчислюють єдине значення для кожного CASE-засобу шляхом множення ваги кожного критерію на його значення (з урахуванням масштабу) і складання всіх частин. CASE-засіб з найвищим результатом отримує перший ранг. Алгоритми, засновані на ранзі, використовують ранжування CASE-засобів - кандидатів за окремими критеріями або групами критеріїв відповідно до значень критеріїв у заданому масштабі. Потім, аналогічно попередньому, ранги зводяться разом і обчислюються загальні значення рангів.

При аналізі результатів вибору передбачається, що процес вибору завершений, CASE-засіб вибрано і рекомендовано до використання. Проте, може знадобитися більш точний аналіз для визначення ступеня залежності

значень ключових критеріїв від відмінностей в значеннях характеристик CASE-засобів - кандидатів. Такий аналіз дозволить визначити, наскільки результат ранжування CASE-засобів залежить від оптимальності вибору вагових коефіцієнтів критеріїв. Він також може використовуватися для визначення суттєвих відмінностей між CASE-засобами з дуже близькими значеннями критеріїв або рангами. Якщо жодне з CASE-засобів не задовольняє мінімальним критеріям, вибір (можливо, разом з оцінкою) може бути повторений для інших CASE-засобів - кандидатів.

Якщо відмінності між самими кращими кандидатами несуттєві, додаткова інформація може бути отримана шляхом повторного вибору (можливо, разом з оцінкою) з використанням додаткових або інших критеріїв.

Рекомендації по вибору повинні бути строго обгрунтовані. У разі відсутності адекватних CASE-засобів, як було зазначено вище, рекомендується розробити новий CASE-засіб, модифікувати існуючий або відмовитися від впровадження.

Критерії оцінки та вибору. Критерії формують базис для процесів оцінки і вибору і можуть приймати різні форми, включаючи:

- числові заходи в широкому діапазоні значень, наприклад, обсяг необхідної пам'яті;
- числові заходи в обмеженому діапазоні значень, наприклад, простота освоєння, виражена в балах від 1 до 5;
- виконавчі заходи (істина / неправда, так / ні), наприклад, здатність генерації документації у форматі Postscript;
- заходи, які можуть приймати одне або більше з кінцевих множин значень, наприклад, платформи, для яких підтримується CASE-засіб.

Типовий процес оцінки і / або вибору може використовувати набір критеріїв різних типів.

Структура набору критеріїв наведена на рисунку 1.7. Кожен критерій має бути обраний і адаптований експертом з урахуванням особливостей конкретного процесу. У більшості випадків невелика кількість тих описаних

нижче критеріїв виявляються прийнятними для використання, при цьому також додаються додаткові критерії. Вибір і уточнення набору використовуваних критеріїв є критичним кроком у процесі оцінки і / або вибору.

Функціональні характеристики. Критерії першого класу призначені для визначення функціональних характеристик CASE-засобу. Вони в свою чергу поділяються на ряд груп і підгруп.

Рис. 1.7. Структура критеріїв оцінки CASE-засобів

Середовище функціонування:

Проектна середовище: підтримка процесів життєвого циклу. Це впливає на доступність процесів ЖЦ, які підтримує CASE-засіб. Прикладами таких процесів є аналіз вимог, проектування, реалізація, тестування і оцінка, супровід, забезпечення якості, управління конфігурацією і управління проектом, причому вони залежать від прийнятої користувачем моделі ЖЦ.

- галузь застосування. Прикладами є системи обробки транзакцій, системи реального часу, інформаційні системи і т.д.

- розмір, який підтримуються. Визначає обмеження на такі величини, як кількість рядків коду, рівнів вкладеності, розмір бази даних, кількість елементів даних, кількість об'єктів конфігураційного управління.

ПЗ/технічні засоби: необхідні технічні засоби. Обладнання, необхідне для функціонування CASE-засобу, включаючи тип процесора, обсяг оперативної і дискової пам'яті.

- підтримувані технічні засоби. Елементи обладнання, які можуть використовуватися CASE-засобом, наприклад, пристрої введення/виводу.

- необхідне ПЗ для функціонування CASE-засобу, включаючи операційні системи і графічні оболонки.

- підтримуване ПЗ. Програмні продукти, які можуть використовуватися CASE-засобом.

Технологічне середовище:

- відповідність стандартам технологічного середовища. Такі стандарти стосуються мови, бази даних, сховища, комунікацій, графічного інтерфейсу користувача, документації, розробки, управління конфігурацією, безпеки, стандартів обміну інформацією та інтеграції за даними, з управління та по призначеному для користувача інтерфейсу.

- сумісність з іншими засобами. Здатність до взаємодії з іншими засобами, включаючи безпосередній обмін даними (прикладами таких засобів є текстові процесори, бази даних та інші CASE-засоби). Можливість перетворення сховища або його частини в стандартний формат для обробки іншими засобами.

- підтримувана методологія. Набір методів і методик, підтримуваних CASE-засобом. Прикладами є структурний або об'єктно-орієнтований аналіз і проектування.

- підтримувані мови. Всі мови, які використовуються CASE-засобом. Прикладами таких мов є мови програмування (Кобол, Ада, С), мови баз даних і мови запитів (DDL, SQL), графічні мови (Postscript, HPGL), мови специфікації

проектних вимог і інтерфейси операційних систем (мови керування завданнями).

Функції, орієнтовані на фази життєвого циклу:

- моделювання: дані критерії визначають здатність виконання функцій, необхідних для специфікації вимог до ПЗ і перетворенню їх в проект: побудова діаграм - можливість створення і редагування діаграм різних типів, що представляють інтерес для користувача.
- графічний аналіз. Можливість аналізу графічних об'єктів, а також зберігання та подання проектної інформації в графічному поданні. У більшості випадків графічні аналізатори інтегровані із засобами побудови діаграм.
- введення і редагування специфікацій вимог і проектних специфікацій. До специфікаціям такого роду відносяться описи функцій, даних, інтерфейсів, структури, якості, продуктивності, технічних засобів, середовища, витрат і графіків.
- мова специфікації вимог і проектних специфікацій. Можливість імпорту, експорту і редагування специфікацій з використанням формальної мови.
- моделювання даних. Можливість введення і редагування інформації, яка описує елементи даних системи і їх відношення.
- моделювання процесів. Можливість введення і редагування інформації, яка описує процеси системи і їх відношення.
- проектування архітектури ПЗ. Проектування логічної структури ПЗ;
- структури модулів, інтерфейсів та ін.
- імітаційне моделювання. Можливість динамічного моделювання різних аспектів функціонування системи на основі специфікацій вимог і / або проектних специфікацій, включаючи зовнішній інтерфейс і продуктивність (наприклад, час відгуку, коефіцієнт використання ресурсів і пропускну здатність).
- прототипування. Можливість проектування і генерації попереднього варіанту всієї системи або її окремих компонент на основі

специфікацій вимог і/або проектних специфікацій. Прототипування в основному стосується зовнішнього призначеного для користувача інтерфейсу і здійснюється при безпосередній участі користувачів.

- генерація екранних форм. Можливість генерації екранних форм на основі специфікацій вимог і / або проектних специфікацій.

- можливість трасування. Можливість наскрізного аналізу функціонування системи від специфікації вимог до кінцевих результатів (встановлення і відстеження відповідностей і зв'язків між функціональними та іншими зовнішніми вимогами до ІС, технічними рішеннями і результатами проектування). Пряме трасування (перевірка обліку всіх вимог) і зворотне трасування (пошук проектних рішень, не пов'язаних ні з якими зовнішніми вимогами).

- синтаксичний і семантичний контроль проектних специфікацій. Контроль синтаксису діаграм і типів їх елементів, контроль декомпозиції функцій, перевірка специфікацій на повноту і несуперечність.

- інші види аналізу. Конкретні додаткові види аналізу можуть включати алгоритми, потоки даних, нормалізацію даних, використання даних, призначений для користувача інтерфейс.

Автоматизоване проектування звітів.

Реалізація: зачіпає функції, пов'язані зі створенням виконуваних елементів системи (програмних кодів) або модифікацією існуючої системи. Багато з перерахованих нижче критеріїв залежать від конкретних мов і включають наступні:

- синтаксично кероване редагування. Можливість введення і редагування вихідних кодів на одному або декількох мовах з одночасним синтаксичним контролем.

- генерація коду. Можливість генерації кодів на одному або декількох мовах на основі проектних специфікацій. Типи генерованого коду можуть включати звичайний програмний код, схему бази даних, запити, екрани/меню.

- компіляція коду. Конвертування вихідного коду. Можливість перетворення коду з однієї мови в інший.

- аналіз надійності. Можливість кількісно оцінювати параметри надійності ПЗ, такі, як кількість помилок і ін.

- реверсний інжиніринг. Можливість аналізу існуючих вихідних кодів і формування на їх основі проектних специфікацій.

- реструктуризація вихідного коду. Можливість модифікації формату і/або структури існуючого вихідного коду.

- аналіз вихідного коду. Прикладами такого аналізу можуть бути визначення розміру коду, обчислення показників складності, генерація перехресних посилань і перевірка на відповідність стандартам.

- налагодження. Типові функції налагодження - трасування програм, виділення вузьких місць і найбільш часто використовуваних фрагментів коду і т.д.

- тестування: Критерії тестування включають наступні: опис тестів. Типові можливості включають генерацію тестових даних, алгоритмів тестування, необхідних результатів і т.д.

- фіксація і повторення дій оператора. Можливість фіксувати дані, що вводяться оператором за допомогою клавіатури, миші і т.д., редагувати їх і відтворювати в тестових прикладах.

- автоматичний запуск тестових прикладів.

- регресійні тестування. Можливість повторення і модифікації раніше виконаних тестів для визначення відмінностей в системі і / або середовищі.

- автоматизований аналіз результатів тестування. Типові можливості включають порівняння очікуваних і реальних результатів, порівняння файлів, статистичний аналіз результатів і ін.

- аналіз тестового покриття. Оснащеність засобами контролю вихідного коду і аналіз тестового покриття. Перевіряються, зокрема, звернення до операторів, процедур і змінних.

- аналіз продуктивності. Можливість аналізу продуктивності програм. Аналізовані параметри продуктивності можуть включати використання центрального процесора, пам'яті, звернення до певних елементів даних і/або сегментів коду, тимчасові характеристики і т.д.

- аналіз виняткових ситуацій в процесі тестування.

-динамічне моделювання середовища. Зокрема, можливість автоматично генерувати модельований вхідні дані системи.

Загальні функції:

Наведені нижче критерії визначають функції CASE-засобів, що охоплюють всю сукупність фаз ЖЦ. Підтримка всіх цих функцій здійснюється за допомогою сховища.

Документування: редагування текстів і графіки. Можливість вводити і редагувати дані в текстовому і графічному форматі.

- редагування за допомогою форм. Можливість підтримувати форми, визначені користувачами, вводити і редагувати дані відповідно до форм.

- підтримка функцій і форматів гіпертексту.

- відповідність стандартам документування.

- автоматичне вилучення даних зі сховищ і генерація документації за специфікаціями користувача.

Управління конфігурацією:

- контроль доступу і змін. Можливість контролю доступу на фізичному рівні до елементів даних і контролю змін. Контроль доступу включає можливості визначення прав доступу до компонентів, а також вилучення елементів даних для модифікації, блокування доступу до них на час модифікації і приміщення назад в репозиторій.

- відстеження модифікацій. Фіксація і ведення журналу всіх модифікацій, внесених в систему в процесі розробки або супроводу.

- управління версіями. Ведення і контроль даних про версії системи і всіх її колективно використовуваних компонентах.

- облік стану об'єктів конфігураційного управління. Можливість отримання звітів про всіх послідовних версіях, вмісті та стан різних об'єктів конфігураційного управління.

- генерація версій і модифікацій. Підтримка користувальницького опису послідовності дій, необхідних для формування версій і модифікацій, і автоматичне виконання цих дій.

- архівування. Можливість автоматичного архівування елементів даних для подальшого використання.

Управління проектом: управління роботами і ресурсами. Контроль і управління процесом проектування ІС в термінах структури завдань і призначення виконавців, послідовності їх виконання, завершеності окремих етапів проекту і проекту в цілому. Можливість підтримки планових даних, фактичних даних і їх аналізу. Типові дані включають графіки (з урахуванням календаря, робочих годин, вихідних і ін.), Комп'ютерні ресурси, розподіл персоналу, бюджет і ін. оцінка. Можливість оцінювати витрати, графік та інші проектні параметри, що вводяться користувачами.

Управління процедурою тестування. Підтримка управління процедурами і програмою тестування, наприклад, управління розкладом планованих процедур, фіксація і запис результатів тестування, генерація звітів і т.д. Управління якістю. Введення відповідних даних, їх аналіз та генерація звітів.

Корегуюча дія. Підтримка управління коригувальних дій, включаючи обробку повідомлень про проблемні ситуації.

Адміністрування сховища. Контроль і забезпечення цілісності проектних даних.

Автоматичне резервування (визначається постачальником або плановане користувачем).

Безпека. Захист від несанкціонованого доступу. Обробка помилок. Виявлення помилок в роботі системи, повідомлення користувача, коректне завершення роботи або збереження стану до моменту переривання.

Простота використання: зручність для користувача інтерфейсу. Зручність розташування та подання часто використовуваних елементів екрану, способів введення даних і ін. локалізація (відповідно до вимог даної країни).

Простота освоєння. Трудові і часові витрати на освоєння засобів. Адаптованість до конкретних вимог користувача. Адаптованість до різних алфавітів, режимах текстового і графічного представлення (зліва-направо, зверху-вниз), різним форматам дати, способам введення/виведення (екранним формам і форматам), змін в методології (зі змінами графічних нотацій, правил, властивостей і складу визначених об'єктів) та ін.

Якість документації (повнота, зрозумілість, легкість для читання, корисність і ін.).

Доступність і якість навчальних матеріалів. Вони можуть включати комп'ютерні навчальні матеріали, навчальні посібники, курси.

Вимоги до рівня знань. Кваліфікація і досвід, необхідні для ефективного використання CASE-засобів.

Простота роботи з CASE-засобом (як для початківців, так і для досвідчених користувачів).

Уніфікованість призначеного для користувача інтерфейсу (по відношенню до інших засобів, що використовуються в даній організації).

Допустимий час реакції на дії користувача (в залежності від середовища). Простота установки і оновлення версій.

Вимоги до технічних засобів. Вимоги до оптимального розміру зовнішньої і оперативної пам'яті, типу і продуктивності процесора, що забезпечує прийнятний рівень продуктивності.

Ефективність робочого навантаження. Ефективність виконання CASE-засобом своїх функцій в залежності від інтенсивності роботи користувача (наприклад, кількість натискань клавіш або кнопки миші, необхідну для виконання певних функцій).

Продуктивність. Час, що витрачається CASE-засобом для виконання конкретних завдань (наприклад, час відповіді на запит, час аналізу 100000

рядків коду). У деяких випадках дані оцінки продуктивності можна отримати з зовнішніх джерел.

Переносимість. Сумісність з версіями ОС (можливість роботи в середовищі різних версій однієї і тієї ж ОС, простота модифікації CASE-засобу для роботи з новими версіями ОС).

Переносимість даних між різними версіями CASE-засобу. Відповідність стандартам переносимості. Такі стандарти включають документацію, комунікації і призначений для користувача інтерфейс, віконний інтерфейс, мови програмування, мови запитів і ін.

Наведені нижче критерії є загальними за своєю природою і не належать до сукупності показників якості, наведеної в стандарті ISO/IEC 9126.

Витрати на CASE-засіб. Включають вартість придбання, установки, початкового супроводу і навчання. Слід враховувати ціну для всіх необхідних змін (включаючи єдину копію, кілька копій, локальну ліцензію, ліцензію для підприємства, мережеву ліцензію).

Оцінний ефект від впровадження CASE-засобу (рівень продуктивності, якості і т.д.). Така оцінка може зажадати економічного аналізу.

Профіль дистриб'ютора. Загальні показники можливостей дистриб'ютора. Профіль дистриб'ютора може включати величину його організації, стаж в бізнесі, фінансове становище, список будь-яких додаткових продуктів, ділові зв'язки (зокрема, з іншими дистриб'юторами даного засобу), планована стратегія розвитку.

Сертифікація постачальника. Сертифікати, отримані від спеціалізованих організацій в області створення ПЗ (наприклад, SEI і ISO), які засвідчують, що кваліфікація постачальника в області створення і супроводу ПЗ задовольняє деяким мінімально необхідним або цілком певним вимогам. Сертифікація може бути неформальною, наприклад, на основі аналізу якості роботи постачальника.

Ліцензійна політика. Доступні можливості ліцензування, право копіювання (носіїв і документації), будь-які обмеження та / або штрафні санкції за вторинне використання (мається на увазі продаж користувачем CASE-засобу

продуктів, до складу яких входять деякі компоненти CASE-засобу, що використовувалися при розробці продуктів).

Профіль продукту. Загальна інформація про продукт, включаючи термін його існування, кількість проданих копій, наявність, розмір і рівень діяльності користувальницької групи, система звітів про проблеми, програма розвитку продукту, сукупність застосувань, наявність помилок і ін.

Підтримка постачальника. Доступність, реактивність і якість послуг, що надаються постачальником для користувачів CASE-засобів. Такі послуги можуть включати телефонну "гарячу лінію", місцеву технічну підтримку, підтримку в самій організації.

Доступність і якість навчання. Навчання може проводитися на території постачальника, користувача або де-небудь в іншому місці. Адаптація, необхідна для впровадження CASE-засобів в організації користувача. Прикладом може бути визначення способу використання централізованого CASE-засобу з єдиної, загальної БД в розподіленому середовищі.

1.4. Специфікація вимог до системи

Специфікація вимог до програмної системи – це специфікація окремого програмного продукту, програми або набору програм, які виконують деякі дії в деякому середовищі. Тобто – це повний опис поведінки системи що розробляється [3].

В загальному випадку специфікація включає наступне:

- глосарій проекту;
- опис варіантів використання.

Наведемо список основних термінів та понять в області розробки програмної системи «Підтримки вибору CASE-засобів» – глосарій. Глосарій – список понять в специфічній області знання з їх визначеннями [3]. Ці поняття та визначення подано у таблиці 1.1.

Таблиця 1.1

Глосарій

Термін	Опис терміну
1. Основні поняття та категорії предметної області та проекту	
CASE-засіб	набір інструментів і методів програмної інженерії для проектування програмного забезпечення, що допомагає забезпечити високу якість програм, відсутність помилок і простоту в обслуговуванні програмних продуктів
Метод проектування	процес створення проекту, прототипу, праобразу майбутнього об'єкта, стану та способів його виготовлення
Модель	відтворення чи відображення об'єкту, задуму (конструкцій), опису чи розрахунків, що відображає, імітує, відтворює принципи внутрішньої організації або функціонування, певні властивості, ознаки чи(та) характеристики об'єкта дослідження чи відтворення (оригіналу)
Методи експертних оцінок	спосіб прогнозування та оцінки майбутніх результатів дій на основі прогнозів фахівців
Програмне забезпечення	сукупність програм системи обробки інформації і програмних документів, необхідних для експлуатації цих програм [3].
Програмний продукт	програмний засіб, програмне забезпечення, які призначені для постачання користувачів (покупцеві, замовникові) [3].
Бізнес-процес	будь-яка діяльність, що має вхідний продукт, додає вартість до нього, та забезпечує вихідний продукт для внутрішнього або зовнішнього споживача [3].
База даних	логічно впорядкований та взаємопов’язаний набір даних, що використовуються спільно та призначені для задоволення інформаційних потреб користувачів.

Продовження таблиці 1.1

2. Користувачі системи	
Користувач	Особа, яка займається безпосереднім користуванням системою на користувацькому рівні.
Експерт	Особа, яка здійснює управлінські функції системою, її наповненням та забезпечує функціонування системи на експертному рівні.
3. Вхідні та вихідні документи	
База даних	логічно впорядкований та взаємопов’язаний набір даних, що використовуються спільно та призначені для задоволення інформаційних потреб користувачів.
Список експертних оцінок	типова форма відображення результатів оцінювання CASE-засобів експертом
Рейтинг CASE-засобів	документ, який забезпечує впорядкування CASE-засобів за певними класифікаційними характристиками

У таблицях 1.2 – 1.8 наведено опис варіантів використання, що реалізують основну функціональність програмної системи. Діаграма варіантів використання представлена на рисунку 1.8.

Таблиця 1.2

Варіант використання «Перегляд відомих CASE-засобів»

Контекст використання	Управління інформацією.
Дійові особи	Користувач
Передумова	Користувач аутентифікований та авторизований.
Тригер	Натиснення кнопки «Перглянути довідник CASE-засобів».
Сценарій	1. Заповнення полів. 2. Перевірка правильності введених даних. 3. Натиснення кнопки «Переглянути».
Постумова	Відображення інформації про CASE-засоби.

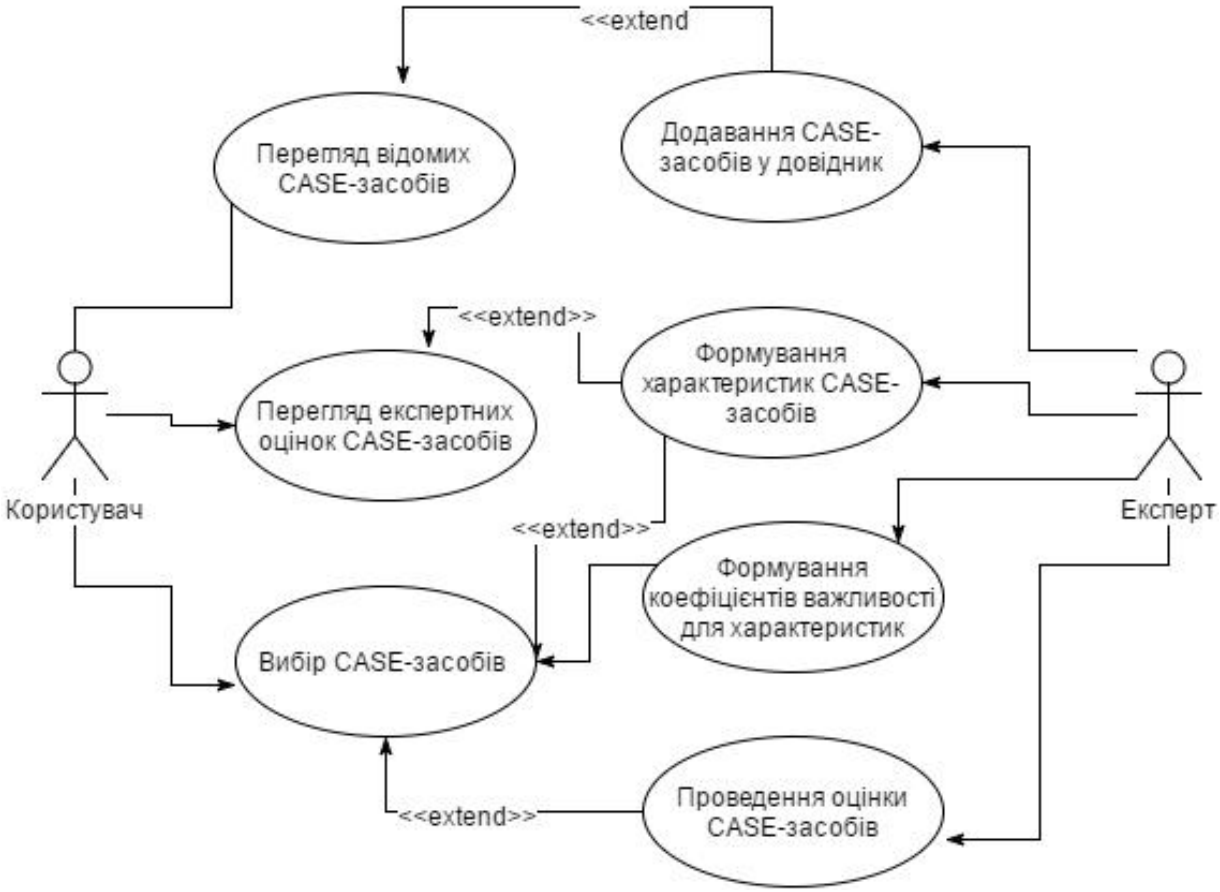

Рис. 1.8. Діаграми варіантів використання системи

Таблиця 1.3

Варіант використання «Перегляд експертних оцінок CASE-засобів»

Контекст використання	Управління інформацією
Дійові особи	Всі користувачі відповідно до прав доступу
Передумова	Користувач аутентифікований та авторизований.
Тригер	Вибір CASE-засобів.
Сценарій	-
Постумова	Відображення інформації про експертні оцінки.

