2. ТЕХНОЛОГИИ КОМПЬЮТЕРНОГО МОДЕЛИРОВАНИЯ

Ю.А. Кева[1], Д.Е. Конева[1], И.Г. Захарова[12]

[1] Тюменский государственный университет, г. Тюмень

[2] Научно-технический университет «Сириус», г. Сочи

УДК 004.423

ИССЛЕДОВАНИЕ И ПРОГРАММНАЯ РЕАЛИЗАЦИЯ МЕТОДА ОПРЕДЕЛЕНИЯ ПАТТЕРНОВ ПРОЕКТИРОВАНИЯ ДЛЯ ИСХОДНОГО КОДА НА ЯЗЫКЕ JAVA

Аннотация. В статье приводится сравнение и реализация методов распознавания паттернов проектирования в исходном коде на языке Java.

Ключевые слова: паттерны проектирования, исходный код, объектно-ориентированный язык программирования, распознавание паттернов проектирования, извлечение признаков.

Введение

В настоящее время IT-специалисты становятся очень востребованными, и зачастую они работают в группах из нескольких человек. Различные методики организации рабочего процесса позволяют им работать слаженно и наиболее эффективно взаимодействовать друг с другом, однако не только это влияет на качество кода, который они пишут. Кроме знания языков программирования, также важно уметь писать код так, чтобы он был понятен другим людям в команде разработчиков.

Работодатель заинтересован в поиске IT-специалистов, которые смогут писать код, понятный уже работающим программистам. Для этого они не только должны знать на определенном уровне те или иные языки программирования, но и писать код в понятном для других стиле. В данной работе под стилем понимается использование паттернов проектирования.

Таким образом, возникает проблема поиска IT-специалистов, которые пишут программный код с использованием паттернов проектирования.

В данный момент эта проблема решается проведением собеседования. IT-специалиста опрашивают по теории, а также он выполняет тестовое задание, которое позволяет выяснить, как он пишет программный код. Однако программист может не знать про паттерны проектирования, но это не помешает ему использовать их в своем коде.

Идея работы состоит в том, чтобы на основе анализа кода выявить паттерны, которые в нем присутствуют.

Цель – разработать программный продукт для распознавания паттернов проектирования в исходном коде, написанном на объектно-ориентированном языке.

При решении данной проблемы возникают следующие задачи:

1. Изучить подходы к выявлению определенных паттернов в программном коде и методы описания паттернов на основе системы признаков.
2. Реализовать парсер для извлечения признаков, описывающих паттерны, и структуру данных для их хранения.
3. Разработать алгоритмы для сопоставления паттернов и программного кода.
4. Разработать и протестировать программный продукт для определения паттернов проектирования в исходном коде.

В настоящее время ведется активное исследование по распознаванию паттернов проектирования в исходном коде. Они применяются во многих проектах для решения задач проектирования. Понимание того, что конкретный паттерн реализуется в том или ином программном модуле, может помочь в понимании программы, улучшить обслуживание программного обеспечения. Паттерны можно найти, глядя на исходный код, однако сложность проектов и различия в стилях написания кода у разработчиков затрудняют поиск.

Автоматическое обнаружение паттернов проектирования поможет разработчикам программного обеспечения быстро и правильно понимать и поддерживать незнакомый исходный код, что приведет к повышению производительности труда разработчиков.

Для решения задачи распознавания паттернов проектирования в исходном коде могут применяться как правила, основанные на признаках тех или иных паттернов, так и методы машинного обучения.

Обзор

Паттерны проектирования – это описание взаимодействия объектов и классов, адаптированных для решения общей задачи проектирования в конкретном контексте. Паттерн проектирования именует, абстрагирует и идентифицирует ключевые аспекты структуры общего решения, которые и позволяют применить его для создания повторно используемого дизайна [6].

Были рассмотрены статьи с похожей тематикой и предложенные в них методы распознавания паттернов. Для решения задачи применяются распознавание по правилам или методы машинного обучения.

В статье [1] с помощью парсера на основе исходного кода строится AST-дерево, после чего применяется статический анализ. Статический анализ – поиск зависимостей в полученном дереве, соответствующих определенному паттерну проектирования, то есть происходит распознавание по определенным признакам, соответствующим каждому паттерну проектирования.

В статье [3] проводится сравнение основных методов, описаны их преимущества и недостатки. В методах, использующих статический анализ, поиск паттернов осуществляется с помощью анализа структуры классов и проверяется ее соответствие определенным условиям. Если структура паттерна хоть немного отличается от заявленных жесткий условий, то он не будет найден, поэтому обычно выполняется либо несколько фаз статического анализа, либо он используется вместе с динамическим анализом и методами машинного обучения.

