Prediksi *Code Defect* Perangkat Lunak Dengan Metode *Association Rule Mining* dan *Cumulative Support Thresholds*

Rizal Setya Perdana[1], Umi Laili Yuhana[2]
Program Studi Teknik Informatika, Institut Teknologi Sepuluh Nopember
Kampus ITS Keputih, Sukolilo, Surabaya 60111
E-mail: [1]rizalespe@ub.ac.id, [2]yuhana@if.its.ac.id

Masuk: 13 Juni 2014; Direvisi: 3 Juli 2014; Diterima: 20 Juli 2014

Abstract. *Software quality is a research field in software engineering that has big effects in development process which result in good software. Researches on software of defect prediction caused by specification fault or fault operation have been done for more than 30 years in software engineering. This research focuses on software code defect prediction. To solve the problem of disability program code, this paper will utilize the code patterns which software potentially causes disability in NASA data sets in predicting disability. The method used in the search pattern is utilizing Association Rule Mining with Cumulative Support Thresholds that automatically generates values of support and confidence the optimal value without requiring input from the testing of the results. Result disability automatically software code has a value of 82.35% accuracy.*
Keywords: *software quality, defect prediction, association rule, cumulative support threshold.*

Abstrak. *Kualitas perangkat lunak merupakan salah satu penelitian pada bidang rekayasa perangkat lunak yang memiliki peranan yang cukup besar dalam terbangunnya sistem perangkat lunak yang berkualitas baik. Prediksi defect perangkat lunak yang disebabkan karena terdapat penyimpangan dari proses spesifikasi atau sesuatu yang mungkin menyebabkan kegagalan dalam operasional telah lebih dari 30 tahun menjadi topik riset penelitian. Makalah ini akan difokuskan pada prediksi defect yang terjadi pada kode program (code defect). Metode penanganan permasalahan defect pada kode program akan memanfaatkan pola-pola kode perangkat lunak yang berpotensi menimbulkan defect pada data set NASA untuk memprediksi defect. Metode yang digunakan dalam pencarian pola adalah memanfaatkan Association Rule Mining dengan Cumulative Support Thresholds yang secara otomatis menghasilkan nilai support dan nilai confidence paling optimal tanpa membutuhkan masukan dari pengguna. Hasil pengujian dari hasil pemrediksian defect kode perangkat lunak secara otomatis memiliki nilai akurasi 82,35%.*
Kata kunci: *Kualitas Perangkat Lunak, Prediksi Code Defect, Association Rule, Cumulative Support Threshold.*

1. Pendahuluan

Keberhasilan dari sistem perangkat lunak tidak hanya bergantung pada penentuan biaya yang besar dan manajemen sumber daya yang baik, namun hal yang paling penting yang harus diperhatikan adalah kualitas perangkat lunak itu sendiri. Kualitas perangkat lunak merupakan salah satu penelitian pada bidang rekayasa perangkat lunak yang memiliki peranan yang cukup besar dalam terbangunnya sistem perangkat lunak yang berkualitas baik. Oleh sebab itu menurut (Okumoto, 2011) disebutkan bahwa prediksi pada *defect* perangkat lunak yang disebabkan karena terdapat penyimpangan dari proses spesifikasi atau sesuatu yang mungkin menyebabkan kegagalan dalam operasional telah lebih dari tiga puluh tahun menjadi topik riset penelitian dalam bidang rekayasa perangkat lunak.

Menurut (Runeson dkk, 2006) *defect* perangkat lunak terdiri dari *defect* kebutuhan (*requirement defect*), defect desain (*design defect*), dan defect kode (*code defect*). Pada makalah ini akan difokuskan pada prediksi *defect* yang terjadi pada kode program yang menyebabkan turunnya kualitas perangkat lunak saat proses pengembangan perangkat lunak. Metode yang digunakan dalam penelitian untuk mendeteksi *defect* adalah dengan memanfaatkan pola-pola kode perangkat lunak yang berpotensi menimbulkan *defect* dengan metode *data mining* yaitu *association rule mining*.

