DETEKSI CODE SMELL PADA KODE PROGRAM DALAM REPRESENTASI AST DENGAN PENDEKATAN BY RULES

Hanson Prpihantoro Putro [1], Inggriani Liem [2]
[1,2]Kelompok Keahlian Rekayasa Perangkat Lunak dan Data Insitut Teknologi Bandung
Jl Ganesha 10, Bandung 40135
Telp. (022) 2500935
E-mail: if29047@students.if.itb.ac.id, inge@informatika.org

ABSTRAKS
Pada makalah ini dijelaskan bagaimana deteksi code smell dilakukan dengan menganalisis beberapa aspek yang berpengaruh pada keberadaan code smell dalam suatu kode program. Tujuan penelitian ini adalah melakukan deteksi code smell yang bisa digunakan untuk semua bahasa pemrograman, khususnya untuk menilai hasil praktikum tugas kuliah pemrograman. Dengan representasi AST dan pendekatan by rules, dilakukan analisis untuk menentukan parameter, proses deteksi untuk kemudian diimplementasikan ke dalam sebuah perangkat detektor. Untuk mempermudah proses pengolahan data, yang digunakan sebagai masukan dan keluaran program detektor ini adalah file dengan format XML. Validasi telah dilakukan dengan memilih 26 smell yang berhasil dideteksi dalam pengujian perangkat detektor ini

Kata Kunci: code smell, AST, code smell detection by rules, XML

1. PENDAHULUAN

Code smell merupakan istilah yang menandakan bahwa di suatu kode program mungkin terdapat suatu masalah (Slinger, 2005). Kode yang dikatakan mengandung *smell* berarti ada yang salah, atau lebih tepatnya tidak seharusnya ada pada kode tersebut. Hal ini disebabkan oleh kebiasaan buruk para *programmer* dalam berlatih pemrograman yang kemudian memunculkan suatu masalah bagi program yang dibuatnya. Sebagai dasarnya, Fowler (2000) telah mendefinisikan 22 *code smell* yang sering muncul pada sebuah kode program. Dari sini kemudian berkembang berbagai jenis *code smell* lain yang dapat ditemukan di berbagai kode program dalam suatu pengembangan perangkat lunak.

Code smell merupakan permasalahan yang termasuk dalam *design defect* pada suatu perangkat lunak. Desain yang buruk ini dapat mengurangi kualitas dari perangkat lunak yang disusunnya. *Code smell* harus diminimalisir dengan mendeteksi keberadaannya pada kode program, kemudian melakukan perbaikan jika diperlukan. Pendeteksian *code smell* sendiri dapat dilakukan secara manual oleh manusia ataupun secara otomatis oleh suatu kakas. Pendeteksian secara manual akan memerlukan waktu yang cukup lama terlebih jika dilakukan untuk mendeteksi ratusan kode hasil pengerjaan tugas pemrograman para mahasiswa seperti yang terjadi di Program Studi Teknik Informatika ITB. Oleh karena itu diperlukan suatu perangkat detektor *code smell* yang dapat digunakan secara otomatis untuk mendeteksi ratusan kode program dalam waktu yang relatif lebih cepat.

Terdapat beberapa cara pendeteksian yang bisa dilakukan untuk mendapati suatu *code smell* pada kode program. Berikut akan disampaikan beberapa langkah yang dilakukan dalam melakukan deteksi *code smell* pada kode program dalam representasi AST dengan pendekatan *by rules*. Pembahasan dimulai dari beberapa penelitian/proyek terkait, analisis deteksi *code smell* termasuk alasan pemilihan model yang dideteksi serta pendekatan deteksinya, implementasi dari paramter deteksi dan perangkat detektornya hingga hasil pengujian yang telah berhasil dilakukan pada perangkat detektor tersebut. Diharapkan dengan adanya perangkat detektor ini, kebiasaan pemrograman yang baik akan terbentuk pada diri para *programmer* khususnya para mahasiswa Program Studi Teknik Informatika ITB, sehingga tidak lagi menghasilkan *code smell* yang berbahaya.

