Pembangunan Model Prediksi *Defect* Menggunakan Metode *Ensemble Decision Tree* Dan *Cost Sensitive Learning*

Satrio Agung Wicaksono, Daniel Oranova S dan Sarwosri

***Abstrak*—** **Rencana project pengembangan perangkat lunak dapat disusun menggunakan *work breakdown structure* (WBS). Pelaksanaan unit terkecil dari WBS pada proses pengembangan perangkat lunak disebut *action*. *Action* dapat menimbulkan *defect* pada perangkat lunak. Sebuah *action* dikategorikan menghasilkan *high defect* jika banyaknya *defect* yang dihasilkan *action* tersebut melebihi *threshold* tertentu.**

***Action-based defect prevention* (ABDP) merupakan metode untuk membangun model prediksi yang dapat meramalkan apakah *action* yang akan dilakukan akan menghasilkan *high defect* atau tidak. Model prediksi pada ABDP menggunakan *single classification tree* yang dibangun berdasarkan catatan *action* yang telah dilakukan dan *defect* yang ditimbulkan *action* tersebut pada suatu proses pengembangan perangkat lunak. Hasil prediksi akan menjadi pertimbangan dalam proses pengembangan perangkat lunak untuk menghindari munculnya *high defect*.**

Penelitian ini mengajukan metode pembuatan model prediksi yang merupakan pengembangan dari ABDP. Metode yang diajukan berusaha meningkatkan akurasi ABDP dan meminimalkan *cost* (kerugian) jika terjadi kesalahan prediksi. Untuk meningkatkan akurasi, model prediksi dibangun dengan *ensemble method* menggunakan *base classifier classification tree*. Untuk memperkecil kerugian yang ditimbulkan oleh kesalahan prediksi, maka pada proses pembuatan *classification tree* digunakan metode *cost sensitive learning*. Hasil uji coba menunjukkan bahwa, metode yang diajukan memiliki *accuracy* dan *recall* yang lebih baik dari *single classification tree*.

***Kata Kunci—work breakdown structure, action, high defect, ABDP, classification tree, ensemble method, cost sensitive learning*.**

I. INTRODUCTION

SEBUAH rencana project pengembangan perangkat lunak dapat disusun dengan *work breakdown structure* atau WBS [3]. *Action* merupakan pelaksanaan unit terkecil WBS, yaitu *step*.

Sebuah *action* dapat menimbulkan *defect* pada perangkat lunak. Berdasarkan banyaknya *defect* yang ditimbulkan, sebuah *action* dapat dikategorikan menghasilkan *high defect* atau *low defect*. Sebuah action dianggap menghasilkan *high defect* jika banyaknya *defect* yang dihasilkan *action* tersebut lebih dari *threshold* tertentu.

Agar proses pengembangan perangkat lunak berjalan dengan lancar, maka harus dilakukan tindakan pencegahan munculnya *high defect*.

Terdapat beberapa pendekatan untuk mencegah munculnya *high defect*. Salah satu pendekatan adalah *action-based defect prevention* (ABDP) [1]. Pada ABDP, *action* dengan pola tertentu dianggap beresiko menghasilkan *high defect*. Pencegahan munculnya *high defect* dilakukan dengan memprediksi apakah sebuah *action* yang akan dilakukan berpotensi menghasilkan *high defect*. Dengan mengetahui hasil prediksi, dapat dipertimbangkan apakah *action* tersebut akan tetap dilakukan, dilakukan dengan perubahan atau dibatalkan untuk menghindari munculnya *high defect*.

Model prediksi ABDP dapat melakukan kesalahan prediksi yang dapat dibagi menjadi dua tipe. Tipe 1 adalah memprediksi *action* yang sebenarnya menghasilkan *low defect* sebagai *high defect*. Tipe 2 adalah memprediksi *action* yang sebenarnya menghasilkan *high defect* sebagai *low defect*. Kesalahan tipe 2 menimbulkan *cost* (kerugian) lebih besar dari tipe 1.

Pada penelitian ini diajukan metode pembuatan model model prediksi yang merupakan pengembangan dari ABDP. Untuk meningkatkan akurasi prediksi, model prediksi dibangun dengan *ensemble method* yang menggunakan banyak *classification tree*. Untuk memperkecil kerugian yang ditimbulkan oleh kesalahan prediksi dan menekan munculnya kesalahan prediksi tipe 2, maka proses pembuatan *classification tree* dilakukan dengan metode *cost sensitive learning*.