Telah banyak metode untuk memprediksi *defect* kode (Runeson dkk, 2006) antar metode tersebut terkait efisiensi dalam melakukan pemrediksian yang masih belum optimal. Hal ini mengindikasikan bahwa prediksi *defect* pada kode perangkat lunak membutuhkan perhatian dan metode yang lebih efisien dalam memprediksi. Apabila dilihat dari sudut pandang metode pencarian *defect* pada perangkat lunak, maka menurut (Shepperd dkk, 2013) terdapat tiga jenis sudut pandang pencarian tersebut. Pertama, *defect* perangkat lunak dapat ditemukan dengan metode pendekatan tata bahasa yaitu menggunakan BNF dan mudah ditemukan oleh *compiler*. Kedua, *defect* perangkat lunak dapat ditemukan dengan mengetahui pola kode perangkat lunak yang berpotensi menimbulkan *defect* dengan data diperoleh dari hasil eksperimen. Ketiga, *defect* perangkat lunak ditemukan dengan memanfaatkan data-data yang didapatkan saat proses penentuan kebutuhan perangkat lunak yang nantinya digunakan sebagai panduan dalam proses *software testing*. Pada makalah ini akan berfokus pada sudut pandang kedua yaitu dengan memanfaatkan pola-pola kode perangkat lunak yang berpotensi menimbulkan *defect*. Metode yang digunakan pada penelitian ini dalam mencari pola adalah metode *data mining*.

Data mining adalah proses mencari informasi atau pola dari sejumlah banyak data sehingga dapat diketahui pola-pola tertentu yang masih tersembunyi untuk pengambilan keputusan. Salah satu aplikasi yang memanfaatkan *data mining* adalah proses pemrediksian terhadap kondisi yang belum terjadi dan dibutuhkan untuk mengetahui kondisi yang akan terjadi. Pada makalah ini akan digunakan metode *data mining* dalam memprediksi pola kode perangkat lunak yang berpotensi terjadi *defect*. Metode *association rule mining* merupakan metode dalam pencarian kondisi terjadinya nilai atribut dari data yang muncul bersama-sama. Pada penelitian sebelumnya (Okumoto, 2011) digunakan metode *association rule mining* untuk melihat pola-pola perangkat lunak yang berpotensi terjadi *defect* dengan menambahkan konstrain berupa relasi antar atribut untuk menaikkan tingkat akurasi. Terdapat permasalahan yang berpotensi untuk diteliti lebih lanjut adalah sejauh mana pengaruh jumlah fitur yang diperhitungkan, nilai *confidence*, dan *support* terhadap akurasi saat proses klasifikasi. Hal ini dikarenakan nilai *confidence* dan *support* tidak secara otomatis muncul serta masukan nilai *confidence* dan *support* dari pengguna belum tentu menghasilkan akurasi pendeteksian *defect* secara akurat. Cara untuk menangani permasalahan tersebut maka diperlukan pengotomatisan dalam menentukan kedua nilai tersebut sehingga pola *defect* perangkat lunak yang muncul merupakan pola yang paling optimal dan otomatis.

Metode *association rule mining* merupakan metode dalam pencarian kondisi terjadinya nilai atribut dari data yang muncul bersama-sama. Pada penelitian sebelumnya (Okumoto, 2011) digunakan metode *association rule mining* untuk melihat pola-pola perangkat lunak yang berpotensi terjadi *defect* dengan menambahkan konstrain berupa relasi antar atribut untuk menaikkan tingkat akurasi. Secara umum dalam menggunakan metode *association rule*, metode ini membutuhkan nilai *support* dan *confidence* yang ditentukan secara manual. Penentuan kedua nilai tersebut mempengaruhi hasil pencarian pola *defect*, sehingga untuk melakukan otomatisasi penentuan nilai *support* dan nilai *confidence* dibutuhkan langkah tambahan yaitu dengan melakukan perhitungan *cumulative support threshold* (Selvi dan Tamilarasi, 2009) yang akan dijelaskan secara mendetail pada bagian selanjutnya.

Pada penelitian ini dibahas mengenai metode yang dapat secara otomatis mendeteksi *defect* yang ada pada kode perangkat lunak dengan memanfaatkan metode *data mining* untuk melakukan pendeteksian secara otomatis. Hal ini sangat berguna untuk mengurangi biaya dalam pengembangan perangkat lunak yang sebelumnya *defect* diketahui saat sistem sudah berjalan dengan pendeteksian ini akan lebih cepat diketahui bagian yang terjadi *defect*.