Таблиця 1.4

Варіант використання «Вибір CASE-засобів»

Контекст використання	Управління інформацією
Дійові особи	Користувач
Передумова	Користувач аутентифікований та авторизований.
Тригер	Вибір зі списку.
Сценарій	1. Вибір CASE-засобів. 2. Визначення характеристик. 3. Натиснення кнопки «Зберегти інформацію про CASE-засіб».
Постумова	Відображення списку працівників із займаними посадами

Таблиця 1.5

Варіант використання «Додавання CASE-засобів у довідник»

Контекст використання	Управління довідниками
Дійові особи	Експерт
Передумова	Користувач аутентифікований та авторизований.
Тригер	Відкриття розділу «Довідники».
Сценарій	1. Вибір довідника. 2. Заповнення необхідних полів. 3. Перевірка введених даних. 4. Натиснення кнопки «Зберегти».
Постумова	Відображення даних довідника CASE-засобів.

Таблиця 1.6

Варіант використання «Формування характеристик CASE-засобів»

Контекст використання	Управління інформацією.
Дійові особи	Експерт
Передумова	Користувач аутентифікований та авторизований.
Тригер	Відкриття розділу «Характеристки CASE-засобів».
Сценарій	1. Вибір характеристик. 2. Перевірка введених даних. 3. Натиснення кнопки «Зберегти».
Постумова	Відображення характеристик CASE-засобів

Таблиця 1.7

Варіант використання «Формування коефіцієнтів важливості для характеристик»

Контекст використання	Управління інформацією.
Дійові особи	Експерт
Передумова	Користувач аутентифікований та авторизований.
Тригер	Відкриття розділу «Характеристки CASE-засобів».
Сценарій	1. Вибір характеристик. 2. Формування коефіцієнтів. 3. Перевірка введених даних. 4. Натиснення кнопки «Зберегти».
Постумова	Відображення характеристик CASE-засобів із коефіцієнтами важливості

Таблиця 1.8

Варіант використання «Проведення оцінки CASE-засобів»

Контекст використання	-
Дійові особи	Експерт
Передумова	Користувач аутентифікований та авторизований.
Тригер	Відкриття розділу «Оцінка CASE-засобів».
Сценарій	1. Вибір CASE-засобів. 2. Заповнення необхідних полів. 3. Перевірка введених даних. 4. Натиснення кнопки «Порівняти».
Постумова	Відображення списку CASE-засобів у формі рейтингового списку

В таблиці 1.9 наведено специфікацію функціональних вимог програмної системи.

Таблиця 1.9

Специфікація функціональних вимог

Іденти-фікатор вимоги	Назва вимоги	Атрибути вимоги		
		Пріоритет	Складність	Контакт
1	Перегляд відомих CASE-засобів	рекомендоване	середня	Користувач
2	Перегляд експертних оцінок CASE-засобів	обов’язкове	висока	Користувач
3	Вибір CASE-засобів	обов’язкове	висока	Користувач
4	Додавання CASE-засобів у довідник	обов’язкове	висока	Експерт
5	Додавання CASE-засобів у довідник	обов’язкове	висока	Експерт
6	Формування коефіцієнтів важливості для характеристик	рекомендоване	середня	Експерт
7	Проведення оцінки CASE-засобів	обов’язкове	висока	Експерт

Специфікацію нефункціональних вимог наведено в таблиці 1.10.

Наведемо специфікацію суттєвих для проекту нефункціональних вимог:

1. Застосовність:
   - мінімальний час для навчання звичайних і досвідчених користувачів;
   - відповідність стандартам графічного інтерфейсу.
2. Надійність:
   - постійна безвідмовна робота;
   - пропускна здатність каналу зв’язку 100 Mb/s;
   - забезпечення можливості віддаленого доступу до комп’ютера, на якому буде встановлена система;
   - доступність – 5%.
3. Робочі характеристики:
   - швидкість завантаження інтернет-ресурсу: 0,1 – 1 с;

- число транзакцій: 100 / 1 с;
- використання ресурсів: від 1 Gb, в залежності від кількості студентів.

4. Проектні обмеження:
   - Операційна система Microsoft Windows 7/8;
   - MySQL;
   - Eclipse IDE for Java EE Developers;

Таблиця 1.10

Специфікація нефункціональних вимог

Ідентифікатор вимоги	Назва вимоги	Атрибути вимог		
		Пріоритет	Складність	Контакт
Застосовність				
1.1	Час, необхідний для навчання звичайних і досвідчених користувачів	Рекомендована	Низька	Адміністратор
1.2	Основні вимоги застосовності нової системи відносно інших систем, які знають користувачі	Опційна	Низька	Адміністратор
1.3	Вимоги по відповідальності стандартам графічного інтерфейсу користувача	Рекомендована	Низька	Адміністратор
Надійність				
2.1	Доступність	Обов’язкова	Середня	Адміністратор
2.2	Середній час безвідмовної роботи	Рекомендована	Середня	Адміністратор
2.3	Точність	Обов’язкова	Середня	Адміністратор
Робочі характеристики				
3.1	Використання ресурсів	Рекомендована	Середня	Адміністратор
4.1	Вимоги до технології програмування	Рекомендована	Середня	Адміністратор

5. Вимоги до документації
   - наявність інтерактивної довідки.
6. Інтерфейси:
   -інтерфейс користувача – Java-інтерфейс.

Висновки до розділу 1

Здійснено опис предметної області, напрями діяльності. Визначено склад функцій, що входять до бізнес-процесу на основі яких розроблено схему управління бізнес-процесом. Проведено аналіз відомих CASE-засобів. Здійснено аналіз вимог до програмної системи.

РОЗДІЛ 2
ПРОЕКТУВАННЯ СИСТЕМИ ПІДТРИМКИ ВИБОРУ CASE-ЗАСОБІВ

2.1. Розроблення архітектури програмної системи

В процесі проектування системи підтримки вибору CASE-засобів для проектування системи управління підприємством ми стикнулися з проблемою вибору архітектури системи. Для вирішення даної бізнес проблеми, була вибрана клієнт-серверна архітектура. Проект, який побудований на клієнт-серверній архітектурі, повинен складатися з трьох частин: зв'язок з базою даних, відображення даних клієнту та бізнес логіка проекту яка обробляє запити користувача і відображає саме ту інформацію, яку захотів користувач. Обробка та збереження даних відбувається на боці сервера, відображення даних і надсилання запитів на їхню модифікацію виконується на боці клієнта.

Статичний аспект такої архітектури зображений, за допомогою наступної діаграми, на рисунку 2.1.

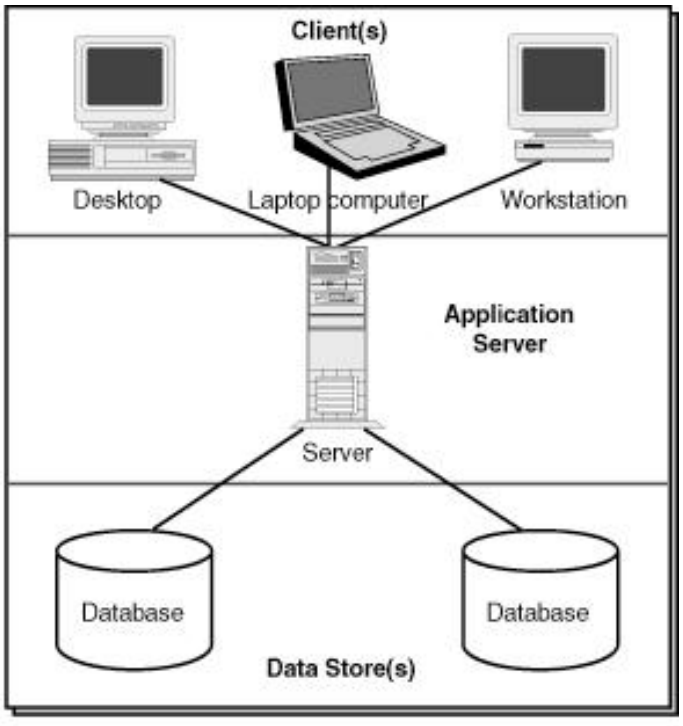

Рис. 2.1.Статичний аспект клієнт-серверної архітектури

На цій діаграмі клієнтською аплікацією виступає клієнт, розроблений на основі, тобто розширення технології Java. Він, у свою чергу, віддає на сервер запити, відповідно до того з якими формами працює простий користувач та експерт. Реалізація взаємодії із базою даних здійснюється за допомогою JDBC connection.

Уся бізнес-логіка проекту виконується на другому ріні. Вся обробка запитів від клієнта і надсилання йому відповіді здійснюється на цьому рівні. Також тут визначається, які запити повинні іти до бази даних. Третій рівень – це власне сама база даних, яка приймає Sql-запити, і у відповідності їм повертає дані.

На рисунку 2.2 представлено схему розміщення основних модулів системи по рівнях їх реалізації із врахування особливостей використання.

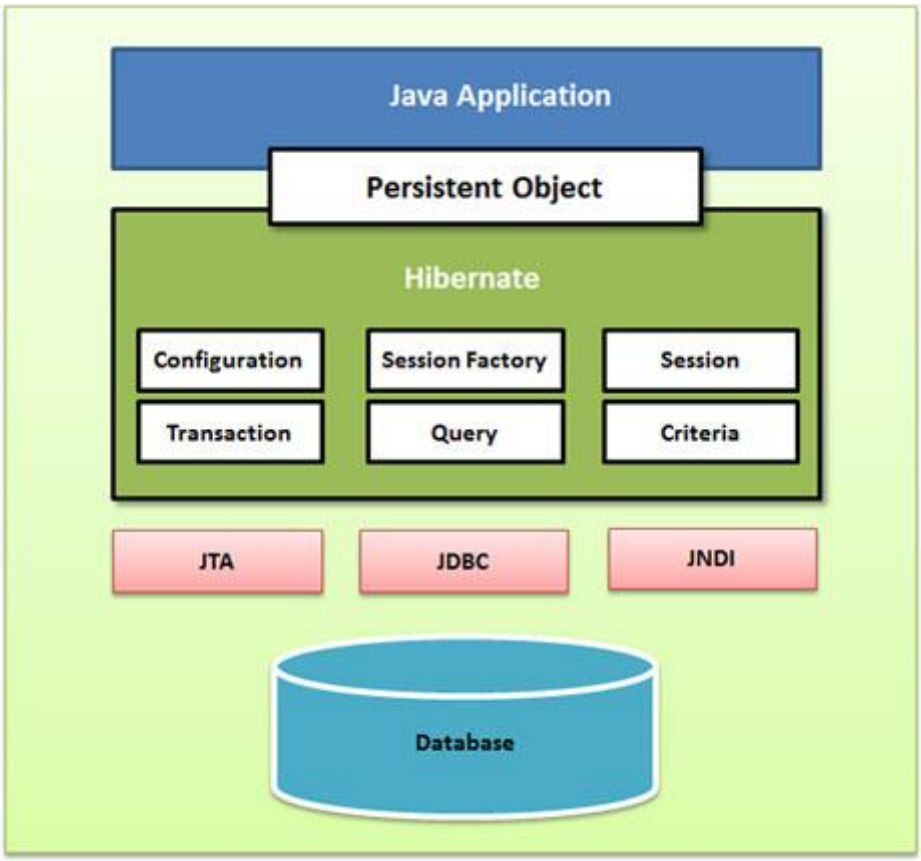

Рис. 2.2. Схема розміщення програмних модулів

На рисунку 2.3 представлено діаграму розміщення системи підтримки вибору CASE-засобів.

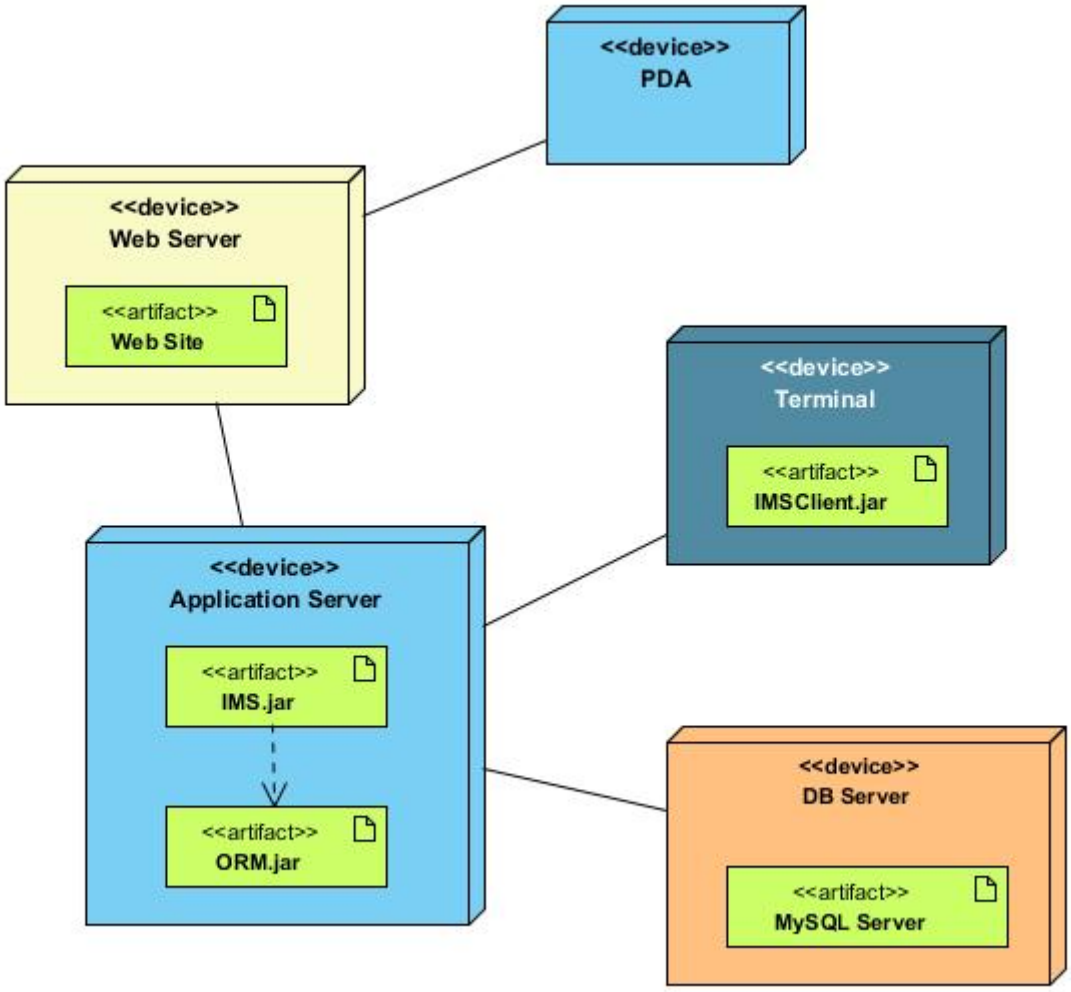

Рис. 2.3. Діаграма розміщення

На рисунку 2.4 представлено діаграму послідовності дій при вході в систему.

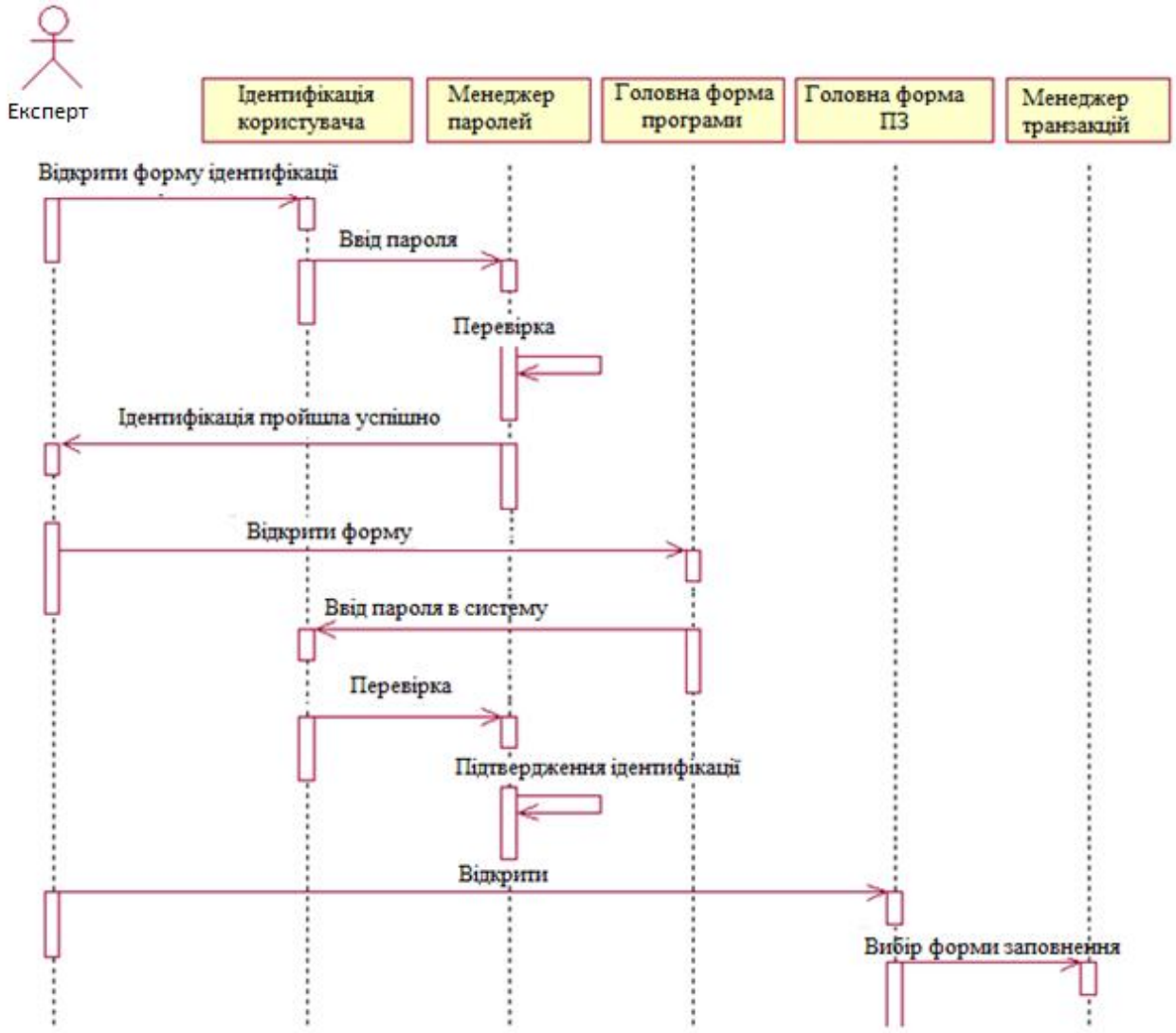

Рис. 2.4. Діаграма послідовності дій при вході в систему

Розгнянемо детальніше основні процеси, які реалізовуються в системі. Для цього використаємо діаграми станів. На рисунку 2.5 представлено діаграму станів проектованої системи.

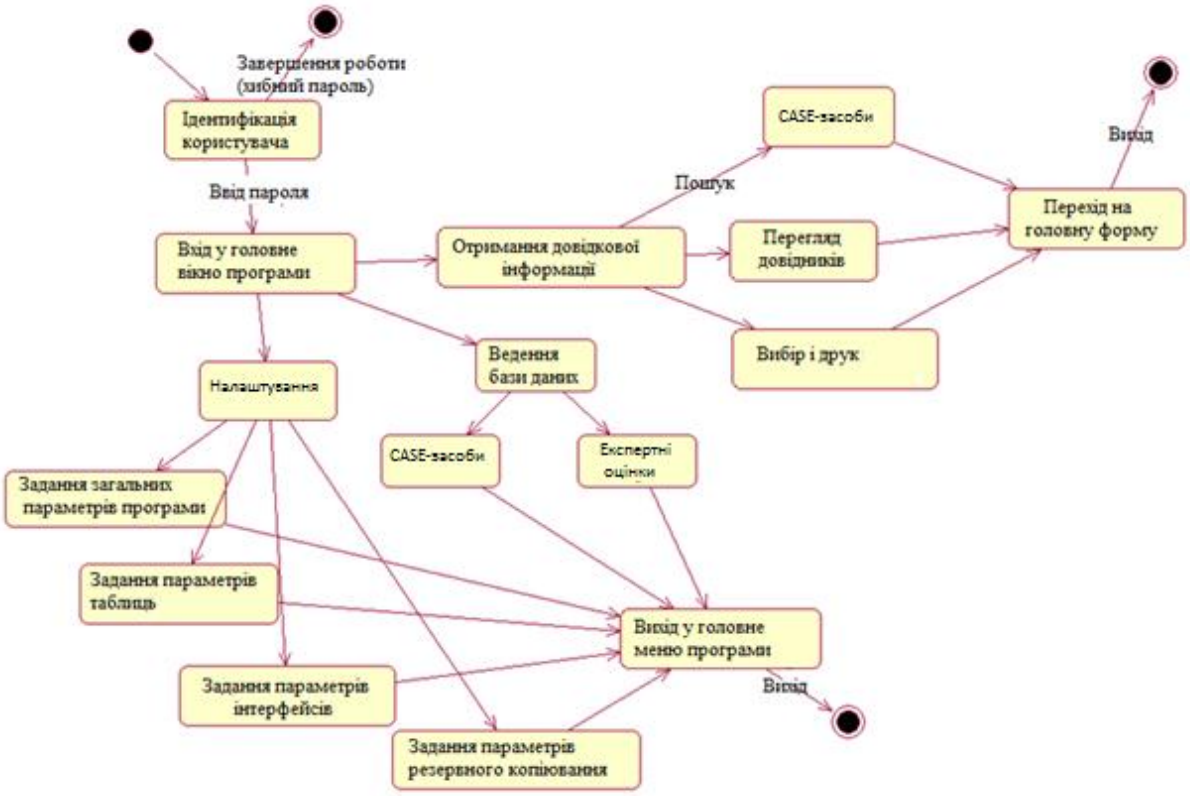

Рис. 2.5. Діаграма станів системи

На рисунку 2.6 представлено діаграму компонентів проекту, реалізованого на java з використанням взаємодії з СУБД MySQL.

