Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK) p-ISSN: 2355-7699
Vol. 3, No. 4, Desember 2016, hlm. 292-299 e-ISSN: 2528-6579

REVIEW METODE PENDETEKSIAN GOD CLASS

Divi Galih Prasetyo Putri[1], Muhammad Shulhan Khairy[2], Siti Rochimah[3]

[1]Departemen Teknik Elektro dan Informatika, Sekolah Vokasi Universitas Gadjah Mada
[2,3]Jurusan Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember
Email: [1]divi.galih@ugm.ac.id, [2]khairy10@mhs.if.its.ac.id, [3]siti@if.its.ac.id

(Naskah masuk: 1 Oktober 2016, diterima untuk diterbitkan: 26 Desember 2016)

Abstrak

Code Smell mengacu pada konsep mengenai pola atau aspek desain pada sistem perangkat lunak yang dapat menimbulkan masalah dalam proses pengembangan, penggunaan, atau perawatan sebagai dampak dari implementasi yang buruk dari desain perangkat lunak. Code Smell dapat menurunkan aspek *understandability* dan *maintainability* program. Program yang mengandung God Class juga cenderung lebih sulit untuk dirawat dibandingkan dengan program yang sama namun tidak mengandung God Class. God Class atau dapat juga disebut Blob merupakan sebuah kelas yang terlalu banyak berisi fungsionalitas didalamnya. Kelas-kelas seperti ini mengolah dan mengakses banyak informasi sehingga sulit dipahami. Pada penelitian ini akan dibahas metode-metode untuk mendeteksi adanya God Class. Selain itu juga dibandingkan kelebihan serta kekurangan metode-metode yang telah dianalisa. Dari pencarian literatur yang dilakukan, didapatkan 3 buah metode, metode pertama menggunakan cara deteksi dalam bentuk *rule card*, metode kedua menggunakan *rule card* dan catatan histori perubahan pada sebuah perangkat lunak, dan metode ketiga adalah pendeteksian berdasarkan contoh kelas yang dideteksi manual sebagai kecacatan perangkat lunak. Dari ketiga metode tersebut, metode ketiga dinilai sebagai yang terbaik berdasarkan nilai presisi dan *recall*-nya.

Kata kunci: *Blob, God Class*

Abstract

Code smell referring to the concept about a pattern or design aspects on a software system that can make a problem in the process of development, using, or maintenance as the impact of bad implementation of software design. Code smell can lower software understandability and maintainability. A software that containing god class will be more difficult to maintain compared with a same software but doesn't have a god class. God class, also called blob is a class that has too many functionality. A god class process and access a lot of information. On this research will be discussed methods to detect a god class. We also compared the advantage and disadvantage about analysed method. From the literature we search, there are 3 methods, first method using detection with a rule card, the second method using rule card and history changes of a software, and the third method is detection by examples classes that detected manually as a software defect. And our research result is the third method is the best method based on its precision and recall.

Keywords: *Blob, God Class*

1. LATAR BELAKANG

Code Smell mengacu pada konsep mengenai pola atau aspek desain pada sistem perangkat lunak yang dapat menimbulkan masalah dalam proses pengembangan, penggunaan, atau perawatan sebagai dampak dari implementasi yang buruk dari desain perangkat lunak. Pada penelitian terdahulu telah dikaji mengenai dampak Code Smell terhadap frekuensi perubahan pada kode program. Code Smell dapat menurunkan aspek *understandability* dan *maintainability* program. Jika sebuah perangkat lunak sulit untuk dimengerti dan dirawat oleh pengembang selanjutnya maupun tim perawat perangkat lunak, maka dapat merugikan pengembang sistem karena proses *maintenance* merupakan proses yang sangat penting dengan biaya yang cukup besar.

Dalam sebuah penelitian diketahui bahwa program yang mengandung God Class dan *Shotgun Surgery* lebih sering mengalami perubahan. Program yang mengandung God Class juga cenderung lebih sulit untuk di-*maintenance* dibandingkan dengan program yang sama namun tidak mengandung God Class. Pada (Sjøberg, et al., 2013) aspek *maintenance* terkena dampak terbesar jika terdapat Code Smell, termasuk God Class. God Class atau dapat juga disebut Blob merupakan sebuah kelas yang berisi terlalu banyak fungsionalitas didalamnya. Kelas-kelas seperti ini mengolah dan mengakses banyak informasi sehingga sulit dipahami. Padahal seharusnya sebuah kelas memiliki fungsi yang spesifik, sehingga konsep modularitas dalam pemrograman berorientasi objek

tetap terjaga dan tidak menimbulkan dampak sulitnya perawatan perangkat lunak kedepannya. God Class juga salah satu jenis Code Smell yang paling sering muncul dalam perangkat lunak. Jenis lainnya adalah *Duplicate Code*, *Data Class*, *Schizophrenic Class* dan *Long Method*, hal ini dibahas pada penelitian (Fontana, et al., 2013). Selain itu terdapat juga penelitian (de F. Carneiro, et al., 2010) yang melakukan visualisasi adanya Code Smell, termasuk God Class.