2. Tinjauan Pustaka

2.1 *Defect Perangkat Lunak*

Defect perangkat lunak (*software defect*) didefinisikan sebagai defect pada perangkat lunak yang mungkin terjadi *defect* pada kode program, *defect* pada dokumentasi, pada desain, dan hal-hal lain yang menyebabkan kegagalan perangkat lunak. *Defect* perangkat lunak dapat muncul pada berbagai tahap proses pengembangan perangkat lunak (Pressman, 2001). *Defect* perangkat lunak merupakan faktor penting yang mempengaruhi kualitas perangkat lunak. Kualitas perangkat lunak dapat ditingkatkan dengan mencegah munculnya *defect* perangkat lunak melalui perbaikan yang mungkin menghasilkan *defect* perangkat lunak pada proses pengembangan perangkat lunak (Boehm dan Basili, 2001). Menurut (Runeson dkk, 2006) *defect* perangkat lunak terdiri dari *defect* kebutuhan (*requirement defect*), *defect* desain (*design defect*), dan *defect* kode (*code defect*).

2.2 Prediksi *Defect* Kode

Prediksi *defect* pada perangkat lunak (Okumoto, 2011) diasumsikan sebagai jumlah *defect* pada perangkat lunak yang dapat ditemukan pada kondisi tertentu. Sehingga apabila ditemukan *defect* maka akan diketahui untuk ditemukan solusi agar tidak menyebabkan kegagalan sistem yang lebih besar. Pada tingkat yang lebih tinggi apabila *defect* perangkat lunak tidak ditangani, maka akan muncul permasalahan yang lebih besar yaitu *system failure* yang menyebabkan berhentinya sistem keseluruhan. Terdapat beberapa macam *defect* pada perangkat lunak yang pada masing-masing *defect* tersebut memiliki fokus masing-masing dalam penelitian. Pada (Czibula dkk, 2014) disebutkan bahwa prediksi *defect* perangkat lunak digunakan adalah proses yang mengidentifikasi *defect* yang terdapat pada modul atau bagian yang terdapat pembaharuan dari sistem perangkat lunak yang memungkinkan terjadi kesalahan. Prediksi *defect* perangkat lunak merupakan proses yang terjadi secara otomatis dalam perangkat lunak yang sampai saat ini merupakan fokus bahasan yang dilakukan oleh peneliti dan komunitas untuk mendapatkan keakuratan dalam melakukan pemrediksian.

Selain diharapkan dapat selesai tepat waktu, perangkat lunak juga membutuhkan proses pemantauan, deteksi *defect* berkelanjutan pada semua tahap proses pengembangan sehingga tidak menyebabkan kesalahan pada fase selanjutnya. *Defect* seperti *fault* atau *bug* merepresentasikan faktor utama dalam perencanaan untuk dapat selesai tepat waktu dan kualitas produk perangkat lunak selama tahap perawatan dan perubahan (evolusi) perangkat lunak.

Pada (Czibula dkk, 2014) disebutkan bahwa prediksi *defect* perangkat lunak digunakan adalah proses yang mengidentifikasi *defect* yang terdapat pada modul atau bagian yang terdapat pembaharuan dari sistem perangkat lunak yang memungkinkan terjadi kesalahan. Prediksi *defect* perangkat lunak merupakan proses yang terjadi secara otomatis dalam perangkat lunak yang sampai saat ini merupakan fokus bahasan yang dilakukan oleh peneliti dan komunitas untuk mendapatkan keakuratan dalam melakukan pemrediksian.

Kualitas perangkat lunak memiliki keterkaitan yang kuat dengan tidak adanya *defect* yang muncul, semakin banyak *defect* yang muncul maka akan semakin rendah kualitas dari perangkat lunak. Hal ini menjadi sangat penting untuk diperhatikan oleh pengembang maupun manajer proyek dalam menjamin terdapat *defect* sesedikit mungkin. Prediksi *defect* perangkat lunak membantu pendeteksian, penelusuran, dan memperbaiki ketidaknormalan yang disebabkan oleh kesalahan yang dilakukan saat pengembangan. Prediksi *defect* perangkat lunak dapat membantu apabila terjadi perubahan maka akan dengan mudah mengambil keputusan saat itu juga tanpa harus menunggu kesalahan tersebut berdampak pada masalah yang lain. Hal ini mengindikasikan bahwa dengan menerapkan pendeteksian *defect* perangkat lunak akan menekan biaya pengembangan serta menaikkan kepuasan klien.