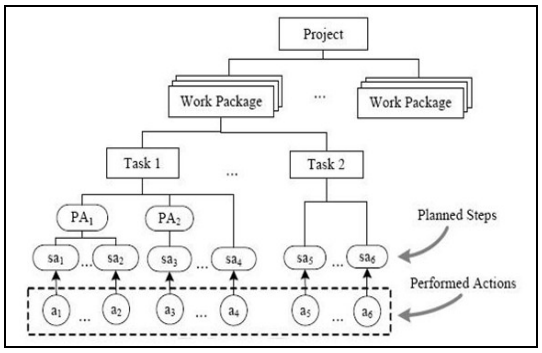
II. DASAR TEORI

A. Action-based defect prevention

Sebuah rencana *project* pengembangan perangkat lunak dapat disusun dengan *work breakdown structure* atau WBS [3]. Pada WBS, sebuah *project* dipecah menjadi beberapa *package*. Sebuah *package* dibagi lagi menjadi beberapa *task*. *Task* dibagi lagi menjadi beberapa *process*. Pada tingkatan paling rendah, sebuah *process* dibagi menjadi unit terkecil yang dinamakan *step*. *Action* merupakan kegiatan eksekusi sebuah *step* pada sebuah proses pengembangan perangkat lunak.

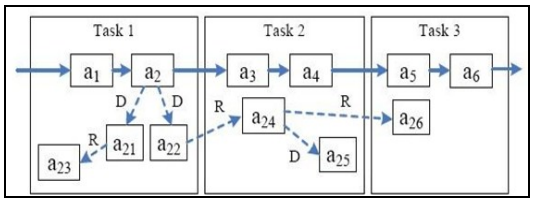
Semua penulis dari Jurusan Teknik Informatika, Fakultas Teknologi informasi, Institut Teknologi Sepuluh Nopember (Email : satrio@cs.its.ac.id, daniel@its-sby.edu, sri@its-sby.edu).

Struktur WBS ditunjukkan pada gambar 1



Gambar. 1. Struktur WBS project perangkat lunak

Pelaksanaan sebuah *action* dapat menimbulkan *defect*. Pada gambar 2 ditunjukkan bahwa *action* a2 menyebabkan beberapa *defect* yang harus ditangani *action* lain. Secara keseluruhan, *defect* yang ditimbulkan oleh a2 pada gambar 2 adalah tiga defect.



Gambar 2. Sebuah *action* dapat menimbulkan *defect.*

Berdasarkan banyaknya *defect* yang ditimbulkan, sebuah *action* dapat dikategorikan menjadi *high defect* dan *low defect*. Sebuah *action* dianggap *high defect* jika menghasilkan *defect* lebih dari *threshold* tertentu.

Action-based defect prevention (ABDP) merupakan sebuah metode untuk memprediksi apakah sebuah *action* yang akan dilakukan berpotensi menghasilkan *high defect* [1]. ABDP mengasumsikan bahwa, *action* dengan pola tertentu beresiko menghasilkan *high defect*. Dengan melakukan prediksi apakah sebuah *action* yang akan dilakukan berpotensi menghasilkan *high defect*, dapat dipertimbangkan apakah *action* tersebut akan tetap dilakukan, dilakukan dengan perubahan atau dibatalkan untuk menghindari munculnya *high defect* tersebut.

Proses prediksi pada ABDP dilakukan oleh model prediksi *classification tree* yang dibangun berdasar attribut - attribut *action* yang telah terjadi dan attribut - attribut *defect* yang ditimbulkan oleh *action* tersebut pada sebuah proses pengembangan perangkat lunak.

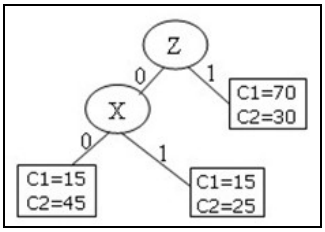
Model prediksi pada ABDP dapat melakukan kesalahan prediksi. Kesalahan tersebut dapat dibagi menjadi dua jenis, kesalahan tipe 1 dan tipe 2. Kesalahan tipe 1 adalah kesalahan memprediksi *action* yang sebenarnya *low defect* sebagai *high defect*. Kesalahan tipe 2 adalah kesalahan memprediksi *action* yang sebenarnya *high defect* sebagai *low defect*. Kesalahan tipe 2 mengakibatkan *cost* (kerugian) yang lebih besar dari tipe 1 karena kesalahan tipe 2 akan mengakibatkan munculnya high *defect* pada proses pengembangan perangkat lunak.

B. Ensemble classification tree

Klasifikasi merupakan proses untuk menggolongkan sebuah obyek menjadi *class* tertentu berdasar nilai attribut – attributnya seperti yang digambarkan pada gambar 3. Bagian yang bertugas untuk melakukan proses klasifikasi disebut model prediksi [4]

Gambar 3. Proses Klasifikasi.