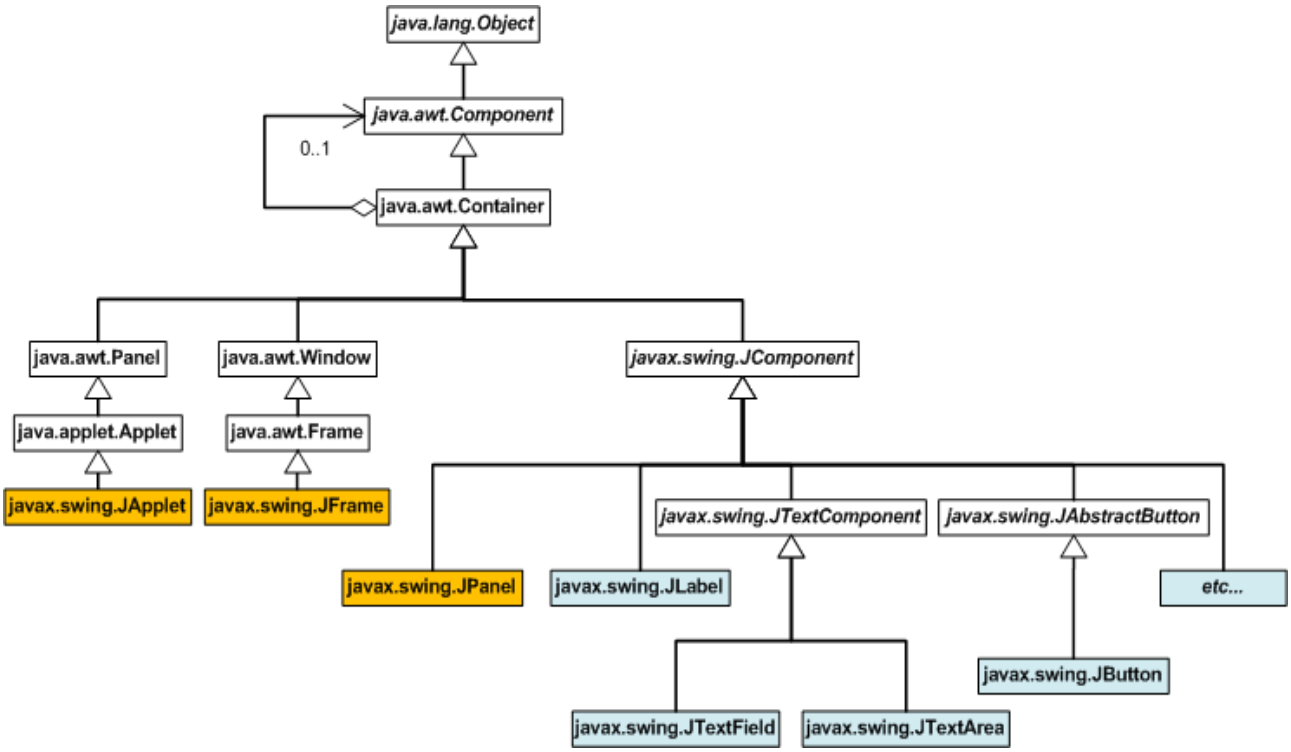

Рис. 2.6. Діаграма компонентів

На рисунку 2.7 представлено діаграму класів системи підтримки вибору CASE-засобів.

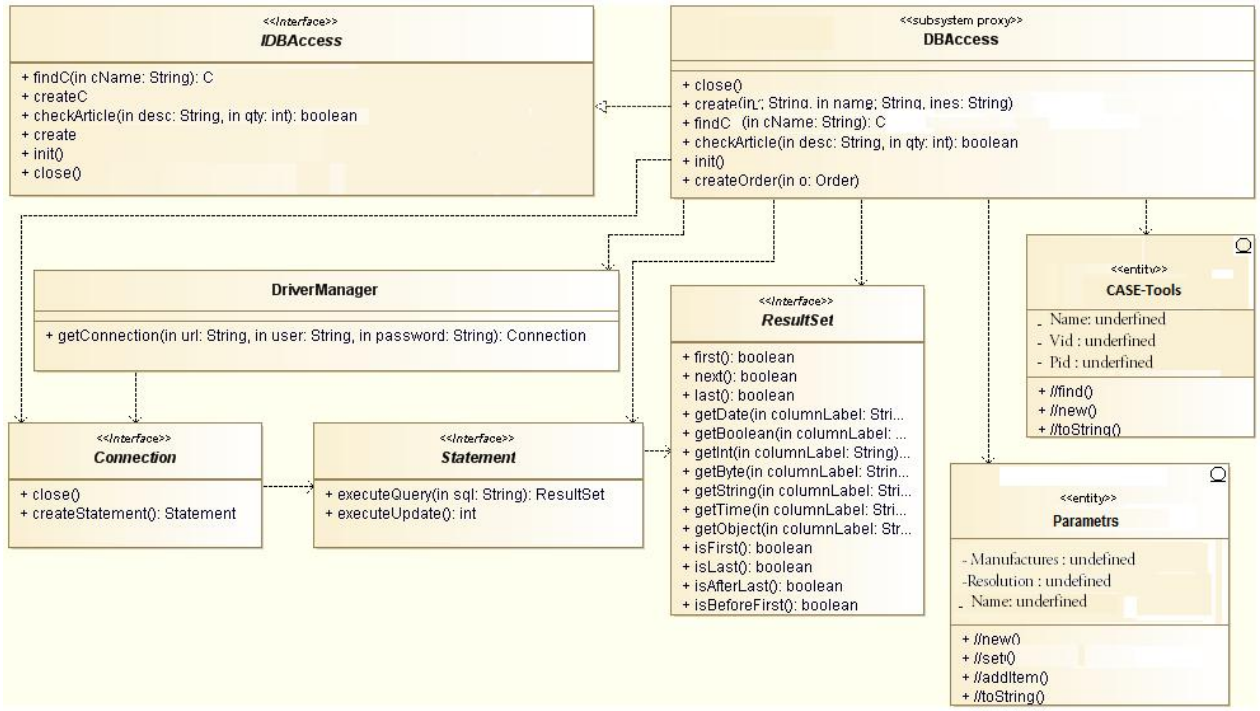

Рис. 2.7. Діаграма класів

2.2. Проектування структури бази даних

Одним з найбільш складних етапів проектування є розробка таблиць бази даних та збереження інформації, тому що результати, які повинна видавати система не завжди дають повну уяву про структуру таблиць.

При розробці, краще керуватися наступними принципами:

- Інформація в таблицях не повинна дублюватися. Коли визначена інформація зберігається тільки в одному місці, то немає необхідності у синхронізації цих даних, та забезпечить ефективність, та виключить можливість не збігу.

- Кожна таблиця повинна містити інформацію тільки на одну тему, у цьому випадку дані набагато легше обробляти, якщо вони утримуються в різних таблицях.

- Кожна таблиця бази даних повинна містити інформацію на окрему тему, а кожне поле таблиці — містити дані по темі таблиці.

При розробці треба враховувати:

- кожне поле повинне бути зв'язане з темою таблиці;
- не рекомендується включати в таблицю результати вираження;
- у таблиці повинна бути присутня уся необхідна інформація;
- інформацію варто розбивати на найменші логічні одиниці.

У зв’язку з великою кількістю інформації не має можливості відобразити та описати всі таблиці та зв’язки між ними. Тому виділимо інформацію про характеристики CASE-засобів.

Модель «Сутність-зв'язок» (Entity-Relationship, або ER- модель) являє собою концептуальну модель даних. Модель даних являє собою набір концепцій, які описують структуру бази даних і пов'язані з нею транзакції модифікації та отримання даних. Основна мета розробки моделі полягає в створенні моделі користувацького сприйняття даних й узгодження великої кількості технічних аспектів пов'язаних із проектуванням бази даних. Концептуальна модель даних не залежить від конкретної системи керування базою даних (СКБД) або апаратної платформи, що використовується для реалізації бази даних, Коли говорять про логічне проектування, уживають такі терміни, як сутність, зв'язок і атрибут.

Сутність - це множина однотипних об'єктів, називаних екземплярами, при цьому кожен екземпляр індивідуальний і відрізняється від всіх інших екземплярів.

Типи сутності можна класифікувати як сильні і як слабкі. Сильна сутність, це сутність яка може існувати незалежно від інших. Слабка сутність, це сутність існування якої залежить від сильної сутності. На ER- діаграмах сильні сутності позначаються прямокутником, а слабкі прямокутником з подвійним контуром [7].

Атрибут це характеристика сутності. Атрибут виражає одна закінчена і визначена властивість сутності. При проектуванні рекомендується створювати

атомарні атрибути. Домен атрибута це набір значень який може бути привласнений атрибуту. Зв'язок - це логічне відношення між сутностями, що виражає деяке обмеження.

Ступінь участі сутності визначає чи залежить існування деякої сутності від участі у зв'язку іншої сутності. Ступінь участі може бути обов'язковою і необов'язковою. Кожен зв'язок зображується у вигляді ромбика із зазначеним на ньому ім'ям зв'язку. Значення зв'язку характеризує його тип. При створенні зв'язків між сутностями в дочірню сутність передаються атрибути, що складають первинний ключ у батьківській сутності. Ці атрибути утворюють у дочірній сутності зовнішній ключ. Потенційний ключ це атрибут або набір атрибутів які однозначно ідентифікують екземпляр сутності. Первинний ключ вибирається з потенційних ключів виходячи з гарантій унікальності його значень [2].

Розглянемо опис структурних одиниць інформації вхідної і вихідної форми "CASE засоби".

Таблиця 2.1

Опис структурних одиниць інформації вхідної і вихідної форми "CASE-засоби"

№	Повне найменування	Тип даних	Кількість символів
1	Ідентифікатор CASE-засобу	Цілочисельний	
2	Розробник CASE-засобу	Цілочисельний	
3	Опис CASE-засобу	Символьний	255
4	Інформація про характеристики	Символьний	255
5	Пам’ять	Символьний	255
6	Процесор	Символьний	255
7	Ціна	Дробовий	255
8	Інтерфейс	Символьний	255
9	Документація	Символьний	255
10	Додаткові хіарактеристики	Символьний	255

Таблиця 2.1. представляє собою інформацію про основні характеристики CASE-засобів. Призначення атрибутів:

- Ідентифікатор CASE-засобу – унікальний код таблиці;
- Розробник CASE-засобу – ідентифікатор, який ідентифікує розробника CASE-засобу;
- Опис CASE-засобу – загальна інформація про CASE-засоби;
- Інформація про характеристики – інформація про характеристики CASE-засобів;
- Процесор – інформація про характеристики процесора для оптимальної роботи CASE-засобів;
- Пам'ять – інформація про пам’ять;
- Інтерфейс – інформація про інтерфейс засобу;
- Документація – інформація про документацію засобу;
- Ціна – інформація про ціну;
- Додаткові характеристики – інформація про додаткові характеристики.

Висновки до розділу 2

У цьому розділі розроблено архітектуру програмного додатку, що дозволить краще зрозуміти функції основних його частин. Створено та описано структурну схему, основними компонентами якої є: рівень клієнта, рівень бізнес-логіки та рівень даних. Описано функціональну структуру системи та її основних елементів – модулів обробки даних. Визначено основні елементи бази даних та встановлено зв’язки між ними. Спроектовано структуру бази даних.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ПІДТРИМКИ ВИБОРУ CASE-ЗАСОБІВ

3.1. Програмна реалізація системи

Для реалізації системи підтримки вибору CASE-засобів обрано мову прорамування JAVA.

Створення мови Java - це дійсно один із самих значних кроків уперед в області розробки середовищ програмування за останні 20 років. Мова програмування Java є засобом для написання вихідного коду. Аплети й додатки, написані на цій мові, при компіляції приводяться до виду, у якому вони будуть запускатися на Java-Платформі.

Досвід багатьох професійних програмістів показує, що мова програмування Java - проста, гнучка і потужна. Вона об’єктно-орієнтована, забороняє множинне спадкування, має строгу систему типів, реалізує багатопотоковість, динамічно зв'язана і містить автоматичний «збирач сміття».

Її синтаксис заснований на C і C++, тому програмісти на C можуть без труднощів освоїти його. Зменшена «надмірність» мови, що означає те, що розробники зможуть значно швидше зрозуміти код, написаний іншим програмістом. Приміром, у мові програмування Java неможливе визначене користувачем перевантаження оператора, що можливо в C.

Мова Java, дає програмістам можливість користуватися трьома різними способами програмування, використовуючи при цьому тільки одну мову. Подібно символічній мові програмування Smalltalk, мова Java об'єктно-орієнтована, має здатність динамічного зв'язування й реалізує ієрархічну структуру класів з одинарним спадкуванням.

Для числових операцій, мова Java має незалежні від платформи типи даних, перевірку границь масиву, і чітко - певну IIEPA (Інститут інженерів по електротехніці й радіоелектроніці) арифметику. Ці здатності становлять гарну

основу для запису стійких числових алгоритмів, що дають повторювані результати. Відносно системного програмування, виражень, інструкцій, і операторів мова Java у більшості випадків нічим не відрізняється від мови С.

Мова Java сприяє «перехопленню» дефектів програми на стадії розробки, поки програмний продукт ще не випущений на ринок. Це здійснюється шляхом строгої типізації даних, автоматичного збору сміття й перевірки границь масиву, неможливістю автоматичного приведення типів і відсутністю посилальних типів (покажчиків). Всі перераховані факти виявляють собою обережності, які допомагають програмістові зменшити час на розробку, що дуже важливо в епоху Інтернету, коли часу на реалізацію й випуск працездатного програмного забезпечення дуже мало.

Мова Java має вбудовану підтримку багатопотоковості, зі строгим визначенням способів синхронізації для критичних ділянок коду з різних потоків, що дозволяє запобігти появі ряду проблем: при розрахунку часу, конкуренції між потоками й ін. В умовах росту потреб у багатопотоковості при обробці даних і в зменшенні витрат ресурсів процесора, мова Java була створена з урахуванням цих вимог сучасності, де ми маємо справу з новим поколінням «конкуруючих» між собою при паралельній роботі додатків і сервісів.

Механізми обробки виняткових ситуацій і потоків інтегровані з мовою і її системою типів (тобто передбачені на рівні мови). Крім того, мова реалізує динамічне зв'язування підкласів з методами, що скасовують або додають функціональні можливості, під час виконання. В інших середовищах подібні можливості були «заховані» і ускладнювали системні сервіси. Замість таких властивостей у мові, ми одержуємо простоту й безліч переваг, у числі яких можливість зробити мову платформонезалежною.

На теперішній час комп'ютерному світу відомо багато різних платформ, серед яких такі як: Microsoft Windows, Macintosh, OS/2, UNIX ® і система Netware ®. Для встановлення на кожну з них програмного забезпечення, воно повинне бути відкомпільоване окремо. Двійковий файл додатка, що

виконується на одній платформі, не може бути запущений на іншій, тому як двійковий файл - специфічний, залежно від платформи.

Платформа Java - нова програмна платформа для транспортування й виконання високо інтерактивних, динамічних і безпечних аплетів і додатків на системах мережевих комп'ютерів. Основною якістю Java-Платформи, що виділяє її серед інших, є те, що вона розташовується на самому верхньому рівні в інших платформах, що дозволяє їй робити компіляцію в байт-коди, не прив'язані до кожної з фізичних машин і представляють собою машинні інструкції для віртуальної машини (virtual machine). Програма, написана мовою Java, компілюється у файл байт-коду, що може працювати скрізь, де присутня Java-Платформа, на кожній з основних операційних систем. Інакше кажучи, той самий файл буде виконуватися на будь-якій операційній системі, на якій присутня Java-Платформа. Подібна мобільність стає можливою завдяки тому, що в основі Java-Платформи лежить віртуальна Java-Машина.

У той час як кожна платформа має у своєму розпорядженні свою власну реалізацію віртуальної Java-Машини, всі віртуальні машини задовольняють вимогою єдиної специфікації. Завдяки цьому, платформа Java може реалізовувати єдиний стандарт - універсальний програмний інтерфейс для аплетів і додатків на будь-яких апаратних засобах. Тому Платформа Java є ідеальною для Інтернету, де та сама програма повинна бути здатна до виконання на будь-якому комп'ютері у світі.

Розробники використовують мову Java при написанні вихідного коду для Java-додатків. Вони компілюють свій код один раз і позбуваються тим самим від необхідності компілювати його для кожної системи окремо. Вихідний текст мови Java компілюється в проміжну, переносну форму байт-коду, що запуститься скрізь, де є присутнім Java-Платформа.

Розробники можуть писати объектно-орієнтовані, багатопоточні, динамічно зв'язані додатки, використовуючи мову Java. Платформа має вбудовані системи захисту, обробки виняткових ситуацій, і автоматичного «збору сміття». Крім того, існує можливість використовувати JIT (just-in-time)

компілятори (компілятори «на льоту») і прискорити виконання програм за допомогою перетворення байт- кодів Java у машинну мову. Також, розробники можуть записувати й викликати так звані нативні методи - методи C, C++ або інших мов, відкомпільовані для певної операційної системи - для підвищення швидкості виконання або для застосування спеціальних функціональних можливостей.

Програми, написані мовою Java і потім відкомпільовані, будуть запускатися на Java-Платформі. Платформа Java має дві основних частини:

- Java Virtual Machine (віртуальна Java-Машина);
- Java API (прикладний програмний інтерфейс Java).

У сукупності, ці частини забезпечують оперативні засоби керування роботою програми для кінцевого користувача при установці інтернет-додатків.

The Java Base Platform - «мінімальна» Java платформа, створена для запуску Java-Аплетів і додатків, що розробники можуть без проблем встановити й використовувати. Дана платформа призначається для мережевих, настільних комп'ютерів і робочих станцій. Платформа містить у собі ту ж віртуальну машину, що описувалася вище й при цьому має мінімальний комплект API, необхідним для запуску основних аплетів і додатків. Згаданий мінімальний комплект відомий, як Java Applet API або Java Base API. Розробники, які пишуть для цього комплекту можуть бути впевнені в тім, що програма запуститься скрізь без необхідності в підключенні додаткових бібліотек класів.

Деякі ліцензіати платформи Java уклали контракти про включення приватних реалізацій Java Base API в Java- платформі. В міру розробки бібліотек класів, Java Base Platform розростаються й нові класи регулярно мігрують у встановлену на кожну ліцензійну операційну систему Java Base Platform.

Інший набір API, що називається Standard Extension API, визначений JavaSoft у партнерстві з провідними промисловими компаніями створений для розширення основних функціональних можливостей. Найближчим часом

планується мігрувати деяку підмножину Standard Extension API в Java Base Platform.

Embedded Java Platform була розроблена для споживача, що використовує прилади з малими ресурсами й з більш спеціалізованою функціональністю, ніж мережевий комп'ютер. Наприклад, принтери, ксерокси, мобільні телефони й ін. Подібна апаратура може мати деякі специфічні властивості, а саме невеликий обсяг пам'яті, відсутність дисплея або неможливість зв'язку по мережі.

API, розроблений для такої платформи, називається Java Embedded API. Java Embedded API - найменший із прикладних програмних інтерфейсів, які можуть бути впроваджені в описані вище прилади й при цьому ефективно працювати. Оскільки дана платформа усе ще допрацьовується, Java Embedded API дотепер не може розглядатися як стандарт. Тому існує деяка невизначеність, пов'язана із складом API. Приблизно, він буде містити в собі пакети java.lang і java.util. Java-Додатки, написані для одного окремого пристрою, зберігають працездатність на широкому діапазоні подібних по своїй специфіці пристроїв.

На сьогоднішній день, Java-Платформа забезпечує підтримку оперативного, інтерактивного наповнення World Wide Web, за допомогою « just-in-time» доступу до програмних ресурсів. Додатки легко доступні на всіх операційних системах одночасно, що звільняє користувача від необхідності вибору операційної системи по факту доступності додатків. Більш маленькі, менш дорогі спеціалізовані системи будуть, в остаточному підсумку, доступні для спеціалізованих додатків.

Переваги для розробника: Мова програмування Java - невелика, проста для вивчення система, оснащена всебічно, що розширюється набором, API. Розробники можуть «написавши один раз, запускати всюди», що дає мові Java величезну перевагу перед іншими мовами на ринку. Крім того, програми на Java на всіх операційних системах при компіляції перетворяться до того самого двійкового формату. Виграш у порівнянні із ситуацією, коли для установки програми на декількох платформах потрібно писати й компілювати код окремо

для кожної платформи, очевидний. Програміст може працювати над додатком під однією платформою, скорочуючи при цьому час і вартість розробки, і бути впевненим, що його код буде працювати скрізь. Можливість «написавши один раз, запускати всюди», для багатьох програмістів є достатньою причиною для переходу до мови Java від таких мов як C або C++, на яких працездатність додатків залежить від платформи а, також, додатки яких «немережеві».

На додаток до цього, можливе створення додатків, що багаторазово використовують загальнодоступні об'єкти, що ще більше зменшує вартість розробок і дозволяє розробникам концентрувати свої зусилля тільки на створенні чогось нового.

Переваги при адмініструванні й підтримці: Java-Платформа має переваги, пов'язані з адмініструванням корпоративних комп'ютерних систем. Контроль версії й оновлення спрощуються, тому що Java-Додатки можуть зберігатися в центральному архіві й тут обслуговуватися для кожного окремо взятого користувача. У багатотипному, багатоплатформенному оточенні кількість Java-Платформ, яким необхідна технічна підтримка зменшується до однієї. У сукупності з тенденцією до зниження вартості мережеих комп'ютерів це приводить до зменшення як основних витрат, так і витрат на обслуговування. З подібними мережевими комп'ютерами керування даними може здійснюватися централізовано, у той час як обробка даних здійснюється локально.

Компанії з великими внутрішніми комп'ютерними мережами, які можуть не приділяти достатньо уваги для встановлення на всіх машинах новітнього програмного забезпечення або останніх операційних систем, можуть запускати Java-Додатки на всіх комп'ютерах у мережі. Крім того, впроваджуючи утримування корпоративних даних у форматах «розпізнаваних» Java додатками, корпорації дає своїм працівникам можливість доступу до будь-яких даних незалежно від платформи.

Якщо на Java-Платформі запустити клієнт, компанії можуть скористатися интерактивністю інтернету й перекласти необхідність відсилання завдань працівникам на клієнтський додаток. Також компанії можуть зменшити час, що

витрачається на облік замовлень за допомогою впровадження спеціальних клієнтів, що містять бланки замовлень на Web сторінках. При цьому клієнти можуть бути запущені на будь-яких платформах.

Java-Платформа дозволяє розробникам створювати два види програм:

Аплети - це програми, для запуску яких необхідно використовувати броузер. Тег <applet> вставляється в HTML-Код Web- сторінки й містить ім'я програми, що повинна бути запущена. Коли користувач заходить на сторінку по корпоративній мережі або по Інтернету, аплет автоматично завантажується із сервера й запускається на клієнтській машині. Через те, що аплети є завантажувальними програмами, вони повинні бути маленькими по розмірах або розбитими на модулі так, щоб на завантаження йшло якнайменше часу.

Додатки - це програми, що запускаються без броузера, в яких відсутній вбудований механізм автоматичного завантаження. У момент виклику додатка, він запускається. Таким чином, додаток являє собою щось схоже на програми, які пишуться на інших мовах. Вони можуть виконувати стандартні прикладні завдання, починаючи з редагування текстів і закінчуючи обробкою графіки. Як і для аплета, для запуску додатка потрібна Java-Платформа, однак платформа може бути доступна, як окрема програма, може бути впроваджена прямо в операційну систему або може бути убудована в сам додаток.

Хоча аплети й додатки запускаються різними способами, проте, вони мають однаковий доступ до більшості можливостей мови. Приміром, як аплет, так і додаток можуть працювати з вилученими серверами баз даних: «витягати» необхідні дані, обробляти їх локально й зберігати результат у БД.