2.3 *Association Rule*

Data mining adalah proses mencari informasi atau pola dari sejumlah banyak data sehingga dapat diketahui pola-pola tertentu yang masih tersembunyi untuk pengambilan keputusan. Salah satu aplikasi yang memanfaatkan *data mining* adalah proses pemprediksian

terhadap kondisi yang belum terjadi dan dibutuhkan untuk mengetahui kondisi yang akan terjadi. Prediksi pada kondisi tersebut dapat dilakukan dengan memanfaatkan data atau fakta yang sudah ada atau prediksi langsung dengan mengenali pola-pola yang ingin ditemukan. Metode *association rule mining* merupakan metode dalam pencarian kondisi terjadinya nilai atribut dari data yang muncul bersama-sama. Metode ini melakukan pencarian pada hubungan yang memiliki nilai dalam kumpulan data. Pada penelitian sebelumnya (Okumoto, 2011) digunakan metode *association rule mining* untuk melihat pola-pola perangkat lunak yang berpotensi terjadi *defect* dengan menambahkan konstrain berupa relasi antar atribut untuk menaikkan tingkat akurasi. Penemuan pola asosiasi yang paling sering muncul, maka dibutuhkan nilai yang harus ditentukan oleh pengguna yaitu nilai *support* dan *confidence*.

Apabila terdapat *itemset* X dan Y, maka nilai *support* merupakan persentase jumlah transaksi dalam kumpulan data yang terdiri dari X sekaligus Y. Sedangkat nilai *confidence* merupakan *itemset* X yang juga mungkin muncul Y, kedua nilai tersebut biasa disebut dengan *minsup* dan *minconf*. Penemuan pola asosiasi yang paling sering muncul. Terdapat permasalahan yang berpotensi untuk diteliti lebih lanjut adalah sejauh mana pengaruh jumlah fitur yang diperhitungkan, nilai *confidence*, dan *support* terhadap akurasi saat proses klasifikasi. Hal ini dikarenakan nilai *confidence* dan *support* tidak secara otomatis muncul serta *input* pengguna belum tentu menghasilkan akurasi

2.4 *Cumulative Support Threshold*

Sebuah *itemset* A = {a1, a2, ..., ak} dengan ai ⊂ I adalah dianggap sebagai *frequent* jika nilai *support* dari A adalah lebih besar daripada seluruh *minimum support* yang paling rendah pada A. *Cumulative Support Threshold* digunakan untuk menunjukkan kumpulan nilai *support* dari *item* pada level sebelumnya. Bagian ini diturunkan dari level kedua dan seterusnya. Sebagai contoh untuk *cumulative support* (cs) untuk dua *itemset* dapat dilihat pada persamaan 1.

$$cs(a_1,a_2) = \{sup(a_1) * sup(a_2)\} \quad (1)$$

Pada level selanjutnya *cumulative support* dari sebuah *itemset* dihitung sebagai nilai terkecil dari *cumulative support* dari himpunan yang ada. Sebagai contoh untuk melihat jika terdapat tiga *itemset* dapat dilihat pada persamaan 2.

$$cs(a_1,a_2,a_3) = min\ \{cs(a_1,a_2),cs(a_2,a_3),cs(a_1,a_3)\} \quad (2)$$

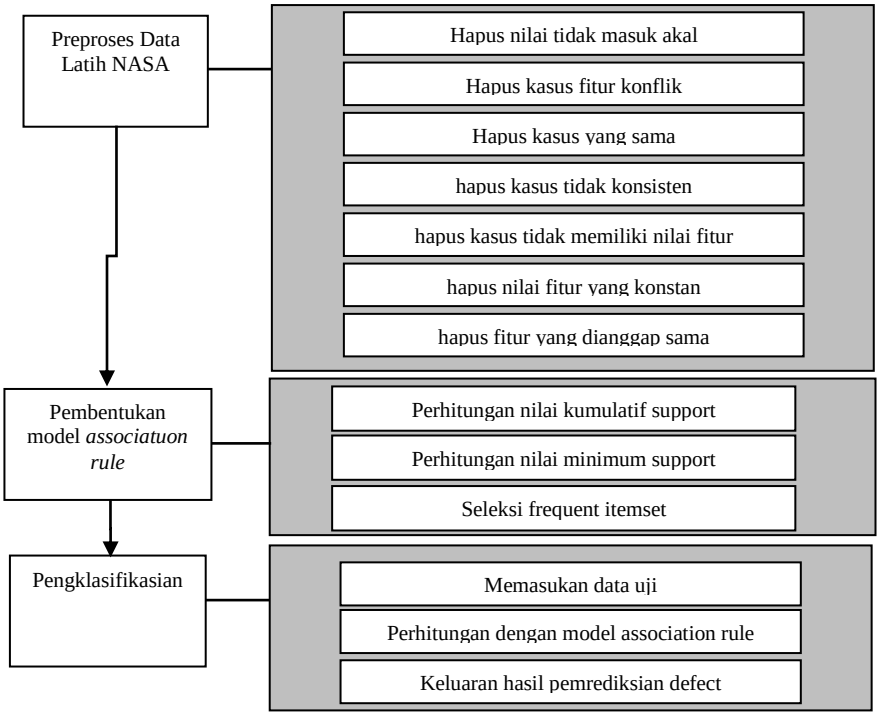
Cumulative Support (cs) untuk level kedua dapat diidentifikasi menggunakan nilai *support* untuk masing-masing item. Pada level ketiga *cumulative support* (cs) ditentukan dengan cs level kedua dan seterusnya.