Terdapat beberapa jenis *learning algorithm* untuk membuat model prediksi. *Classification tree* merupakan sebuah *learning algorithm* yang menggunakan *tree* untuk membangun model prediksinya. Gambar 4 menunjukkan sebuah *classification tree* sederhana. *Classification tree* tersebut dapat digunakan untuk memprediksi *class* sebuah *object*.



Gambar 4. Contoh sebuah classification tree sederhana .

Untuk meningkatkan akurasi dari proses klasifikasi yang menggunakan *classification tree*, salah satu cara yang dapat dilakukan adalah menggunakan model prediksi yang terdiri atas banyak *tree* sebagai *base classifier*. Teknik ini dinamakan *ensemble method*. Pada *ensemble method* proses klasifikasi dengan melakukan *voting* hasil prediksi *base classifier*-nya.

TABEL 1 METODE BAGGING

1	K adalah banyaknya *bootstrap*
2	For i = 1 to k do
3	Buat sebuah *bootstrap sample* Di berukuran
4	N
5	Buatlah sebuah *classification tree* Ci
6	menggunakan *bootstrap sample* Di
7	End for
8	$C^*(x) = \arg\max_y \sum_i \delta(C_i(x) = y)$
	$(\delta(.) = 1)$ jika argumennya bernilai true dan sebaliknya

Bagging merupakan salah satu teknik yang mengimplementasikan ensemble *method* [4]. Pada Tabel 1 ditunjukkan algoritma pembuatan *ensemble classification tre*e menggunakan metode *bagging*. Mula – mula ditentukan dulu banyaknya *bootstrap* atau *classifier* yang akan dibuat seperti yang ditunjukkan pada baris 1. Pada baris 2 sampai 7, untuk setiap iterasi dilakukan pembuatan *bootstrap* menggunakan teknik *sampling with replacement* dengan *uniform probability distribution.* Dari *bootstrap* yang didapatkanpada setiap iterasi, dibuat sebuah *classification tree* untuk sebuah

bootstrap. Model prediksi yang terbentuk terdiri atas banyak classifier. Jika terdapat data yang akan diklasifikasikan, maka proses klasifikasi dilakukan dengan melakukan voting antara *classifier – classifier* penyusun model prediksi yang telah dibuat seperti yang ditunjukkan pada baris 8.

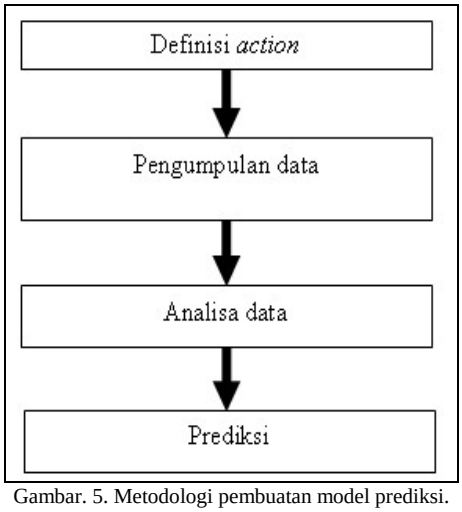
C. Cost sensitine learning

Cost sensitive learning merupakan teknik untuk membangun *classifier* yang mampu meminimalkan kerugian jika terjadi kesalahan prediksi. Untuk mengimplementasikan *cost sensitive learning*, harus diketahui *cost function*. *Cost function* merupakan bobot yang diberikan pada *cost* (kerugian) yang ditimbulkan oleh kesalahan model klasifikasi dalam memprediksi *class* dari *object* tertentu. C(i, j) menotasikan *cost* atau kerugian jika mengklasifikasikan sebuah data yang sebenarnya berasal dari *class* i menjadi *class* j.

Pada *classification tree*, *cost sensitive learning* dapat diintegrasikan dalam beberapa cara [4]. (i) *cost function* digunakan sebagai kriteria untuk pemilihan attribut yang akan displit (ii) *cost function* digunakan untuk memilih bagian *tree* yang akan dipangkas (iii) *cost function* digunakan memanipulasi bobot dari data training sehingga menghasilkan *tree* dengan *cost minimal* (iv) *cost function* digunakan sebagai kriteria untuk pelabelan *leaf node.*

III. PEMBUATAN MODEL PREDIKSI

Langkah - langkah untuk membuat model prediksi yang dapat meramalkan apakah sebuah action pada proses pengembangan perangkat lunak akan menghasilkan high defect atau tidak terdiri atas (i) definisi action (ii) pengumpulan data (iii) analisa data (iv) proses prediksi. Alur metodologi ditunjukkan pada gambar 5.