Dari referensi literatur terkait pendeteksian Code Smell, telah banyak dikembangkan metode untuk mendeteksi adanya Code Smell pada sebuah sistem. *Paper* ini memiliki kontribusi dalam pembahasan metode-metode untuk mendeteksi adanya God Class. Selain itu juga membandingkan kelebihan serta kekurangan metode-metode yang telah dianalisa. Dari kontribusi ini diharapkan akan terdapat studi-studi selanjutnya tentang pendeteksian God Class agar permasalahan tersebut dapat diminimalisir untuk pengembang perangkat lunak kedepannya.

2. PENELITIAN TERKAIT

Telah dibahas mengenai 22 tipe dari Code Smell dengan karakteristik yang berbeda dan efek yang berbeda pada sistem, hal ini dibahas pada (Fowler, 1999). Berbagai macam Code Smell tersebut mempengaruhi struktur dari kode dan membuka peluang untuk dilakukan *refactoring*. Salah satu jenis Code Smell adalah God Class yang mengacu pada kelas yang menjadi sentral dari sistem, dibahas pada penelitian (Lanza, et al., 2005). Penelitian tersebut juga mengajukan sebuah matriks untuk mendeteksi God Class.

Pada penelitian sebelumnya (Zhang & Hall, 2011) telah dibuat studi literatur mengenai perkembangan penelitian terhadap Code Smell dari tahun 2000-2009. Penelitian lainnya (Hamid & Ilyas, 2013) menyusun sebuah studi literatur terhadap kakas bantu pendeteksi Code Smell. Disini dibandingkan kemampuan dua buah kakas bantu dalam mendeteksi dua jenis Code Smell yaitu Feature Envy dan God Class.

Dari beberapa penelitian terkait tersebut, belum ada penelitian yang secara khusus melakukan studi terhadap metode-metode pendeteksian Code Smell terutama untuk jenis God Class. Oleh karena itu, diharapkan penelitian ini dapat memberikan kontribusi.

3. METODE PENELITIAN

3.1. Pertanyaan Penelitian

Berdasarkan studi literatur pada penelitian-penelitian sebelumnya, *paper* ini bertujuan untuk mempelajari metode pendeteksian God Class dan melakukan analisis terhadap hasilnya. Untuk itu dibentuklah *research questions* sebagai berikut:

RQ1: Metode-metode apa saja yang telah dikembangkan untuk mendeteksi adanya God Class?
RQ2: Bagaimana hasil dalam pendeteksian God Class pada setiap metode?
RQ3: Apa saja efek yang ditimbulkan oleh God Class terhadap sistem?

3.2. Prosedur Pencarian

Sebelum melakukan proses studi literatur perlu dilakukan proses pencarian literatur yang memiliki kesesuaian dengan topik dan *research question* yang telah ditentukan. Pencarian dilakukan pada sumber-sumber elektronik berikut:

- IEEExplore (http://ieeexplore.ieee.org), dan
- Screince Direct (http://sciencedirect.com).

Pencarian hanya dilakukan pada dua sumber karena kedua sumber tersebut adalah sumber jurnal maupun *paper* konferensi yang banyak dikenal dan menjadi sumber rujukan penelitian dimanapun, peneliti juga hanya memiliki akses pada dua sumber elektronik tersebut. Untuk pencarian pada repositori literatur Sciencedirect dilakukan pencarian hanya untuk literatur dengan akses terbuka, karena literatur dengan akses tertutup tidak dapat dibuka kecuali memiliki akses penuh pada sumber tersebut. Berbeda halnya dengan literatur IEEE Xplore yang dapat diakses secara penuh dengan koneksi internet dalam kampus. Pada proses pencarian literatur ditetapkan beberapa kata kunci pencarian:

- "Software Evolution"
  AND
  "Bad Smell Code"
- "Software Evolution"
  AND
  "Bad Smell Detection"
- "Software Evolution"
  AND
  "Code Smell Detection"
- "Bad Smell Detection"
  AND
  "God Class"
- "Code Smell Detection"
  AND
  "God Class"
- "Software Quality"
  AND
  "God Class"

3.3. Proses Seleksi Studi

Pada subbab ini akan dibahas tentang hasil pencarian kata kunci diatas pada dua sumber elektronik yang telah diterangkan sebelumnya.

