УДК 004.415.2.045 (076.5)

Нечай О. С.

ЗАСОБИ ДІАГНОСТИКИ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У статті досліджено засоби реалізації методу діагностики об'єктно-орієнтованого програмного забезпечення, сутність якого полягає в моніторингу дефектів проектування. Визначено функції та режими роботи засобів, розроблено архітектуру та її компоненти.

Ключові слова: діагностика програмного забезпечення, архітектура програмного забезпечення, засоби візуалізації, якість програмного забезпечення.

Вступ

Витрати на супровід програмного забезпечення досягають 85–90 % бюджету життєвого циклу інформаційних систем [1]. Значну частину зусиль під час супроводу направлено на визначення уражених дефектами проектування конструкцій програмного забезпечення та їх відновлення [2, 3]. Дефект проектування – це невідповідність правилу проектування структурних характеристик елемента або фрагмента конструкції програми [6].

Нинішні засоби діагностики об'єктно-орієнтованого програмного забезпечення дають велику кількість помилкових результатів [4] або результатів, що не збігаються з оцінками розробників [5]. Тому виявлення дефектів проектування залишається ресурсномістким завданням, яке виконується переважно вручну.

Таким чином, є необхідність досліджувати та розробляти нові засоби діагностики, які б підвищували рівень автоматизації виявлення дефектів проектування та знижували вартість супроводу програмного забезпечення.

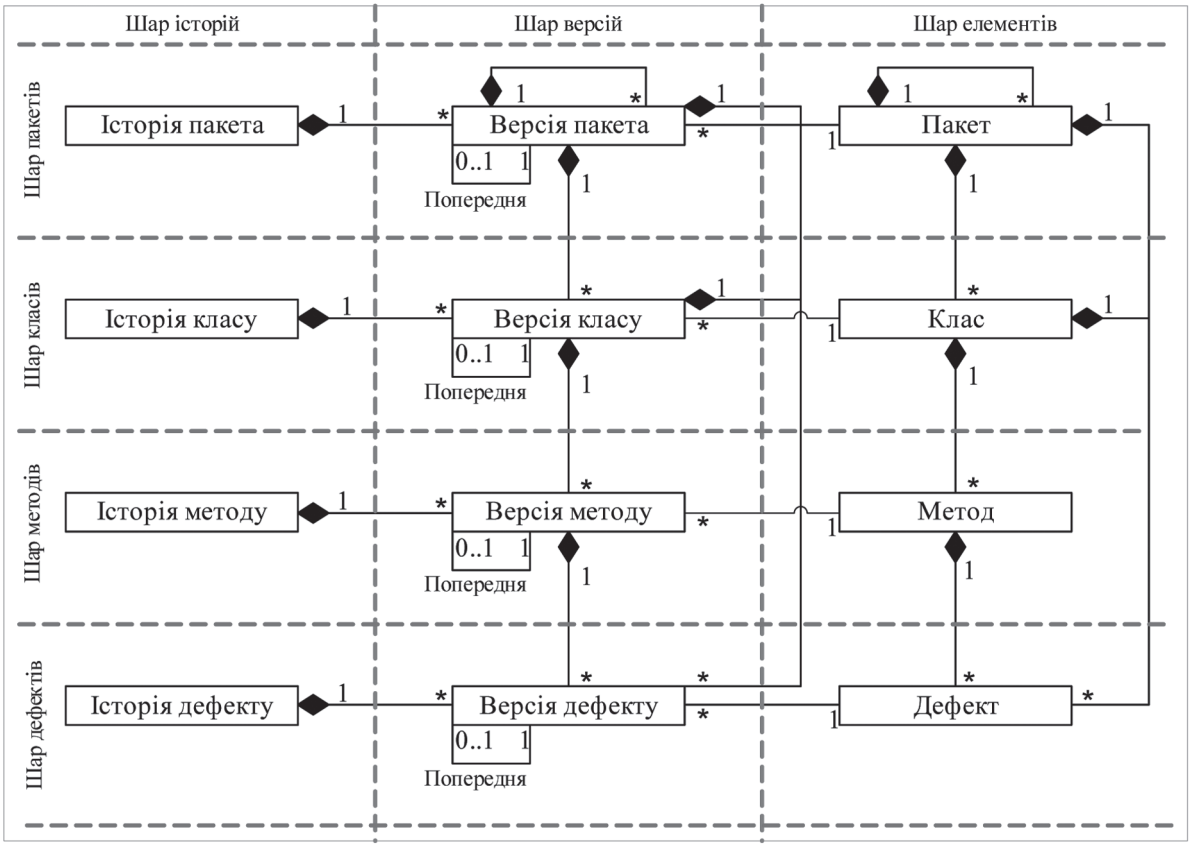
Огляд останніх досліджень і публікацій

У літературі описано значну кількість засобів, розроблених для реалізації методів виявлення дефектів проектування [7-13]. Засіб з відкритим вихідним кодом Findbugs [7] реалізує метод виявлення дефектів шляхом пошуку помилкових ідіом коду, які автори називають зразками дефектів. Інтегрована платформа оцінки якості об'єктно-орієнтованого проектування iPlasma [8] забезпечує автоматизацію аналізу, діагностики і поліпшення структури об'єктно-орієнтованого програмного забезпечення, підтримує всі необхідні фази аналізу від вилучення моделей до обчислення метрик і виявлення дублікатів коду. У роботі [9] запропоновано засіб виявлення дефектів проектування SMELLFW. Дефекти проектування специфікуються за допомогою предметно-орієнтованої мови програмування й алгоритми виявлення генеруються автоматично на основі специфікації.

