Implementasi Mesin Turing untuk Identifikasi Kebutuhan *State Pattern* Berdasarkan *Source Code*

Dziky Dwi Rahmanto[1], Eko Darwiyanto[2], Sri Widowati[2]

[1,2,3]Fakultas Teknik Informatika Institut Teknologi Telkom, Bandung
[1]dziky.bear@gmail.com, [2]ekodarwiyanto@yahoo.com, [3]swd@ittelkom.ac.id

Abstrak

Identifikasi *design pattern* merupakan suatu proses yang dilakukan pada saat mendesain perangkat lunak, khususnya s*tate pattern* identifikasi kebutuhannya dilihat pada *source code*. Hal ini dikarenakan pola *code smell* pada *state pattern* yang hanya dapat terlihat pada *source code*-nya. Mencari suatu pola pengkodean dalam suatu *source code* bukanlah hal yang mudah dan cepat, selain diharuskan memahami isi *source code*, terkadang efek *human error* seperti terlewatnya suatu baris kode saat membaca *source code* kerap kali terjadi. Oleh karena itu dibutuhkan suatu perangkat lunak yang dapat mencari pola tersebut secara otomatis.

Dalam Tugas Akhir ini telah diimplementasikan mesin turing untuk menyelesaikan permasalahan pencarian pola – pola *code smell state pattern*.. Mesin turing dipilih karena kemampuannya sangat mendukung proses pencarian pola tersebut dalam suatu *source code*.

Hasil analisis nilai akurasi hasil identifikasi perangkat lunak yang dilihat berdasarkan hasil identifikasi para pakar *OO* membuktikan bahwa mesin turing hanya dapat membaca pola secara sintaktik saja, namun tidak semantiknya.

Kata kunci : state pattern, identifikasi, code smell, mesin turing, source code, akurasi.

Abstract

Identify design pattern is a process performed when designing software, especially state pattern, its identification of needs seen in the source code. It is because the pattern of state pattern's code smell can be detected only from its source code. Finding a code pattern in a source code is not easy and fast, beside we need to understand the source code, sometime the human error effect like skipping a line of code when we read the source code, often happen. That's why a software to find its pattern automatically is needed.

In this Final Project, a turing machine has been implemented to solve the problem of finding pattern of state pattern's code smell's. Turing machine was chosen for its ability to strongly support the process of finding those patterns in a source code.

The results of the identification accuracy value analysis software that is seen on the results of the identification of OO experts proved that a Turing machine can only read a syntactic patterns, but not semantic.

Keywords: state pattern, identification, code smell, turing machine, source code, accuration.

1. Pendahuluan

Mengetahui kebutuhan akan suatu *design pattern* dalam suatu perangkat lunak dapat meningkatkan pemahaman pengembang dalam memahami desain perangkat lunak tersebut [7]. Terutama apabila pengembang ingin mengembangkan perangkat lunak tersebut lebih lanjut. Kebutuhan akan suatu design pattern ditunjukkan dengan kemunculan *code smell* pada suatu perangkat lunak. Setiap *design pattern* memiliki *code smell* yang berbeda dari *design pattern* lainnya.

Pada *State Pattern*, salah satu pola *code smell* yang muncul adalah adanya percabangan kompleks. Berbeda dari *design patter*n lainnya yang dapat diidentifikasi kebutuhannya dari diagram kelas, *state pattern* mengharuskan pengembangnya untuk mengidentifikasi dari *source code*. Hal ini dikarenakan pola percabangan pada *code smell state pattern*, hanya dapat dilihat dari *source code*. Mungkin mencari percabangan dalam suatu *source code* yang panjang bukanlah suatu hal yang sulit, namun untuk mencari percabangan yg sesuai dengan pola *code smell state pattern* hal ini akan sedikit memakan waktu. Salah satu cara menemukan pola – pola tersebut dalam suatu *source code* tanpa harus membacanya secara manual adalah dengan memanfaatkan otomata, salah satunya adalah mesin Turing.

Mesin Turing memiliki kemampuan untuk menggerakkan pointernya (*head*) ke arah kanan dan kiri pita. Mesin Turing juga memiliki kemampuan untuk membaca dan menulis di lebih dari satu pita simbol (mesin Turing *Multiple-Tape*). Jika dilihat dari karakteristik pencarian pola *code smell state pattern* yang mengharuskan kita untuk mencari suatu pola pada bagian tertentu secara berulang – ulang , yaitu memastikan suatu percabangan adalah *code smell state pattern*, kemampuan mesin turing

tersebut sangat mendukung. Karena kita dapat menyalin bagian tertentu dari *source code* ke dalam pita lain (kemampuan *Multiple-Tape*) untuk dicek secara berulang – ulang (kemampuan menggerakkan pointer ke kiri dan kanan) tanpa harus mengganggu pita utamanya.