Gambar. 5. Metodologi pembuatan model prediksi.

A. Definisi action dan pengumpulan data

Tujuan utama dari definisi *action* adalah mendefinisikan attribut - attribut yang akan dikumpulkan dari catatan *action* dan laporan *defect* pada proses pengembangan perangkat lunak. Attribut – attribut ini menjadi kandidat attribut dataset yang digunakan untuk membangun model prediksi dan uji coba.

Attribut - attribut didefinisikan berdasar domain expert, yaitu saran dari pihak yang berkecimpung pada proses pengembangan perangkat lunak tentang hal apa saja yang mempengaruhi sebuah action akan menimbulkan defect. Setelah ditentukan attribut – attributnya, maka langkah yang harus dilakukan ialah megumpulkan data untuk mengisi attribut – attribut tersebut dari dokumentasi action dan defect yang telah terjadi pada proses pengembangan perangkat lunak

B. Analisa data

Analisa data bertujuan untuk membangun model prediksi. Tahap analisa data terdiri atas dua bagian, yaitu (i) tahap preprocessing data dan (ii) tahap pembuatan model prediksi

***Preprocessing* data**

Data preprocessing merupakan kegiatan mengubah data yang telah terkumpul pada tahap sebelumnya menjadi dataset yang digunakan untuk membangun model prediksi. Pada tahap ini, attribut – attribut action dan attribut – attribut defect yang telah didapatkan dari tahap sebelumnya digabung menjadi attribut dataset. Tidak semua attribut action dan defect yang telah didefinisikan sebelumnya digunakan pada dataset. Attribut – attribut yang digunakan pada dataset ditunjukkan pada tabel 2 [3]

Attribut - attribut dataset yang ditunjukkan pada Tabel 2 dibagi menjadi dua bagian yaitu attribut *antecedent*, yaitu attribut nomor 1 - 20 dan attribut *consequent*, attribut nomor 21. Attribut *antecedent* digunakan pada tahap prediksi sebagai input bagi model prediksi untuk memprediksi nilai dari attribut consequent. Attribut *consequent* 21 merupakan attribut yang menentukan apakah *action* menghasilkan *high defect* atau tidak. Pada penelitian ini, sebuah *action* dikategorikan menghasilkan high defect jika nilai attribut *number of defect* lebih atau sama dengan 4

Dataset yang dihasilkan dibagi menjadi dua, menjadi training set dan testing set. Training set digunakan untuk membangun model prediksi. Sedangkan testing set digunakan pada proses uji coba untuk mengukur kinerja model prediksi

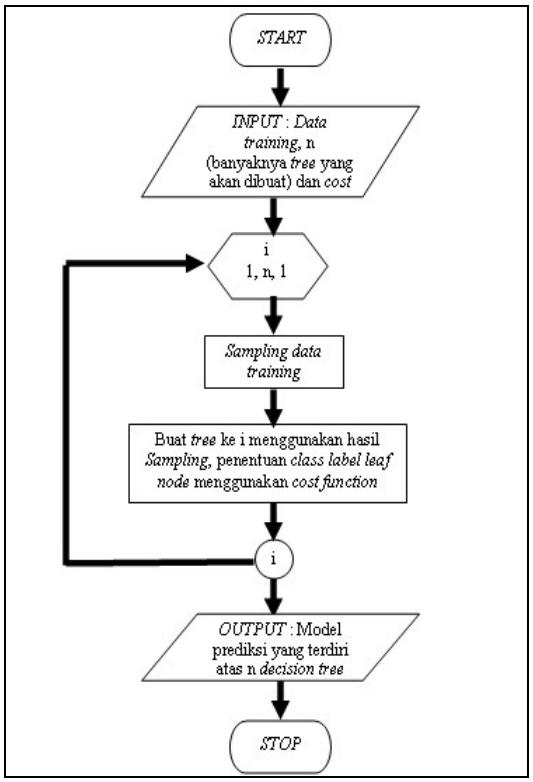
Tahap pembuatan model prediksi

.Gambar 6 menunjukkan *flowchart* proses pembangunan model prediksi yang didasarkan pada algoritma di tabel 1. Input dari proses pembangunan model prediksi adalah *data training*, banyaknya *classification tree* yang digunakan dan dan *cost function*. Data *training* didapatkan dengan mengambil sebagian *dataset* yang telah dihasilkan dari tahap sebelumnya. Banyaknya *tree* yang akan dibangun merupakan parameter yang ditentukan sendiri oleh pengguna sistem. Sedangkan *cost function* juga merupakan parameter yang ditentukan oleh pengguna sistem.