3. Metodologi Penelitian

Secara garis besar pendeteksian *defect* pada perangkat lunak merupakan proses yang membutuhkan data pelatihan karena *metode association rule* merupakan salah satu metode *supervised* di dalam klasifikasi *data mining*. Data latih tersebut nantinya akan diproses sehingga menghasilkan sebuah model yang nantinya akan digunakan sebagai pendeteksi dari *defect* perangkat lunak. Pada Gambar 1 terlihat bahwa data latih yang digunakan dalam membentuk model *association rule* berasal dari NASA yang berisi data karakteristik *defect* kode perangkat lunak.

Model klasifikasi menggunakan metode *association rule* dengan proses sebelum data latih tersebut diproses untuk membentuk model *association rule* data latih tersebut terlebih dahulu diolah terlebih dahulu untuk melakukan beberapa penyesuaian. Setelah dianggap siap data latih tersebut diproses untuk membentuk sebuah model klasifikasi yang nantinya dipersiapkan agar secara otomatis mampu menghasilkan deteksi *defect* perangkat lunak secara akurat. Penambahan metode dengan melakukan perhitungan *cumulative support threshold* dilakukan agar perhitungan *association rule* tidak perlu lagi menentukan nilai *confidence* dan

support yang sebelumnya harus dimasukkan secara manual oleh pengguna. Selain berfungsi mengotomatiskan perhitungan *association rule*, metode ini juga mencari nilai *confidence* dan *support* yang paling optimal.



Gambar 1. Skema Proses Pendeteksian *Defect* Perangkat Lunak

Pada bagian pertama yaitu *preprocessing* dengan melakukan penyesuaian terhadap data latih sebelum nantinya digunakan sebagai pembentukan model pelatihan. Tahap *preprocessing* yang dilakukan banyak melakukan penghapusan pada data yang kurang representatif. Sebelum dibentuk menjadi sebuah model pengklasifikasian terlebih dahulu dilakukan pemrosesan awal sehingga data latih tersebut siap untuk dibentuk menjadi model klasifikasi. Data latih terkait deteksi *defect* perangkat lunak telah diketahui sekitar 96 *dataset* pada tahun 2012. Data latih yang digunakan disini adalah *dataset* yang disediakan secara gratis oleh *website* NASA Metrics Data Program (MDP).

Permasalahan yang harus diselesaikan sebelum menggunakan *dataset* adalah penggunaan *dataset* NASA ini harus melalui tahap *preprocessing*. Menurut (Runeson dkk, 2006) disebutkan ada beberapa tahap yang harus dilakukan dalam melakukan *preprocessing* yaitu menghapus kasus nilai-nilai yang tidak masuk akal, menghapus kasus dengan nilai fitur yang terjadi konflik, menghapus kasus yang sama atau redundan, menghapus kasus yang tidak konsisten, menghapus kasus yang tidak memiliki nilai fitur, menghapus nilai fitur yang konstan, dan menghapus fitur yang dianggap sama. Hasil setelah dilakukan preproses terhadap *dataset* NASA adalah memperbaiki *dataset* sehingga tidak ada data yang dianggap kurang representatif dan sebagai *noise* pada proses pendeteksian *defect*.

