Улучшение структуры и быстродействия модуля поиска заимствований в системе проверки учебных работ по программированию

Н. Д. Козулин, email: 301nikotrin301@gmail.com[1]

[1] Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский авиационный институт (национальный исследовательский университет)»

Аннотация. *Статья посвящена модернизации модуля поиска заимствований в практических заданиях студентов по программированию. В статье описан используемый в данном модуле подход обработки кодов программ, а также описаны реализованные изменения по ускорению его работы и внедрению возможностей расширения для дальнейшего развития.*

Ключевые слова: *статический анализ исходного кода программы, обнаружение заимствований.*

Введение

В процессе обучения программированию выдается множество практических заданий, требующих не только корректности решения, но и его оригинальности, которые обеспечивают проверку освоения предоставляемого материала. Все это ведет к появлению достаточно больших объемов работы для преподавателей. В рамках автоматизации выполнения данной задачи тестирование правильности выполнения программ частично возможно реализовать, в то время как задача определения наличия заимствований представляет из себя более комплексную проблему. Хоть системы антиплагиата в целом уже прошли огромный путь в своем развитии, основной их направленностью выступает поиск заимствований в текстах, написанных на естественных языках (ЕЯ). Подход, используемый в сервисах поиска плагиата в текстах на ЕЯ, не применим к анализу кода программ, поскольку он не учитывает специфики языков программирования, предоставляющих обширные возможности, которые могут быть использованы при сокрытии чужого кода в своей работе, что приводит к необходимости применения специализированных средств поиска заимствований.

На кафедре 319 МАИ разработан и используется комплекс инструментов [1] для автоматизированной проверки связанных с

программированием работ студентов. В этот комплекс уже внедрен модуль обнаружения заимствований [2], направленный на разбор работ, относящихся к стандартным учебным задачам на языке программирования Java. Одним из недостатков модуля является его относительная невысокая скорость работы. Также поскольку практическими заданиями по программированию учебный процесс не ограничивается, возникает потребность как в расширении данного модуля, например, адаптации его к смежным дисциплинам через внедрение поддержки других языков программирования, так и дополнении его возможностями, например, сравнения работ с источниками из сети Интернет. Для этого необходимо улучшить внутреннюю структуру модуля с целью упрощения внесения в него дальнейших изменений и повышения производительности с сохранением требуемого качества работы.

Описание используемого подхода в анализе работ

В основе работы ранее разработанного модуля поиска заимствований лежит лексический подход, суть которого состоит в преобразовании исходного кода программы в аналогичную ему последовательность строк, состоящих из токенов – лексем, соответствующих элементам и конструкциям использованного языка программирования. Сам же процесс поиска заимствований в программах представлен в модуле в виде следующих шагов:

1. Подготовительный этап, заключающийся в очистке или замене элементов исходного кода программ способных помешать преобразованию его в последовательность токенов и анализу структуры программы.
2. Токенизация – процесс преобразование исходного кода программы, проводимый на основе корневого блока с подстановкой токенизированных представлений методов после инструкции их вызова, в представление для осуществления сравнения.
3. Сравнение – определение доли уникальности между исходными кодами программ, основывающийся на редакционном расстоянии Левенштейна [3], определяемом при помощи алгоритма Вагнера-Фишера [4].

В ходе применения старой версии модуля выяснилось, что с учетом планируемого расширения количества данных его скорость работы недостаточна, особенно, если учитывать в проверках решения прошлых лет. В рамках улучшения производительности и возможностей по расширению модуля предлагается внедрение механизма построения абстрактного синтаксического дерева (АСД) для работы с элементами

кода программы в виде отдельных узлов, а не последовательно. В рамках реализации данного подхода были рассмотрены следующие библиотеки, позволяющие создавать АСД:

- Compiler Tree API [5]
- JavaParser [6]
- Eclipse JDT API [7]
- Spoon [8]

Сравнение библиотек построения АСД

Для тестирования были использованы 973 работы студентов, полученные в рамках одного учебного семестра и распределенные по 56 учебным задачам. Результаты тестирования производительности разбора работ данными библиотеками представлены в таблице 1.

Таблица 1

Время работы библиотек построения АСД

Библиотека	Время разбора, мс
Compiler Tree API	275984
JavaParser	3064
Eclipse JDT API	2577
Spoon	31872

На основе приведенного тестирования (табл. 1) и анализа работы была выбрана библиотека JavaParser. Ключевыми определяющими критериями выбора JavaParser являются:

- Отсутствие нормальной документации у Compiler Tree API из чего следует невозможность получения АСД отдельно от компиляции всей программы.
- Медленная скорость работы и отсутствие возможности прохода по дереву, основанной на шаблоне «Посетитель», у библиотеки Spoon.
- Простое использование и большое разнообразие классов-посетителей у JavaParser для прохода по дереву в сравнении Eclipse JDT API, а также возможность анализа только части выражения, представленного в виде строки.