Также была рассмотрена статья [4], в которой наиболее подробно описана предварительная обработка исходного программного кода и описаны выделяемые в программе признаки для распознавания паттернов. В этой статье для поиска паттернов используется машинное обучение, а признаки паттернов представляются в виде текста на естественном языке.

Постановка решаемой задачи

Под признаками программы понимается множество *P*, которое состоит из элементов, представляющих собой описания классов.

$P = \{C_1, C_2, \dots, C_N\}$, где C_i – описание класса, N – количество классов в программе.

Описание классов *C* включает в себя описания полей, методов и наследуемых классов.

$C = CI \cup F \cup M$, где

- $F\ (Field) = \{F_1, F_2, \dots, F_M\}$, где F_i – описание полей класса, M – количество полей в классе.
- $M\ (Method) = \{M_1, M_2, \dots, M_L\}$, где M_i – описание методов классов, *L* – количество методов в классе.
- $CI\ (Class\ Inheritance) = \{CI_1, CI_2, \dots, CI_K\}$, где CI_i – описание наследуемого класса, *K* – количество наследуемых классов, *CI* – подмножество *P*.

Описание поля класса содержит имя, атрибуты и тип значения.

$F_i = (F_{i1}, F_{i2}, \dots, F_{ix})$, $x =$ 1 (имя) + количество атрибутов поля + 1 (тип значения).

Сигнатура метода может быть описана, как имя метода, атрибуты, тип возвращаемого значения и входные параметры.

$M_i = \left(M_{i1}, M_{i2}, \dots, M_{iy}\right), y =$ 1 (имя) + количество атрибутов метода + 1 (тип возвращаемого значения) + количество параметров.

В разных объектно-ориентированных языках программирования может быть различное число атрибутов, зависящее от выбора того или иного языка, причем они могут присутствовать не в любых сочетаниях.

Паттерн – набор признаков *Pp (Program pattern)*.

$Pp = \{C_1, C_2, \dots, C_{Np}\}$, C_i – описание класса,

$C = CI \cup F \cup M$, где *F* – описание полей класса, *M* – описание методов класса, *CI* – описание наследуемых классов.

Задача распознавания паттернов сводится к тому, чтобы сопоставить признаки программы *P* и признаки паттерна *Pp*.

Материалы и методы решения

В данной работе используется корпус программ на языке программирования Java, ссылка на который была приведена в статье [4].

Для выделения признаков программы необходимо реализовать парсер.

После изучения статей с похожей тематикой были выбраны следующие признаки для классов: ClassName (имя класса), ClassModifiers (модификаторы доступа к классу), ClassImplements (интерфейсы, которые реализует данный класс), ClassExtends (наследуемые классы), Fields (поля класса), Methods (методы класса).

У методов были выделены следующие признаки: MethodName (имя метода), MethodModifiers (модификаторы доступа), MethodParams (входные параметры метода), MethodReturnType (тип возвращаемого значения), MethodIncomingMethod (количество методов, который вызывает данный метод), MethodIncomingName (имена вызываемых методов), MethodOutgoingMethod (количество методов, вызывающих данный метод), MethodOutgoingName (имена методов, вызывающих этот метод).

Выбор именно этих признаков обусловлен тем, что они необходимы как при распознавании паттернов с помощью методов статического анализа, так и при распознавании методами машинного обучения. В качестве метода распознавания паттерна был выбран статический анализ – распознавание по признакам. Паттерны имеют определенную структуру и иерархию классов, по которым их можно распознать.

Каждому паттерну проектирования соответствуют определенные признаки, указывающие на присутствие этого паттерна в коде, поэтому в полученной коллекции деревьев для каждого паттерна достаточно будет проверять определенный набор условий.

Первым рассмотренным паттерном был singleton – порождающий паттерн проектирования. Он гарантирует, что в программе будет только один экземпляр

класса с глобальной точкой доступа к нему. Его можно определить по следующим признакам класса: приватный конструктор, при этом в классе нет публичных конструкторов, приватное статическое поле, хранящее ссылку на объект этого класса, публичный статический метод для доступа к объекту класса.

Алгоритм распознавания паттерна singleton можно записать с помощью псевдокода.