Багато засобів виконують візуалізацію програмного забезпечення для виявлення дефектів проектування [10-13]. Наприклад, засіб CodeCrawler [10], який поєднує метрики та візуалізацію, відображає властивості елементів конструкції програмного забезпечення на властивості графічних зображень, дозволяючи таким чином візуально виявити аномалії в структурі програмного забезпечення. Засіб Codecity [11] відображає конструкцію програмної системи, використовуючи тривимірну графіку та метафору міста. Конструкції міста, що представляють уражені дефектом проектування елементи конструкції програмного забезпечення, виділяються кольорами. Таким чином формується зображення, яке автори назвали «картою дефектів».

Розглянуті засоби аналізують лише одну версію програмного забезпечення, тому не можуть надати важливу інформацію про його історію, а однією з причин втрати працездатності програмного забезпечення є накопичення помилкових проектних рішень, тобто саме розвиток дефектів проектування з часом. Засіб Van [12] дозволяє використовувати історичну інформацію про уражені дефектами проектування елементи конструкції для підвищення точності виявлення дефектів проектування. Проте не існує засобів діагности об'єктно-орієнтованого програмного забезпечення, які б дозволяли визначати дефекти проектування на етапі їхнього зародження, спостерігати за їхнім розвитком і своєчасно планувати роботу з реструктуризації.

У зв'язку з цим, у роботі [13] запропоновано метод діагностики об'єктно-орієнтованого програмного забезпечення, сутність якого полягає в



Мал. 1. Метамодель історії дефектів проектування

моніторингу його дефектів проектування. Метод реалізується шляхом використання метамоделі історії дефектів проектування (DDHM – Design Flaws History Model) об'єктно-орієнтованого програмного забезпечення та багатоаспектної візуалізації дефектів проектування елементів конструкції програмного різного рівня абстракції. Для реалізації цього методу розроблено відповідні засоби.

Мета статті

Основною метою нашої статті є розробка та дослідження засобів для реалізації запропонованого в роботі [13] методу діагностики об'єктно-орієнтованого програмного забезпечення. Для досягнення мети потрібно:

- визначити функції та необхідні режими роботи засобів;
- розробити архітектуру засобів;
- розробити та реалізувати основні компоненти засобів архітектури.

Метамодель історії дефектів проектування

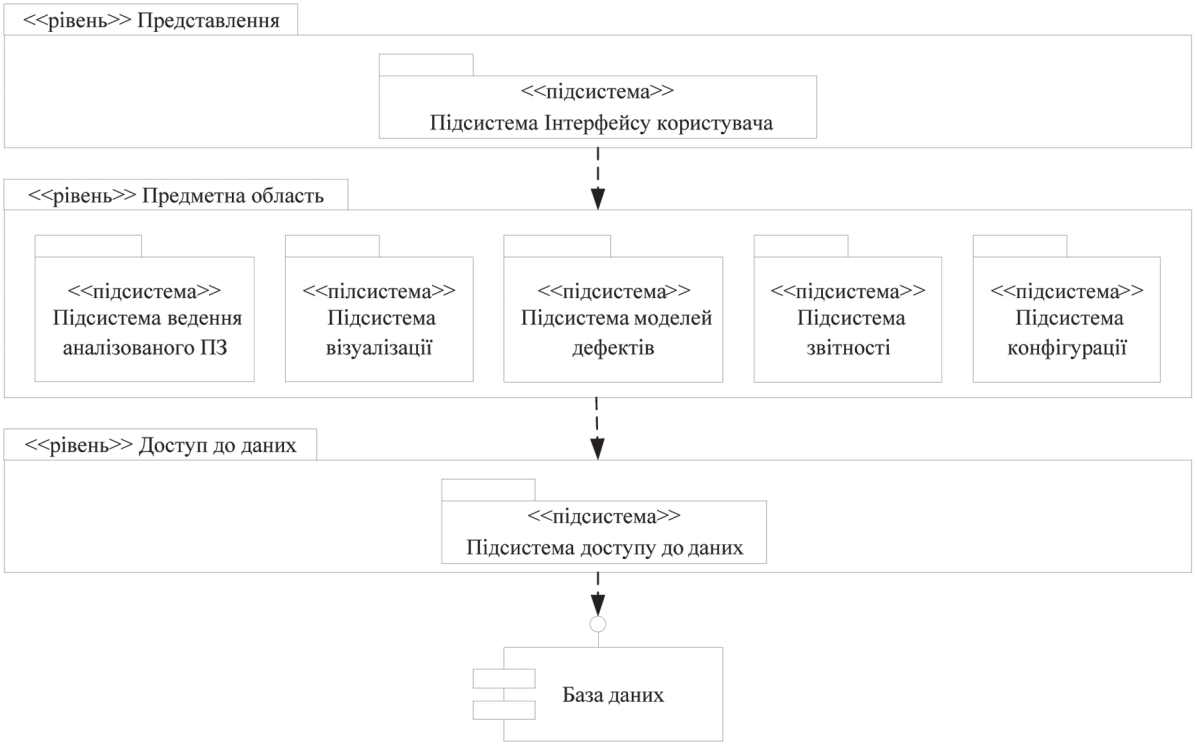
Перш ніж перейти до розгляду засобів діагностики програмного забезпечення, зупинимося на DDHM (мал. 1), на якій ґрунтується їхня робота. Метамодель зображено у вигляді шарів. Вертикальні шари містять сутності, що є екземплярами узагальнених сутностей метамоделі HISMO [12]. Горизонтальні шари містять сутності, які зображують елементи модельованого програмного забезпечення, їхні версії й історії.

Метамодель складається з таких вертикальних шарів:

- шар елементів – містить сутності метамоделі, що представляють елементи конструкції програмного забезпечення, історія яких вивчається;
- шар версій – містить сутності метамоделі, що представляють версії елементів конструкції програмного забезпечення;
- шар історій – містить сутності метамоделей, що представляють історії елементів конструкції програмного забезпечення.