2. PEKERJAAN TERKAIT

Terdapat tiga detektor yang sebelumnya telah dikembangkan dan dipublikasikan untuk mendeteksi *code smell*. Yang pertama yaitu **CodeNose**. Pengembangan deteksi otomatis ini dilakukan oleh Stefan Slinger (2005) pada tesisnya di Delft University of Technology. Ia membangun deteksi sebagai sebuah *plug in* untuk *framework* pada **IDE Eclipse**. Bahasa yang dideteksi adalah bahasa pada kode program Java.

Selanjutnya terdapat **TRex** (*TTCN-3 Refactoring and Metrics Tools*) yang merupakan plugin **IDE Eclipse** yang bersifat *open source* bagi siapa saja yang bersedia mengikuti proyek pengembangannya. Perangkat ini menyediakan fungsionalitas-fungsionalitasnya dalam notasi **TTCN-3** (*Testing and Test Control Notation version 3*) (Bisanz, 2006). **TRex** mendukung penilaian dan restrukturisasi otomatis pada kasus uji **TTCN-3** dengan menyediakan beberapa *metric* dan *refactoring* yang cocok.

Perangkat lunak ketiga yaitu **Checkstyle**. **Checkstyle** merupakan perangkat pengembangan yang membantu para *programmer* Java menuliskan kode program yang terikat pada standar pemrograman (Burn, 2007). Perangkat ini secara otomatis melakukan proses pengecekan kode program bahasa Java termasuk mendeteksi setiap kode yang tidak sesuai dengan *style*. Perangkat ini cocok digunakan untuk proyek yang ingin ketat terhadap aturan, *style*, *coding standard* yang disepakati oleh anggota proyek. **Checkstyle** disediakan sebagai *plugin* untuk berbagai IDE berbahasa Java seperti **Eclipse**, **Netbeans**, **BlueJ**, dan perangkat pengembangan yang lain.

Selain berbagai perangkat detektor yang telah dijelaskan sebelumnya, terdapat juga penelitian mengenai deteksi *code smell* ini yang telah mendefinisikan suatu metode dan teknik deteksi suatu *code smell* (Moha, 2009). Metode ini adalah metode generik yang mendeskripsikan beberapa langkah yang dapat digunakan dalam melakukan spesifikasi dan deteksi suatu *smell*. Metode deteksi ini dimulai dengan analisis deksripsi *smell*, spesifikasi, proses, deteksi hingga validasi

3. ANALISIS DETEKSI CODE SMELL

Terdapat banyak sekali *code smell* yang bermunculan di lingkungan pengembangan perangkat lunak. Dari beragam *smell* yang ada tersebut, cara yang dilakukan untuk mendeteksi keberadaanya juga beragam. Berikut disampaikan analisis mengenai *code smell*, parameter deteksi serta proses deteksi yang dilakukan untuk menemukan suatu *code smell* pada sebuah kode program

3.1 Code smell

Code smell yang dianalisis pada penelitian ini disarikan dari berbagai sumber yang menyertakan proses deteksi *code smell*. *Code smell* utama yang disampaikan oleh Fowler (2000) dipilih pada *smell* yang dinyatakan oleh Mantylä (2003) sebagai *smell* yang dapat dideteksi. Kemudian diambil juga *smell* yang berhubungan dengan *style* atau konvensi dalam menuliskan kode program (Wirth, 1973; Liem, 2000; van der Vlugt, 2003; Duta, 2003; Cannon, 2008; Henricson, 1992; Geotechnical Sofware, 2008; Sun Microsystems, 1999). Terakhir ditambahkan *smell* sebagai referensi diperoleh dari yang telah dapat dideteksi pada perangkat-perangkat deteksi yang telah dikembangkan sebelumnya (Slinger, 2005; Bisanz, 2006; Burn, 2007).