Tabel 1. Hasil pencarian literatur dengan penyaringan kata kunci

Sumber Elektronik	Kata Kunci	Hasil Pencarian
IEEExplore	Software Evolution AND Bad Smell Code	12
Science Direct		4
	Total	16
IEEExplore	Software Evolution AND Bad Smell Detection	3
Science Direct		4
	Total	7
IEEExplore	Software Evolution AND Code Smell Detection	9
Science Direct		11
	Total	20
IEEExplore	Bad Smell Detection AND God Class	0
Science Direct		26
	Total	26
IEEExplore	Code Smell Detection AND God Class	3
Science Direct		25
	Total	28
IEEExplore	Software Quality AND God Class	7
Science Direct		6
	Total	13
	Total	110

Total hasil pencarian secara keseluruhan berjumlah 110 dokumen. Pencarian ini hanya dilakukan dengan penyaringan kata kunci. Kemudian dilakukan seleksi berdasarkan *inclusion* dan *exclusion criteria* sebagai berikut.

Tabel 2. Daftar *inclusion criteria* dan *exclusion criteria*

Inclusion Criteria	*Exclusion Criteria*
Literatur merupakan *paper* jurnal atau *paper conference*.	Literatur dengan topik yang sama.
Literatur diterbitkan pada rentang tahun 2010-2014	Literatur sebelum tahun 2010.
Literatur menggunakan bahasa inggris.	Literatur yang tidak berkaitan dengan God Class dan metode deteksinya.
Literatur dapat diakses secara *online*.	
Literatur yang berkaitan dengan God Class dan metode deteksinya.	

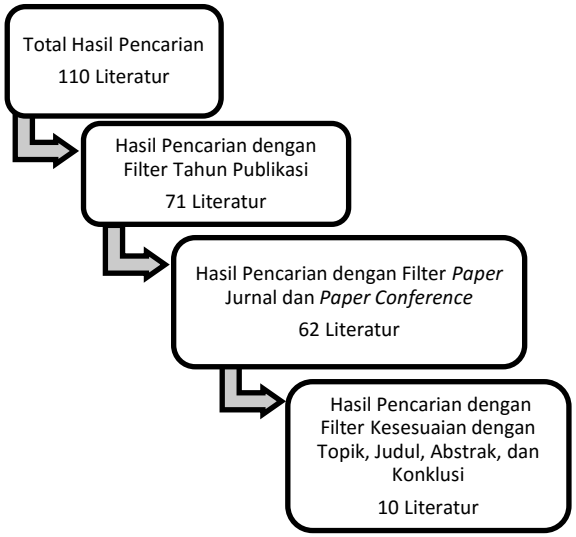
Tabel 3. Hasil pencarian literatur dengan penyaringan tambahan tahun publikasi

Sumber Elektronik	Kata Kunci	Tahun Publikasi	Hasil
IEEExplore	Software Evolution AND Bad Smell Code	2010-Sekarang	3
Science Direct			8
	Total		11
IEEExplore	Software Evolution AND Bad Smell Detection	2010-Sekarang	2
Science Direct			3
	Total		5
IEEExplore	Software Evolution AND Code Smell Detection	2010-Sekarang	5
Science Direct			6
	Total		11
IEEExplore	Bad Smell Detection AND God Class	2010-Sekarang	0
Science Direct			19
	Total		19
IEEExplore	Code Smell Detection AND God Class	2010-Sekarang	2
Science Direct			18
	Total		20
IEEExplore	Software Quality AND God Class	2010-Sekarang	1
Science Direct			4
	Total		5
		Total	71