Метамодель складається з таких горизонтальних шарів:

- шар пакетів – містить сутності метамоделей, що представляють пакети, їхні версії й історії;
- шар класів – містить сутності метамоделі, що представляють класи, їхні версії й історії;
- шар методів – містить сутності метамоделі, що представляють методи, їхні версії й історії;
- шар дефектів – містить сутності метамоделі, що представляють дефекти проектування, їхні версії й історії.



Мал. 2. Архітектура засобів діагностики

Функції і режими роботи засобів

Засоби забезпечують моніторинг дефектів проектування елементів конструкції на різних рівнях абстракції в різних аспектах [13] шляхом реалізації наступних функцій:

- ведення моделей програмного забезпечення, що діагностується;
- ведення моделей дефектів проектування;
- вилучення екземпляра DDHM з вихідних кодів програмного забезпечення що діагностується;
- збереження екземпляра DDHM в базі даних;
- побудова візуальних зображень для моніторингу дефектів проектування в різних аспектах;
- взаємодія користувача із зображеннями;
- побудова звітів про проведену діагностику.

Засоби працюють в таких основних режимах:

- накопичення й аналіз даних – аналіз вихідного коду, обчислення метрик, побудова та збереження в базу даних DDHM;
- діагностування – моніторинг користувачем дефектів проектування шляхом вивчення зображення та взаємодії з ними.

Архітектура засобів

Для реалізації перерахованих вище функцій пропонується архітектура. Для забезпечення здатності до перенесення, розширюваності, якостей, необхідних для дослідницьких інструментів, засоби побудовано на архітектурному стилі «багаторівнева архітектура», вперше описаному в роботі [14], яка включає наступні рівні (мал. 2):

- показування – найвищий рівень, який забезпечує взаємодію користувача із засобами в діалоговому режимі, а також ведення і виведення даних з підсистем рівня предметної області;
- предметна область – середній рівень, забезпечує основну функціональність засобів;
- доступ до даних – нижній рівень, забезпечує доступ до даних в базі даних підсистем рівня предметної області. Рівні показування і доступу до даних містять по одній підсистемі (підсистема інтерфейсу користувача та підсистема доступу до даних відповідно).

Рівень предметної містить:

- підсистему ведення програмного забезпечення, що діагностується, забезпечує ведення моделей аналізованих програмних систем, їхніх версій і елементів конструкції;
- підсистему візуалізації, яка забезпечує побудову графічних зображень і функції масштабування, навігації, фокусування;
- підсистему моделей дефектів, яка забезпечує ведення і налаштування моделей дефектів проектування;
- підсистему звітності, що забезпечує налаштування і формування звітів про проведення діагностики програмного забезпечення;
- підсистему конфігурації, що забезпечує ведення установок, які задають режими функціонування засобів.

Вилучені екземпляри DDHM, результати вимірів, налаштування моделей дефектів проектування, конфігураційна та службова інформація зберігається в базі даних.

Оскільки підсистеми інтерфейсу користувача і доступу до даних є власне засобами взаємодії підсистем рівня предметної області з оточенням (користувачі, бази даних), далі у статті вони детально не розглядаються.

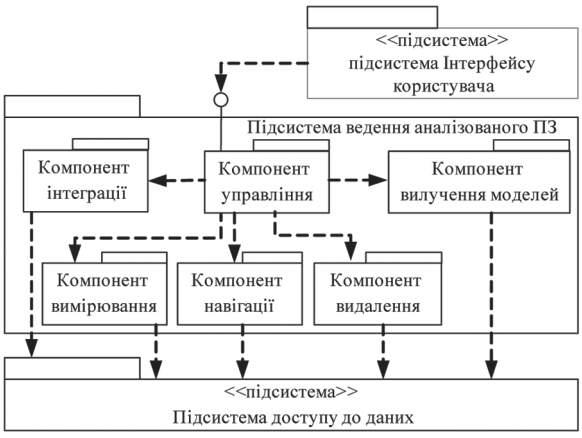
Підсистеми рівня предметної області

Для того, щоб зменшити кількість залежності підсистеми інтерфейсу користувача від компонентів підсистем рівня предметної області до однієї застосовується шаблон проектування «Фасад» [15]. Роль фасаду в кожній підсистемі грає компонент управління, який надає спрощений інтерфейс підсистеми. Компонент управління обробляє запити, що надходять від підсистеми інтерфейсу користувача, і забезпечує спільну роботу компонентів відповідної підсистеми.

Підсистема ведення програмного забезпечення, що діагностується, забезпечує вирішення наступних завдань:

- додавання, зміна, видалення моделей систем програмного забезпечення, що діагностується;
- додавання, змінення, видалення версій систем, що діагностуються програмним забезпеченням;
- перегляд і навігація по елементах конструкції кожної версії систем програмного забезпечення, що діагностується;
- пошук по елементах конструкції кожної версії систем програмного забезпечення, що діагностується.

Завдання вирішуються за допомогою таких компонентів підсистеми (мал. 3): управління, вилучення моделей, інтеграція, вимірювання, видалення, навігація і пошук.



Мал. 3. Структура підсистеми ведення програмного забезпечення

Компонент вилучення моделей приймає на вхід шлях до файлів вихідного коду версії системи програмного забезпечення, що аналізується. Компонент зчитує з вказаного місця файлової системи вихідний код, виконує його аналіз, створює екземпляр метамоделі FAMIX [16]. Як ядро цього компонента використовується інструмент iPlasma [8]. На виході компонент створює файл формату MSE, в якому зберігає отриману модель.

Компонент інтеграції викликається компонентом управління після отримання файлу формату MSE від компонента вилучення моделей. Файл передається на вхід компоненту інтеграції, який для інтеграції моделі версії системи та моделі історії дефектів проектування (екземпляр DDHM) виконує наступні кроки:

- завантаження моделі версії системи програмного забезпечення в базу даних;
- аналіз походження кожної сутності з моделі версії системи шляхом пошуку сутностей, ідентичних даній, в моделі більш ранньої версії системи програмного забезпечення, що діагностується;
- встановлення зв'язку між виявленими ідентичними сутностями, що фіксується в базі даних.