Langkah pertama pada tahap analisis dilakukan dengan mendefinisikan bahan utama yang akan dideteksi yaitu suatu kode program. Sebuah kode program dapat dituliskan dalam berbagai bahasa pemrograman. Bahasa pemrograman yang digunakan inilah yang mempengaruhi *code smell* seperti apa yang ada dalam kode program. Terdapat *code smell* yang berlaku di satu bahasa dan *code smell* yang berlaku untuk semua bahasa. Bahasa yang digunakan dipengaruhi oleh paradigma yang teraplikasikan pada bahasa tersebut.

Setiap bahasa memiliki file kode program dengan sintaks yang berbeda-beda untuk merepresentasikan satu program yang sama (Aho, 2007). Deteksi yang dilakukan langsung pada kode program akan bergantung pada bahasa yang digunakan. Namun demikian, secara umum struktur dari kode program tersebut akan sama jika dilihat dari model AST (*Abstract Syntaxt Tree*). Oleh karena itu, dengan mendeteksi bentuk AST kode program, dimungkinkan untuk membangun detektor yang bebas bahasa.

Selanjutnya dari beberapa pendekatan deteksi yang ada, dipilih pendekatan *detection by rules*. Pendekatan ini dipilih karena secara alami, *code smell* sendiri didefinisikan sebagai aturan-aturan yang dilanggar untuk suatu kode program yang baik. Hampir semua *smell* dapat didefinisikan sebagai aturan yang menunjukkan keberadaanya dalam sebuah kode program. Selain itu, deteksi *code smell* dengan menggunakan pendekatan *rule* memungkinkan pengguna untuk dapat memodifikasi aturan deteksi *code smell* sesuai konvensi yang berlaku di lingkungan pengguna

3.2 Parameter deteksi

Setiap *code smell* memiliki kriteria yang menjelaskan kode seperti apa yang mengandung *smell*. Contoh ini dapat dijelaskan dengan *smell Incorrect Using Function*. Suatu fungsi dikatakan mengandung *smell* ini jika dalam deklarasi fungsi tersebut tidak ada *statement* `return`, yang berarti bertentangan degan kegunaan suatu fungsi yang harusnya memiliki nilai kembalian. Itulah kriteria *smell Incorrect Function,* yang dalam hal ini akan dijelaskan dengan pendekatan *by rules* sebagai teknik deteksi yang digunakan.

Aturan-aturan yang menandakan *code smell* beserta kode program yang akan dideteksi disediakan oleh pengguna sebagai parameter masukan untuk perangkat detektornya. Dalam hal ini karena deteksi dilakukan pada level AST maka kode program hasil ketikan *programmer* harus ditransformasi dahulu ke bentuk AST-nya oleh sebuah transformator. Dengan menggunakan AST maka sintaks dari semua bahasa akan menjadi satu elemen yang sama untuk struktur pohon sintaks yang dibangun.

3.3 Proses deteksi

Setelah dikumpulkan berbagai kriteria yang digunakan untuk mendeteksi setiap *code smell* dengan menggunakan pendekatan *by rules* pada AST kode program, kriteria-kriteria tersebut dapat dikelompokkan ke dalam enam klasifikasi menurut jenis aturan yang akan diimplementasikan serta posisi dari struktur AST yang akan dideteksi. Dengan pengelompokan seperti yang kami usulkan,

diharapkan implementasi dapat dilakukan dengan lebih mudah.

Enam kelompok tersebut beserta algoritma dasar yang digunakan untuk melakukan deteksi dijelaskan sebagai berikut:

a. *Count Child*
   Kelompok ini merupakan kelompok pendeteksian yang paling sederhana. Pendeteksian dilakukan dengan menghitung banyaknya elemen anak yang muncul dari suatu lingkup yang diberikan dengan satu kali penelusuran tanpa harus melakukan proses rekursif ke dalam elemen anaknya lagi. Teknik ini dilakukan untuk *smell* yang berhubungan dengan bagian dari suatu perintah *source code* seperti menghitung jumlah elemen `case` dalam sebuah elemen `switch`.
b. *Count Element*
   Teknik pada kelompok ini hampir sama dengan kelompok sebelumnya. Perbedaanya terletak pada kedalaman penelusurannya, bahwa di kelompok ini dilakukan secara rekursif hingga elemen terdalam. Pada kelompok ini, beberapa *rule* membutuhkan atribut pencarian baru di mana pencarian tidak hanya terletak pada kesamaan nama elemen saja tetapi dapat juga dengan mengecek kesamaan nilai ataupun tipenya seperti pada *smell JavaDoc*. *Smell* ini menyatakan bahwa pada kode program Java atau bahasa yang mendukung pendokumentasian otomatis, harus terdapat komentar yang menggunakan standar tersendiri.
c. *Count Grandchild*
   Teknik pendeteksian yang dilakukan pada kelompok ini juga hampir sama dengan teknik sebelumnya. Namun proses rekursif tidak dilakukan hingga elemen habis melainkan berhenti setelah berada pada kedalaman kedua. Beberapa *smell* memerlukan dua kali tahap penulusuran yaitu yang berhubungan dengan *control statement* di mana *smell* baru muncul di dalam blok kode program yang merupakan anak dari setiap *statement* perintah. Sebagai contoh kasus pada perintah `break` yang harus ada pada setiap *block scope* sebuah `case` atau `break` yang tidak boleh ada pada *block scope* sebuah `for`.
d. *Count Element Value*
   Pada kelompok ini, pendeteksian dilakukan dengan terlebih dahulu mencari sesuatu nilai yang nantinya digunakan untuk menghitung elemen lain dengan nilai yang sama. Proses ini dapat dilakukan secara rekursif. Contoh *smell* yang termasuk dalam kelompok ini yaitu *Unused Definition* di mana nama setiap pendefinisan diambil dari setiap deklarasi yang dilakukan untuk kemudian dicari keberadaannya pada setiap ekspresi yang ada pada blok program yang sama.
e. *Compare Tree*
   Jika pada kelompok sebelumnya kesamaan ditentukan dari nilai setiap elemen yang ditemukan maka pada kelompok ini kesamaan ditentukan dari struktur pohon yang membentuk sebuah potongan kode program. Sebagai contoh pada *smell Duplicate Code*, potongan kode program akan terlihat sebagai sebuah duplikasi jika bentuk pohon keduanya dan juga elemen penyusunnya sama, walaupun *value* yang digunakan beda.
f. *Check Order*
   Pada kelompok pendeteksian ini, *smell* diperoleh jika struktur AST yang diberikan tidak sesuai dengan aturan. Aturan yang didefinisikan adalah aturan yang benar di mana suatau kode program harus memiliki urutan elemen seperti yang didefinisikan tersebut. Sebagai contoh pada *smell General File Content*, suatu struktur AST dari sebuah file harus memiliki elemen-elemen yang disusun secara terurut mulai dari pendeklarasian tipe atau variabelnya hingga pendeklarasian program utamanya.

4. IMPLEMENTASI DETEKTOR

Perangkat detektor ini diimplementasikan dalam bahasa pemrograman Java. Java dipilih karena bahasa ini memiliki banyak *library* termasuk dalam pengolahan bahasa pemrograman.

4.1 Input-output

Masukan detektor berupa AST kode program dan aturan yang menandakan *code smell* serta keluaran berupa laporan keberadaan *code smell.* Masukan dan keluaran dari detektor ini diimplementasikan dalam format XML. XML dipilih karena format ini menyediakan aturan yang baku untuk mendefinisikan suatu struktur, termasuk struktur pohon AST, struktur kriteria *rules* untuk *code smell* serta struktur file pelaporan. Java juga telah menyediakan *library* untuk pengolahan XML sehingga implementasi dapat dilakukan dengan lebih mudah. Masukan berupa kode program yang dideteksi, direpresentasikan dalam format XML seperti yang diperlihatkan oleh Kode 1. Format XML dari kode program ini dihasilkan dari *transformator source code* dalam berbagai bahasa pemrograman yang merupakan bagian dari penelitian lain.