$C \coloneqq \emptyset$

$foreach\ class\ c\ do$

$bool\ haveStaticfield, havePrivateCtor, haveGetMethod \coloneqq false$

$foreach\ field\ f\ in\ c.fields\ do$

$haveStaticfield \coloneqq haveStaticfield \vee isPrivate(f) \wedge isStatic(f)$

$foreach\ method\ m\ in\ c.methods\ do$

$havePrivateCtor$

$\coloneqq havePrivateCtor \vee isConstructor(m)$

$\wedge isPrivate(m)$

$haveGetMethod$

$\coloneqq haveGetMethod \vee isPublic(m) \wedge isStatic(m)$

$\wedge m.returnType == c.className$

$if\ haveStaticfield \wedge havePrivateCtor \wedge haveGetMethod\ do$

$C \coloneqq C \cup c$

Был также рассмотрен альтернативный способ представления и проверки признаков паттернов.

В данном подходе паттерн – это набор правил, состоящих из названия правила и определенных выражений.

Pattern: Rules

Rules: Rule*

Rule:

Name ‘{‘

(RuleExpr)*

‘}’

Выражения правил содержат операцию и тело правила.

RuleExpr: Op RuleBody

Для операций предусмотрена операция require, означающая искомый признак, указывающий на присутствие паттерна в коде.

Op: ‘require’

RuleBody:

Name

‘+’ Attributes (FieldRule | MethodRule | CtorRule)

Тело метода может содержать название правила, которое необходимо проверить, или некоторый набор определенных признаков: атрибуты, правила для полей класса, методов или конструкторов.

Attributes: (Attribute)*

Attribute: (‘!’)?AttributeName

Атрибуты состоят из названий атрибутов и отметки критичности. Если такая метка присутствует, то условие должно выполняться для всех полей класса, методов или конструкторов.

FieldRule: ‘fld’ ‘->’ Type

Правило для поля класса содержит тип хранимого в поле значения.

MethodRule: ‘mtd’ ‘(‘ Type* ‘)’ ‘->’ Type

Правило для методов состоит из типов параметров и типа возвращаемого значения.

CtorRule: ‘ctor’ ‘(‘ Type* ‘)’

Правило для конструктора содержит типы входных параметров.

Type: ‘_’ | ‘this’ | Name

Тип может содержать отметку, что тип для данного правила не имеет значения, как и количество параметров, this – тип значения должен совпадать с названием класса, для которого проверяется условие, и Name – имя правила.

AttributeName: ‘private’ | ‘public’ | ‘static’ | ‘abstract’ | ‘override’

Name: [A-Za-z][A-Za-z0-9]

В данной грамматике паттерн singleton можно представить с помощью следующих правил:

```
Singleton

cls {

    require static !private fld -> this

    require !private ctor (_)

    require static public mtd (_) -> this

}
```

Результаты

Для выделения признаков был реализован парсер на языке Python 3 с использованием библиотеки Plyj для парсинга исходного кода на языке Java 7. Описанные методы поиска паттернов были реализованы на языке программирования C#, для интерфейса использовались Windows Forms.

Первый подход поиска паттернов в деревьях признаков программ прост в реализации, однако он усложняет добавление новых паттернов для распознавания, так как для каждого нового паттерна придется реализовывать алгоритм поиска.

Второй подход с использованием грамматики для описания паттернов позволяет упростить распознавание новых паттернов и более гибко описывать искомые паттерны. При этом алгоритм поиска будет более общим и сведется к проверке правил, записанных для каждого паттерна с использованием введенной грамматики. Для этого подхода требуется реализовать парсер, который будет выделять записанные в файле правила и строить по ним дерево правил.

Заключение

В дальнейшем можно добавить в приложение большее количество распознаваемых паттернов проектирования, протестировать реализованный метод поиска паттернов на проектах, где в классах может быть использовано несколько паттернов проектирования. Функционал приложения можно расширить и добавить подсказки с описанием найденных паттернов, чтобы его можно было использовать в учебных целях, когда пользователь на готовых примерах смотрит, как в исходном коде реализован тот или иной паттерн.

Благодарности

Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 19-37-51028.

СПИСОК ЛИТЕРАТУРЫ

1. Heuzeroth D. [et al.] Automatic design pattern detection.
2. Uchiyama S. [et al.] Design pattern detection using software metrics and machine learning. First international workshop on model driven software migration, 2011.
3. Uchiyama S. [et al.] Detecting design patterns in object-oriented program source code by using metrics and machine learning. Journal of Software Engineering and Applications, 2014.
4. Nazar N., Aleti A. Feature-based software design pattern detection.
5. Mhawish M.Y., Gupta M. Software metrics and tree-based machine learning algorithms for distinguishing and detecting similar structure design patterns.
6. Гамма Э. [и др.] Паттерны объектно-ориентированного программирования.