Tabel 4. Hasil pencarian literatur dengan penyaringan tambahan jenis literatur

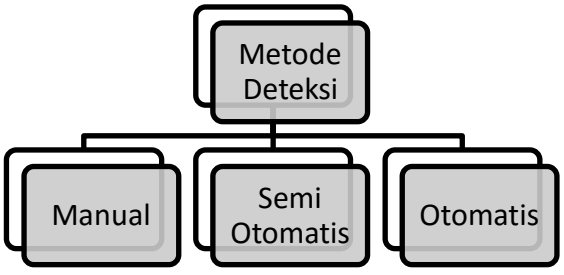
Sumber Elektronik	Kata Kunci	Tahun Publikasi	Filter *Paper*	Hasil
IEEExplore	Software Evolution AND Bad Smell Code	2010-Sekarang	Ya	8
Science Direct				3
	Total			11
IEEExplore	Software Evolution AND Bad Smell Detection	2010-Sekarang	Ya	2
Science Direct				3
	Total			5
IEEExplore	Software Evolution AND Code Smell Detection	2010-Sekarang	Ya	5
Science Direct				6
	Total			11
IEEExplore	Bad Smell Detection AND God Class	2010-Sekarang	Ya	0
Science Direct				16
	Total			16
IEEExplore			Ya	2

Science Direct	Code Smell Detection AND God Class	2010-Sekarang		12
	Total			14
IEEExplore	Software Quality AND God Class	2010-Sekarang	Ya	1
Science Direct				4
	Total			5
Total		62		



Gambar 1. Diagram hasil penyaringan literatur

Dari seleksi yang dilakukan berdasarkan tahun publikasi, tipe *paper*, dan kesesuaian judul maupun abstrak dengan topik didapatkan jumlah *paper* yang akan digunakan untuk dilakukan analisa sebanyak 10 buah *paper*. Penyeleksian *paper* dilakukan dengan tahun publikasi mulai 2010 sampai saat ini (2014). Kemudian dari jumlah yang didapat dari seleksi pertama dilakukan seleksi referensi, yang diambil adalah *paper* jurnal ilmiah dan *paper* konferensi. Hal ini dilakukan karena penulis memandang bahwa topik yang akan dibahas seharusnya berkaitan dengan perkembangan beberapa tahun terakhir, dan pastinya penelitian yang berkembang beberapa tahun terakhir sudah mencakup penelitian pada tahun-tahun sebelumnya.



Gambar 2. Skema pembagian kategori metode deteksi God Class

Tabel 5. Daftar literatur utama

Literatur	
Detecting Bad Smell in Source Code Using Change History Information	(Palomba, et al., 2013)
DECOR: A method for the specification and detection of code and design smells	(Moha, et al., 2010)
Design Defect Detection and Correction by Example	(Kessentini, et al., 2011)
Mining Version History for Detecting Code Smell	(Palomba, et al., 2014)

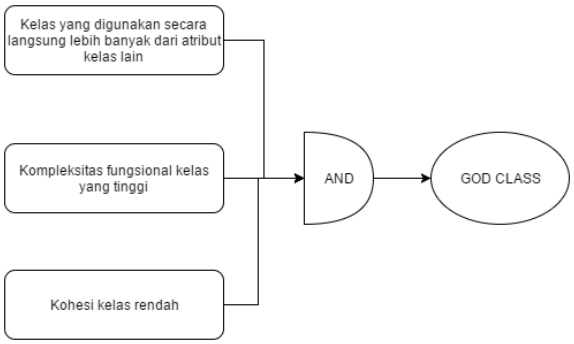
Setelah itu penyeleksian didasarkan pada kesesuaian literatur yang ada dengan topik yang dibahas, kemudian dilanjutkan dengan kesesuaian topik dengan abstraksi, dan isi secara umum. Literatur mengenai metode deteksi God Class yang akan digunakan dalam *review* ini adalah metode-metode yang tergolong *semi-automated* atau *automated*. Sedangkan metode yang tergolong manual tidak digunakan dalam *review*. Seleksi studi atau dokumen pada dasarnya terdiri dari beberapa tahap. Menghilangkan kesamaan isi atau metode antar dokumen dan melakukan penyaringan yang lebih dalam mengenai faktor-faktor eksklusi dan inklusi yang telah didefinisikan tidak dapat dilakukan hanya dengan memilah berdasarkan tahun publikasi dan jenis *paper*. Oleh karena itu, perlu dilakukan studi lebih dalam dengan mengkaji judul, abstrak, metode, hasil, dan konklusi dari setiap literatur sehingga didapatkan literatur yang paling sesuai untuk kemudian dapat di-*review*.

Beberapa dokumen pada Tabel 5 merupakan hasil dari studi kualitas dan literatur-literatur ini yang akan dijadikan literatur utama untuk melakukan *review*.