4.2 Paramter deteksi

Seperti yang sudah disampaikan sebelumnya, parameter sebagai masukan perangkat detektor ini diformulasikan menjadi aturan-aturan yang menandakan kemunculan *code smell* serta kode program dalam bentuk AST. Karena aturannya akan berlaku untuk semua *smell* dan AST-nya berlaku untuk semua bahasa maka dalam implementasinya perlu didefinisikan format XML yang seragam untuk setiap file masukan detektor.

```
program helloworld;
begin
writeln('Hello World');
end.
```

```
<ast_scheme project="Sample" lang="PASCAL">
 <source value="hello.pas">
  <program value="helloworld" line="1">
   <main_decl line="2">
    <method_call value="writeln" line="3">
     <argument line="3">
      <constant value="Hello World" line="3" />
     </argument>
    </method_call>
   </main_decl>
  </program>
 </source>
</ast_scheme>
```

Gambar 1. Contoh kode program mentah bahasa Pascal (atas) dan AST-nya dalam format XML (bawah)

Kode 1 memperlihatkan contoh masukan suatu file berisi AST (bawah) dalam format XML yang dibentuk dari sebuah kode program dalam bahasa Pascal (atas). Dari kode tersebut diperlihatkan bahwa suatu file AST akan diawali dengan elemen `ast_scheme` sebagai *root* penanda dokumen XML ini. Setelah itu struktur akan dilanjutkan dengan `source` yang menunjukkan file asal AST yang diikuti berbagai elemen yang menunjukkan struktur sintaks kode programnya. Semua elemen selanjutnya akan memiliki atribut `line` yang menunjukkan posisi di mana sintaksnya berada dan beberapa atribut `value` sebagai nilai tambahan untuk sintaks tersebut.

```
<smell_scheme>
  <module value="count_child">
    <rule value="Long Parameter List">
      <scope tag="function_method_decl" />
      <count tag="parameter_decl" />
      <limit value="5" cond="greater_than" />
      <notes value="Jumlah parameter dalam sebuah fungsi terlalu banyak. Coba untuk menstrukturkannya." />
    </rule>
  </module>
</smell_scheme>
```

Kelompok deteksi:
Count Child
Nama:
Long Parameter List
Lingkup ditemukan:
`function_method_decl`
Indikator *smell*:
`parameter_decl`
Dikatakan *smell* jika indikator:
Greater Than 5
Catatan untuk perbaikan:
`Jumlah parameter dalam sebuah fungsi terlalu banyak. Coba untuk menstrukturkannya. (misalnya membentuknya menjadi kelas baru)`

Gambar 2. Contoh pendefinisian *rule* dari *code smell Long Parameter List* (atas) dan penjelasannya (bawah)

Aturan *code smell* dalam format XML diperlihatkan melalui contoh pada Kode 2. File skema *rule* akan memiliki struktur yang terdiri dari 3 (tiga) elemen utama. Elemen yang pertama yaitu `smell_scheme` sebagai *root* dokumen. Elemen kedua yaitu `module` yaitu anak dari `smell_scheme`, menunjukkan kelompok deteksi yang berlaku. Elemen ketiga yaitu `rule` yang berisi aturan-aturan yang nantinya akan diimplementasikan pada deteksi sebagai anak dari elemen `module`.

Bahan utama yang digunakan untuk melakukan deteksi berada di dalam elemen `rule`. Contoh pada Kode 2 terdapat elemen `scope` yang menyatakan *smell* tersebut biasanya muncul di lingkup elemen apa pada AST yang dideteksi. Kemudian elemen `count` digunakan sebagai elemen apa yang mengindikasikan suatu kode program mengandung *smell*. Selanjutnya elemen `limit` menyatakan batasan dari elemen indikator yang ditemukan sehingga bisa dikatakan potongan stuktur kode program tersebut *smell* atau tidak. Terakhir elemen `notes` digunakan sebagai keterangan atas *smell* yang ditemukan tersebut