Seperti halnya pada mesin otomata lainnya, *state – state* pada mesin Turing nantinya akan dijadikan pedoman oleh perangkat lunak dalam mencari pola – pola tersebut.

2. Landasan Teori

2.1 State Pattern

State Pattern merupakan *Design Pattern* yang memungkinkan sebuah objek untuk mengubah (*alter*) perilakunya (*behaviour*) saat *state* internalnya berganti [5]. Objek tersebut nantinya akan berganti kelas [5].

State Pattern digunakan bila ada kasus seperti berikut [5] :

1. Sebuah perilaku objek tergantung dari *state*-nya, dan objek tersebut harus mengubah perilakunya pada saat *runtime* tergantung dari *state*-nya.
2. Sebuah operasi memiliki perkondisian yang besar dan bercabang yang tergantung dari *state* objek tersebut. *State* ini biasanya direpresentasikan oleh satu atau lebih konstanta. Terkadang, beberapa operasi akan memiliki struktur perkondisian yang sama. *State Pattern* akan memisahkan setiap cabang perkondisian ke dalam kelas – kelas yang terpisah. Hal ini memungkinkan kita untuk memperlakukan *state* objek sebagai sebuah objek terpisah yang dapat bervariasi dan berbeda dari objek yang lain.

2.2 Mesin Turing

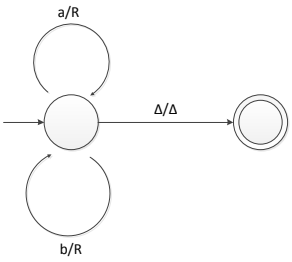
Mesin turing merupakan salah satu kelas otomata yang dikenalkan oleh Alan M. Turing tahun 1936. Mesin turing memiliki kemiripan dengan otomata berhingga dimana mereka sama – sama terdiri dari sebuah mekanisme kontrol dan sebuah masukan simbol yang kita sebut pita. Perbedaan antar keduanya adalah mesin turing dapat menggerakkan pointer pita (tape heads) ke kanan dan ke kiri. Selain itu mesin turing juga memungkinkan kita untuk membaca dan menulis pada pita yang dibacanya.

Definisi formal untuk mesin turing adalah sebagai berikut :

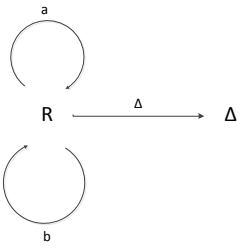
Sebuah mesin turing adalah sebuah *sextuple* dari bentuk (S, Σ, Γ, δ, ι, h), dimana :

a. S adalah himpunan berhingga *state*.
b. Σ adalah himpunan berhingga simbol yang bukan *blank*. Disebut juga *machine's alphabet.*
c. Γ adalah himpunan berhingga simbol, termasuk yang ada pada Σ. Disebut juga *machine's tape symbols.*
d. δ adalah fungsi transisi mesin.
e. ι adalah anggota S yang merupakan *initial state*.
f. h adalah anggota S yang merupakan *halt state*.

Untuk penggambaran model mesin turing, terdapat 2 cara, yaitu dengan diagram transisi dan dengan diagram komposit. Berikut pada sebuah mesin turing yang akan berhenti dan menulis Δ apabila bertemu Δ :



Gambar 2(a) Diagram Transisi Mesin Turing

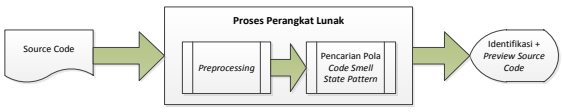


Gambar 2(b) Diagram Komposit Mesin Turing

3. Gambaran Umum Sistem

Perangkat lunak yang dibangun adalah sebuah perangkat lunak yang dapat mengidentifikasi pola – pola *code smell state pattern* dalam suatu *source code* dengan menggunakan mesin turing. Dengan menggunakan perangkat lunak ini pengguna dapat mengetahui apakah suatu source code membutuhkan *state pattern* atau tidak. Selain itu, pengguna juga dapat mengetahui posisi pola *code smell state pattern* yang ditemukan.