4. GOD CLASS

God Class dapat diartikan sebagai kelas yang menjadi pusat dari sistem. Kelas ini menjalankan sebagian besar pekerjaan dan hanya mendelegasikan peran-peran kecil pada kelas lain. Kelas ini juga banyak menggunakan data dari kelas lain.

Pendeteksian God Class memiliki 3 faktor, yaitu penggunaan kelas secara langsung yang lebih banyak daripada beberapa atribut dari kelas lain, kompleksitas fungsional sebuah kelas yang tinggi, dan kohesi antar kelas rendah. Dari ketiga faktor tersebut apabila dipenuhi, maka sebuah kelas dapat dikatakan terdapat Code Smell jenis God Class.



Gambar 3. Faktor yang menjadi parameter untuk mendeteksi God Class

5. METODE DETEKSI GOD CLASS

5.1. DETEX

DETEX adalah sebuah teknik deteksi yang memungkinkan *software engineers* untuk mendefinisikan Code Smell secara spesifik pada abstraksi tingkat tinggi menggunakan *unified vocabulary* dan *domain-specific language* yang didapatkan dari domain analisis. Teknik ini digunakan untuk menghasilkan algoritma deteksi secara otomatis. Berikut ini adalah langkah-langkah dalam melakukan deteksi:

- *Domain analysis*

Pada langkah ini dilakukan analisa secara mendalam terhadap domain dari Code Smell untuk menentukan konsep utama dari deskripsi yang diberikan. Dari konsep utama tersebut didapatkan *vocabulary*, *taxonomy*, dan klasifikasi dari Code Smell. Pada Gambar 4 dan Gambar 5 adalah contoh konsep utama dan *taxonomy*.

- *Specification*

Spesifikasi dilakukan menggunakan Domain Specification Language (DSL) dalam bentuk *rule card* berdasarkan *vocabulary* dan *taxonomy* yang didapatkan dari proses sebelumnya. *Rule card* mendefinisikan properti yang harus dimiliki oleh sebuah kelas untuk dapat digolongkan sebagai Code Smell. Penggunaan DSL memungkinkan software engineer atau expert untuk menentukan sendiri spesifikasi *rule* dari Code Smell.

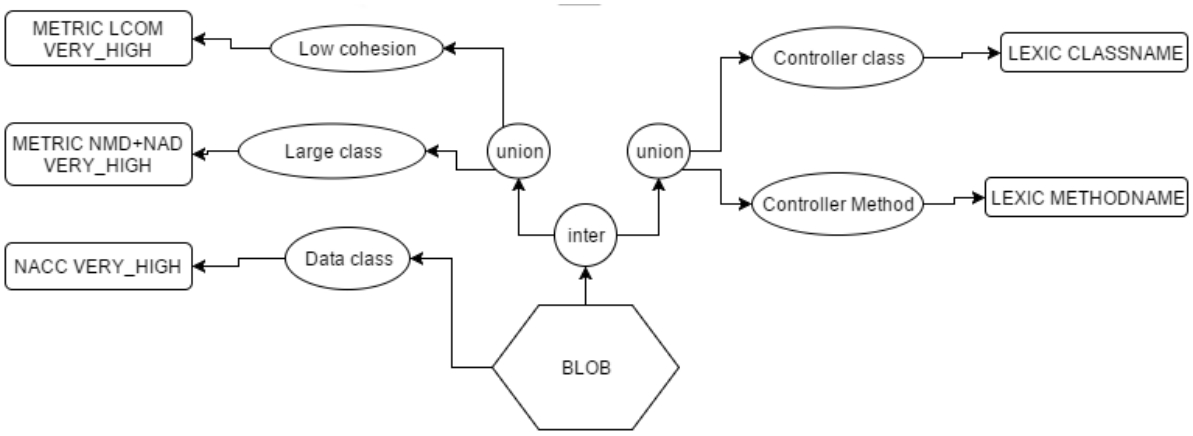
- *Algorithm Generation*

Algoritma dibuat secara otomatis menggunakan metamodel dan *parser*. Sebuah *framework* digunakan untuk memungkinkan algoritma dibuat secara otomatis.

- *Detection*

Deteksi dilakukan pada model dari sistem yang didapatkan baik melalui proses *forward engineering* ataupun *reverse engineering* pada sebuah sistem.