4.3 Proses deteksi

```
...
function nCalc (a, b, c, d, e, f:real)
: real
begin
  nCalc := ((a+b)/c);
end;
```

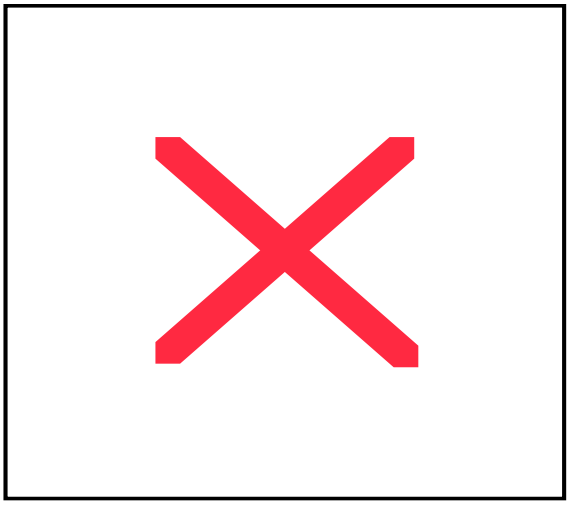
Gambar 3. Contoh kode program Pascal yang mengandung *code smell Long Paramter List*

Deteksi dilakukan dengan membaca *rules* yang menjadi masukan untuk kemudian diberlakukan pada AST yang akan dideteksi. Jika salah satu aturan ini cocok dengan beberapa bagian dari AST yang diberikan maka dikatakan bahwa kode program dari AST tersebut mengandung *code smell*. Sebagai contoh untuk *smell Long Parameter List*. Aturan yang telah didefinisikan seperti pada Kode 2 akan dapat mendeteksi keberadaan *smell* yang muncul pada contoh Kode 3. Kecocokan diperoleh karena pada kode program tersebut ditemukan enam deklarasi parameter dari suatu fungsi sedangkan pada aturannya dikatakan, suatu fungsi dikatakan mengandung *smell Long Paramter List* jika memiliki lebih dari lima deklarasi parameter. Untuk lebih jelasnya kode program pada Kode 3 memiliki struktur pohon sintaks seperti yang diperlihatkan pada Gambar 1.

Pertama, detektor akan menelusuri seluruh elemen AST untuk menemukan

ISSN: 1907-5022

`function_method_decl`. Karena *smell Long Parameter List* termasuk ke dalam kelompok *Count Child*, maka untuk setiap deklarasi fungsi, detektor akan menghitung elemen `parameter_decl`. Jika jumlah elemen tersebut lebih dari 5 maka dapat dikatakan bahwa dalam deklarasi fungsi tersebut terdapat *smell Long Parameter List*. Pencarian ini berlaku untuk setiap *rule* yang telah didefinisikan.