Ідентичними називаються сутності, які є в різні моменти часу двома версіями з однієї й тієї ж історії [12]. Компонент використовує найпростіший метод встановлення ідентичності сутностей, який полягає у використанні їхніх імен. Якщо існують дві сутності одного типу з однаковим ім'ям у двох різних версіях системи програмного забезпечення, що діагностується, то вони розглядаються як дві версії з однієї й тієї ж історії.

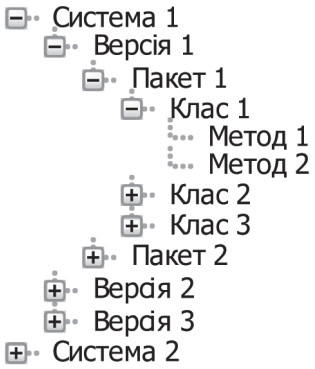
Компонент вимірювань одержує на вхід ідентифікатор системи і виконує вимірювання, використовуючи збережений в базі даних екземпляр DDHM системи і моделі дефектів проектування. Властивості сутностей моделі [13] оновлюються результатами вимірювання та фіксуються в базі даних. Обчислюються метрики елементів конструкції програмного забезпечення їх історії. Прикладами метрик історії є вік елемента конструкції програмного забезпечення, кількість змін елементу, кількість авторів, які вносили зміни в елемент. Використовуючи моделі дефектів проектування, обчислюються властивості дефектів проектування [13, 17]. Компонент викликається компонентом управління після будь-яких змін екземпляра DDHM або моделей дефектів проектування.

Компонент видалення одержує на вхід ідентифікатор версії системи, яку необхідно видалити, і виконує її безпечне видалення з бази даних. Під безпечним видаленням розуміємо таке видалення частини екземпляра DDHM, після якого

екземпляр зберігає відповідність як DDHM, так і історії системи, яку представляє. Для цього компонент видалення використовує компонент інтеграції для відновлення зв'язків, розірваних при видаленні, після чого викликає компонент вимірювань, оскільки екземпляр DDHM був змінений. Компонент видалення виконує видалення всієї моделі системи програмного забезпечення, що діагностуються, але в цьому випадку забезпечувати безпечне видалення немає необхідності, тому що видаляється весь екземпляр DDHM з бази даних.

Компонент навігації надає підсистемі для користувача інтерфейсу абстракцію систем програмного забезпечення, що діагностується, у вигляді дерева для спрощення перегляду. Вузлами дерева є системи, їхні версії й елементи конструкції різного рівня абстракції (мал. 4).

Користуючись деревом, інженер може вивчати склад систем програмного забезпечення, що діагностується. По заданим користувачем ключовим словам виконується пошук необхідних вузлів дерева.



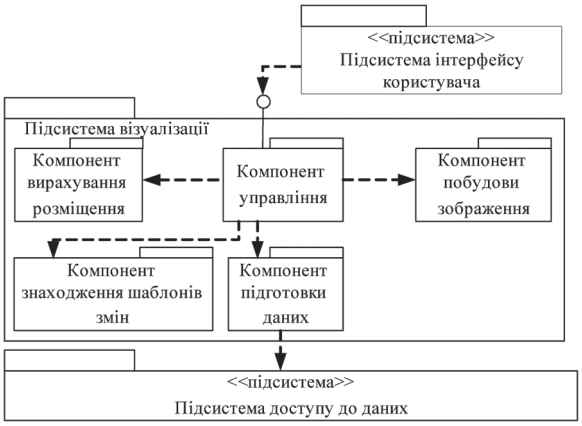
Мал. 4. Дерево систем аналізованого програмного забезпечення

Підсистема візуалізації забезпечує рішення наступних завдань:

– підготовка даних для побудови графічних зображень з урахуванням обмежень на тип і кількість відображених на зображенні дефектів проектування або конструкцій програмного забезпечення;
– виявлення шаблонів зміни дефектів проектування;
– побудова графічних зображень;
– взаємодія користувача із зображеннями.

Підсистема візуалізації забезпечує побудову таких зображень [13]: «Рентгенограма», «Історія дефекту», «Історія ознак дефекту».

Завдання підсистеми вирішуються за допомогою наступних компонентів (мал. 5): управління, підготовку даних, виявлення шаблонів зміни, обчислення розміщення, побудови зображень.



Мал. 5. Структура підсистеми візуалізації

Компонент підготовки даних для побудови графічних зображень одержує на вхід інформацію про тип необхідного зображення і відповідні йому параметри. Використовуючи збережений у базі даних екземпляр DDHM і вхідну інформацію, компонент виконує алгоритми, запропоновані в роботі [13] і створює на виході структури даних, необхідні для побудови того чи іншого графічного зображення.

Компонент виявлення категорій одержує на вхід дані для побудови зображень і виконує їхній аналіз, щоб знайти категорії дефектів проектування, таких як збільшення, зменшення, пульсування або стабільність. Категорії можуть вказати на найбільш проблематичні з точки зору супроводу елементи програмного забезпечення або на місця проведеної реструктуризації. За результатами проведеного аналізу дані оновлюються так, щоб при побудові зображення знайдені категорії дефектів були виділені і передавалися на вихід компонента.

Компонент обчислення розміщення отримує на вхід дані для побудови зображень і визначає координати розташування на екрані для кожного елементу майбутнього зображення.

Компонент побудови зображення отримує на вхід дані для побудови зображень, дані про розміщення елементів зображення на екрані. Потім виконує побудову зображення і виведення його на екран.