Validasi pada metode ini dilakukan oleh *independent engineer* untuk memastikan apakan sebuah kelas memang merupakan Code Smell. Subjek validasi adalah 4 *antipattern* beserta spesifikasinya yang didapatkan dari (Brown, et al., 1998) dan (Fowler, 1999). Spesifikasi yang didapatkan tersebut kemudian diproses secara otomatis untuk membentuk algoritma deteksi.Validasi dilakukan terhadap model hasil *reverse engineering* dari 10 *open source* sistem dengan bahasa pemrograman java, antara lain: ARGOUML, AZUREUS, GANTTPROJECT, LOG4J, LUCENE, NUTCH, PMD, QUICKUML, dan 2 versi dari XERCES. Hasil validasinya menunjukkan bahwa metode ini memiliki presisi sebesar 88.6% dan recall sebesar 100%.



Gambar 4. Contoh taksonomi metode DETEX

The Blob (also called God class) corresponds to a large controller class that depends on data stored in surroundung data classes. A large class declares many fields and methods with a low cohesion. A controller class monopolises most of the processing done by a system, takes most of the decisions, and closely directs the processing of other classes. Controller classes can be identified using suspicious names such as Process, Control, Manage, System, and so on. A data class contains only data and performs no processing on these data. It is composed of highly cohesive fields and accesors.

Gambar 5. Contoh konsep utama metode DETEX

```
RULE_CARD : Blob {
   RULE : Blob {ASSOC: associated FROM: mainClass ONE TO: DataClass MANY } ;

   RULE : mainClass {UNION LargeClassLowCohesion ControllerClass } ;

   RULE : LargeClassLowCohesion {UNION LargeClass LowCohesion } ;

   RULE : LargeClass { (METRIC: NMD + NAD, VERY_HIGH, 20) } ;

   RULE : LowCohesion { (METRIC: LCOM5, VERY_HIGH, 20) } ;

   RULE : ControllerClass {UNION (SEMANTIC: METHODNAME, {Process, Control, Ctrl,
Command, Cmd, Process, Proc, UI, Manage, Drive})
                                 (SEMANTIC: CLASSNAME,
{Process, Control, Ctrl, Command, Cmd, Process, Proc, UI, Manage, Drive, System,
Subsystem})} ;

   RULE : DataClass {(STRUCT: METHOD_ACCESSOR, 90) } ;

} ;
```

Gambar 6. Rule card yang digunakan untuk deteksi God Class

5.2. History Changes Analysis

Metode ini dijalankan berdasarkan *history data* pada setiap perubahan yang dilakukan dalam sebuah program. Hipotesa yang diajukan adalah bahwa ketika sebuah perubahan terjadi, kelas yang tergolong Blob atau God Class juga akan mengalami perubahan sebanyak α% dan melibatkan paling tidak satu kelas lain. HIST memanfaatkan analisa terhadap histori program untuk mendapatkan data-data yang diperlukan dalam melakukan deteksi. Untuk melakukan deteksi terhadap Blob atau God Class digunakan *rule card* pada Gambar 6.

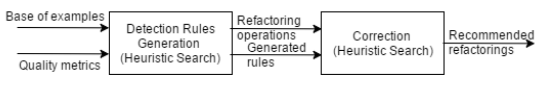
5.3. Detection Defect by Example

Metode ini memanfaatkan *repository* yang berisi data *code* yang sudah digolongkan sebagai defect atau smell. Dari data yang didapat kemudian dihasilkan aturan pendeteksian berbentuk kombinasi dari matriks kualitas perangkat lunak. Skema pendeteksiannya dapat dilihat pada Gambar 8. Metode ini memanfaatkan *Genetic Programming* untuk mencari aturan yang sesuai. Populasi pada algoritma ini merepresentasikan kemungkinan aturan yaitu berupa kombinasi matriks kualitas. Seperti algoritma genetic pada umumnya, populasi pertama yang telah terbentuk dihitung nilai *fitness*-nya. Setelah itu, individu dengan nilai *fitness* tertinggi akan digunakan dalam proses *crossover*. Hasil dari proses *crossover* kemudian dimutasi untuk menghasilkan populasi yang baru. Proses ini dilakukan sampai memenuhi batas iterasi dan dihasilkan aturan terbaik untuk melakukan deteksi. Fungsi yang digunakan untuk melakukan perhitungan *fitness value* pada metode ini dapat dilihat pada persamaan dibawah ini.