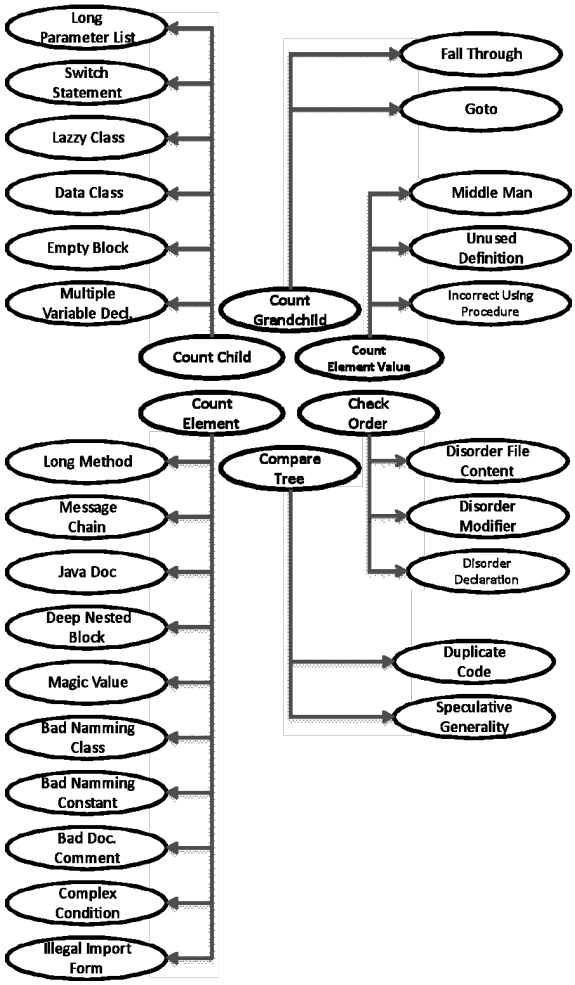


Gambar 4. Struktur AST dari kode program yang mengandung *code smell*.

5. PENGUJIAN

Pengujian dilakukan dengan menyediakan *test case* file AST serta *rules* terkait *smell* yang akan dideteksi dalam format XML. Pengujian telah dilakukan pada 26 *smell* yang dipilih di mana setiap pengujian yang dilakukan mendapatkan hasil yang bisa diterima. 26 *smell* tersebut dibagi ke dalam enam kelompok deteksi yang memperlihatkan dengan algoritma apa *smell* tersebut bisa dideteksi. Pembagian ini diperlihatkan oleh Gambar 2. Selanjutnya, setiap *smell* diuji dengan minimal dua *test case* (dua file masukan), file yang mengandung *smell* dan yang tidak menggandung *smell* tersebut.

Secara keseluruhan, pengujian berhasil dilakukan pada 54 file *test case* masukan tekait dengan 26 *smell* yang diujikan pada perangkat detektor *code smell* ini. 54 *test case* ini terbagi dalam 7 kelas uji, 6 kelas uji diperoleh dari 6 kelompok deteksi yang diimplementasikan sedangkan satu kelas uji merupakan kombinasi dari keenam kelompok uji tersebut. Hasil pengujian dapat diinterpretasi dari output yang dihasilkan oleh AST yang mengandung *smell* dideteksi (dieksekusi) dengan komponen detektor yang kami kembangkan. Hasil deteksinya dilaporkan pada keluaran bahwa kode program tersebut memang mengandung *smell* yang dimaksud atau bebas dari smell sesuai spesifikasi *test case*. Dengan demikian dapat disimpulkan bahwa deteksi *code smell* pada kode program dalam representasi AST dengan pendekatan *by rules* mampu melakukan deteksi untuk relatif sebagian besar *code smell* dari 22 *smell* yang didefinisikan oleh Fowler (2000) dan beberapa *smell* tambahan lainnya.



Gambar 5. Pembagian kelompok deteksi untuk code smell yang diuji.

6. KESIMPULAN

Dari kegiatan-kegiatan yang telah dilakukan, dihasilkan perangkat detektor *code smell* yang menggunakan masukan berupa AST dari kode program dan juga kriteria yang disusun berdasarkan proses deteksi dengan satu pendekatan saja, yaitu menggunakan pendekatan *by rules*. Sesuai literatur Bisanz (2006) dibuktikan bahwa deteksi *code smell* pada level AST memungkinkan deteksi dilakukan bebas bahasa. Pengujian telah dilakukan terhadap AST dari kode program Pascal, C, C++ dan Java dalam format XML. Sesuai literatur Burn (2007) dibuktikan bahwa dengan menggunakan *rule*, kriteria dan parameter deteksi *code smell* mudah untuk dimodifikasi sehingga dapat digunakan untuk mendeteksi *code smell* yang beragam. Dengan demikian dapat diambil kesimpulan bahwa pendeteksian pada AST kode program dengan menggunakan pendekatan *by rules* mampu mendeteksi keberadaan *code smell* untuk relatif banyak jenis *code smell* yang ada