Однак, для запуску аплета, на відміну від додатка, необхідна мережа. Крім того, додатки мають більше волі при роботі із системою, а саме - повний доступ до її сервісів. Простий приклад: додатки можуть нормально записувати й читати файли, а аплети - ні. З моменту завантаження аплета з «ненадійної» сторінки, він обмежується в правах доступу до читання й запису файлів на будь-якій системі, за винятком тієї, з якої він прибув. Для зняття або ослаблення подібних обмежень для аплетів використовуються цифрові підписи,

завдяки яким кінцевий користувач може бути впевнений у тім, що завантаження завжди відбувається з ресурсу, якому він довіряє. Зараз, там, де існує необхідність локального зберігання даних у файлах, потрібні додатки.

Java-Платформа складається із двох основних частин, Java Virtual Machine (віртуальна машина Java) і Java API (рисунок 3.1).

Java Virtual Machine - Java Virtual Machine - це «запрограмований» комп'ютер, що може бути реалізований у програмному забезпеченні або в апаратних засобах. Це абстрактний пристрій, спроектований так, щоб бути реалізованим на як можна більшому числі сучасних процесорів. Інтерфейс переносів і адаптери дають JVM можливість бути перенесеною на нові операційні системи без необхідності в повнім переписуванні.

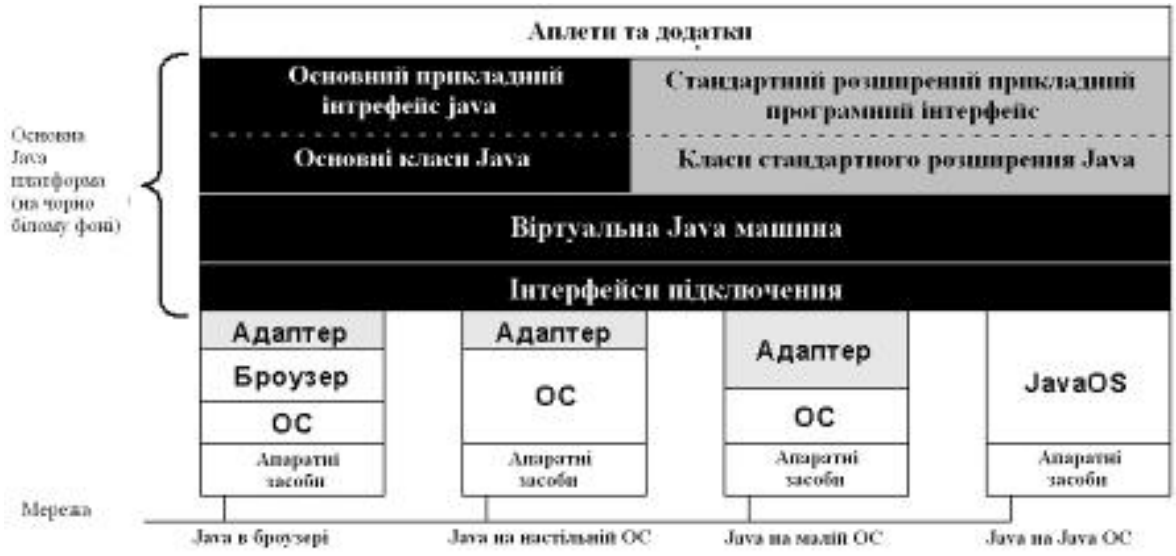

Рис.3.1. Java Base Platform для операційної системи

Java API - Java API визначає стандартний інтерфейс для аплетів і додатків, не звертаючи уваги на встановлену на комп'ютер операційну систему. Java API - основа, каркас при розробці додатка. Даний API визначає набір стандартних інтерфейсів для використання в ключових областях, кількість котрих збільшується, у яких програмісти звичайно вибудовують свої Java додатки.

Java Base API (Основний прикладний програмний інтерфейс Java)

забезпечує можливість роботи з різними допоміжними класами, з уведенням/виведенням, з мережею, з GUI (графічний інтерфейс користувача), і аплетами. Компанії-Виробники операційних систем, які мають ліцензію Java, підписали контракти, що зобов'язують їх включати Java Base API у будь-яку Java-Платформу, що вони встановлюють.

Java Standard Extension API (Стандартний розширений прикладний програмний інтерфейс Java). Є розширенням описаного вище Java Base API. Передбачається, що деякі розширення будуть згодом мігровані в Java Base API. Інші нестандартні API будуть підтримуватися в додатках, аплетах і основних операційних системах. При публікації специфікації будь-якого розширювального API до його остаточного виходу, в обов'язковому порядку публікуються промислові огляди з можливістю зворотного зв'язка з розроблювачами.

На представленому малюнку, Java Base Platform представлена частинами, зафарбованими чорним, включаючи блоки, підписані «Адаптер». Java API включає як Java Base API так і Java Standard Extension API. Класи є реалізаціями API. Java Virtual Machine лежить в основі платформи. Інтерфейс переносу розташовується між Java Virtual Machine і операційною системою (ОС) або броузером. Інтерфейс переносу складається із платформонезалежної частини (зафарбована чорним) і залежної від платформи частини з написом «Адаптер». ОС і JavaOS забезпечують роботу з віконним інтерфейсом, зберіганням даних і взаємодією по мережі. Як показано, різні машини можуть приєднуватися по мережі.

Основна частина Java API відкрита й розширювана. Специфікації для кожного інтерфейсу розвиваються всегалузевими фахівцями у всіх областях. Підготовлювані специфікації видаються й відкриваються для рецензування різними галузями промисловості. Реалізації специфікацій API надходять від JavaSoft і інших підприємств. У сучасному середовищі постійних інновацій, структура Java API дозволяє будь-якому нововведенню легко існувати у вигляді розширення до Платформи Java.

Середовище розробки мови Java включає і середовище компіляції, і середовище виконання, як показано на рисунку 3.2. Java-Платформа представлена середовищем виконання. Розроблювач пише мовою програмування Java вихідний код (файли з розширенням java) і компілює їх у байт-коди (файли з розширенням class). Байт-Коди є інструкціями для Java Virtual Machine. При створенні аплета розроблювач, крім того, розміщає файл із байт-кодом на HTTP сервері й додає до HTML-Коду Web-сторінки тег <applet code=filename>, що вказує на точку входу для файлу з байт-кодом.

Байт-Коди можуть, на вибір, інтерпретуватися інтерпретатором або звертатися в машинний код just-in-time (JIT) генератором коду, частіше іменованим JIT Compiler (Компілятор «на льоту»). Інтерпретатор і JIT компілятор працюють в оточенні системи виконання (потоки, пам'ять, інші системні ресурси). Будь-які класи динамічно довантажуються в момент, коли це потрібно аплету.

Середовище виконання: Середовище компіляції(Java-Платформа}

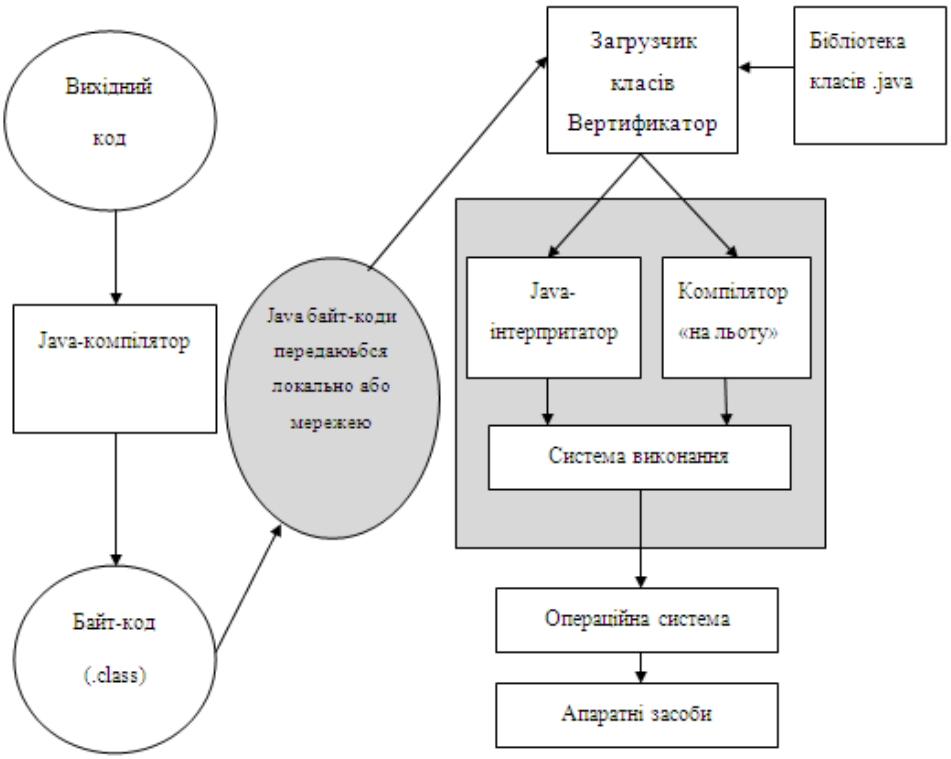

Рис. 3.2 Процес компіляції вихідного коду в байт-код

Лістинг основних модулів системи представлено в додатку А.

3.2. Програмна реалізація бази даних

Фізичне проектування це створення опису реалізації бази даних на запам'ятовувальних пристроях з визначенням структур зберігання й методів доступу, використовуваних для організації ефективної обробки даних. Фізичне проектування є фазою процесу створення проекту бази даних при виконуванні якої проектувальник приймає рішення щодо способів реалізації бази даних. Приступаючи до фізичного проектування бази даних необхідно, насамперед, вибрати конкретну СУБД. Взагалі, основною метою фізичного проектування бази даних є опис способу фізичної реалізації логічного проекту бази даних. У випадку реляційної моделі під цим мається на увазі наступне:

- створення набору реляційних таблиць та обмежень для них на основі інформації представленої в глобальній логічній моделі даних;
- визначення конкретних структур зберігання даних і методів доступу до них, що забезпечують оптимальну продуктивність системи з базою даних;
- розробка засобів захисту створюваної системи від несанкціонованого доступу.

Для програмної реалізації бази даних необхідна СУБД з відкритим вихідним кодом, яка підтримує обсяги даних до 1 Тб та достатню продуктивність. Із рішень з відкритим вихідним кодом, що відповіжають вказаним вимогам, найбільш розповсюдженими є MySQL.

Система керування реляційними базами даних MySQL була розроблена компанією «TcX» для підвищення швидкодії обробки великих баз даних. Ця система керування базами даних (СКБД) з відкритим кодом була створена як альтернатива комерційним системам. MySQL з самого початку була дуже схожою на mSQL. У 1999 році MySQL обігнала mSQL за популярністю. Остання версія mSQL- 3.8 - була випущена 9 червня 2006.

MySQL виникла як спроба застосувати mSQL до власних розробок компанії: таблицям, для яких використовувалися ISAM — підпрограми низького рівня. У результаті був вироблений новий SQL-інтерфейс, але API-інтерфейс залишився в спадок від mSQL.

З часом MySQL все розширювалася і зараз вона — одна з найпоширеніших систем керування базами даних. Вона використовується, в першу чергу, для створення динамічних веб-сторінок, оскільки має впевнену підтримку з боку різноманітних мов програмування.

Станом на квітень 2015 року, MySQL пропонувала версію MySQL 5.6 в двох різних варіантах: з відкритим вихідним кодом MySQL Community Server і комерційний Server Enterprise. Вони мають спільний програмний код і включають в себе серед іншого наступні можливості:

- Крос-платформна підтримка.
- Збережувані процедури та функції.
- Тригери.
- Курсори.
- Оновлювані подання (представлення).
- Інформаційна схема (так званий системний словник, що містить метадані).
- Підтримка SSL.
- Кешування запитів.
- Вкладені запити SELECT.
- Підтримка реплікації.
- Повноцінна підтримка Юнікоду (UTF-8 і UCS2).
- Сегментування таблиць.

MySQL являється компактним багатопоточним сервером баз даних та характеризується великою швидкістю, стійкістю і простотою використання.

Ця система вважається гарним рішенням для малих і середніх додатків. Вихідний код сервера компілюється на безлічі платформ. Найбільш повно можливості сервера виявляються в UNIX-системах, де є підтримка

багатопоточності, що підвищує продуктивність системи в цілому. Для некомерційного використання MySQL є безкоштовним.

Можливості сервера MySQL:

- простота у встановленні та використанні;
- підтримується необмежена кількість користувачів, що одночасно працюють із БД;
- кількість рядків у таблицях може досягати 50 млн.;
- висока швидкість виконання команд;
- наявність простої і ефективної системи безпеки.

Недоліками сервера MySQL залишається не реалізована підтримка транзакцій, замість якої пропонується використовувати LOCK/UNLOCK TABLE, а також відсутність підтримки для зовнішніх (foreign) ключів, тригерів, збережених процедур та представлень (VIEW). Та для розробки малих та середніх інформаційних систем, призначених для використання в межах робочих груп ці недоліки є фактично невідчутними.

У текстовому режимі робота з базою даних виглядає просто як введення команд у командний рядок, а результати вибірок повертаються у вигляді своєрідних таблиць, поля в яких налазять один на одного, якщо дані не вміщаються на екрані.

Розглянемо детальніше процедуру реалізації відношень спроектованої бази дани в СУБД MySQL. На рисунку 3.3 представлено реалізацію відношення CASE_tools, яке містить інформацію про CASE-засоби.

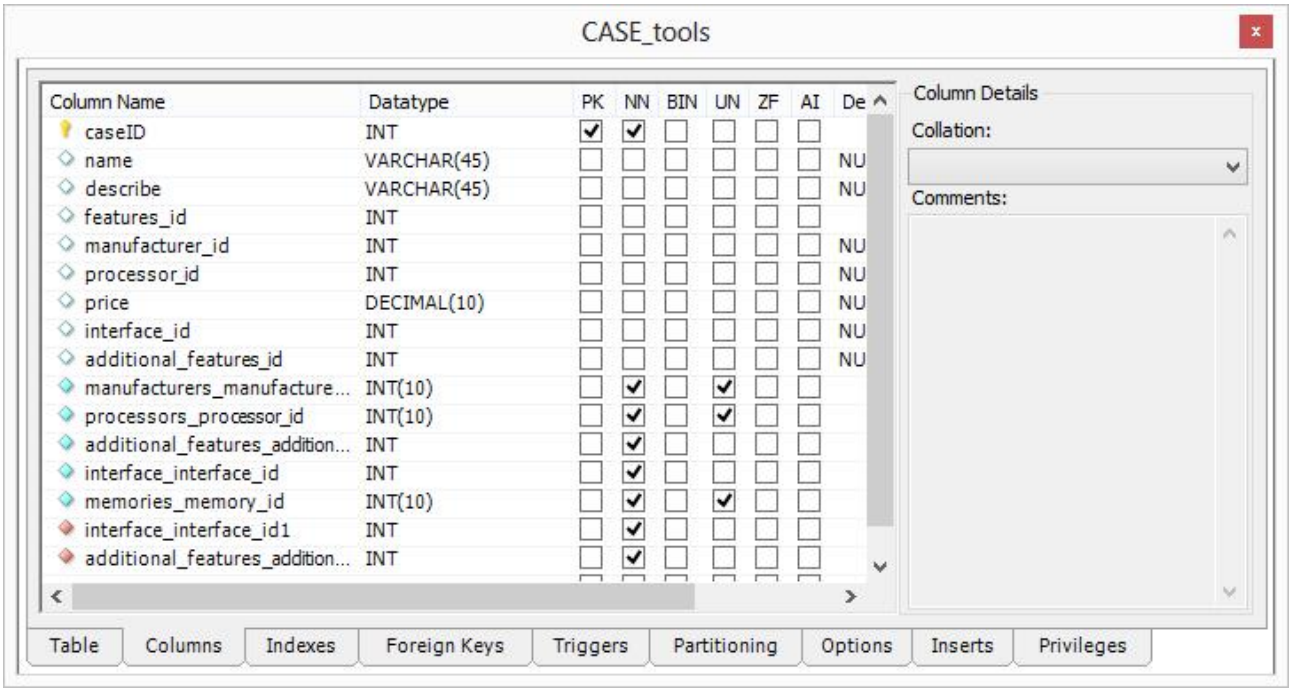

Рис. 3.3. Реалізація відношення CASE_tools

Нижче наведено DDL інтерпретацію для даного відношення CASE_tools, а в додатку Б представлено DDL код всієї бази даних.

```
CREATE  TABLE IF NOT EXISTS `mydb`.`CASE_tools` (
  `caseID` INT NOT NULL ,
  `name` VARCHAR(45) NULL DEFAULT NULL ,
  `describe` VARCHAR(45) NULL DEFAULT NULL ,
  `features_id` INT NULL ,
  `manufacturer_id` INT NULL DEFAULT NULL ,
  `processor_id` INT NULL DEFAULT NULL ,
  `price` DECIMAL(10) NULL DEFAULT NULL ,
  `interface_id` INT NULL DEFAULT NULL ,
  `additional_features_id` INT NULL DEFAULT NULL ,
  `manufacturers_manufacturer_id` INT(10) UNSIGNED NOT NULL ,
  `processors_processor_id` INT(10) UNSIGNED NOT NULL ,
  `additional_features_additional_features_id` INT NOT NULL ,
  `interface_interface_id` INT NOT NULL ,
  `memories_memory_id` INT(10) UNSIGNED NOT NULL ,
  `interface_interface_id1` INT NOT NULL ,
  `additional_features_additional_features_id1` INT NOT NULL ,
  PRIMARY KEY (`caseID`) ,
  INDEX `fk_CASE_tools_manufacturers1` (`manufacturers_manufacturer_id` ASC) ,
  INDEX `fk_CASE_tools_processors1` (`processors_processor_id` ASC) ,
  INDEX `fk_CASE_tools_additional_features1`
(`additional_features_additional_features_id` ASC) ,
  INDEX `fk_CASE_tools_interface1` (`interface_interface_id` ASC) ,
  INDEX `fk_CASE_tools_memories1` (`memories_memory_id` ASC) ,
  INDEX `fk_CASE_tools_interface1` (`interface_interface_id1` ASC) ,
```

```
        INDEX `fk_CASE_tools_additional_features1`
(`additional_features_additional_features_id1` ASC) ,
        CONSTRAINT `fk_CASE_tools_interface1`
          FOREIGN KEY (`interface_interface_id1` )
          REFERENCES `mydb`.`interface` (`interface_id` )
          ON DELETE NO ACTION
          ON UPDATE NO ACTION,
        CONSTRAINT `fk_CASE_tools_additional_features1`
          FOREIGN KEY (`additional_features_additional_features_id1` )
          REFERENCES `mydb`.`additional_features` (`additional_features_id` )
          ON DELETE NO ACTION
          ON UPDATE NO ACTION)
      ENGINE = InnoDB
```

На рисунку 3.4 представлено реалізацію відношення manufacturers, яке містить усю інформацію про розробника вказаного засобу.

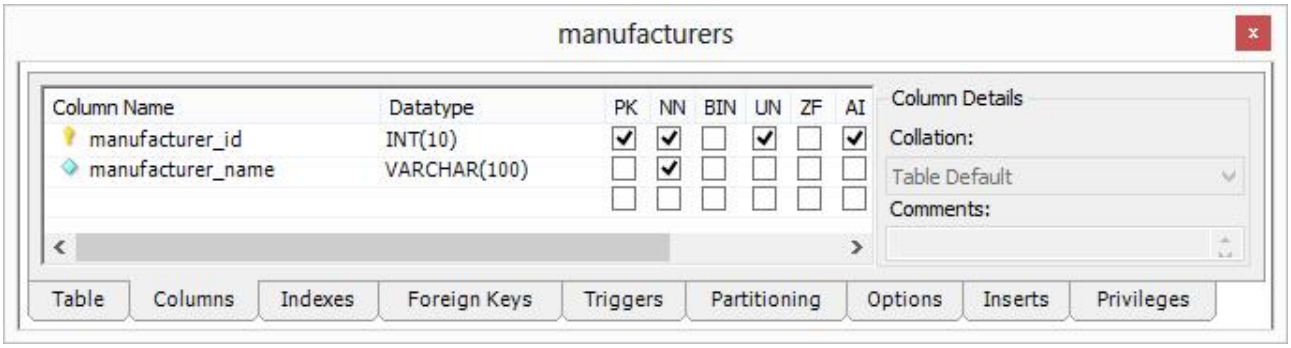

Рис. 3.4. Реалізація відношення manufacturers

На рисунку 3.5 – реалізація відношення memories, в якому зберегіються інформація про пам'ять, яка необхідна для нормального функціонування CASE-засобу.

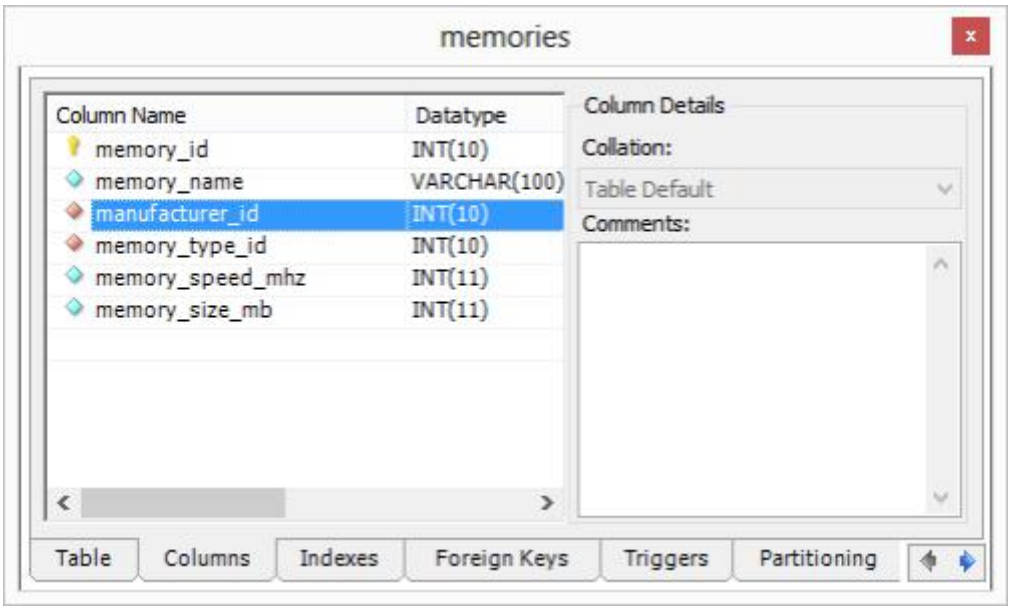

Рис. 3.5. Реалізація відношення memories

На рисунку 3.6 представлено реалізацію відношення processors, в якому зберегаються дані про процесор.