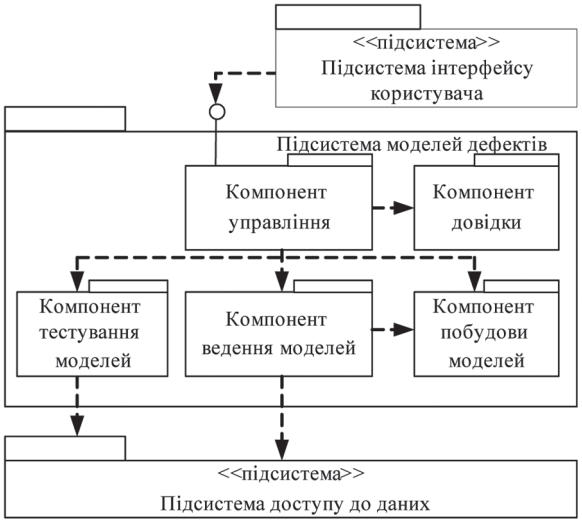
Підсистема моделей дефектів забезпечує вирішення наступних завдань:

– перегляд списку моделей дефектів проектування [17], що зберігаються в базі даних, та їхніх описів;
– встановлення граничних значень метрик, які служать опорною точкою для визначення інтенсивностей прояв ознак;
– додавання, зміна, видалення, пошук і тестування моделей дефектів;
– перегляд списку метрик, які можуть бути отримані за допомогою засобів;

– вибір метрик, за допомогою яких можуть бути оцінені інтенсивності прояву ознак;
– додавання, зміна, видалення ознак дефектів проектування.

Завдання вирішуються за допомогою таких компонентів підсистеми (мал. 6): управління, побудови моделей, ведення моделей, тестування моделей, довідки.

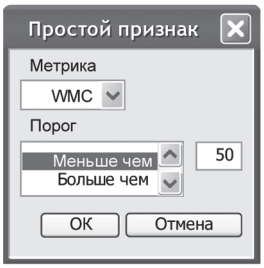
Компонент ведення моделей дефектів отримує на вхід ідентифікатор моделі й ідентифікатор операції, яку необхідно виконати (додати, змінити, видалити, знайти). Якщо компонент запускається без параметрів, то він працює в режимі перегляду моделей, що вже існують у базі даних.



Мал. 6. Структура підсистеми моделей дефектів

Для створення або зміни моделі використовується компонент побудови моделей, що розглядається далі. Після того, як користувач завершує дії з додавання або зміни, компонент побудови моделей виконує збереження змін в базі даних. Видалення моделі компонент виконує самостійно. При отриманні команди пошуку компонент бере на вхід ключове слово і виконує пошук моделей дефектів проектування в базі даних. Після внесення змін в будь-яку модель, компонент викликає компонент вимірювань з підсистеми ведення програмного забезпечення, що діагностується, для збереження несуперечності властивостей сутностей екземпляра DDHM і результатів моделювання дефектів проектування.

Компонент побудови моделі є засобом візуальної побудови і редагування моделей дефектів проектування. Компонент надає користувачеві список і опис метрик, які можна отримати за допомогою засобів. Користувач вводить ознаки дефекту, розбиває їх на більш прості ознаки, поки не отримає набір ознак, інтенсивність яких можна оцінити за допомогою однієї метрики зі списку. Такі ознаки називаються простими. Далі компонент дає можливість зв'язати кожну просту ознаку з метрикою та встановити для неї поріг (мал. 7).



Мал. 7. Редактор простої ознаки

Простіші ознаки за допомогою компонентів об'єднуються в складніші операторами «І» та «АБО». Ознаки дефекту можуть бути так само змінені після створення.

Компонент тестування моделей одержує на вхід ідентифікатор моделі дефекту проектування, яку необхідно протестувати. Далі витягує з бази даних вказану модель і, проаналізувавши та визначивши її необхідні тестові вхідні дані, запитує їх у користувача. Отримавши тестові дані, компонент виконує обчислення значень функцій, що входять в модель дефекту проектування. Обчислені значення компонент передає на вихід.

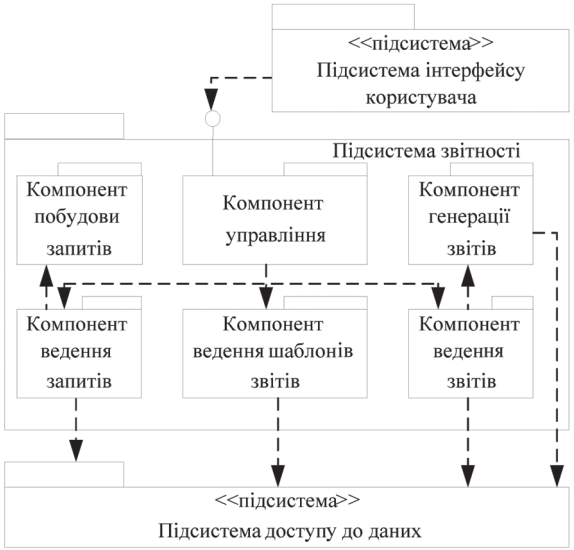
Компонент довідки одержує на вхід ключові слова для пошуку по довідниках або ідентифікатори розділів довідки, які необхідно відобразити. Компонент зберігає і надає користувачеві інформацію про принципи, евристики, кращі практики, шаблони та антишаблони проектування, дефекти об'єктно-орієнтованого проектування, метрики, способи їхнього обчислення, а також джерела, де їх вперше запропонували. Довідники зберігаються у форматі XML.

Підсистема звітності забезпечує рішення наступних завдань:
– додавання, зміна, видалення шаблонів звітів;
– додавання, зміна, видалення звітів;
– додавання, зміна, видалення запитів до даних про результати діагностики;
– побудова запитів;
– генерація звітів;
– перегляд звітів;
– пошук звітів;
– перегляд результатів вимірювань програмного забезпечення, що діагностується.