7. RENCANA KE DEPAN

Seperti yang telah disampaikan sebelumnya bahwa terdapat banyak sekali teknik yang dapat digunakan untuk mendeteksi keberadaan *code smell*. Setiap *code smell* memiliki karakteristik yang berbeda-beda sehingga ia harus dideteksi dengan teknik yang berbeda-beda pula menurut jenis karakteristiknya. Agar semua *code smell* dapat dideteksi maka diperlukan analisis lebih mendalam untuk setiap teknik deteksi *code smell*. Arahan untuk penelitian selanjutnya adalah mendefinisikan model dan pendekatan deteksi yang lain berikut kombinasinya serta menentukan *code smell* seperti apa yang dapat dideteksi pada model dan pendekatan deteksinya. Diharapkan dengan spesifikasi tersebut dapat dibangun suatu perangkat detektor yang mampu mendeteksi keberadaan *code smell* secara lebih komprehensif.

Selanjutnya komponen pendeteksian *by rules* yang sudah diuji pada penelitian ini akan diintegrasikan dengan komponen lain pada penelitian selanjutnya. Selain itu, transformator dari kode program menjadi format XML yang mampu mengolah kode program dalam berbagai bahasa pemrograman yang diajarkan pada mata kuliah pemrograman di Program Studi Teknik Informatika ITB juga dikembangkan dalam penelitian lain.

PUSTAKA

Aho, Alfred V. dan Lam, Monica S. (2007). *Compilers: Principle, Techniques, and Tools*. Addison Wesley.

Bisanz, Martin (2006). *Pattern-Bassed Smell Detection in TTCN-3 Test Suites*. Geor-August-Universität Göttingen, Göttingen, Jerman.

Burn, Oliver (2007). *Checkstyle 4.4*. Source Forge.
Diakses pada 22 Januari 2009 dari
http://checkstyle.sourceforge.net

Dutta, Shiv dan Hook, Gary (2003). *Best Practice for Programming in C*. IBM.
Diakses pada 25 Agustus 2008 dari
http://www.ibm.com/developerworks/aix/library/au-hook_duttaC.html

Fowler, Martin; Beck, Kent (2000). *Refactoring: Improving the Design of Existing Code*. Addison Wesley.

Geotechnical Sofware (2008). *C++ Programming Style Guidelines*. Geotechnical Sofware, Stavanger, Norwegia.
Diakses pada 19 Januari 2009 dari
http://geosoft.no/development/cppstyle.html

Henricson, Mat dan Nyquist, Erik (1992). *Programming in C++ Rules and Recomendations*. Ellemtel Telecomunication System Laboratories, Alvsjo, Swedia.

Liem, Inggriani (2000). *Standar Setoran Program*. Politeknik DEL, Medan, Indonesia.

Mäntylä, Mika (2003). *Bad Smells in Software – A Taxonomy and an Empirical Study*. Helsinki University of Technology, Helsinki, Finlandia.

Moha, Naouel dan Guéhéneuc, Yann-Gaël (2003). *DECOR: A Method for the Spesification and Detection of Code and Design Smells*. University of Montreal, Quebec, Kanada.

Cannon, L.W.; Elliott , R.A. (2008). *Recommended C Style and Coding Standards*. Sundland, Goddard Space Flight Center, NASA.
Diakses pada 2 November 2008 dari
http://sunland.gsfc.nasa.gov/info/cstyle.html

Slinger, Stevan (2005). *Code Smell Detection in Eclipse*. Delft University of Technology, Delft, Belanda.

Stallings, William (2002). *Network Security Essentials (2nd Edition)*. Prentice Hall.

SUN Microsystem (1999). *Java Code Conventions*. SUN Microsystem.
Diakses pada 19 Januari 2009 dari
http://java.sun.com/docs/codeconv/CodeConventions.pdf

van der Vlugt, Peter (2003). *C Style Guide and Programming Guidelines*. C Kernel Page, Belanda.
Diakses pada 3 Januari 2009 dari
http://www.chris-lott.org/resources/cstyle/Peter_CStyleGuide.pdf

Wirth, Niklaus (1973). *The Programming Language Pascal.* Eidgenössische Technische Hochschule, Zürich, Swiss.