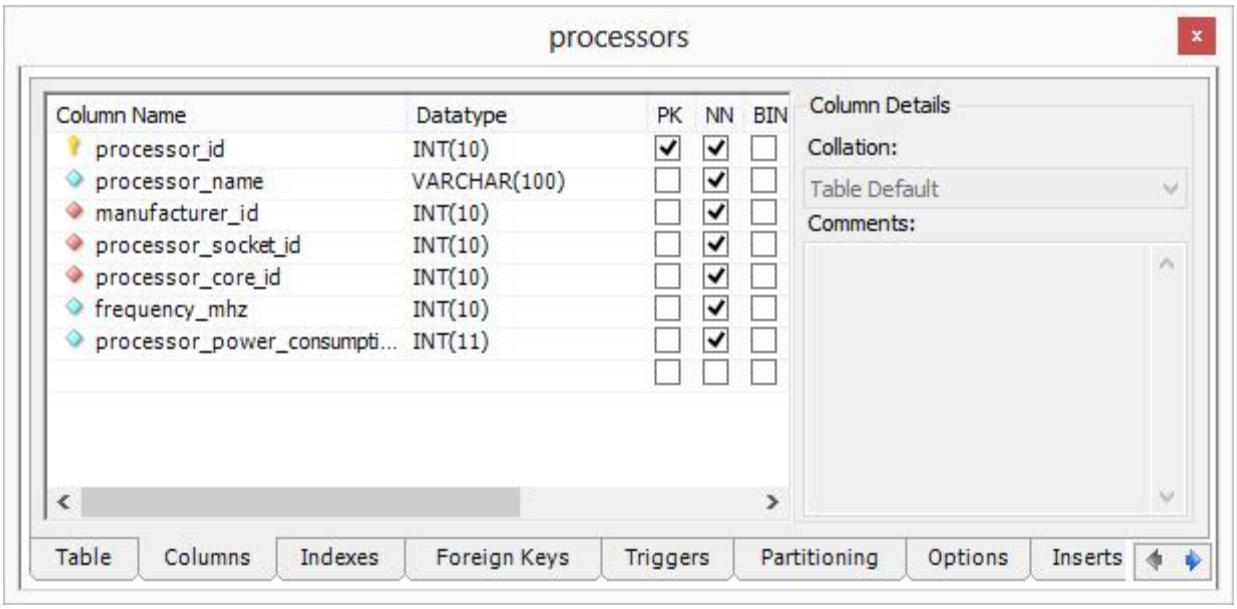

Рис. 3.6. Реалізація відношення processors

На рисунку 3.7 наведено структуру відношення features, в якому зберігаються дані про основні параметри CASE-засобів та їх оцінки.

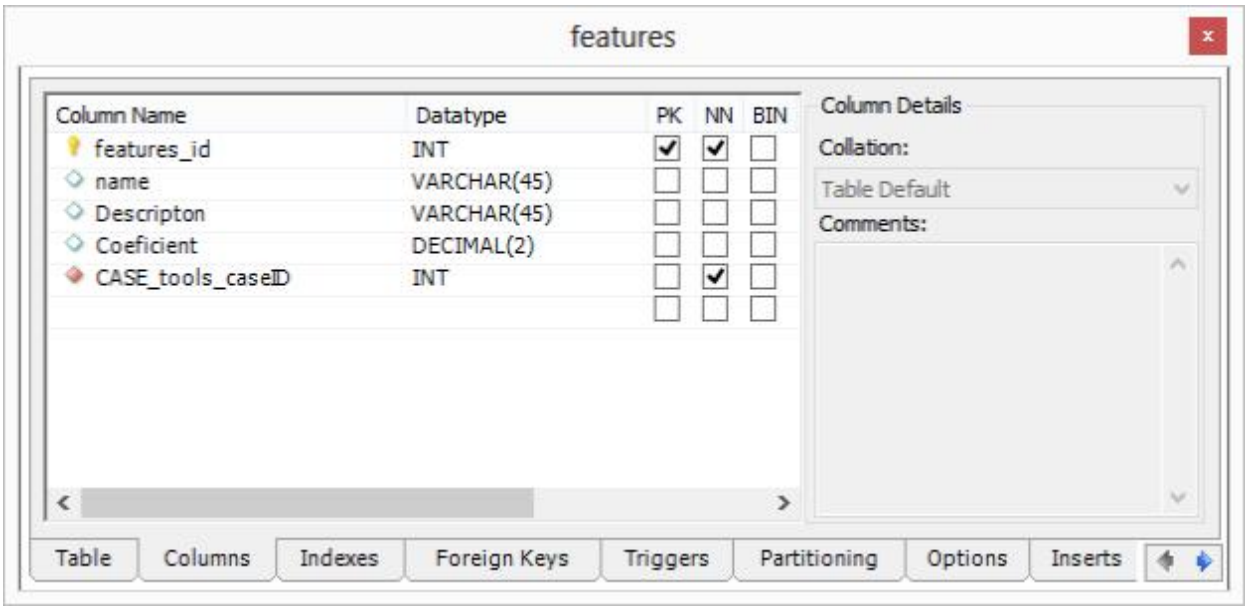

Рис. 3.7. Реалізація відношення features

На рисунку 3.8 представлено реалізацію відношення additional_features, яке містить інформацію про додаткові характеристики CASE-засобів

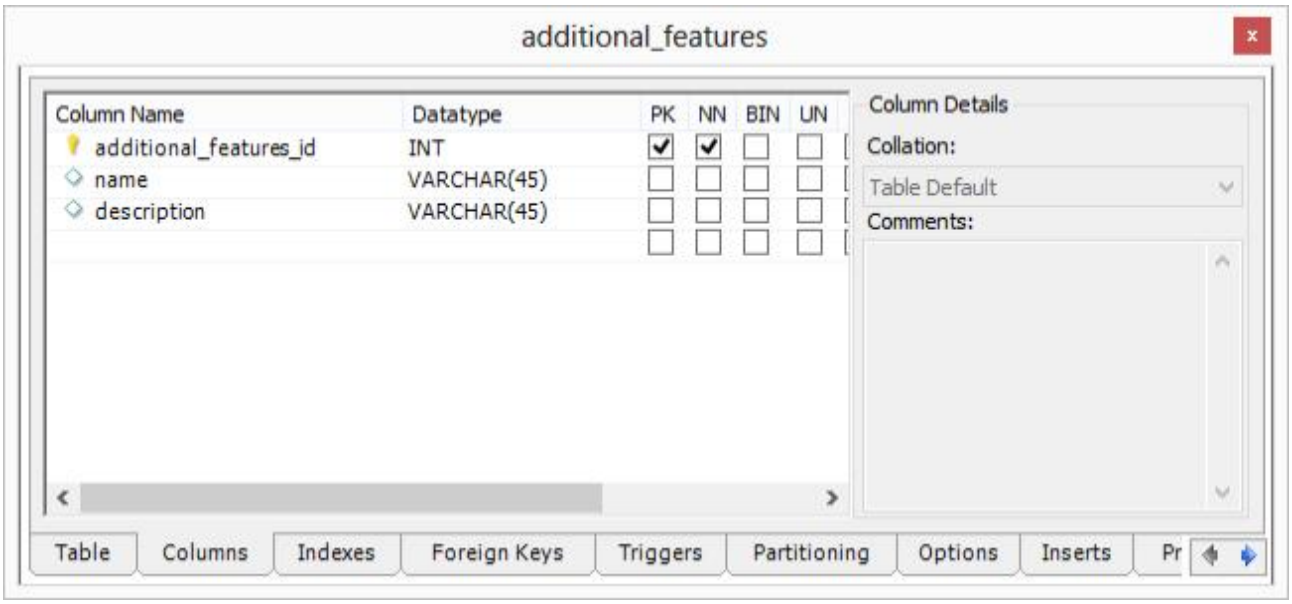

Рис. 3.8. Реалізація відношення additional_features

На рисунку 3.9 представлено реалізацію відношення interface, яке містить інформацію про використовуваний інтерфейс.

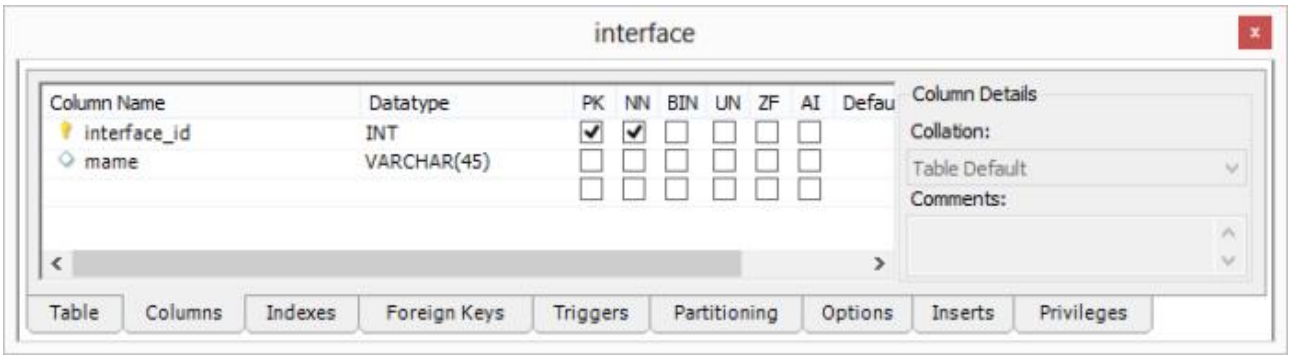

Рис. 3.9. Реалізація відношення interface

На рисунку 3.10 – реалізація відношення users, в якому зберегіється інформація про користувачів системи.

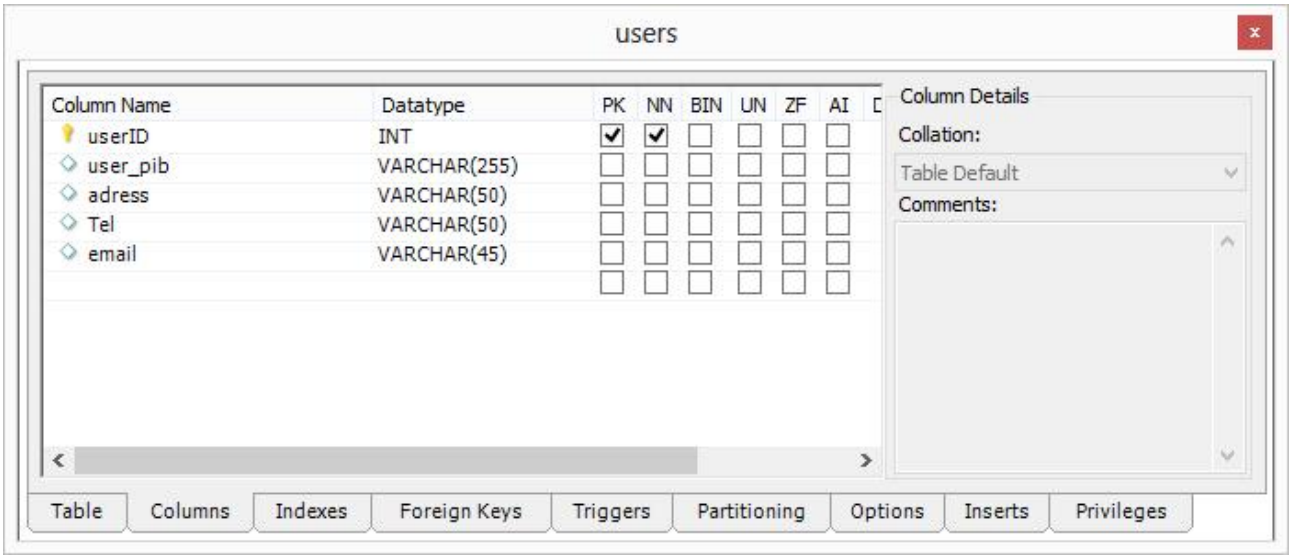

Рис. 3.10. Реалізація відношень users

Підсумкову ER діаграму представлено на рисунку 3.11. З даного рисунка видно представлення зв’язків міє таблицями БД, та їх конкретну реалізацію в середовищі MySQL.

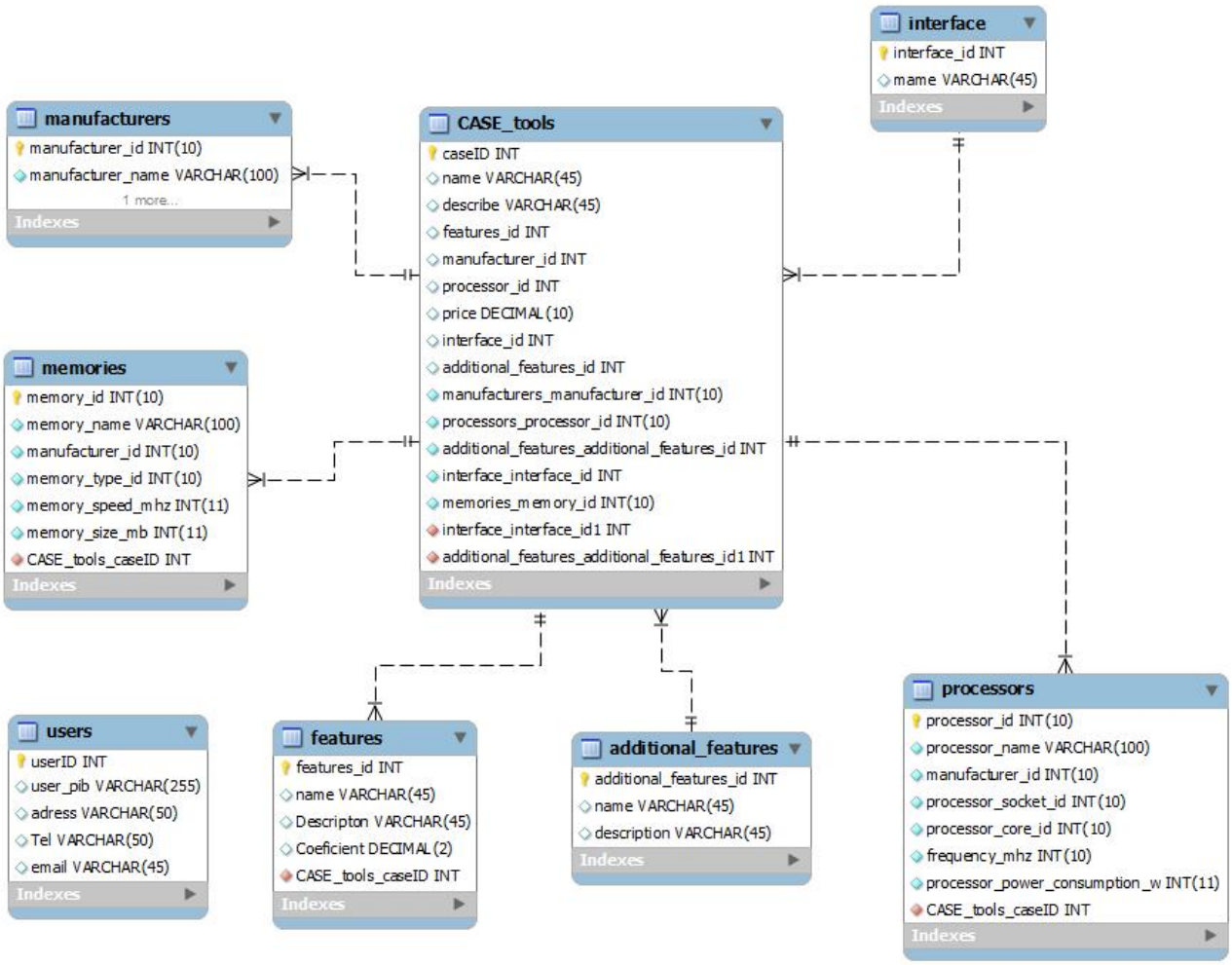

Рис. 3.11. ER-діаграма бази даних

У додатку Б до дипломної роботи представлено DDL statement бази даних системи для підтримки вибору CASE-засобів.

Висновки до розділу 3

У даному озділі обґрунтовано технологію, мову програмування та розроблено програмну систему. Обґрунтовано засоби розробки бази даних та створено програмну реалізацію бази даних програмної системи за допомогою механізму збережених процедур.

РОЗДІЛ 4
ТЕСТУВАННЯ ТА ДОСЛІДНА ЕКСПЛУАТАЦІЯ СИСТЕМИ

4.1. Тестування

Тестування програмного забезпечення (англ. Software Testing) — це процес технічного дослідження, призначений для виявлення інформації про якість продукту відносно контексту, в якому він має використовуватись. Техніка тестування також включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою оцінки. Може оцінюватись:

- відповідність вимогам, якими керувалися проектувальники та розробники
- правильна відповідь для усіх можливих вхідних даних
- виконання функцій за прийнятний час
- практичність
- сумісність з програмним забезпеченням та операційними системами
- відповідність задачам замовника.

Оскільки число можливих тестів навіть для нескладних програмних компонент практично нескінченне, тому стратегія тестування полягає в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів. Як результат програмне забезпечення (ПЗ) тестується стандартним виконанням програми з метою виявлення багів (помилок або інших дефектів).

Тестування ПЗ може надавати об'єктивну, незалежну інформацію про якість ПЗ, ризики відмови, як для користувачів так і для замовників.

Тестування може проводитись, як тільки створено виконуваний код (навіть частково завершено). Процес розробкизазвичай передбачає коли та як буде відбуватися тестування. Наприклад, при поетапному процесі, більшість тестів відбувається після визначення системних вимог і тоді вони реалізуються

в тестових програмах. На противагу цьому, відповідно до вимог гнучкої розробки ПЗ, програмування і тестування часто відбувається одночасно.

В процесі реалізації системи підтримки вибору CASE-засобів було проведено процедуру тестування. Розглянемо детальніше процедуру модульного тестування системи, оскільки вона є дуже важливою з точки зору успішності реалізації проекту.

Спочатку запускаємо інтегроване середовище розробки Eclipse (integrated development environment - IDE), створюємо базовий Java-проект, в який імпортуємо JAR-бібліотек JUnit, jMock і RMock. Java-проект - TestingPersonal. На рисунку 4.1 представлено процедуру стоврення тестового проекту в середовищі Eclipse.

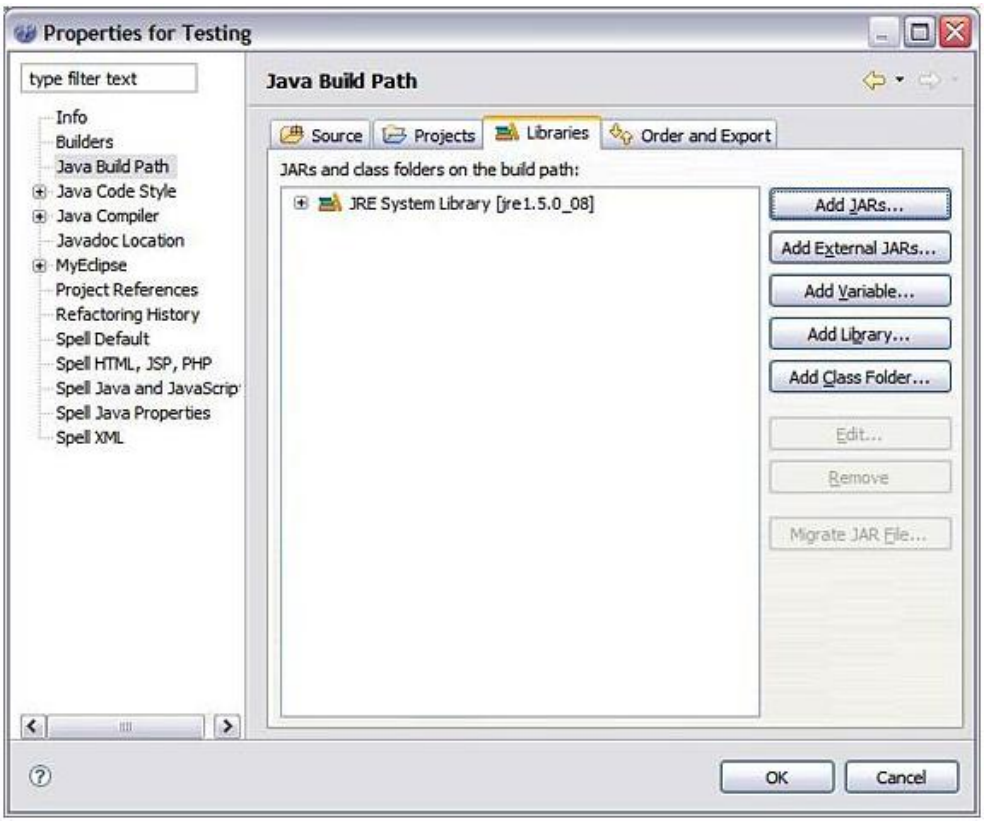

Рис. 4.1. Процедура створення тестового проекту

Використовуємо кнопку Add JARs, якщо JAR-файли вказані в Java classpath (Java Runtime Environment (JRE), налаштована в Eclipse). Кнопка Add Variable працює з конкретним каталогом файлової системи (локальної або

віддаленої), де розміщені ресурси (включаючи JAR-файли), на які можна послатися. Використовуємо кнопку Add Library, якщо потрібно послатися на такі спеціалізовані ресурси, які використовуються в Eclipse за замовчуванням або налаштовані на спеціалізоване середовище робочої області Eclipse. Натискаємо кнопку Add Class Folder, щоб додати ресурс з однієї з папок існуючих проектів, уже налаштованих як частину проекту (рисунок 4.2).

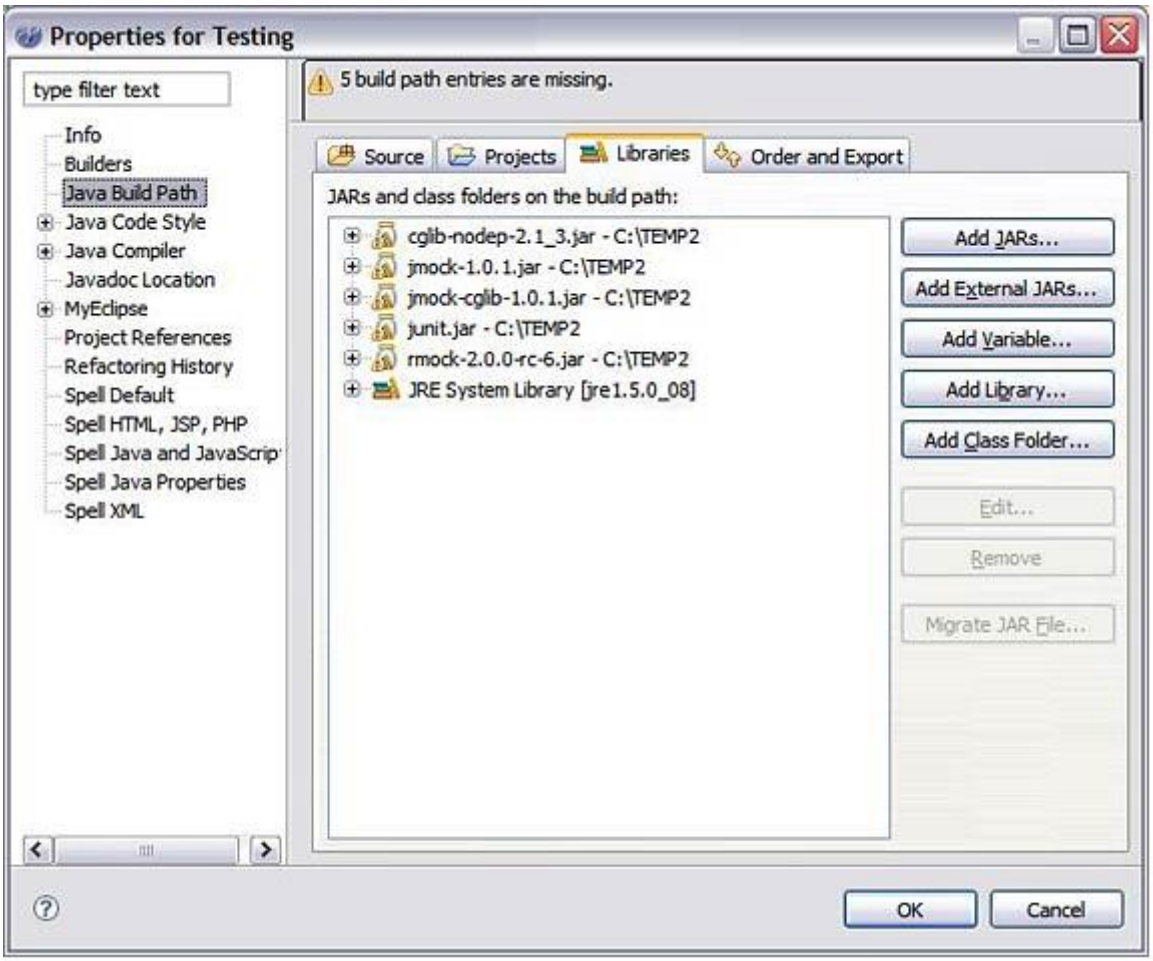

Рис. 4.2. Процедура підключення необхідних бібліотек

Розглянемо детальніше процедуру тестування модуля Testing CaseTools, а саме з вихідним кодом чотирьох класів: ServiceClass.java, CaseTools.java, ICaseTools.java, ServiceClassTest.java. Тестованим класом є ServiceClass, який містить один метод: runService (). Метод service приймає об'єкт CaseTools, який реалізує простий інтерфейс ICaseTools. Один метод реалізований в конкретному класі CaseTools:executeCase ().CaseTools. Це клас, який ми

повинні імітувати відповідним чином. Четвертий клас - це тестовий клас ServiceClassTest (реалізація максимально спрощена). У лістингу нижче показаний вихідний код цього четвертого класу.

```
public class ServiceClass {
  public ServiceClass(){
  //конструктор без аргументів
  }

  public boolean runService(ICaseTools CaseTools){
  if("success".equals(CaseTools.executeCase())){
            return true;
  }
  else
  {
            return false;
  }
}
}
```

В класі ServiceClass блок коду if ... else є простим логічним переходом, що допомагає відобразити, чому тест завершиться невдало або успішно при виборі одного (а не іншого) шляху, відповідно до очікуваних результатів. Вихідний код класу CaseTools показаний нижче.

```
public class CaseTools implements ICaseTools{
   public CaseTools(){
  //конструктор без аргументів
   }
   public String executeCase(){
     return "success";
   }
}
```

Клас CaseTools з конструктором без аргументів і простою змінною String, що повертається з методу executeCase(), теж не складний. Нижче показаний код класу ICaseTools.

```
public interface ICaseTools {
     public abstract String executeCase ();
}
```

Інтерфейс ICaseTools має один метод, який повинен бути реалізований в класі CaseTools. Маючи наведений вище код, переходимо до розгляду того, як можна успішно виконати тест класу ServiceClass в різних сценаріях.