Завдання вирішуються за допомогою таких компонентів підсистеми (мал. 8): управління, побудова запитів, ведення запитів, ведення шаблонів звітів, ведення звітів, генерація звітів.

Компонент ведення запитів одержує на вхід ідентифікатор запиту та ідентифікатор операції,

яку необхідно виконати (додати, змінити, видалити, знайти). Якщо компонент запускається без параметрів, то він працює в режимі перегляду запитів, вже існуючих в базі даних. Для створення або зміни запиту використовується компонент побудови запитів, що розглядається далі. Після того, як користувач завершує роботу з додавання або зміни з компонентом побудови запитів, виконується збереження змін в базі даних. Видалення запиту компонент виконує самостійно. При отриманні команди пошуку компонент бере на вхід ключове слово і виконує пошук запитів в базі даних за їхніми текстовими описами.



Мал. 8. Структура підсистеми звітності

Компонент побудови запитів забезпечує візуальну побудову та редагування запитів до даних про результати діагностики. Інтерфейс компонента має в своєму розпорядженні область, яка відображає запит в графічному вигляді, і областю, яка містить текст запиту. Надається можливість працювати або в графічній області, або в текстовій області. Забезпечується синхронізація зображень, так що вони завжди відображають поточний стан. Графічні засоби в діалоговому вікні компонента дозволяють будувати запити, використовуючи операції перетягування. Елементи запиту об'єднуються операторами «І» та «АБО», параметри елементів запиту налаштовуються за допомогою полів вибору.

Компонент ведення шаблонів звітів отримує на вхід ідентифікатор шаблону звіту та ідентифікатор операції, яку необхідно виконати (додати, змінити, видалити, знайти). Якщо компонент запускається без параметрів, то він працює в режимі перегляду шаблонів звітів, що вже існують в базі даних. Щоб створити або змінити шаблон використовується текстовий редактор. У тих місцях шаблону, де при генерації звіту будуть вставлятися дані про результати діагностики, зберігаються метадані, повернуті запитами й ідентифікатори запитів. Після того, як користувач завершує роботу з доданням або зміною шаблона звіту в текстовому редакторі, виконується збереження шаблона в базі даних.

Компонент ведення звітів отримує на вхід ідентифікатор звіту та ідентифікатор операції, яку необхідно виконати (додати, змінити, видалити, знайти). Якщо компонент запускається без параметрів, то він працює в режимі перегляду звітів, що вже існують в базі даних. Щоб створити або змінити звіт, використовується компонент генерації звітів, що розглядається далі. Після того, як користувач завершує роботу з додавання або зміни, виконується збереження змін в базі даних. Видалення звіту компонент виконує самостійно. При отриманні команди пошуку компонент бере на вхід ключове слово і виконує пошук звітів у базі даних за їхніми текстовими описами та вмістом.

Компонент генерації звітів отримує на вхід ідентифікатор шаблону звіту, за яким необхідно згенерувати звіт. Далі виконується наступна послідовність дій для створення звіту:

- вилучення відповідного ідентифікатору звіту з бази даних;
- виконання всіх запитів, пов'язаних із шаблоном;
- підстановку результатів виконання запитів в шаблони, в результаті чого створюється звіт;
- створений звіт зберігається в базі даних як документ і може бути роздрукований чи переданий через мережу.

Підсистема конфігурації забезпечує рішення наступних завдань:

- перегляд і пошук налаштувань засобів;
- зміну параметрів засобів;
- збереження та відновлення налаштувань засобів.

Завдання вирішуються за допомогою таких компонентів підсистеми (мал. 9): управління, ведення налаштувань, збереження та відновлення налаштувань, довідки.

Компонент ведення налаштувань забезпечує перегляд налаштувань в діалоговому режимі, їхню зміну і збереження в базі даних. Компонент збереження і відновлення налаштувань дозволяє користувачеві зберігати поточні налаштування засобів і за необхідності відновляти їх.

Компонент пошуку та довідки забезпечує підтримку користувача при конфігурації засобів, шляхом надання довідки про кожному з налаштувань і можливості виконувати пошук за довідкою і за існуючими налаштуваннями.

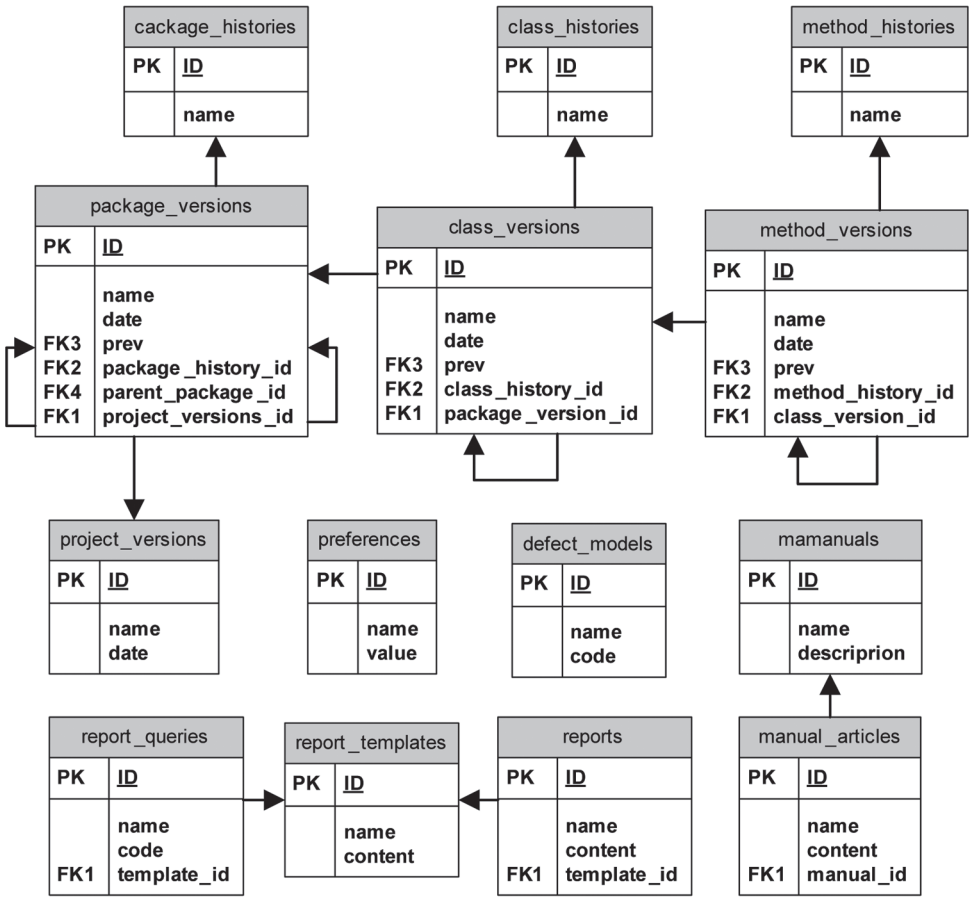
Мал. 9. Структура підсистеми конфігурації