В дополнение к построению АСД у JavaParser имеется встроенная библиотека JavaSymbolSolver, позволяющая определять тип выражения и определять необходимые для вызова методы на основе переданных в них выражений в виде аргументов на основе пользовательских и встроенных в Java типов, а также на основе типов данных из подключаемых библиотек.

Изменение подготовительного этапа и этапа токенизации

С использованием JavaParser подготовительный этап работы инструмента был модифицирован следующим образом:

- Процесс очистки полностью убран из-за его наличия в JavaParser, а следовательно, в новой версии не производится предварительное редактирование исходного кода программ.
- Непосредственно анализ структуры программы осуществляется при помощи обхода АСД на основе паттерна «Посетитель», что позволяет разделить алгоритмы обработки по методам класса-посетителя принимающие соответствующие элементы языка программирования.

Этап токенизации был в значительной степени изменен, что позволило разделить обработку непосредственно выражений по отдельным методам. Одно из ключевых изменений является построение иерархии классов-токенизаторов, позволяющей определить базовую реализацию токенизации, при которой в представление подставляется соответствующий токен элемента выражения без дополнительной обработки. Процесс токенизации может быть переопределен дочерними классами с перезаписью только ряда ключевых методов для настройки под конкретный ряд задач.

Сам по себе JavaParser не отслеживает присвоения переменным объектов различных типов из-за чего он не может разрешать динамический полиморфизм. Разработанный класс-токенизатор с обобщением представления всех идентификаторов, заменяет переменные в выражениях-аргументах методов объектами типов, содержащиеся в них в действительности на момент выполнения данной инструкции, при помощи специального класса-посетителя.

Сравнение производительности старой и новой версий

Старая версия модуля обнаружения заимствований и новая были протестированы на работах студентов для выявления разницы их производительности, результаты чего представлены в таблице 2.

Таблица 2

Сравнение производительности старой и новой версий модуля

Версия	**Подготовительный этап, мс**	**Токенизация, мс**	**Итог, мс**
Старая	438200	27972	466172
Новая	118928	79045	197973

На основе результатов тестирования (табл. 2) видно, что подготовительный этап ускорился примерно в 3,7 раз, а токенизация замедлилась в 2,8 раз, что в итоге дает общее ускорение работы

программы примерно в 2,4 раза. Также ручная проверка работы модуля показала, что новая версия не снижает качество непосредственного сравнения программ.

Заключение

В рамках модернизации модуля поиска заимствований были проанализированы библиотеки построения АСД на основе исходных кодов программ, написанных на языке Java. Также было произведено их сравнение, на основе которого выбрана библиотека для использованная.

Применение библиотеки по построению АСД на основе исходного кода позволила разделить обработку элементов программы по отдельным методам, что делает структуру модуля более читаемой и гибкой, а также позволило увеличить скорость работы инструмента целиком. Структура модуля обнаружения заимствований также стала готова к дальнейшему расширению.

Список литературы

1. Данилова, И. И. Применение автоматизированных тестов и инструментов статического анализа в информационной системе проверки программ в рамках обучения программированию / И. И. Данилова, С. А. Полицын // Информатика: проблемы, методология, технологии : Сборник материалов XIX международной научно-методической конференции, Воронеж, 14-15 февраля 2019 года / Под ред. Д.Н. Борисова. – Воронеж: Издательство «Научно-исследовательские публикации» (ООО «Вэлборн»), 2019. – С. 921-926.
2. Козулин, Н. Д. Проектирование системы обнаружения заимствований в программах студентов / Н. Д. Козулин // Гагаринские чтения – 2022 : Сборник тезисов работ международной молодёжной научной конференции XLVIII, Москва, 12-15 апреля 2022 года. – Москва: Издательство "Перо", 2022. – С. 253-254.
3. Левенштейн, В. И. Двоичные коды с исправлением выпадений, вставок и замещений символов / В. И. Левенштейн // Доклады Академий Наук СССР. – 1965. – Т. 163. – №4. – С. 845-848.
4. Wagner, R. A. The String-to-String Correction Problem / R A. Wagner, J. F. Michael // Journal of the ACM. – 1974. – № 21. – С. 168-173.
5. Compiler Tree API [Электронный ресурс] : официальный сайт программного продукта. – Режим доступа : https://docs.oracle.com/javase/8/docs/jdk/api/javac/tree/
6. JavaParser [Электронный ресурс] : официальный сайт программного продукта. – Режим доступа : https://javaparser.org/
7. Eclipse JDT [Электронный ресурс] : официальный сайт программного продукта. – Режим доступа : https://www.eclipse.org/jdt/

8. Spoon [Электронный ресурс] : официальный сайт программного продукта. – Режим доступа : https://spoon.gforge.inria.fr/index.html