Secara umum proses tahapan proses dalam perangakat lunak adalah sebagai berikut :



Gambar 3(a) : Arsitektur Perangkat Lunak

Terdapat 2 proses utama dalam perangkat lunak, yaitu :

1. *Preprocessing*

Proses *preprocessing* yang dilakukan perangkat lunak adalah mem-*filter* dan memindahkan semua karakter yang ada di dalam *source code* kedalam

pita pertama mesin turing (penjelasan pita simbol yang digunakan dalam mesin turing perangkat lunak akan dijelaskan pada sub bab Desain Mesin Turing) disesuaikan dengan himpunan simbol yang ada pada mesin turing perangkat lunak. Untuk karakter yng tidak terdaftar pada daftar simbol mesin turing perangkat lunak, maka akan dikonversi menjadi karakter spasi (‘ ‘).

2. Pencarian Pola *Code Smell State Pattern*

Proses pencarian pola *code smell state pattern* dilakukan dengan mengacu pada desain mesin turing perangkat lunak. Perangkat lunak akan berhenti melakukan pencarian ketika telah mencapai karakter akhir dari *source code*.

4. Pengujian Sistem

Pengujian perangkat lunak dilakukan untuk mengetahui apakah mesin turing yang didesain sudah dapat merepresentasikan ciri khas pattern dan apakah hasil identifikasi yang dikeluarkan perangkat lunak sudah sesuai dengan hasil identifikasi para pakar *OO*. Berdasarkan pengujian yang sudah dilakukan terhadap 12 data uji yang berupa *source code* Java (.java) didapat bahwa 8 dari 12 data uji terbaca sesuai dengan hasil identifikasi oleh para pakar *OO*, dimana 7 dari 7 data uji yang membutuhkan *state pattern* terbaca sesuai dan 1 dari 5 data uji yang tidak membutuhkan *state pattern* terbaca sesuai. Empat data yang terbaca tidak sesuai kemudian dianalisis kembali dengan cara mengidentifikasi secara manual.

Berikut adalah bagian dari keempat data yang terbaca salah oleh perangkat lunak, yaitu terbaca sebagai sebagai pola *code smell state pattern* :

```
public class GumballMachine  {
...
void releaseBall()  {
        ...
        if (count != 0)  {
                count = count - 1;
        }
}
...
}
```

```
public class StatePatternView extends FrameView  {
...
if (jadwal  != null)  {
        ...
} else {
        ...
        jadwal = dbManager.getLastJadwal();
        ...
}
...
}
```

```
public class DependentPizzaStore  {
 ...
if (style.equals("NY"))  {
        if (type.equals("cheese")) {
                ...
        } else if (type.equals("veggie")) {
                ...
        } else if (type.equals("clam"))  {
                ...
        } else if (type.equals("pepperoni")) {
                ...
        }
} else if (style.equals("Chicago")) {
        ...
} else {
        ...
}
...
}
```

```
class State0 extends State{
        State handle(int no, String s) {
          if ((s.equals("a")) && (no==0)){
                return state1;
          }else{
                error(s, no);
          }return null;
        }
  }
```

Jika kita perhatikan pada kedua kasus awal, sebenarnya pola *code smell state pattern* memang muncul, yaitu sebuah perilaku objek tergantung dari *state*-nya (pada bagian yang dicetak merah), dan objek tersebut harus mengubah perilakunya pada saat *runtime* tergantung dari *state*-nya (pada bagian yang dicetak biru). Namun, pada kenyataanya pola tersebut bukanlah pola *code smell state pattern*, karena variabel **count** pada data pertama dan **jadwal** pada data kedua menurut pemilik data bukanlah sebuah *state*. Sehingga meskipun memiliki pola yang sama, tetapi kedua data tersebut tidak membutuhkan *state pattern*.

Hal ini lah yang menjadi kelemahan mesin turing. Mesin turing hanya membaca tiap – tiap karakter pada *source code* untuk mencari sebuah pola tanpa mengetahui arti dari pola tersebut. Dengan kata lain, mesin turing hanya dapat membaca suatu pola secara sintaktik saja, namun tidak secara semantiknya. Sementara dalam penentuan kebutuhan suatu *design pattern* dibutuhkan pemahaman tentang *source code* itu sendiri.

Kemudia pada kasus ketiga, terdapat operasi yang memiliki perkondisian yang besar dan bercabang yang tergantung dari *state* objek tersebut yang bergantung pada state tertentu. Hal ini memenuhi pola *code smel state pattern* yang kedua. Namun pada kenyataannya, pakar yang menganalis

data tersebut menyatakan bahwa data tersebut membutuhkan *design pattern* lain, yaitu *factory pattern*. Padahal jika kita perhatikan pada data Person.java berikut :

```
public class Person {
...
   public void giveMeMoney(){
      if (character.equals("GoodCharacter")){
         ...
      }else if (character.equals("BadCharacter")){
         ...
      }else if (character.equals("OtherCharacter")){
         ...
      }
   }
}
```

dapat dilihat bahwa data tersebut memiliki pola yang sama dengan data yang ada pada kasus ketiga. Namun pakar yang menganalisa data Person.java tersebut menyatakan bahwa data tersebut membutuhkan *state pattern*.