Proses yang kedua adalah pembentukan model *association rule* yang merupakan bagian proses *data mining* untuk mencari pola-pola dari *dataset* yang telah diproses sebelumnya sebagai dasar dalam melakukan pendeteksian. Pembentukan model ini melakukan tiga tahapan lain yaitu perhitungan nilai *cummulative support*, pembentukan nilai *minimum support*, dan seleksi *frequent itemset*. Ketiga proses dalam pembentukan model ini akan menghasilkan model klasifikasi tanpa perlu menerima masukan nilai *confidence* dan nila *support* seperti pada proses *association rule* pada umumnya. Kontribusi utama dalam penelitian ini adalah menggabungkan beberapa metode yang digunakan sebelumnya untuk dapat menghasilkan pola yang diinginkan secara otomatis tanpa harus mendapat masukan dari pengguna sistem berupa nilai *support* dan

nilai *confident*. Ketiga *itemset* yang telah disebutkan sebelumnya memiliki frekuensi 3,4, dan 3 sehingga ketiga *itemset* tersebut memenuhi batas ambang nilai *support*. Secara sederhana *frequent closed itemset* merupakan *itemset* jika misalkan *itemset* Y adalah *frequent* dan tidak terdapat *superset* langsung sedemikian sehingga sup(Y')=sup(Y). Hal ini mengindikasikan bahwa prediksi *defect* pada kode perangkat lunak membutuhkan perhatian dan metode yang lebih efisien dalam memprediksi.

Pada (Okumoto, 2011) sebuah himpunan kemungkinan {I0, I1,...,IN} dengan kejadian kemunculan telah melebihi nilai minimal ambang atau *threshold support*. Dengan kata lain *frequent itemset* adalah *itemset* yang memiliki *support* yang melebihi nilai minimum yang ditetapkan. *Support* atau nilai *support* dari sebuah *itemset* menyatakan persentase kemunculan *itemset* tersebut jika dibandingkan dengan jumlah keseluruhan transaksi.

Merupakan *itemset* yang terdiri dari *itemset* yang berbeda-beda level dan tidak termasuk *itemset* yang memiliki pasangan *ancestor-descendant* yang memenuhi minimum nilai *support*. Sebuah transaksi dianggap sebagai sebuah *itemset*. Apabila terdapat istilah k-*itemset* yang menyatakan jumlah *item* yang dalam *set* tersebut adalah k. Sebagai contoh untuk memperlihatkan bagaimana nilai *support*, apabila S adalah *itemset* misalkan {roti, susu, keju} dan T adalah seluruh transaksi misalnya {{roti}, {roti, mie},{sarden, kacang}, ...} dan U adalah kumpulan transaksi yang mengandung S. Contoh lain dari *frequent itemset* adalah apabila ditentukan batas ambang *threshold* 3 sebuah *itemset* {2-1-1}, {1-1-1} dan {1-1-1,2-1-1}. Ketiga *itemset* yang telah disebutkan sebelumnya memiliki frekuensi 3,4, dan 3 sehingga ketiga *itemset* tersebut memenuhi batas ambang nilai *support*. Secara sederhana *frequent closed itemset* merupakan *itemset* jika misalkan *itemset* Y adalah *frequent* dan tidak terdapat *superset* langsung sedemikian sehingga sup(Y')=sup(Y). Sebuah *itemset* A = {a1, a2, ..., ak} dengan ai $\subset$ I adalah dianggap sebagai *frequent* jika nilai *suport* dari A adalah lebih besar daripada seluruh *minimum support* yang paling rendah pada A. *Cumulative Support Threshold* digunakan untuk menunjukkan kumpulan nilai *support* dari *item* pada level sebelumnya. Bagian ini diturunkan dari level kedua dan seterusnya. Pada (Czibula dkk, 2014) disebutkan bahwa prediksi *defect* perangkat lunak digunakan adalah proses yang mengidentifikasi *defect* yang terdapat pada modul atau bagian yang terdapat perbaruan dari sistem perangkat lunak yang memungkinkan terjadi kesalahan.

3.1 Perhitungan Nilai *Support* Kumulatif

Perhitungan nilai penunjang (*support*) kumulatif dapat dihitung dengan persamaan 3. Pada bagian diatas dituliskan untuk dua *itemset* saja, apabila terdapat lebih dari dua *itemset* yang muncul maka rumus perhitungan nilai penunjang kumulatif menjadi persamaan 4.

$$cs(a1,a2) = \{sup(a1) * sup(a2)\} \quad (3)$$

$$cs(a1,a2,a3) = \min \{cs(a1,a2),cs(a2,a3),cs(a1,a3)\} \quad (4)$$

3.2 Perhitungan Nilai *Support* Minimum

Perhitungan *minimum support* didapatkan untuk melihat masing-masing *item* berdasarkan persamaan 5. Misalkan ms(a) menjelaskan *minimum support* dari sebuah *item* a dengan a $\subset$ I dan nantinya a1$\leq$a2$\leq$a3$\leq$ ... $\leq$ak. Apabila ms(a1,a2) menggambarkan *minimum support* dari transaksi *itemset* {a1,a2}, maka rumusannya dapat dilihat pada persamaan 6. Perhitungan *minimum support* pada tiga *itemset* dapat dilihat pada persamaan 7.