Використання jMock для імітації інтерфейсів. Тестуємо метод service в класі ServiceClass. Припустимо, що предметом тестування є твердження, що метод runService() не виконувався, або, іншими словами, що повернений Boolean-результат дорівнює false. У цьому випадку імітується передача в метод runService() об'єкт ICaseTools для очікування виклику його методу executeCase() і повернення рядка, відмінного від "success". Таким чином гарантуємо, що

Boolean-рядок false повертається в тест. Код класу ServiceClassTest, який наведений нижче, містить логіку тесту.

```
import org.jmock.cglib.MockObjectTestCase;
public class ServiceClassTest extends MockObjectTestCase {
        private ServiceClass serviceClass;
        private Mock mockCaseTools;
        private ICaseTools CaseTools;
                public void setUp(){
                serviceClass = new ServiceClass();
                mockCaseTools = new Mock(ICaseTools.class);
        }
                public void testRunServiceAndReturnFalse(){
                mockCaseTools.expects(once()).method\
               ("executeCase").will(returnValue("failure"));
                CaseTools = (ICaseTools)mockCaseTools.proxy();
                boolean result = serviceClass.runService(CaseTools);
                assertFalse(result);
```

Зазвичай гарною ідеєю є включення в тести методу setUp(), якщо в різних прикладах тестів виконуються спільні операції. Метод tearDown() теж годиться, але він не настільки необхідний доти, поки ви не будете виконувати інтегровані тести. В jMock середовище перевіряє всі очікувані результати по всіх фіктивним об'єктах в кінці (або під час) виконання тесту. Немає реальної необхідності включати метод verify() для кожного очікуваного результату. При виконанні тесту як JUnit, тест завершується успішно (рисунок 4.3).

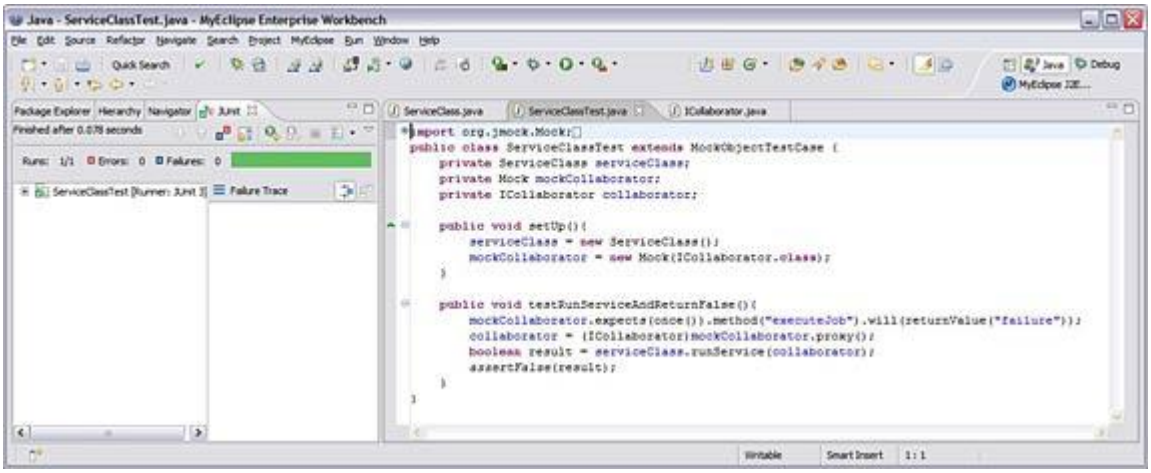

Рис. 4.3. Результати тестування

Клас ServiceTestClass розширює клас org.jmock.cglib.MockObjectTestCase jMock CGLIB. mockCaseTools - це простий клас org.jmock.JMock. Зазвичай є

два способи створення фіктивних об'єктів в jMock: для імітації інтерфейсу використовується новий метод Mock (Class.class). Для імітації конкретного класу використовується метод mock (Class.class, "identifier").

Важливо відзначити, як імітованим proxy передається в метод runService() класу ServiceClass. У jMock можна витягти реалізації proxy із створених фіктивних об'єктів, для яких очікувані результати вже були встановлені.

4.2. Розгортання програмного продукту

Для розгортання системи складемо список необхідних вимог до програмного забезпечення:

- Операційна система Microsoft Windows 7/8;
- MySQL;
- Eclipse IDE for Java EE Developers;
- Драйвера JDBC-MySQL;

Вимоги до апаратних ресурсів визначатимуться кількістю користувачів системи. Для невеликої кількості (до 100 одночасно) користувачів достатньо наступної конфігурації для апаратних ресурсів:

- процесор з тактовою частотою не менше 2 Ghz;
- оперативна пам’ять 4 Gb;
- об’єм жорсткого диску має бути не менше 512 Gb; необхідна пропускна здатність залежить від кількості користувачів.

Розглянемо процедуру розгортання проекту, яка складається з двох частин:

- розгортання Java додатку;
- налатування доступу до сервера баз даних MySQL;

Адаптер сервера WTP Eclipse - це інструмент для розгортання та тестування ресурсів Java EE на сервері WebSphere Application Server Community Edition. Перед тим, як приступити до розгортання ресурсів Java EE, слід

визначити новий сервер і середовище виконання сервера. Після створення або імпорту проекту Java EE, в інтегроване середовище розробки (IDE) Eclipse, вказуємо середовище виконання сервера WebSphere Application Server Community Edition в якості цільової середовища. Ця дія додає бібліотеки класів сервера в шлях компонування проекту.

Ресурси, розгорнуті за допомогою функцій Eclipse, рекомендується видаляти і повторно розгортати також за допомогою функцій Eclipse. Наприклад, якщо розгорнути ресурс в Eclipse і потім видалити його за допомогою Web-консолі або команди deploy, то Eclipse буде вважати, що ресурс розгорнутий. Для усунення такої неполадки слід видалити ресурси, які були опубліковані на сервері і вилучені за межами середовища Eclipse.

Для того щоб розгорнути ресурси Java EE на локальному сервері, виконаємо нступні дії:

- У проекції Java EE вибераємо панель Проект і правою кнопкою миші на потрібному проекті Java EE. Вибераємо Виконати як, Запустити на сервері. На панелі Запустити на сервері, якщо сервер встановлено, вибераємо опцію Вибрати існуючий сервер і вибрати потрібний сервер. Якщо сервер Community Edition не заданий, вибераємо опцію Задати новий сервер вручну щоб задати новий сервер. Натискаємо кнопку Готово. Адаптер сервера WTP незабаром розгорне ресурси Java EE. Якщо сервер не запущений, адаптер сервера WTP запустить його і розгорне ресурс Java EE після ініціалізації сервера.

При необхідності ресурс, опублікований на сервері, можна видалити за допомогою опції Додати/Видалити проект. Якщо просто видалити ресурс без видалення пов'язаного проекту, ресурс залишиться розгорнутим на сервері. На панелі Сервер клацаємо правою кнопкою миші на сервері, в якому вимагається розгорнути ресурс. У контекстному меню вибераємо Додати/Видалити проекти.

Процедура розгортання ресурсу Java EE на віддаленому сервері аналогічна розглянутої вище, однак перед її виконанням рекомендується звернути увагу на додаткові особливості.

Для визначення віддаленого сервера необхідно спершу задати локальний сервер і потім в атрибуті імені хоста вказати ім'я хоста віддаленого сервера. Це обов'язкова процедура, оскільки середовище Eclipse використовує бібліотеки класів локального сервера.

За допомогою Eclipse не можна запустити, зупинити або перезапустити віддалений сервер. За допомогою Eclipse не можна запустити віддалений сервер в режимі налагодження. Як правило, внаслідок такого обмеження зручно вибрати підхід, що передбачає розробку і налагодження ресурсів Java EE на локальному сервері з подальшим переходом на віддалений сервер, призначений для інтеграції ресурсів кількох розробників. Відома неполадка Apache Geronimo може викликати непередбачену виняткову ситуацію java.net.UnknownHostExc.

Незалежно від того, яким чином вказано ім'я хоста цільового сервера (навіть у тому випадку, якщо вказано повне ім'я або IP-адреса), цільовий сервер повертає неповне ім'я хоста. Іншими словами він повертає клієнту своє коротке ім'я. Відповідно з цим ім'ям клієнт запускає операцію передачі файлу. Якщо мережа не може перетворити це ім’я, видається повідомлення про виняткову ситуацію. Якщо команда ping дозволяє звернутися до цільової системі за неповним іменем хоста, віддалене розгортання повинно працювати правильним чином. Опис способу обходу цієї неполадки наведено в розділі Усунення неполадок.

Якщо локальна система та цільової сервер розділені брандмауером, необхідно додатково налаштувати передачу запитів HTTP і RMI між ними. Після встановлення сервера за замовчуванням застосовуються порт HTTP 8080 і порт RMI 1099. Якщо в конфігурації сервера вказані інші порти, необхідно додатково налаштувати брандмауер для застосування цих портів. Під час виклику команди deploy повинен виконуватися віддалений сервер. Перед розгортанням або оновленням ресурсу Java EE пов'язані файли передаються по мережі і зберігаються в якості тимчасових файлів.

Налаштування драйвера JDBC-MySQL. Драйвер jdbc треба встановлювати та підключати додатково. Наприклад, для MySQL його можна

взяти на http://dev.mysql.com/downloads/mirror.php?id=412737. Для роботи драйвера JDBC необхідно налаштувати OpenOffice.org/LibreOffice для роботи з Java. Отриманий файл-архів треба розпакувати в каталог. Потім підключити його до OpenOffice.org/LibreOffice. Сервіс-Параметри -OpenOffice.org - Java - Шлях класу-Додати архів і вибрати файл драйвера (mysql-connector-java-5.1.25-bin.jar).

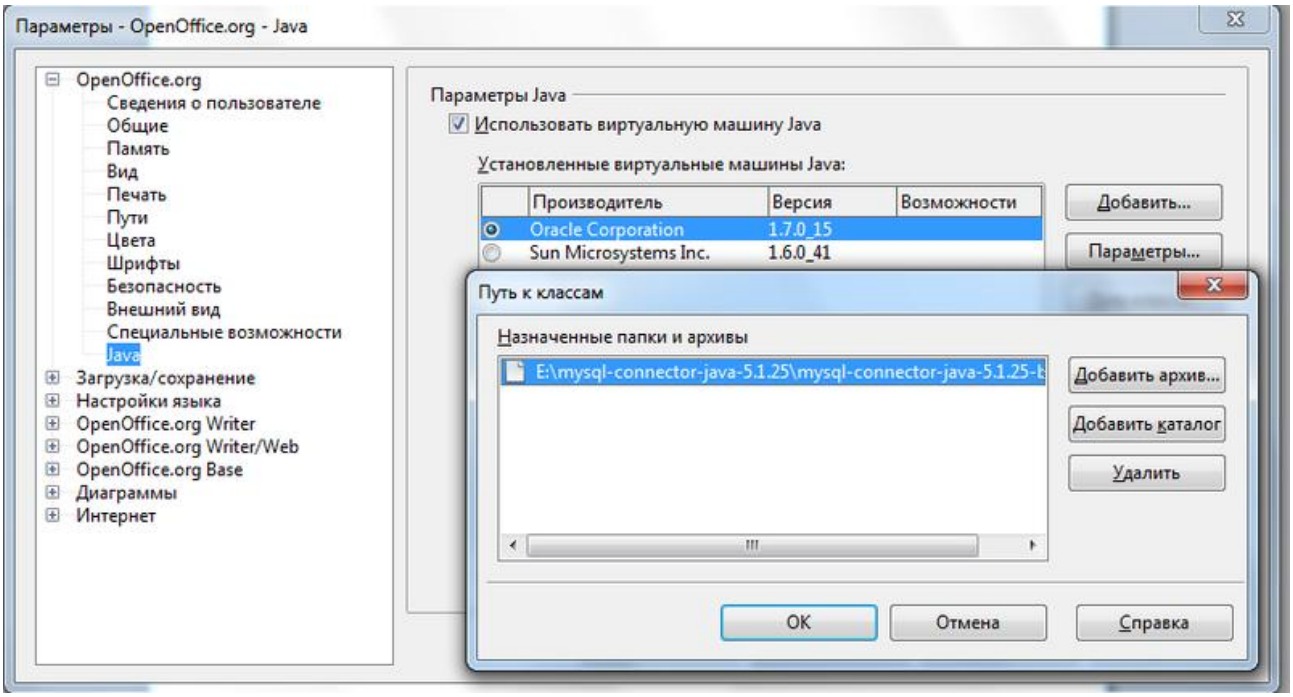

Рис. 4.4. Встановлення драйвера JDBC для MySQL

Далі перевіряємо роботу драйвера. Створюємо нову базу даних. Запускається майстер створення БД. Тут в якості джерела даних вибираємо JDBC. На другому кроці майстра необхідно виконати настройку з'єднання. Задаємо параметри налаштування з'єднання з базою MySQL через JDBC:

- Вказуємо Url джерела даних у форматі mysql://ім'я_сервера: 3306/ Імя_бази_данних_в_MySQL;
- Вказуємо клас драйвера JDBC: Для драйвера JDBC-MySQL це com.mysql.jdbc.Driver

Натискаємо кнопку "Перевірити клас". При натисканні кнопки відбувається перевірка завантаження JDBC – драйвера (рисунок 4.5).

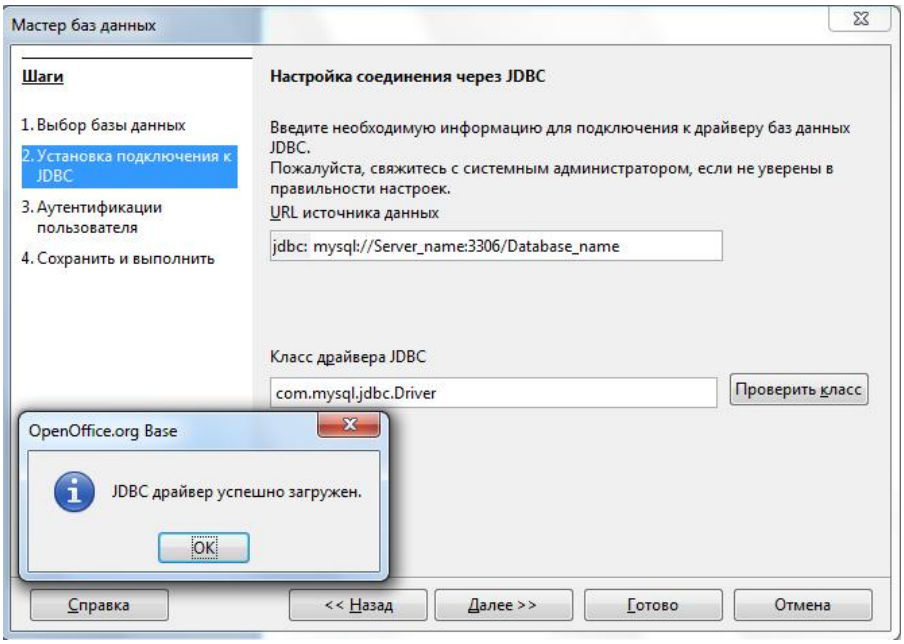

Рис. 4.5. Налаштування драйвера JDBC для MySQL

На наступному кроці пропонується ввести пароль та ім’я користувача з'єднання з MySQL для перевірки та встановлення з'єднання. Налаштування драйвера та проекту виконанено.

4.3. Інструкція користувача

Робота із системою підтримки вибору CASE-засобів для проектування систем управління підприємствами розпочинається із авторизації користувача. Цей функціонал реалізований з метою розподілу функціональних особливостей між простим користувачем та експертом з питань оцінки якісних характеристик CASE-засобів. На рисунку 4.6 представлено форму авторизації користувача в системі.

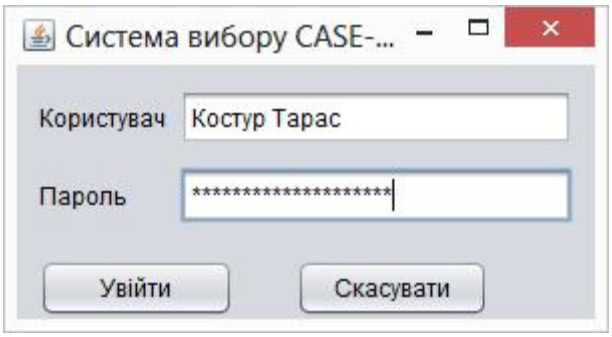

Рис. 4.6. Форма авторизації користувача системи

На рисунку 4.7 представлено екранну форму довідника CASE-засобів, яка дозволяє переглянути внесені в систему програмні продукти. По деяких продуктах була проведена експертна оцінка, а деякі наведені виключно для ознайомлення.

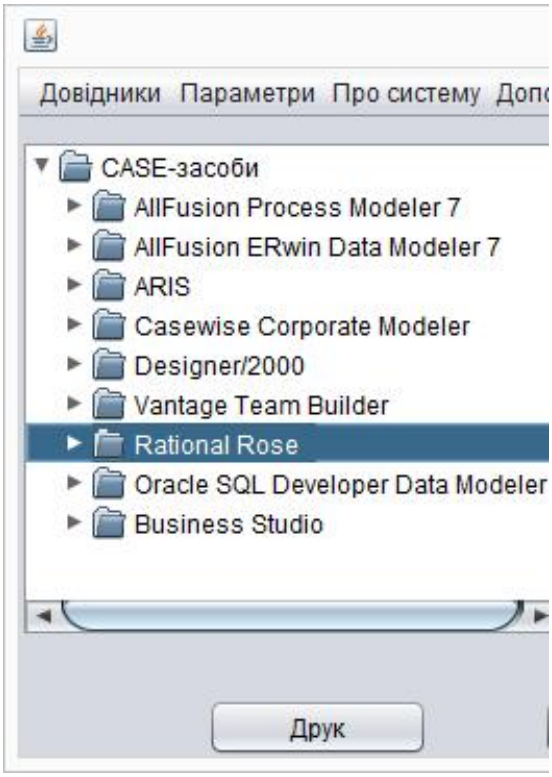

Рис. 4.7. Довідник CASE-засобів

На рисунку 4.8 представлено екранну форму перегляду основних характеристик CASE-засобів, яка дозволяє переглянути внесені в систему характеристики. Ця інформація може використовуватися в якості як довідкових даних, так і для подальшої допомогоги експертам в оцінці, а користувачам у виборі того чи іншого CASE-засобу.

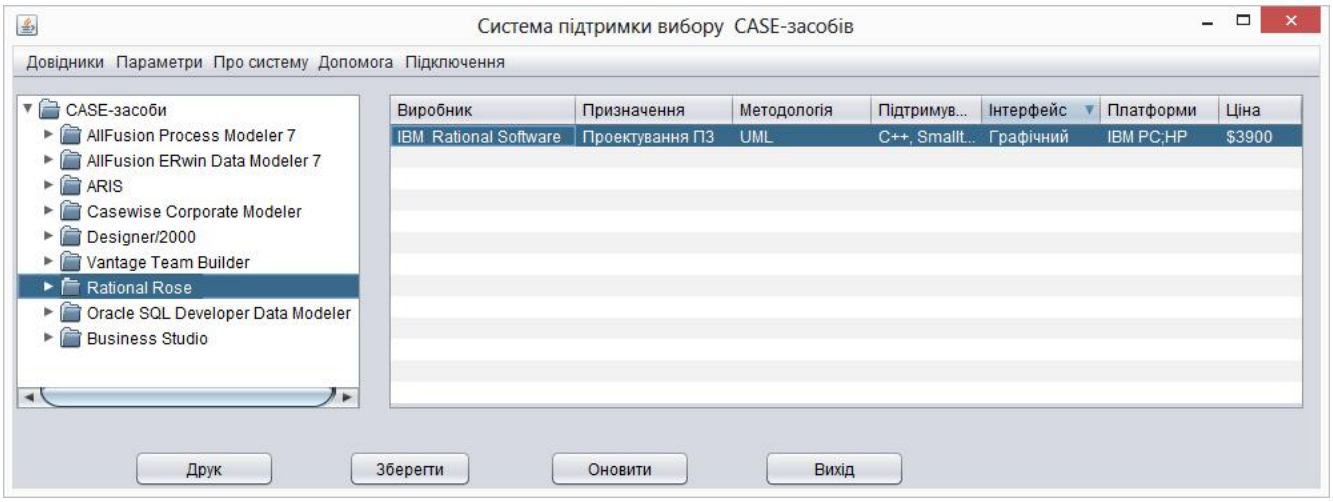

Рис. 4.8. Основні характеристики CASE-засобів

Для кожного CASE-засобу експерт може визначити основні характритстики, які на його думку є присутні у тому чи іншому програмному засобі. На рисунку 4.9 представлено форму вибору характеристик для CASE-засобу із загальної бази характеристик CASE-засобів.

