Секція: Технічні науки

Булах Богдан Вікторович
кандидат технічних наук,
доцент кафедри системного проектування
Національний технічний університет України
"Київський політехнічний інститут імені Ігоря Сікорського"
м. Київ, Україна

Лашко Олена Вікторівна
викладач кафедри приладів та систем неруйнівного контролю
Національний технічний університет України
"Київський політехнічний інститут імені Ігоря Сікорського"
м. Київ, Україна

АНАЛІЗ ТА РЕФАКТОРИНГ ПРОГРАМНОГО КОДУ ДЛЯ ПОТРЕБ ВИКЛАДАЧА ПРОГРАМУВАННЯ

Аналіз та рефакторинг програмного коду - не лише суто "промислова" задача, що постає перед професійними розробниками програмного забезпечення. Застосування подібних технологій може бути виправданим і у навчальному процесі. Засоби аналізу програмного коду слугуватимуть у цьому випадку не задачі контролю якості програмного продукту, а будуть інструментом для оцінювання якості засвоєння студентами матеріалу та оцінки їх компетентності. В той же час рефакторинг ("оптимізація" коду для підвищення якості продукту) може бути зручним засобом для швидкого виправлення допущених студентом помилок. Окрема задача для інструментів аналізу коду - виявлення плагіату у програмних кодах студентів.

Мабуть, перший і найпростіший вибір викладача - розвинуте інтегроване середовище розробки (англ. IDE). Нині більшість інтегрованих

середовищ розробки (Eclipse, Visual Studio, IntelliJ IDEA та багато інших, включаючи хмарні середовища з веб-інтерфейсом), та і просто редакторів коду пропонують певний набір інструментів для аналізу коду та здійснення його рефакторингу. Причому для викладача корисними можуть бути як прості, не прив'язані до мови програмування засоби пошуку та заміни фрагментів у коді (за допомогою регулярних виразів), так і засоби, що використовують особливості конкретної мови програмування (такі як швидкий пошук коду методу класу прямо у місці його виклику - в об'єктно-орієнтованих мовах). Наприклад, це дозволить викладачу швидше зорієнтуватись у часто дуже недосконалому та заплутаному коді студентів та провести швидке коригування коду (скажімо, приведення стилю написання коду до вимог певної конвенції), а також дозволить швидше відшукати подібні фрагменти у роботах різних студентів.

Однак цих базових інструментів часто недостатньо для нетривіальних операцій з виявлення складних конструкцій коду (певних "шаблонів"). Причина в тому, що пошук з використанням регулярних виразів не враховує семантики мови (результати пошуку потребують подальшого уточнення), а семантично-орієнтовані засоби IDE досить обмежені. Тобто, існує проблема "автоматизації процесу розуміння смислу коду". Для вирішення цієї проблеми пропонується скористатися технологіями семантичного веб. Опис семантики коду для його аналізу сам по собі не є новим підходом, й існує ряд публікацій на цю тематику [1,2]. Вже пропонувалося застосувати ці технології для аналізу програмного коду в промисловій розробці програмного забезпечення [3]. Метою даної публікації є синтез підходу, що, після певної підготовки, дозволить спростити роботу викладача програмування щодо оцінювання коду та виявлення плагіату в студентських роботах.

База знань шаблонів коду. Типовий набір засобів для реалізації семантичних програмних систем включає:

- опис онтологій (понять певної предметної області та їх взаємозв'язків) на мовах родини OWL2,
- зберігання аксіом та фактів у вигляді триплетів "суб'єкт - предикат - об'єкт" у форматі RDF,
- взаємодія з базою знань за посередництвом мови SPARQL,
- використання логічних правил на мові SWRL для доповнення онтологій,
- використання OWL/SWRL-різонерів (програм логічного виведення фактів з онтологій) для пошуку усіх можливих прихованих взаємозв'язків для усіх сутностей в онтології також є частиною подібних програм.