$$ms(ai) = \begin{cases} sup(ai)*\max\{minconf, sup(ai+1)\}, & \text{if } 1 \leq i \leq n-1 \\ sup(ai), & \text{if } i=1 \end{cases} \quad (5)$$

$$ms(a1, a2) = \min\{sup(a1, a2), cs(a1, a2)\} \quad (6)$$

$$ms(a1,a2,a3) = \min\{sup(a1,a2,a3),cs(a1,a2,a3)\} \quad (7)$$

Minimum support (ms) pada level kedua diidentifikasi menggunakan *support*(sup) dan *cummulative support* (cs) dari *itemset*. Untuk level ketiga, *minimum support* (ms) yang digunakan sebagai penanda adalah *support* (sup) dan *cummulative support* (cs) dari *itemset*.

3.3 Seleksi *Frequent Itemsets*

Frequent itemset atau pola frekuensi tinggi dihasilkan berdasarkan nilai kumulatif *support* dan nilai minimum *support* pada setiap level. *Itemset* disebut *frequent* jika dan hanya jika memenuhi kondisi yang ada pada persamaan 8. Apabila terdapat tiga *itemset* (a, b, c) maka kondisi disebut *frequent* jika dan hanya jika memenuhi kondisi yang ada pada persamaan 9.

$$cs(a,b) \geq \min\{ms(a),ms(b)\} \quad (8)$$

$$cs(a, b, c) \geq \min\{ms(a, b), ms(b,c)), ms(a,c)\} \quad (9)$$

Proses ketiga adalah proses pengklasifikasian yang merupakan tahap utama dalam melakukan pendeteksian *defect* kode perangkat lunak. Proses *association rule* merupakan salah satu proses dalam *data mining* yang dalam melakukan penentuan kelas harus membutuhkan data latih. Pembentukan model sudah dilakukan pada tahap sebelumnya, sehingga pada tahap ini proses yang dilakukan adalah melakukan pengujian pada model yang terbentuk pada tahap kedua. Sehingga tahap ini melakukan pengujian dengan memasukan data yang belum memiliki kelas target (belum diketahui apakah merupakan *defect* atau tidak) untuk diketahui *defect*nya. Keluaran dari proses ini adalah hasil akurasi dari proses pengklasifikasian dengan metode *association rule*.

4. Hasil dan Pembahasan

4.1. *Dataset*

Pada penelitian menggunakan *dataset* NASA CM1 yang berasal dari perangkat lunak terkait *spacecraft* yang ditulis dalam bahasa pemrograman C. Pemilihan *dataset* NASA CM1 ini dilakukan karena *dataset* ini cukup sederhana digunakan pada domain prediksi *defect* karena memiliki nilai *false* yang sangat rendah. Selain itu probabilitas deteksi naik sejalan dengan naiknya proses pencarian. *Dataset* terdiri dari 42 data yang *defect* dan 285 data yang tidak *defect* (*non-defect*), dapat dikatakan bahwa terdapat 12,84% data yang merupakan *defect* dan sejumlah 87,16% data yang bukan merupakan *defect*. Masing-masing data memiliki 22 fitur atau atribut dan juga kelas label. Pengujian yang dilakukan adalah dengan membagi *dataset* dengan perbandingan 70% data digunakan sebagai data latih dan 30% digunakan sebagai data uji. Sehingga pembagian jumlah data dapat dilihat pada Tabel 1. Atribut yang digunakan dalam dataset NASA seperti yang dijelaskan pada (Shepperd dkk, 2013) dapat dilihat pada Tabel 2.

4.2. Skenario Pengujian

Pengujian dilakukan dengan memanfaatkan *dataset* NASA dengan membagi menjadi prosentase seperti pada Tabel 1 yaitu dengan komposisi data latih sebesar 70% dan data uji sebesar 30%. Data yang digunakan sebagai data uji tidak akan digunakan sebagai data uji pada tahap pengujian. Proses pengujian dilakukan secara sederhana dengan mengulang *generate* data uji dan data latih secara *random* efektif dilakukan karena dengan metode pengujian ini cukup representatif dalam menunjukkan hasil pengujian. Pengujian dilakukan dengan memperlihatkan nilai akurasi ketepatan dalam memprediksi *defect*. Data uji akan diproses untuk menghasilkan nilai akurasi yang diperoleh dengan membandingkan hasil prediksi dengan hasil kelas yang sudah ada yang hasilnya akan ditampakkan pada bagian selanjutnya.