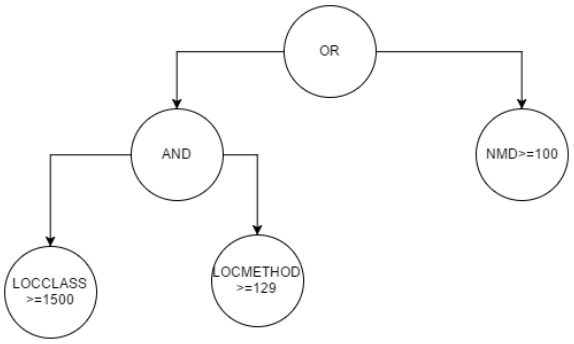
Pada Persamaan (1) t adalah jumlah kelas yang dideteksi, p adalah jumlah kelas contoh yang tergolong *bad smell*. a bernilai 1 ketika kelas yang terdeteksi terdapat pada contoh, dan sebaliknya untuk a yang bernilai 0.

Validasi untuk metode ini dilakukan terhadap 4 *open source* sistem dengan bahasa pemrograman java antara lain, Xerces-J, ArgoUML,Quick UML, dan GanttProject. Prosedur validasinya menggunakan 4-*fold cross validation*, yaitu teknik model validasi untuk menilai hasil analisis statistikal yang menghasilkan dataset independen. Hasil deteksi sebuah sistem dievaluasi menggunakan 3 sistem lain sebagai contoh *defect*. Nilai presisi dan *recall* yang dihasilkan dari metode ini sebesar 96% dan 99.5%.Pada penelitian ini validasi dilakukan oleh beberapa mahasiswa pascasarjana untuk memastikan hasil deteksi dengan dasar definisi dari setiap Code Smell. Deteksi dilakukan terhadap *git history* dari 8 *project* java antara lain: Apache Ants, Apache Tomcat, Jedits, dan 5 *project* Android API. Hasil deteksi menunjukkan tingkat presisi dan *recall* dari metode sebesar 76% dan 61%.

$$f_{norm} = \frac{\frac{\sum_{i=1}^{p} a_i}{t} + \frac{\sum_{i=1}^{p} a_i}{p}}{2} \quad (1)$$



Gambar 7. Skema pendeteksian *defect*



Gambar 8. Tree yang digunakan untuk merepresentasikan *detection rule*

6. Hasil Komparasi

Dapat dilihat pada Tabel 6 bahwa metode dengan nilai presisi dan *recall* terbaik berhasil didapatkan oleh metode ketiga, yaitu pendeteksian God Class secara otomatis berdasarkan contoh kelas yang terdeteksi manual sebagai *defect*. Berdasarkan studi yang dilakukan, hal ini disebabkan pembangunan aturan pendeteksian dilakukan berdasarkan contoh kelas yang sudah dideteksi sebagai God Class. Semakin baik dan lengkap data yang digunakan sebagai contoh maka aturan pendeteksian akan menjadi semakin lengkap dan mampu mendeteksi dengan lebih baik. Namun hal ini juga yang menjadi kekurangan utama dari metode ini, yaitu aturan yang dihasilkan bergantung pada data training yang digunakan.

7. DAMPAK DARI GOD CLASS

God Class berpotensi membahayakan desain sebuah sistem karena berupa agregasi dari berbagai abstraksi dan kelas-kelas lain baik yang digunakan maupun tidak. Hal ini seringkali dikarenakan banyaknya penggunaan kelas yang tidak mematuhi kaidah pemrograman berbasis objek, dimana satu kelas seharusnya hanya memiliki sebuah tugas tertentu.

Dari implementasi yang tidak mengikuti kaidah pemrograman berbasis objek akan mengakibatkan terhambatnya evolusi perangkat lunak karena struktur perangkat lunak akan sulit dimengerti dan dirawat kedepannya. Dampak lainnya adalah tingginya biaya perawatan perangkat lunak, hal ini diakibatkan sulitnya perawatan yang berkorelasi dengan tingginya biaya perawatan.

8. KESIMPULAN

Studi literatur dilakukan untuk mengetahui metode-metode pendeteksi God Class dan dampak dari God Class tersebut pada sistem. Dari proses seleksi literatur yang dilakukan, didapatkan 3 metode untuk mendeteksi salah satu jenis Bad Smell, God Class. Metode-metode tersebut adalah metode pendeteksian dengan dasar pendeteksian berupa aturan deteksi. Perbedaan antar metode tersebut adalah proses pendeteksiannya, terdapat metode yang dapat mendeteksi God Class secara semi-otomatis dan terdapat juga yang dapat mendeteksi