Untuk kasus keempat, hal ini mirip dengan pola kasus ketiga, bedanya pada kasus keempat menurut pakar yang menganalisisnya, data tersebut sudah diimplementasikan state pattern, padahal jika kita lihat pola yang ditemukan oleh perangkat lunak, sebenarnya hal ini sesuai dengan pola kedua yang dikeluarkan oleh GoF [5], yaitu sebuah operasi memiliki perkondisian yang besar dan bercabang yang tergantung dari *state* objek tersebut.

Hal ini membuktikan bahwa penggunaan / pemilihan suatu *deisgn pattern* bersifat subjektif. Maksudnya, terkadang pada data yang sama atau pada pola yang sama, dapat dinyatakan memiliki *design pattern* yang berbeda oleh pakar yang berbeda. Dan hal ini tidak dapat dideteksi oleh mesin turing yang hanya dapat mendeteksi secara sintaktik saja.

Dengan demikian dapat diketahui bahwa mesin turing hanya dapat mempercepat penemuan pola *code smell state pattern*, namun tidak menjamin bahwa pola tersebut merupakan *code smell state pattern*, sehingga analisis manual dari manusia terhadap pola tersebut masih dibutuhkan untuk menentukan apakah pola yang ditemukan mesin turing merupakan *code smell state pattern*.

5. Kesimpulan

Berdasarkan analisis terhadap hasil pengujian, diperoleh kesimpulan sebagai berikut:

1. Mesin turing dapat digunakan untuk merepresentasikan pola – pola *code smell state pattern*.
2. Mesin turing hanya dapat membaca pola pengkodean secara sintaktik, namun tidak secara semantik.
3. Pola yang diidentifikasi oleh mesin turing belum tentu merupakan *code smell state pattern*.
4. Identifikasi suatu *design pattern* tidak dapat dilihat hanya dari sisi sintaktik pengkodean saja, tetapi juga harus dilihat sisi semantiknya.
5. Untuk menentukan kebutuhan suatu *source code* terhadap *state pattern*, mesin turing saja tidak cukup, dibutuhkan bantuan manusia untuk menganalisis pola *code smell* yang ditemukan oleh mesin turing.

Daftar Pustaka

[1] Anonymous. 2008. *Turing machine simulator (Java)*. Diakses : 13 Juni 2011, [online]. Available : http://en.literateprograms.org/Turing_machine_simulator_%28Java%29#chunk%20def:TuringSimulator%20constructor.

[2] Anonymous. 2011. *Automata Theory*. Dikutip : 15 September 2011, [online]. Available : http://en.wikipedia.org/wiki/Automata_theory.

[3] Anonymous. 2011. *Java Syntax*. Dikutip : 14 September 2011, [online]. Available : http://en.wikipedia.org/wiki/Java_syntax.

[4] Brookshear, J. Glenn. 1989. *THEORY OF COMPUTATION FORMAL LANGUAGES, AUTOMATA, AND COMPLEXITY*, The Benjamin/Cummings Publishing Company, Inc.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[6] Freeman, Eric & Freeman, Elisabeth. 2004. *Head First Design pattern*, First Edition, O Reilly Media, Inc.

[7] Fukaya, Kazuhiro, Kubo, Atsuto, Washizaki, Hironori, dan Fukazawa, Yoshiaki. *Design pattern Detection Using Source code of Before Applying Design patterns. Eighth IEEE/ACIS International Conference on Computer and Information Science*, 2009.

[8] Hariyanto, Bambang , Ir., MT. 2007. *Esensi – Esensi Bahasa Pemrograman JAVA*. Informatika Bandung.

[9] Hendro, Steven. 2008. *Aturan Identifier*. Dikutip : 14 September 2011, [online]. Available : http://sinau-java.blogspot.com/2008/05/aturan-identifier.html.

[10] Jacobs, Aaron. *GoF State Pattern*. Diakses : 14 September 2011. [online]. Available : h*ttp://www.cse.wustl.edu/~cdgill/courses/cse432/Klein.State.pptx*.

[11] Muttaqin, Fahrul. *Tugas Akhir : Analisis dan Implementasi State Pattern*. Fakultas Teknik Informatika Institut Teknologi Telkom.

[12] Tar, Bob. *The State and Strategy Pattern*. Dikutip : 23 Juli 2009, [online]. Available : *userpages.umbc.edu/~tarr/dp/lectures/StateStrategy-2pp.pdf*.