Tabel 1. Persebaran *Dataset*

Jenis	Defect +	Defect -
Data Latih	30	200
Data Uji	12	85

Tabel 2. Atribut *Dataset* NASA

Atribut	Keterangan
Loc	McCabe LOC
V(g)	McCabe cyclomatic complexity
Ev(g)	McCabe essential complexity
iv(g)	McCabe design complexity
N	Halstead total operators + operands
V	Halstead volume
L	Halstead program length
D	Halstead difficulty
I	Halstead intelligence
E	Halstead effort
B	Halstead

Atribut	Keterangan
T	Halstead's time estimator
loCode	Halstead's line count
loComment	Halstead's count of line
loBlank	Halstead's count of blank
loCodeAndCOmment	
uniq_Op	unique operator
uniq_Opnd	unique operands
total_Op	total operators
total_Opnd	total operands
branchCount	number of flow graph
defect	reported defect

4.3. Hasil Pengujian

Pengujian dilakukan dengan membandingkan nilai hasil prediksi yang dilakukan oleh metode yang diajukan dengan hasil prediksi yang sudah ada pada *dataset* NASA. Hasil pengujian menunjukkan nilai prosentase akurasi kebenaran dari prediksi adalah 82,35%. Nilai akurasi tersebut cukup besar sehingga dapat disimpulkan bahwa pencarian pola *defect* berhasil membedakan mana kode perangkat lunak yang berpotensi terjadi *defect* atau tidak.

5. Kesimpulan dan Saran

Penelitian ini membahas model klasifikasi berdasarkan metode *association rule* untuk mendeteksi kode perangkat lunak yang berpotensi terjadi *defect*. Metode yang menggabungkan metode *association rule* dengan *cumulative support threshold* sehingga proses klasifikasi tidak membutuhkan nilai *support* dan *confidence* masukan dari pengguna. Hasil pengujian menunjukkan akurasi yang dihasilkan adalah 82,35%, hal ini menunjukkan nilai akurasi pemrediksian cukup tinggi.

Saran untuk penelitian selanjutnya adalah permasalahan *dataset* tidak seimbang antara kelas kondisi positif terdapat *defect* dan negatif *defect* dapat mempengaruhi hasil pembentukan model *association rule* yang berdampak pada hasil akurasi pemrediksian. Penelitian lebih lanjut, sebelum dilakukan proses-proses untuk mendapatkan model, seharusnya dilakukan penilaian terhadap *dataset* untuk menangani ketidakseimbangan tersebut.

Referensi

Boehm, B. dan Basili, V.R. 2001. Software Defect Reduction Top 10 List. *Journal Computer 34 IEEE,* 1 (January 2001), pp. 135-137.

Czibula, G., Marian, Z. & Czibula, I.G. 2014. Software Defect Prediction Using Relational Association Rule Mining. *Inf. Sci. (Ny).*, vol. 264, pp. 260–278, April 2014.

Okumoto, K. 2011. Software Defect Prediction Based on Stability Test Data. *Quality, Reliability, Risk, Maintenance, and Safety Engineering (ICQR2MSE), 2011 International Conference on* , vol., no., pp.385,387, 17-19 June 2011.

Pressman, R.S. 2001. *Software Engineering: A Practitioner's Approach*, Fifth Edition. The McGraw-Hill Companies, Inc, Singapore.

Runeson, P., Andersson, C., Thelin, T., Andrews, A., Berling, T. 2006. What Do We Know About Defect Detection Methods? [Software Testing]. *Software, IEEE*, vol.23, no.3, pp.82, 90, May-June 2006.

Selvi, C. S. K. & Tamilarasi, A. 2009. An Automated Association Rule Mining Technique With Cumulative Support Thresholds. *Int. J. Open Problems in Compt. Math*, vol. 2, no. 3. ICRSR Publication.

Shepperd, M., Qinbao Song, Zhongbin Sun, Mair, C. 2013. Data Quality: Some Comments on the NASA Software Defect Datasets. *Software Engineering, IEEE Transactions* vol.39, no.9, pp.1208, 1215, Sept. 2013.