База даних

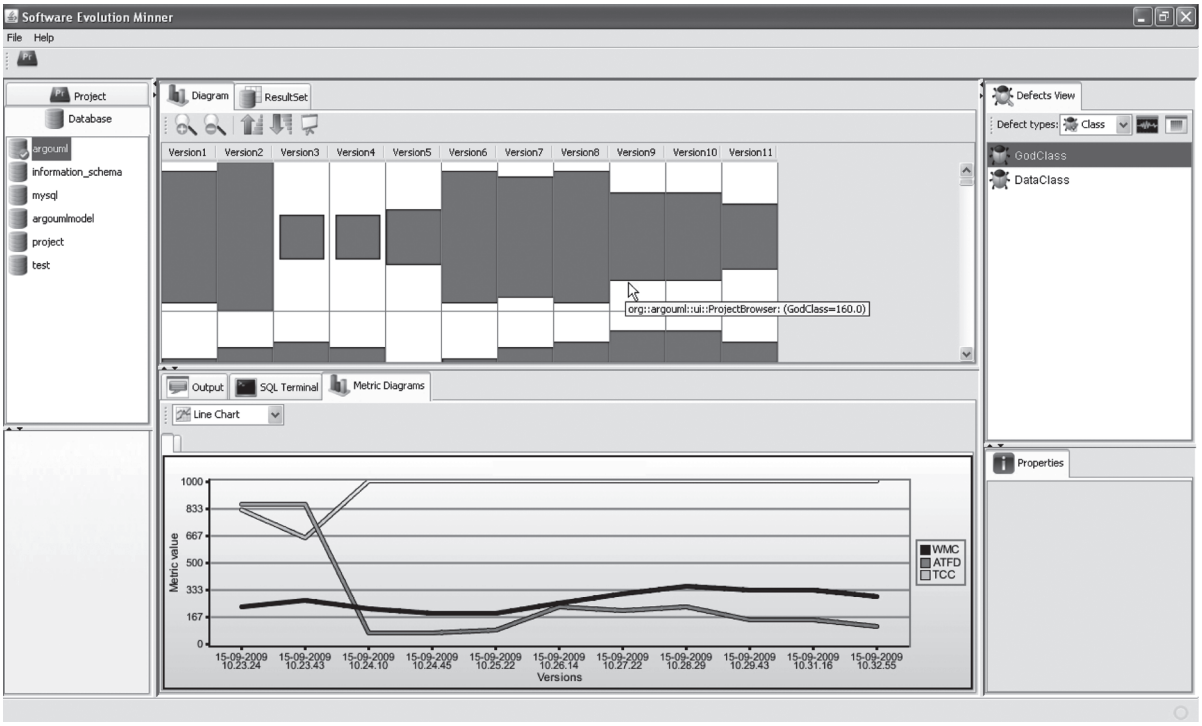
База даних призначена для зберігання екземплярів DDHM, моделей дефектів проектування, шаблонів та звітів з діагностики, налаштувань і службової інформації.

Схема бази даних представлена наступними сутностям (мал. 10):

- елементи екземплярів DDHM – сутності, що описуються DDHM – версії систем програмного забезпечення, що діагностується (project_versions), пакетів (package_versions), класів (class_versions), методів (method_versions), історії пакетів (package_histories), класів (class_histories), методів (method_histories);
- моделі дефектів проектування, визначені користувачем (defect_models);
- шаблони звітів – заготовки, на основі яких будуються звіти про діагностику (report_templates);
- запити до результатів діагностики, при виконанні яких засоби генерують звіти (report_queries);
- звіти про проведену діагностику (reports);
- налаштування засобів (preferences);
- довідники для допомоги користувачеві при веденні моделей дефектів проектування, використання засобів та їхнього налаштування (manuals);
- статті довідників (manual_articles) .



Мал. 10. Схема бази даних



Мал. 11. Знімок засобів інтерфейсів користувача

Застосування засобів

Продемонструємо роботу засобів шляхом моніторингу дефектів проектування «God Class» [18] програмного забезпечення ArgoUML (http://argouml.tigris.org/). ArgoUML – інструмент моделювання, що підтримує стандарт UML 1.4, з відкритими вихідними кодами, написаний мовою програмування Java. Технічні характеристики інструменту, отримані за допомогою розглянутих засобів, показані в табл. 1.

В успадкованому програмному забезпеченні дефектами «God Class» вражені класи, що концентрують значну частину системної логіки в своїх методах, використовуючи інші класи як постачальників даних, та мають слабке зчеплення.