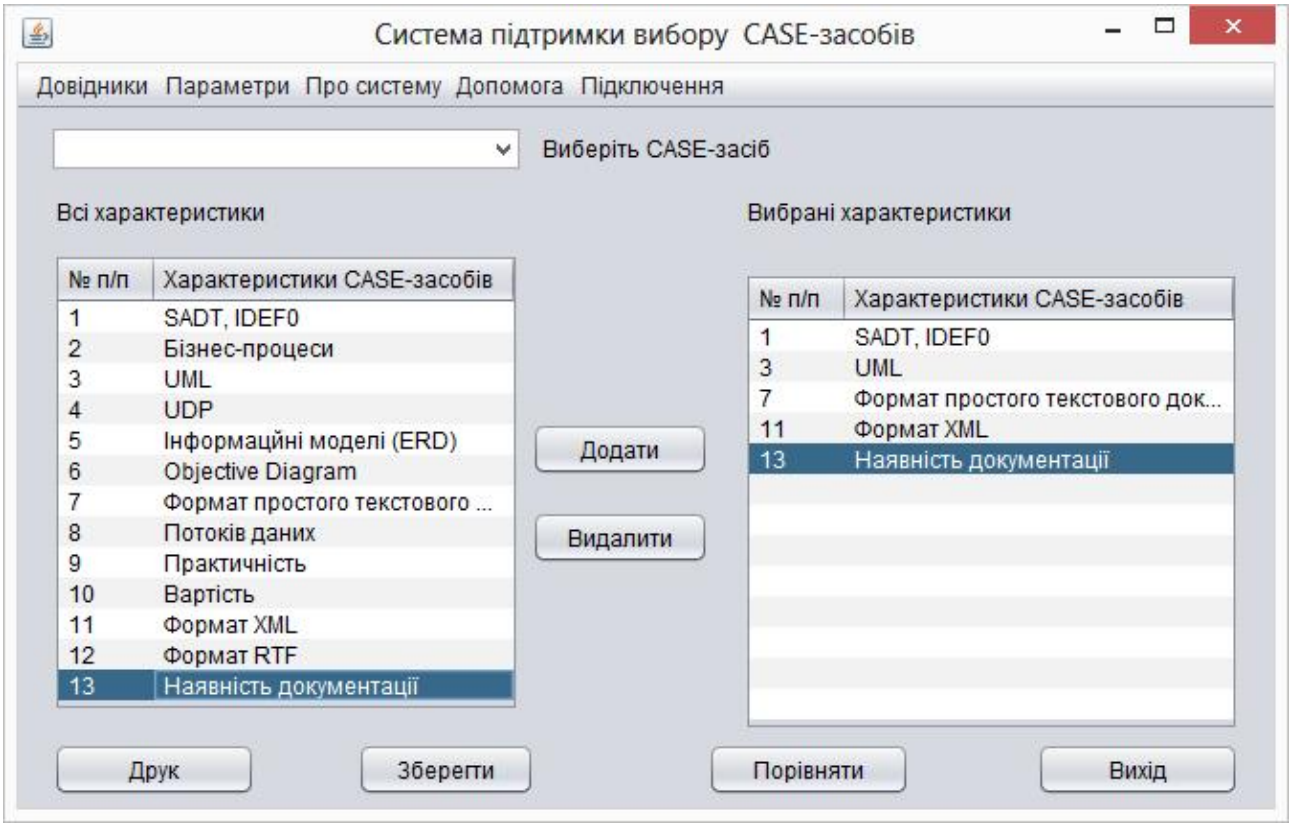

Рис. 4.9. Процедура визначення характеристик вибраного CASE-засобу

Після вибору характеристик експерт може визначити вагові коефіцієнти для даних характеристик CASE-засобів, виходячи із конкретних особливостей того чи іншого підприємства, яке в подальшому планує використовувати вказаний CASE-засіб. На рисунку 4.10. представлено форму встановлення коефіцієнтів експертом.

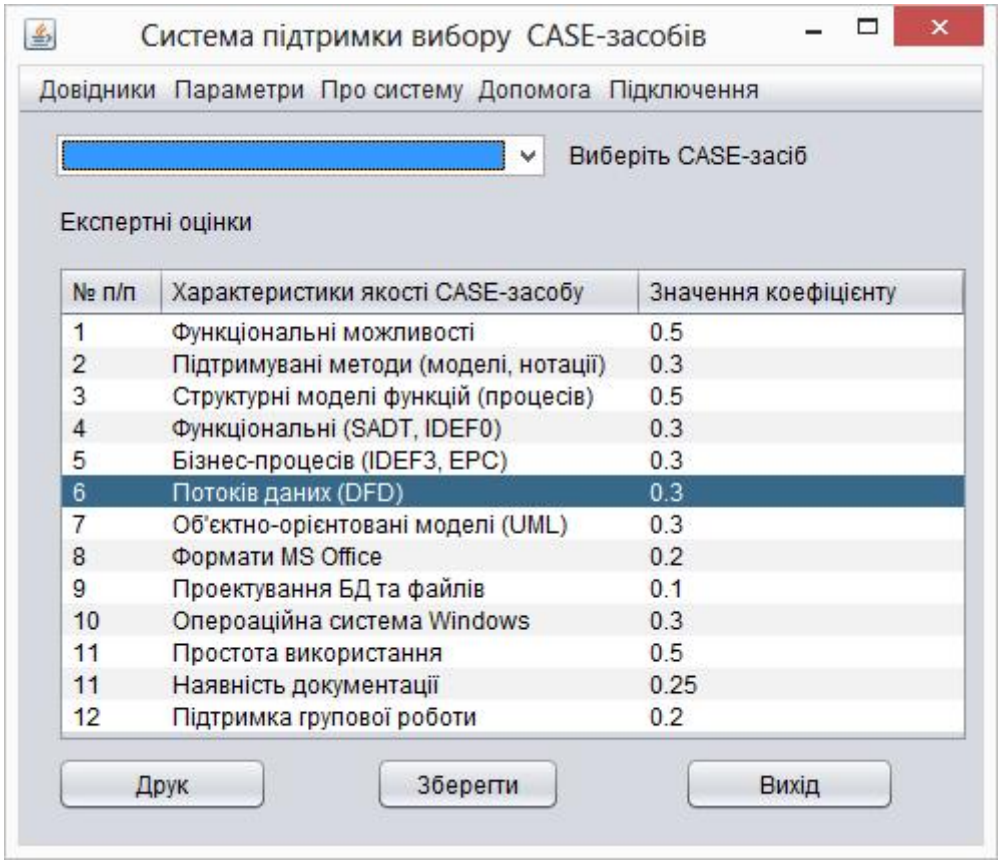

Рис. 4.10. Форма встановлення коефіцієнтів експертом

Після проведення усіх підготовчих етапів (вибір характеристик, встановлення коефіцієнтів) система дозволяє оцінити найкращий CASE-засіб для досліджуваного підприємства. Ця оцінка представлена на рисунку 4.11 у вигляді стовпчикової діаграми. На даному прикладі представлено ранжування трьох найкращих CASE-засобів для використання на досліджуваному підприємстві.

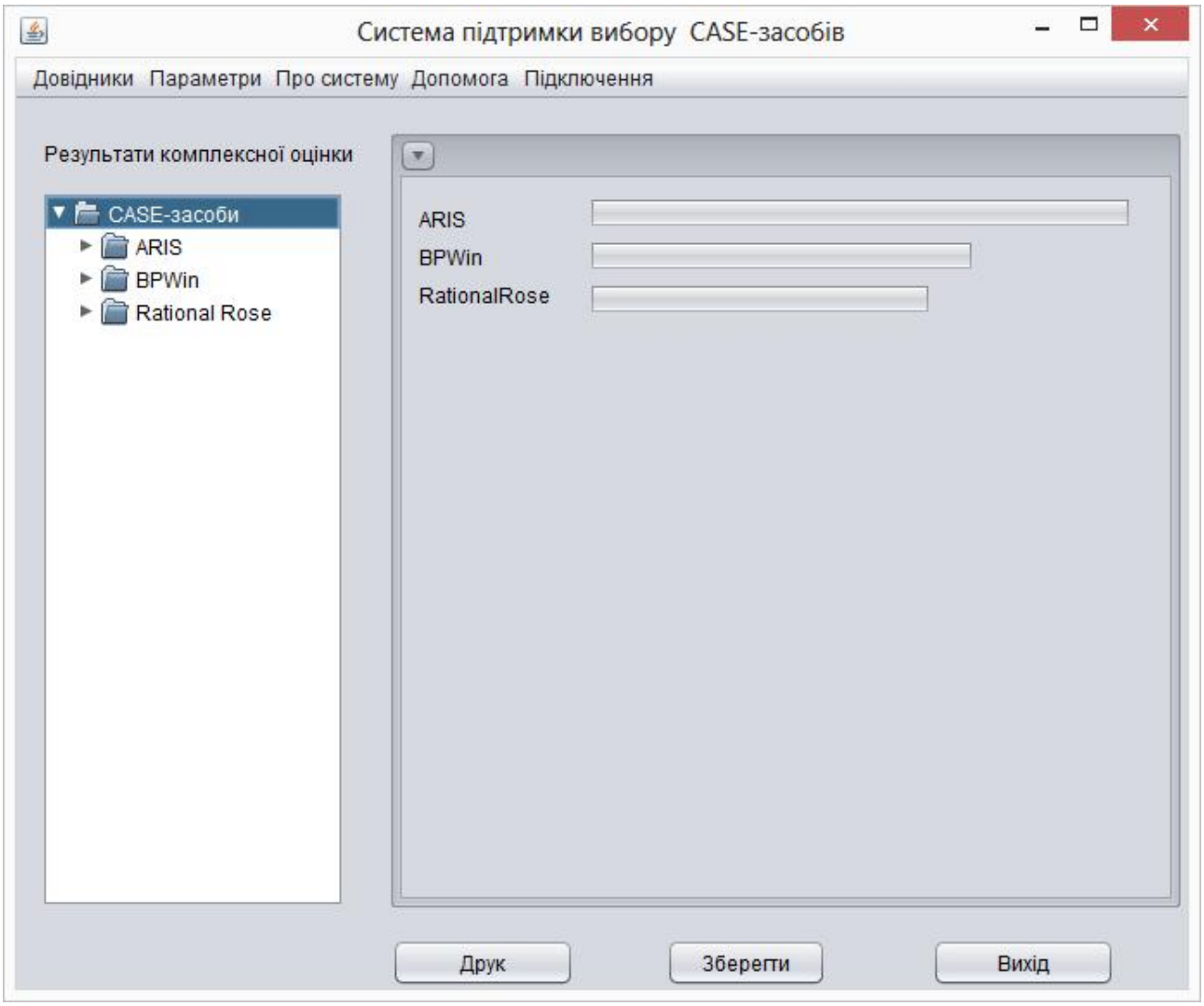

Рис. 4.11. Форма відображення результатів комплексної оцінки якості CASE-засобів

На рисунку 4.12. представлено форму, яка описує інорфмацію про систему, та її розробників.

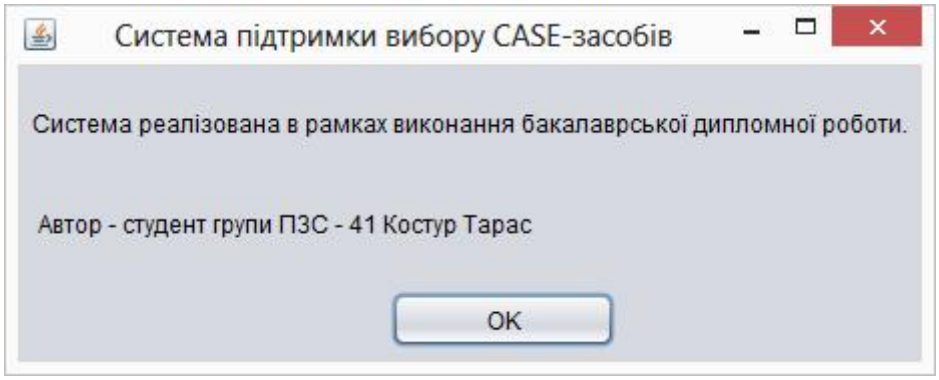

Рис. 4.12. Форма відображення інформації про систему

Висновки до розділу 4

Здійснено опис процедур тестування та їхніх результатів, описані тест-вимоги до програмного забезпечення, а також виявлені дефекти. Розкрито питання встановлення та налаштування програмного забезпечення на сервері, а також вказані вимоги, дотримання яких необхідно для користування системою, описана інструкція користувача для роботи із системою.

ВИСНОВКИ

Вибір ефективних і адекватних об'єкту методів і CASE-засобів, які застосовуються при при аналізі, проектуванні, розробці або впровадженні систем управління підприємствами, являє собою складну і відповідальну задачу.

Запропонована класифікація методологій і методів проектування допоможе орієнтуватися у множині методів при вирішенні завдань вибору оптимального набору моделей, необхідних для аналізу і проектування систем управління конкретної предметної області. Наведена в роботі методика комплексної оцінки якості програмних засобів буде корисна підприємствам і організаціям, які займаються розробкою, впровадженням та ріінжинірингом систем управління різних класів. Прикладні результати рішення задачі обгрунтування вибору CASE-засобів:

- структура і масштаб метрики якості, а також результати проведеного аналізу, будуть корисні фірмам, які займаються проектуванням і впровадженням систем управління підприємствами при виборі CASE-засобів для конкретного проекту.

- розроблена в роботі система допоможе автоматизувати основні бізнес-процеси, які пов’язані із формуванням підсумкової оцінки стосовно вибору того чи іншого CASE-засобу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ландэ Д. В., Снарский А. А., Безсуднов И. В. Интернетика: Навигация в сложных сетях: модели и алгоритмы. — М.:Либроком (Editorial URSS), 2009. — 264 с. ISBN=978-5-397-00497-8.
2. Буров Є. В. Комп’ютерні мережі: підручник / Євген Вікторович Буров. — Львів: «Магнолія 2006», 2010. — 262 с. ISBN 966-8340-69-8
3. Ландэ Д. В., Снарский А. А., Безсуднов И. В. Интернетика: Навигация в сложных сетях: модели и алгоритмы. — М.:Либроком (Editorial URSS), 2009. — 264 с. ISBN=978-5-397-00497-8.
4. Комп’ютерні мережі: [навчальний посібник] / А. Г. Микитишин, М. М. Митник, П. Д. Стухляк, В. В. Пасічник. — Львів: «Магнолія 2006», 2013. — 256 с. ISBN 978-617-574-087-3
5. Инженерное программирование для проектирования программного обеспечения. -М.: Радио і связь, 1985, -512с.
6. Бойков.В., Савинков В.М. Проектирование баз данных информационных систем. М. Мир 1997
7. Бердтис А. Структуры данних. - М.: Статистика, 1974, - 408 с.
8. Горбань О.М., Бахрушин В.Є. Основі теорії систем та системного аналізу. - Запоріжжя, ГУ "ЗІДМУ", 2004, ISBN 966-8227-23-9
9. Майо Д. Самоучитель Microsoft Visual Studio 2010 = Microsoft Visual Studio 2010: A Beginner’s Guide (A Beginners Guide). — С.: «БХВ-Петербург», 2010. — С. 464. — ISBN 978-5-9775-0609-0
10. Алекс Макки Введение в .NET 4.0 и Visual Studio 2010 для профессионалов = Introducing .NET 4.0: with Visual Studio 2010. — М.: «Вильямс», 2010. — С. 416. — ISBN 978-5-8459-1639-6
11. Кен Хендерсон Професійне керівництво з SQL Server: структура та реалізація. — М.: Издательский дом «Вильямс», 2006. — С. 1056. ISBN 5-8459-0912-0

12. Чураков Михаил. Муравьиные алгоритмы [Електронний ресурс] / Михаил Чураков,Андрей Якушев. // Режим доступу: http://rain.ifmo.ru/cat/data/theory/unsorted/ant-algo-2006/article.pdf.
13. Бормашов Д. А. “Кластерный анализ текстов”: Дипломная работа [Електронний ресурс] /Д. А. Бормашов. Режим доступу: http://inf.tsu.ru/library/DiplomaWorks/CompScience/2006/bormashov/diplom.pdf.
14. Чубукова И.А. Data Mining БИНОМ. Лаборатория знаний, Интернет-университет информационных технологий - ИНТУИТ.ру, 2006.
15. Дюк В.А., Самойленко А.П. Data Mining: учебный курс. – СПб.: Питер, 2001.
16. Барсегян А. А., Куприянов М. С., Степаненко В. В., Холод И. И. Методы и модели анализа данных: OLAP и Data Mining. – СПб.: БХВ-Петербург, 2004. – 336 с.
17. Дивак М. П., Шпінталь М.Я., Козак О.Л., Струбицька І.П., Спільчук В.М., Піговський Ю.Р. Методичні рекомендації до виконання дипломної роботи освітньо кваліфікаційного рівня «бакалавр» студентам усіх форм навчання для напряму підготовки 6.050103 – «Програмна інженерія» // Тернопіль : ФОП Шпак П.П. - 2013. - 54 с.