Tabel 6. Hasil komparasi metode

Literatur	Metode	Hasil	Input & Output	Dataset	Imple-mentasi	Kekurangan	Kelebihan
(Moha, et al., 2010) (Palomba, et al., 2013) (Olbrich & Cruzes, 2010)	Algoritma deteksinya dibuat dalam bentuk *rule card* menggunakan DSL yang didapatkan dari analisis domain setiap bad smell secara mendalam.	*Precision*: 52% *Recall*: 49%	Input: Deskripsi mengenai Code Smell Output:	10 Open source java system	Ya	Pembangunan *rule card* secara manual membutuhkan usaha yang lebih	*Expert* dan *engineer* dapat dengan mudah menspesifikasi-kan aturan pendeteksian.
(Palomba, et al., 2013) (Olbrich & Cruzes, 2010) (Palomba, et al., 2014)	Memanfaatkan *rule card* pada (Moha, et al., 2010) dan data perubahan yang terjadi pada software.	*Precision*: 76% *Recall*: 61%	Input: Versioning system address Output: Affected Components	8 Java Project	Ya	Tidak bisa mendeteksi kelas yang belum pernah mengalami perubahan	Nilai presisi yang lebih baik dari algoritma sebelumnya.
(Kessenti ni, et al., 2011)	Membangun aturan pendeteksian secara otomatis berdasarkan contoh kelas yang terdeteksi manual sebagai *defect*.	*Precision*: 96% *Recall*: 99.5%	Input: Contoh *defect class* & quality metrics Output: Generated rules	4 Open source java project	Tidak	Aturan-aturan yang dihasilkan sangat bergantung pada data training	Aturan pendeteksian dibangun secara otomatis.

secara otomatis. Pada metode DECOR (Moha, et al., 2010) menggunakan cara deteksi semi-otomatis dalam pembentukan algoritma deteksi dari rule card. Metode ini menghasilkan prosentase presisi sebesar 88.6% dan recall 100%. Metode kedua (Palomba, et al., 2013) menggunakan rule card dan catatan histori perubahan pada sebuah perangkat lunak. Hasil yang didapat adalah prosentase presisi sebesar 76% dan recall 61%. Metode ketiga (Kessentini, et al., 2011) adalah pendeteksian berdasarkan contoh class yang dideteksi manual sebagai defect perangkat lunak. Hasil dari metode ini adalah yang terbaik dibandingkan dengan kedua metode sebelumnya yaitu dengan prosentase presisi 96% dan recall 99.5%. God Class memiliki dampak terhadap *understandability* dan *maintainability* yang mempengaruhi biaya yang dikeluarkan untuk perawatan.

9. DAFTAR PUSTAKA

BROWN, W. J., MALVEAU, R. C. & MOWBRAY, T. J., 1998. Anti Patterns: Refactoring Software, Architectures, and Project in Crisis. s.l., s.n.

DE F. CARNEIRO, G. et al., 2010. Identifying Code Smells with Multiple Concern Views. s.l., s.n.

FONTANA, F. A. et al., 2013. Investigating the Impact of Code Smells on System's Quality: An Empirical Study on Systems of Different Application Domains. s.l., s.n.

FOWLER, M., 1999. Refactoring-Improving the Design of Existing Code. s.l., s.n.

HAMID, A. & ILYAS, M., 2013. A Comparative Study on Code Smell Detection Tools. s.l., s.n.

KESSENTINI, M., KESSENTINI, W. & SAHRAOUI, H., 2011. Design Defect Detection and Correction by Example. s.l., s.n.

LANZA, M., MARINESCU, R. & S, D., 2005. Object Oriented Metrics in Practice. New York: Springer.

MOHA, N., GUE´HE´NEUC, Y.-G. & DUCHIEN, L., 2010. DECOR: A method for the specification and detection of code and design smells. s.l., s.n.

OLBRICH, S. M. & CRUZES, D. S., 2010. Are all Code Smells Harmful? A Study of God Classes and Brain Classes in the Evolution of three Open sources Systems. s.l., s.n.

PALOMBA, F., BAVOTA, G. & PENTA, M. D., 2013. Detecting Bad Smell in Source Code Using Change History Information. s.l., s.n.

PALOMBA, F., BAVOTA, G. & PENTA, M. D., 2014. Mining Version Histories for Detecting Code Smells. s.l., s.n.

SJØBERG, D. I. et al., 2013. Quantifying the Effect of Code Smells on Maintenance Effort. IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, Volume 39, pp. 1144-1196.

ZHANG, M. & HALL, T., 2011. Code Bad Smells: a review of current knowledge. Journal of Software Maintenance and Evolution, 23(3), pp. 179-202 .