Таблиця 1. **Характеристики ArgoUML**

Характеристика	Значення
Кількість рядків коду	136,000
Кількість методів/інтерфейсів	14,221
Кількість класів	2,522
Кількість пакетів	143

На екрані засобів, що відображають ділянку зображення «Історія дефекту» (мал. 11), показано один із найнебезпечніших дефектів «God Class» в ArgoUML, яким вражений клас org.argouml.ul.ProjectBrowser. Моніторинг показує, що ступінь розвитку дефекту пульсує, що свідчить, по-перше, про те, що клас з цим дефектом не стабільний і на його зміни витрачаються ресурси при підготовці 80 % версій, по-друге, робляться спроби реструктуризації з метою усунення дефекту, однак дефект з'являється повторно. Червоний колір дефекту говорить про те, що значення всіх метрик, які використовуються для оцінки інтенсивності ознак дефекту, перевищили гранично допустимий поріг.

Висновок

Одним із завдань супроводу є моніторинг розвитку дефектів проектування. Для її вирішення запропоновано метод діагностики програмного забезпечення [13].

Для реалізації методу розроблено засоби діагностики об'єктно-орієнтованого програмного забезпечення SEM (Software Evolution Miner) [19]. Засоби використовуються в Національному авіаційному університеті у навчальному процесі при викладанні дисципліни «Архітектура та проектування програмного забезпечення» для студентів напряму «Програмна інженерія».

1. Erlikh L. Leveraging legacy system dollars for E-business / L. Erlikh // (IEEE) IT Pro. – 2000. – Vol. 2, No. 3. – P. 17–23.
2. Сидоров Н. А. Восстановление, повторное использование, переработка программного обеспечения / Н. А. Сидоров // УСИМ. – 1998. – №3. – С. 74–84.
3. Eick S. Does Code Decay? Assessing the Evidence from Change Management Data / Stephen G. Eick, Todd L. Graves, Alan F. Karr, J. S. Marron, Audris Mockus // IEEE Transactions on Software Engineering. – 2001. – Vol. 27, No. 1. – P. 1–12.

4. Kim S. Which warnings should I fix first? / Sunghun Kim, Michael D. Ernst // Proceedings of the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering (ESEC/FSE'07). – New York : ACM, 2007. – P. 45–54.
5. Mantyla M. Bad Smells – Humans as Code Critics / Mika V. Mantyla, Jari Vanhanen, Casper Lassenius // Proceedings of the 20th IEEE International Conference on Software Maintenance (ICSM'04). – Washington : IEEE Computer Society, 2004. – P. 399–408.
6. Нечай О. С. Методи та засоби виявлення дефектів проектування об'єктно-орієнтованого програмного забезпечення / О. С. Нечай, М. О. Сидоров // Вісник НАУ. – 2009. – № 3. – С. 200–205.
7. Hovemeyer D. Finding bugs is easy / David Hovemeyer, William Pugh // ACM SIGPLAN Notices. – 2004. – Vol. 39, No. 12. – P. 92–106.
8. Lanza M. Object-Oriented Metrics in Practice / Michele Lanza, Radu Marinescu. – Berlin : Springer-Verlag, 2006. – 206 p.
9. Moha N. Détection et correction des défauts dans les systèmes orientés objet : Ph.D thesis / Naouel Moha. – Université de Montréal – Université des Sciences et Technologies de Lille, 2008. – 155 p.
10. Lanza M. CodeCrawler – Lessons Learned in Building a Software Visualization Tool / Michele Lanza // Proceedings of the Seventh European Conference on Software Maintenance and Reengineering (CSMR'2003). – Washington : IEEE Computer Society, 2003. – P. 409–418.
11. Wettel R. Visually Localizing Design Problems with Disharmony Maps / Richard Wettel, Michele Lanza // Proceedings of 4th International ACM Symposium on Software (Visualization Softvis'2008). – New York : ACM Press, 2008. – P. 155–164.
12. Gîrba T. Modeling history to analyze software evolution: Research Articles / Tudor Gîrba, Stéphane Ducasse // Journal of Software Maintenance and Evolution: Research and Practice. – 2006. – Vol. 18, No. 3. – P. 207–236.
13. Нечай О. С. Метод діагностики об'єктно-орієнтованого програмного забезпечення / О. С. Нечай // Вісник НАУ. – 2009. – № 5. – С. 100–111.
14. Dijkstra E. The structure of the «THE»-multiprogramming system / Edsger W. Dijkstra / Communications of the ACM. – 1968. – Vol. 11, No. 5. – P. 341–346.
15. Гамма Э. Приемы объектно-ориентированного проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес ; [пер. с англ. А. Слинкин]. – СПб : Питер, 2001. – 368 с.
16. Tichelaar S. FAMIX and XMI / Sander Tichelaar, Stéphane Ducasse, Serge Demeyer // Proceedings of the Seventh Working Conference on Reverse Engineering (WCRE'00). – Washington : IEEE Computer Society, 2000. – P. 296-306.
17. Нечай О. С. Метод побудови моделей дефектів проектування об'єктно-орієнтованого програмного забезпечення / О. С. Нечай, М. О. Сидоров // Наукоємні технології. – 2009. – № 2. – С. 58–64.
18. Reil A. J. Object-Oriented Design Heuristics / A. J. Reil. – Addison Wesley. – 1996. – 400 p.
19. А. с. Комп'ютерна програма «Software Evolution Miner» («SEM») / О. С. Нечай (Україна). – № 29953 ; заявл. 19.06.09 ; опубл. 19.08.09.

A. Nechay

TOOLS FOR OBJECT-ORIENTED SOFTWARE DIAGNOSTIC

This paper presents the tool implementation of object-oriented software diagnostics method, main point of which is design flaws monitoring. Tool's functions and operation modes are defined; architecture and its components are discussed.

Keywords: software diagnostics, design flaws, software architecture, software visualization tools, software quality.