ДОДАТОК А

ЛІСТИНГ ОСНОВНИХ МОДУЛІВ СИСТЕМИ

```
/* Виклик методів вибору CASE-засобів
 */

public class CallingMethodsInSameClass
{
        public static void main(String[] args) {
                printOne();
                printOne();
                printTwo();
        }

        public static void printOne() {
                System.out.println("Hello World");
        }

        public static void printTwo() {
                printOne();
                printOne();
        }
}

import java.util.Arrays;
import java.awt.Rectangle;

/**
 * A sample of a ploymorphic method.
 * @author scottm
 *
 */
public class CreateASet {

    public static void main(String[] args){
        String[] words = {"A", "B", "B", "D", "C", "A"};
        System.out.println( "original: " + Arrays.toString(words));
        System.out.println( "as a set: " + Arrays.toString(makeSet(words)));

        Rectangle[] rectList = {new Rectangle(), new Rectangle(),
                    new Rectangle(0, 1, 2, 3), new Rectangle(0, 1, 2, 3)};
        System.out.println( "original: " + Arrays.toString(rectList));
        System.out.println( "as a set: " + Arrays.toString(makeSet(rectList)));


        Object[] mixed = {"A", "C", "A", "B", new Rectangle(),
                new Rectangle(), "A", new Rectangle(0, 1, 2, 3), "D"};
        System.out.println( "original: " + Arrays.toString(mixed));
        System.out.println( "as a set: " + Arrays.toString(makeSet(mixed)));
    }

    /**
     * An example of polymorphism in action. The method relies
     * on Java's inheritance requirement and polymorhphism to call
     * the correct equals method.
     * @param data != null, no elements of data are null
     * @return a Set (no duplicates) of the elements in data.
     */
    public static Object[] makeSet(Object[] data){
```

```
        assert data != null : "Failed precondition makeSet. parameter cannot be
null";
        assert noNulls(data) : "Failed precondition makeSet. no elements of
parameter can be null";
        Object[] result = new Object[data.length];
        int numUnique = 0;
        boolean found;
        int indexInResult;
        for(int i = 0; i < data.length; i++){
            // maybe should break this out into another method
            indexInResult = 0;
            found = false;
            while(!found && indexInResult < numUnique){
                found = data[i].equals(result[indexInResult]);
                indexInResult++;
            }
            if( ! found ){
                result[numUnique] = data[i];
                numUnique++;
            }
        }
        Object[] result2 = new Object[numUnique];
        System.arraycopy(result, 0, result2, 0, numUnique);
        return result2;
    }

    // pre: data != null
    // return true if all elements of data are non null,
    // false otherwise
    private static boolean noNulls(Object[] data){
        assert data != null : "Failed precondition makeSet. parameter cannot be
null";
        boolean good = true;
        int i = 0;
        while( good && i < data.length ){
            good = data[i] != null;
            i++;
        }
        return good;
    }

}
/**
 * A class that represents a node to be used in a linked list.
 * These nodes are singly linked.
 *
 * @author Mike Scott
 * @version July 27, 2015
 */

 public class ListNode
 {
         // instance variables

        // the data to store in this node
        private Object myData;

        // the link to the next node (presumably in a list)
        private ListNode myNext;

        /**
         * default constructor
         * pre: none<br>
```

```
         * post: getData() = null, getNext() = null
         */
        public ListNode()
        {       this(null, null);
        }

        /**
         * create a ListNode that holds the specified data and refers to the
specified next element
         * pre: none<br>
         * post: getData() = item, getNext() = next
         * @param item the  data this ListNode should hold
         * @param next the next node in the list
         */
        public ListNode(Object data, ListNode next)
        {       myData = data;
                myNext = next;
        }


        /**
         * return the data in this node
         * pre: none<br>
         * @return the data this ListNode holds
         */
        public Object getData()
        {       return myData; }


        /**
         * return the ListNode this ListNode refers to
         * pre: none<br>
         * @return the ListNode this ListNode refers to (normally the next one
in a list)
         */
        public ListNode getNext()
        {       return myNext; }


        /**
         * set the data in this node
         * The old data is over written.<br>
         * pre: none<br>
         * @param data the new data for this ListNode to hold
         */
        public void setData(Object data)
        {       myData = data; }

        /**
         * set the next node this ListNode refers to
         * pre: none<br>
         * @param next the next node this ListNode should refer to
         */
        public void setNext(ListNode next)
        {       myNext = next; }
 }
package Solution;


import java.io.File;
import java.lang.reflect.Method;
import java.util.Arrays;
import java.util.Collection;
```

```
import java.util.HashSet;
import java.util.Scanner;
import java.util.TreeSet;

public class UnsortedSetTest {

    public static void main(String[] args) throws Exception {
        String[] allFileNames = {"hounds.txt", "huckfinn.txt", "oz.txt",
"war.txt", "ciaFactBook2008.txt"};
        String[] noCIA = {"hounds.txt", "huckfinn.txt", "oz.txt", "war.txt"};
        countWords(new BinarySearchTree<String>(), allFileNames[0]);
        for(String s : allFileNames) {
            System.out.println(s);
            countWordsOurUnsortedSet(s);
            countWordsOurBinarySearchTree(s);
            countWordsOurHash(s);
            countWordsCollection(new TreeSet<String>(), s);
            int[] result = countWordsCollection(new HashSet<String>(), s);
            System.out.println(result[0] + " total words.");
            System.out.println(result[1] + " distinct words.");
            System.out.println();

        }
    }

    // return total num words, and num distinct words
    public static int[] countWordsCollection(Collection<String> c, String
fileName) throws Exception{
        c.clear();
        Scanner fileScanner = new Scanner(new File(fileName));
        Stopwatch st = new Stopwatch();
        st.start();
        int total = 0;
        while(fileScanner.hasNext()){
            c.add(fileScanner.next());
            total++;
        }
        st.stop();
        System.out.println("Time for " + c.getClass() + " : \n" + st);
//        System.out.println(c.size() + " distinct words");
//        System.out.println(total + " total words including duplicates: ");
        assert total >= c.size();
        System.out.println();
        return new int[]{total, c.size()};
    }


    // GACKY GACKY GACKY repition. Look into removing repetition with reflection
    // we assume there will be add and size methods
    public static int[] countWordsOurHash(String fileName) throws Exception {
        Scanner fileScanner = new Scanner(new File(fileName));
        Stopwatch st = new Stopwatch();
        UnsortedHashSet<String> c = new UnsortedHashSet<String>();
        st.start();
        int total = 0;
        while(fileScanner.hasNext()) {
            c.add(fileScanner.next());
            total++;
        }
        st.stop();
        System.out.println("Time for our hashtable (closed address hashing): \n"
+ st);
//        System.out.println(c.size() + " distinct words");
```

```
//        System.out.println(total + " total words including duplicates: ");
        assert total >= c.size();
        System.out.println();
        return new int[]{total, c.size()};
    }

    public static int[] countWordsOurUnsortedSet(String fileName) throws
Exception {
        Scanner fileScanner = new Scanner(new File(fileName));
        Stopwatch st = new Stopwatch();
        UnsortedSet<String> c = new UnsortedSet<String>();
        st.start();
        int total = 0;
        while(fileScanner.hasNext()){
            c.add(fileScanner.next());
            total++;
        }
        st.stop();
        System.out.println("Time for our unsorted set based on ArrayList: \n" +
st);
//        System.out.println(c.size() + " distinct words");
//        System.out.println(total + " total words including duplicates: ");
        assert total >= c.size();
        System.out.println();
        return new int[]{total, c.size()};
    }

    public static int[] countWordsOurBinarySearchTree(String fileName) throws
Exception {
        Scanner fileScanner = new Scanner(new File(fileName));
        Stopwatch st = new Stopwatch();
        BinarySearchTree<String> c = new BinarySearchTree<String>();
        st.start();
        int total = 0;
        while(fileScanner.hasNext()){
            c.add(fileScanner.next());
            total++;
        }
        st.stop();
        System.out.println("Time for our binary search tree: \n" + st);
//        System.out.println(c.size() + " distinct words");
//        System.out.println(total + " total words including duplicates: ");
        assert total >= c.size();
        System.out.println();
        return new int[]{total, c.size()};
    }


    // a try at reflection. Not working on Binary Search tree from class.
    // Hunch. Due to add method taking in Comparable, not Object!
    // Alterantives: search list of methods for name?
    public static int[] countWords(Object c, String fileName) throws Exception {
        Scanner fileScanner = new Scanner(new File(fileName));
        Stopwatch st = new Stopwatch();
        System.out.println(Arrays.toString(c.getClass().getMethods()));
        Method addMethod = c.getClass().getMethod("add", Object.class);
        st.start();
        int total = 0;
        while(fileScanner.hasNext()){
            addMethod.invoke(c, fileScanner.next());
            total++;
        }
        st.stop();
```

```
        System.out.println("Time for " + c.getClass() + ": "+ st);
        Method sizeMethod = c.getClass().getMethod("size");
        int distictWords = (Integer) sizeMethod.invoke(c);
//        System.out.println(distictWords + " distinct words");
//        System.out.println(total + " total words including duplicates: ");
        System.out.println();
        return new int[]{total, distictWords};
    }
}
import java.util.Arrays;
import java.util.ArrayList;

public class EightQueens {


        public static void main(String[] args) {
                        solveNQueens(8);
                        ArrayList<char[][]> solutions = getAllNQueens(8);
                        System.out.println( solutions.size() );
                        for( int i = 0; i < solutions.size(); i++){
                                System.out.println("\n\nSolution " + (i+1));
                                if( queensAreSafe(solutions.get(i)) )
                                        printBoard(solutions.get(i));
                                else
                                        System.out.println("UH OH!!!!! BETTER FIX
IT!!!!!");

        }

        /**
         * determine if the chess board represented by board is a safe set up
         * <p>pre: board != null, board.length > 0, board is a square matrix.
         * (In other words all rows in board have board.length columns.),
         * all elements of board == 'q' or '.'. 'q's represent queens, '.'s
         * represent open spaces.<br>
         * <p>post: return true if the configuration of board is safe,
         * that is no queen can attack any other queen on the board.
         * false otherwise.
         * @param board the chessboard
         * @return true if the configuration of board is safe,
         * that is no queen can attack any other queen on the board.
         * false otherwise.
        */
        public static boolean queensAreSafe(char[][] board)
        {       char[] validChars = {'q', '.'};
                assert (board != null) && (board.length > 0)
                                && isSquare(board) && onlyContains(board,
validChars)
                                : "Violation of precondition: queensAreSafe";


                return true;
        }

        public static ArrayList<char[][]> getAllNQueens(int size){
                ArrayList<char[][]> solutions = new ArrayList<char[][]>();
                char[][] board = blankBoard(size);
                solveAllNQueens(board, 0, solutions);
                return solutions;
        }

        public static void solveAllNQueens(char[][] board, int col,
ArrayList<char[][]> solutions){
```

```
                // am I done? if so, add this solution to the ArrayList of
solutions
                if( col == board.length){
                        solutions.add( makeCopy(board));
                        // all done
                } else {
                        for(int row = 0; row < board.length; row++){
                                // place queen
                                board[row][col] = 'q';
                                if( queensAreSafe(board) )
                                        // if safe go on to next column
                                        solveAllNQueens(board, col + 1,
solutions);
                                board[row][col] = '.';
                        }
                }
        }

        // pre: mat != null, mat is rectangular
        public static char[][] makeCopy(char[][] mat){
                assert mat != null;
                char[][] copy = new char[mat.length][mat[0].length];
                for(int r = 0; r < mat.length; r++)
                        for(int c = 0; c < mat[0].length; c++)
                                copy[r][c] = mat[r][c];
                return copy;
        }

        public static void printBoard(char[][] board){
                for(int r = 0; r < board.length; r++){
                        for(int c = 0; c < board[r].length; c++)
                                System.out.print(board[r][c]);
                        System.out.println();
                }
        }

        public static void solveNQueens(int n){
                char[][] board = blankBoard(n);
                //start in column 0
                boolean solved = canSolve(board, 0);
                if( solved ){
                        System.out.println("Solved the " + n + " queen
problem.");
                        printBoard(board);
                }
                else
                        System.out.println("Can't solve the " + n + " queen
problem.");
        }

        public static boolean
                canSolve(char[][] board, int col){


                //know when you are done!
                if( col == board.length)
                        return true; // solved!!!!!

                // not done, try all the rows
                boolean solved = false;
                for(int row = 0; row < board.length && !solved; row++){
                        //System.out.println(row + " " + col);
                        // place queen
```

```
                        board[row][col] = 'q';
                        if( queensAreSafe(board) )
                                solved = canSolve(board, col + 1);
                        if( !solved )
                                board[row][col] = '.';
                }
                return solved; //could be true(solved) or false(not solved)!!
        }


        private static char[][] blankBoard(int size){
                char[][] result = new char[size][size];
                for(int r = 0; r < size; r++)
                        Arrays.fill(result[r], '.');
                return result;
        }

        private static boolean inbounds(int row, int col, char[][] mat){
                return row >= 0 && row < mat.length && col >= 0 && col <
mat[0].length;
        }

        /* pre: mat != null
           post: return true if mat is a square matrix, false otherwise
        */
        private static boolean isSquare(char[][] mat)
        {       assert mat != null : "Violation of precondition: isSquare";

                final int numRows = mat.length;
                int row = 0;
                boolean square = true;
                while( square && row < numRows )
                {       square = ( mat[row] != null) && (mat[row].length ==
numRows);
                        row++;
                }
                return square;
        }

        /* pre: mat != null, valid != null
           post: return true if all elements in mat are one of the characters in
valid
        */
        private static boolean onlyContains(char[][] mat, char[] valid)
        {       assert mat != null && valid != null : "Violation of precondition:
onlyContains";

                int row = 0;
                int col;
                boolean correct = true;
                while( correct && row < mat.length)
                {       col = 0;
                        while(correct && col < mat[row].length)
                        {       correct = contains(valid, mat[row][col]);
                                col++;
                        }
                        row++;
                }
                return correct;
        }

        /* pre: list != null
           post: return true if c is in list
```

```
        */
        private static boolean contains(char[] list, char c)
        {       assert ( list != null ) : "Violation of precondition: contains";

                boolean found = false;
                int index = 0;
                while( !found && index < list.length)
                {       found = list[index] == c;
                        index++;
                }
                return found;
        }

}
public class SortedIntList extends IntListVer3{

    public SortedIntList(int initialCap){
        //call IntList constructor
        super(initialCap);
    }

    public SortedIntList(){
        super();
    }

    //override add
    public void add(int value){
        //search for location to insert value
        int pos = 0;
        while( pos < size() && value > get(pos) ){
            pos++;
        }
        super.insert(pos, value);
    }

}
import java.util.Iterator;

/**
 * Interface for a simple List. Random access to all items in the list is
provided.
 * The numbering of elements in the list begins at 0.
 *
 */
public interface IList<E> extends Iterable<E>{

        /**
         * Add an item to the end of this list.
         * <br>pre: none
         * <br>post: size() = old size() + 1, get(size() - 1) = item
         * @param item the data to be added to the end of this list
         */
        void add(E item);

        /**
         * Insert an item at a specified position in the list.
         * <br>pre: 0 <= pos <= size()
         * <br>post: size() = old size() + 1, get(pos) = item, all elements in
         * the list with a positon >= pos have a position = old position + 1
         * @param pos the position to insert the data at in the list
         * @param item the data to add to the list
        */
        void insert(int pos, E item);
```

```
        /**
         * Change the data at the specified position in the list.
         * the old data at that position is returned.
         * <br>pre: 0 <= pos < size()
         * <br>post: get(pos) = item, return the
         * old get(pos)
     * @param pos the position in the list to overwrite
         * @param item the new item that will overwrite the old item
         * @return the old data at the specified position
         */
        E set(int pos, E item);

        /**
         * Get an element from the list.
         * <br>pre: 0 <= pos < size()
         * <br>post: return the item at pos
         * @param pos specifies which element to get
         * @return the element at the specified position in the list
         */
        E get(int pos);


        /**
         * Remove an element in the list based on position.
         * <br>pre: 0 <= pos < size()
         * <br>post: size() = old size() - 1, all elements of
         * list with a positon > pos have a position = old position - 1
         * @param pos the position of the element to remove from the list
         * @return the data at position pos
         */
        E remove(int pos);

        /**
         * Remove the first occurrence of obj in this list.
         * Return <tt>true</tt> if this list changed as a result of this call,
<tt>false</tt> otherwise.
         * <br>pre: none
         * <br>post: if obj is in this list the first occurence has been removed
and size() = old size() - 1.
         * If obj is not present the list is not altered in any way.
         * @param obj The item to remove from this list.
         * @return Return <tt>true</tt> if this list changed as a result of this
call, <tt>false</tt> otherwise.
         */
        boolean remove(E obj);

        /**
         * Return a sublist of elements in this list from <tt>start</tt>
inclusive to <tt>stop</tt> exclusive.
         * This list is not changed as a result of this call.
         * <br>pre: <tt>0 <= start < size(), start <= stop <= size()</tt>
         * <br>post: return a list whose size is stop - start and contains the
elements at positions start through stop - 1 in this list.
         * @param start index of the first element of the sublist.
         * @param stop stop - 1 is the index of the last element of the sublist.
         * @return a list with <tt>stop - start</tt> elements, The elements are
from positions <tt>start</tt> inclusive to
         * <tt>stop</tt> exclusive in this list.
         */
        IList<E> getSubList(int start, int stop);

        /**
```

```
         * Return the size of this list. In other words the number of elements
in this list.
         * <br>pre: none
         * <br>post: return the number of items in this list
         * @return the number of items in this list
         */
        int size();

        /**
         * Find the position of an element in the list.
         * <br>pre: none
         * <br>post: return the index of the first element equal to item
         * or -1 if item is not present
         * @param item the element to search for in the list
         * @return return the index of the first element equal to item or a -1
if item is not present
         */
        int indexOf(E item);

        /**
         * find the position of an element in the list starting at a specified
position.
         * <br>pre: 0 <= pos < size()
         * <br>post: return the index of the first element equal to item
starting at pos
         * or -1 if item is not present from position pos onward
         * @param item the element to search for in the list
         * @param pos the position in the list to start searching from
         * @return starting from the specified position return the index of the
first element equal to item or a -1 if item is not present between pos and the
end of the list
         */
        int indexOf(E item, int pos);

        /**
         * return the list to an empty state.
         * <br>pre: none
         * <br>post: size() = 0
         */
        void makeEmpty();

        /**
        * return an Iterator for this list.
        * <br>pre: none
        * <br>post: return an Iterator object for this List
        */
    Iterator<E> iterator();

    /**
     * Remove all elements in this list from <tt>start</tt> inclusive to
<tt>stop</tt> exclusive.
     * <br>pre: <tt>0 <= start < size(), start <= stop <= size()</tt>
     * <br>post: <tt>size() = old size() - (stop - start)</tt>
     * @param start position at beginning of range of elements to be removed
     * @param stop stop - 1 is the position at the end of the range of elements
to be removed
     */
    void removeRange(int start, int stop);

    /**
     * Return a String version of this list enclosed in
     * square brackets, []. Elements are in
     * are in order based on position in the
```

```
     * list with the first element
     * first. Adjacent elements are seperated by comma's
     * @return a String representation of this IList
     */
    public String toString();

    /**
     * Determine if this IList is equal to other. Two
     * ILists are equal if they contain the same elements
     * in the same order.
     * @return true if this IList is equal to other, false otherwise
     */
    public boolean equals(Object other);
}
/**
 * A class to provide a simple list.
 * List resizes automatically. Used to illustrate
 * various design and implementation details of
 * a class in Java.
 *
 * @author scottm
 *
 */
public class GenericList{
    // class constant for default size
    private static final int DEFAULT_CAP = 10;

    //instance variables
    // iValues store the elements of the list and
    // may have extra capacity
    private Object[] iValues;
    private int iSize;

    /**
     * Default add method. Add x to the end of this IntList.
     * Size of the list goes up by 1.
     * @param x The value to add to the end of this list.
     */
    public void add(Object x){
        insert(iSize, x);
    }

    public Object get(int pos){
        return iValues[pos];
    }

    /**
     * Insert obj at position pos.
     * post: get(pos) = x, size() = old size() + 1
     * @param pos 0 <= pos <= size()
     * @param obj The element to add.
     */
    public void insert(int pos, Object obj){
        ensureCapcity();
        for(int i = iSize; i > pos; i--){
            iValues[i] = iValues[i - 1];
        }
        iValues[pos] = obj;
        iSize++;
    }

    public Object remove(int pos){
        Object removedValue = iValues[pos];
```

```
    for(int i = pos; i < iSize - 1; i++)
        iValues[i] = iValues[i + 1];
    iValues[iSize - 1] = null;
    iSize--;
    return removedValue;
}

private void ensureCapcity(){
    // is there extra capacity available?
    // if not, resize
    if(iSize == iValues.length)
        resize();
}

public int size(){
    return iSize;
}

// resize internal storage container by a factor of 2
private void resize() {
    Object[] temp = new Object[iValues.length * 2];
    System.arraycopy(iValues, 0, temp, 0, iValues.length);
    iValues = temp;
}

/**
 * Return a String version of this list. Size and
 * elements included.
 */
public String toString(){
    // we could make this more effecient by using a StringBuffer.
    // See alternative version
    String result = "size: " + iSize + ", elements: [";
    for(int i = 0; i < iSize - 1; i++)
        result += iValues[i].toString() + ", ";
    if(iSize > 0 )
        result += iValues[iSize - 1];
    result += "]";
    return result;
}

// Would not really have this and toString available
// both included just for testing
public String toStringUsingStringBuffer(){
    StringBuffer result = new StringBuffer();
    result.append( "size: " );
    result.append( iSize );
    result.append(", elements: [");
    for(int i = 0; i < iSize - 1; i++){
        result.append(iValues[i]);
        result.append(", ");
    }
    if( iSize > 0 )
        result.append(iValues[iSize - 1]);
    result.append("]");
    return result.toString();
}

/**
 * Default constructor. Creates an empty list.
 */
public GenericList(){
    //redirect to single int constructor
```

```
        this(DEFAULT_CAP);
        //other statments could go here.
    }

    /**
     * Constructor to allow user of class to specify
     * initial capacity in case they intend to add a lot
     * of elements to new list. Creates an empty list.
     * @param initialCap > 0
     */
    public GenericList(int initialCap) {
        assert initialCap > 0 : "Violation of precondition. IntListVer1(int
initialCap):"
            + "initialCap must be greater than 0. Value of initialCap: " +
initialCap;
        iValues = new Object[initialCap];
        iSize = 0;
    }

   /**
    * Return true if this IntList is equal to other.<br>
    * pre: none
    * @param other The object to comapre to this
    * @return true if other is a non null, IntList object
    * that is the same size as this IntList and has the
    * same elements in the same order, false otherwise.
    */
    public boolean equals(Object other){
        boolean result;
        if(other == null)
            // we know this is not null so can't be equal
            result = false;
        else if(this == other)
            // quick check if this and other refer to same IntList object
            result = true;
        else if( this.getClass() != other.getClass() )
            // other is not an IntList they can't be equal
            result = false;
        else{
            // other is not null and refers to an IntList
            GenericList otherList = (GenericList)other;
            result = this.size() == otherList.size();
            int i = 0;
            while(i < iSize && result){
                result = this.iValues[i].equals( otherList.iValues[i] );
                i++;
            }
        }
        return result;
    }

}
```

ДОДАТОК Б

DDL БАЗИ ДАНИХ

```
-- База даних: `CASE TOOLS`
--

CREATE  TABLE IF NOT EXISTS `mydb`.`CASE_tools` (
  `caseID` INT NOT NULL ,
  `name` VARCHAR(45) NULL DEFAULT NULL ,
  `describe` VARCHAR(45) NULL DEFAULT NULL ,
  `features_id` INT NULL ,
  `manufacturer_id` INT NULL DEFAULT NULL ,
  `processor_id` INT NULL DEFAULT NULL ,
  `price` DECIMAL(10) NULL DEFAULT NULL ,
  `interface_id` INT NULL DEFAULT NULL ,
  `additional_features_id` INT NULL DEFAULT NULL ,
  `manufacturers_manufacturer_id` INT(10) UNSIGNED NOT NULL ,
  `processors_processor_id` INT(10) UNSIGNED NOT NULL ,
  `additional_features_additional_features_id` INT NOT NULL ,
  `interface_interface_id` INT NOT NULL ,
  `memories_memory_id` INT(10) UNSIGNED NOT NULL ,
  `interface_interface_id1` INT NOT NULL ,
  `additional_features_additional_features_id1` INT NOT NULL ,
  PRIMARY KEY (`caseID`) ,
  INDEX `fk_CASE_tools_manufacturers1` (`manufacturers_manufacturer_id` ASC) ,
  INDEX `fk_CASE_tools_processors1` (`processors_processor_id` ASC) ,
  INDEX `fk_CASE_tools_additional_features1`
(`additional_features_additional_features_id` ASC) ,
  INDEX `fk_CASE_tools_interface1` (`interface_interface_id` ASC) ,
  INDEX `fk_CASE_tools_memories1` (`memories_memory_id` ASC) ,
  INDEX `fk_CASE_tools_interface1` (`interface_interface_id1` ASC) ,
  INDEX `fk_CASE_tools_additional_features1`
(`additional_features_additional_features_id1` ASC) ,
  CONSTRAINT `fk_CASE_tools_interface1`
    FOREIGN KEY (`interface_interface_id1` )
    REFERENCES `mydb`.`interface` (`interface_id` )
```

```
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_CASE_tools_additional_features1`
    FOREIGN KEY (`additional_features_additional_features_id1` )
    REFERENCES `mydb`.`additional_features` (`additional_features_id` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
CREATE  TABLE IF NOT EXISTS `mydb`.`manufacturers` (
  `manufacturer_id` INT(10) UNSIGNED NOT NULL AUTO_INCREMENT ,
  `manufacturer_name` VARCHAR(100) NOT NULL ,
  `CASE_tools_caseID` INT NOT NULL ,
  PRIMARY KEY (`manufacturer_id`) ,
  UNIQUE INDEX `manufacturer_name` (`manufacturer_name` ASC) ,
  INDEX `fk_manufacturers_CASE_tools1` (`CASE_tools_caseID` ASC) ,
  CONSTRAINT `fk_manufacturers_CASE_tools1`
    FOREIGN KEY (`CASE_tools_caseID` )
    REFERENCES `mydb`.`CASE_tools` (`caseID` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8
CREATE  TABLE IF NOT EXISTS `mydb`.`interface` (
  `interface_id` INT NOT NULL ,
  `mame` VARCHAR(45) NULL DEFAULT NULL ,
  PRIMARY KEY (`interface_id`) )
ENGINE = InnoDB
CREATE  TABLE IF NOT EXISTS `mydb`.`memories` (
  `memory_id` INT(10) UNSIGNED NOT NULL AUTO_INCREMENT ,
  `memory_name` VARCHAR(100) NOT NULL ,
  `manufacturer_id` INT(10) UNSIGNED NOT NULL ,
  `memory_type_id` INT(10) UNSIGNED NOT NULL ,
  `memory_speed_mhz` INT(11) NOT NULL ,
  `memory_size_mb` INT(11) NOT NULL ,
  `CASE_tools_caseID` INT NOT NULL ,
```

```
  PRIMARY KEY (`memory_id`) ,
  INDEX `manufacturer_id` (`manufacturer_id` ASC) ,
  INDEX `memory_type_id` (`memory_type_id` ASC) ,
  INDEX `fk_memories_CASE_tools1` (`CASE_tools_caseID` ASC) ,
  CONSTRAINT `fk_memories_CASE_tools1`
    FOREIGN KEY (`CASE_tools_caseID` )
    REFERENCES `mydb`.`CASE_tools` (`caseID` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8
CREATE  TABLE IF NOT EXISTS `mydb`.`users` (
  `userID` INT NOT NULL ,
  `user_pib` VARCHAR(255) NULL ,
  `adress` VARCHAR(50) NULL ,
  `Tel` VARCHAR(50) NULL ,
  `email` VARCHAR(45) NULL ,
  PRIMARY KEY (`userID`) )
ENGINE = InnoDB
CREATE  TABLE IF NOT EXISTS `mydb`.`features` (
  `features_id` INT NOT NULL ,
  `name` VARCHAR(45) NULL ,
  `Descripton` VARCHAR(45) NULL ,
  `Coeficient` DECIMAL(2) NULL ,
  `CASE_tools_caseID` INT NOT NULL ,
  PRIMARY KEY (`features_id`) ,
  INDEX `fk_features_CASE_tools1` (`CASE_tools_caseID` ASC) ,
  CONSTRAINT `fk_features_CASE_tools1`
    FOREIGN KEY (`CASE_tools_caseID` )
    REFERENCES `mydb`.`CASE_tools` (`caseID` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
CREATE  TABLE IF NOT EXISTS `mydb`.`additional_features` (
  `additional_features_id` INT NOT NULL ,
```

```
  `name` VARCHAR(45) NULL DEFAULT NULL ,
  `description` VARCHAR(45) NULL DEFAULT NULL ,
  PRIMARY KEY (`additional_features_id`) )
ENGINE = InnoDB
CREATE  TABLE IF NOT EXISTS `mydb`.`processors` (
  `processor_id` INT(10) UNSIGNED NOT NULL AUTO_INCREMENT ,
  `processor_name` VARCHAR(100) NOT NULL ,
  `manufacturer_id` INT(10) UNSIGNED NOT NULL ,
  `processor_socket_id` INT(10) UNSIGNED NOT NULL ,
  `processor_core_id` INT(10) UNSIGNED NOT NULL ,
  `frequency_mhz` INT(10) UNSIGNED NOT NULL ,
  `processor_power_consumption_w` INT(11) NOT NULL ,
  `CASE_tools_caseID` INT NOT NULL ,
  PRIMARY KEY (`processor_id`) ,
  INDEX `processor_id` (`processor_id` ASC) ,
  INDEX `manufacturer_id` (`manufacturer_id` ASC) ,
  INDEX `processor_socket_id` (`processor_socket_id` ASC) ,
  INDEX `processor_core_id` (`processor_core_id` ASC) ,
  INDEX `fk_processors_CASE_tools1` (`CASE_tools_caseID` ASC) ,
  CONSTRAINT `fk_processors_CASE_tools1`
    FOREIGN KEY (`CASE_tools_caseID` )
    REFERENCES `mydb`.`CASE_tools` (`caseID` )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8
```