Для вирішення проблеми пошуку фрагментів коду за їх абстрактним описом через певні поняття та їх зв'язки слід, відповідно, побудувати онтологію для предметної області конкретної мови програмування [3]. Так для об'єктно-орієнтованих мов слід визначити поняття класу, підкласу, об'єкту, області видимості, оператору та складеного оператору, інтерфейсу, методу, поля, наслідування та багато іншого для того, щоб мати змогу описати код та шукати у ньому фрагменти, описані на високому рівні абстракції.

Шаблони у навчальних кодах. Розглянемо лише деякі приклади шаблонів чи фрагментів у студентських кодах, що потребують уваги викладача об'єктно-орієнтованого програмування, і можуть бути реалізовані з використанням бази знань.

1. *Консольне введення-виведення у класах логіки*. Часто студенти нехтують принципом розділення бізнес-логіки та організації інтерфейсу з користувачем, що ускладнює повторне використання класів бізнес-логіки. Шаблон полягає у наявності викликів методів роботи з консоллю всередині методів окремих класів.

2. *Глобальні змінні*. Часто студенти зловживають їх використанням, посилюючи зв'язність програми, і, автоматично, ускладнюючи собі та іншим пошук помилок у ній. Шаблон: оголошення змінних поза класом (у тих мовах, де це дозволено).

3. Занадто "довгі", *переобтяжені класи та методи*. Студенти часто мають проблеми з об'єктною декомпозицією задач, тобто мінімізують кількість класів, створюючи "монстр-класи", відповідальні за багато різних задач одночасно. У класах же часто мають місце занадто довгі методи, що важко читати (особливо, якщо студенти попередньо вчили процедурне програмування). Шаблон: кількість рядків коду всередині методів та кількість методів у класі більше певної межі.

4. Дотримання *конвенцій написання коду*. Тут семантичне навантаження мінімальне, але теж є, наприклад у випадках, коли назви методів та класів починаються то з великої, то з малої літери, або коли для закритих членів класу прийнята окрема конвенція щодо іменування, а студенти цим нехтують. Звичайно, дотримання конвенцій - не головне в навчанні програмуванню, але воно дисциплінує студентів та спрощує їх перехід до професійної діяльності.

5. *Порушення інкапсуляції*. Опосередковано на це вказує наявність великої кількості відкритих полів класу, особливо, якщо ці поля використовуються у внутрішній логіці класу.

6. *Наявність поліморфізму*. Один з відносно складних прикладів: необхідно упевнитись, що студент реалізував поліморфну поведінку. Шаблон: ієрархія наслідування класів, перевизначення віртуальних методів, виклик цих методів, посилаючись на базовий клас.

7. Наявність чи коректність застосованих *патернів проектування* ("одинак", "фабрика", "стратегія" тощо) .

Звичайно, результати пошуку таких шаблонів потребують подальшої експертної оцінки викладача, але викладачу набагато легше працювати вже

з підготовленими фактами, які було автоматично виявлено. Таким чином, поруч з тривіальним автоматичним виявленням синтаксичних помилок та нетривіальною перевіркою на плагіат (з використанням поширених шаблонів), пошук подібних фрагментів у кодах студентів буде корисним доповненням до інструментарію викладача.

Склад семантичного інструментарію аналізу коду. Загальна програмна інфраструктура для аналізу коду пропонується на основі [3]. Код проходить етап синтаксичного розбору, результатом якого є абстрактне дерево синтаксису (AST, за допомогою генератору аналізаторів формальних мов ANTLR). Далі модуль "генератору знань" приймає на вхід AST та описує усі вжиті елементи мови та відношення між ними як факти у формі триплетів з урахуванням понять OWL-онтології для конкретної мови програмування. Пошук шаблонів відбувається як пошук пов'язаних фактів у базі знань.

Література

1. Mattia Atzeni and Maurizio Atzori. CodeOntology: RDF-ization of Source Code. The semantic web - ISWC 2017 : 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings. Part II. P. 20-28.
2. Tappolet, Jonas & Kiefer, Christoph & Bernstein, Abraham. Semantic web enabled software analysis. Web Semantics: Science, Services and Agents on the World Wide Web. 8.2010. P. 225-240.
3. Булах Б. В. Застосування семантичних технологій для аналізу та рефакторингу програмного коду // Міжнародний науковий журнал "Інтернаука". №12. 2018.