Nilai dari *cost function* didasarkan pertimbangan

kerugian yang ditimbulkan oleh *misclassification error*. *Cost function* yang penting adalah c(l, h) dan c(h, l). c(l, h) adalah *cost* atau kerugian memprediksi sebuah *action* yang sebenarnya tidak menghasilkan *high defect* sebagai *action* yang menghasilkan *high defect*. c(h, l) adalah adalah kebalikan dari c(h, l). Nilai c(h, l) harus lebih dari c(l,h) karena kesalahan memprediksi sebuah *action* yang sebenarnya *high defect* sebagai *action low defect* akan menyebabkan kerugian yang cukup besar, yaitu munculnya *high defect* pada proses pengembangan perangkat lunak

Pada setiap iterasi di gambar 6, dilakukan pembuatan sebuah *classification tree* menggunakan data *training* baru yang didapatkan dari *sampling with replacement data training* asli menggunakan *uniform probability distribution*. Setiap iterasi menghasilkan satu *classification tree*. Pada akhir iterasi, terbentuk sebuah model prediksi yang terdiri atas banyak *classification tree*



Gambar. 6. Proses pembuatan model produksi

Tabel 3 menunjukkan implementasi *flowchart* pada Gambar 6 menggunakan MATLAB. Proses pembuatan model prediksi ditunjukkan pada baris 2 sampai baris 9. Pada setiap iterasi, dibuat data *training* baru dengan *sampling with replacement* pada data *training* asli menggunakan *uniform probability distribution* seperti yang ditunjukkan pada baris 3 sampai baris 7. Pembuatan *classification tree* pada setiap iterasi ditunjukkan pada baris 8. Pada penelitian ini, proses pembuatan *classification tree* dilakukan dengan algoritma *default* yang disediakan oleh MATLAB (*classregtree*). Dapat dilihat pada baris 8 bahwa pada pembuatan *classification tree* input yang diperlukan adalah data *training* (*variabel newAttributes* dan *newComplexity*) dan *cost function* (*variabel cost*).

TABEL 2 ATRIBUT YANG DIGUNAKAN DIDATASET

ID	Attribut	Keterangan
1	*Action state*	0 : terjadwal 1, : tidak terjadwal
2	*Action type*	N : membuat modul baru, M : modifikasi modul yang telah ada, D : menghapus modul, A : menambahkan modul, - : none
3	*Caused type*	- : general *action*, R : *reaction*, D: *defect action*
4	*Action complexity*	0 : rendah, 5 : sedang, 10 : tinggi
5	*Object type*	0 : none, 1 : dokumen, 2 : database, 3 : aplikasi, 4 : konfigurasi sistem
6	*Effort expected*	Numerik, estimasi dari waktu untuk menyelesaikan *action*
7	*Action originator*	0 : none, 1 : customer, 2 : pengguna, 3 : manajer, 4 : programmer
8	*Action target*	0 : tidak ada, 1 : requirement, 2 : desain pendahuluan, 3 : desain, 4 : coding, 5 : testing, 6 : maintenance, 7 : support
9	*Number of object*	Banyaknya *object* yang berhubungan dengan *action* yang dilakukan
10	*Task*	Id *task* dari *action* yang dilakukan
11	*Task status*	0 : sesuai jadwal, 1 : setelah jadwal, 2 : tidak diketahui
12	*Task effort estimated*	Estimasi usaha yang diperlukan untuk menyelesaikan *task*
13	*Task action*	Banyaknya *action* yang dilakukan untuk menyelesaikan *task*
14	*Task modification*	Banyaknya *action* berjenis modifikasi yang dilakukan untuk menyelesaikan *task*
15	*Task creation*	Banyaknya *action* yang digunakan untuk membuat modul baru
16	*Task reaction*	Banyaknya *action* R yang diperlukan untuk menyelesaikan *task*
17	*Task D reaction*	Banyaknya *action* D yang diperlukan untuk menyelesaikan *task*
18	*Task H defect*	Banyaknya *action* yang menyebabkan *high defect*
19	*Task defect effort*	Total usaha yang diperlukan untuk memperbaiki *defect* pada *task*
20	*Task progress*	Usaha yang telah dilakukan / estimasi usaha keseluruhan
21	*Number of defect*	L jika *defect* 0 – 3 dan H jika lebih dari 3

TABEL 3 KODE MATLAB UNTUK MEMBUAT MODEL PREDIKSI

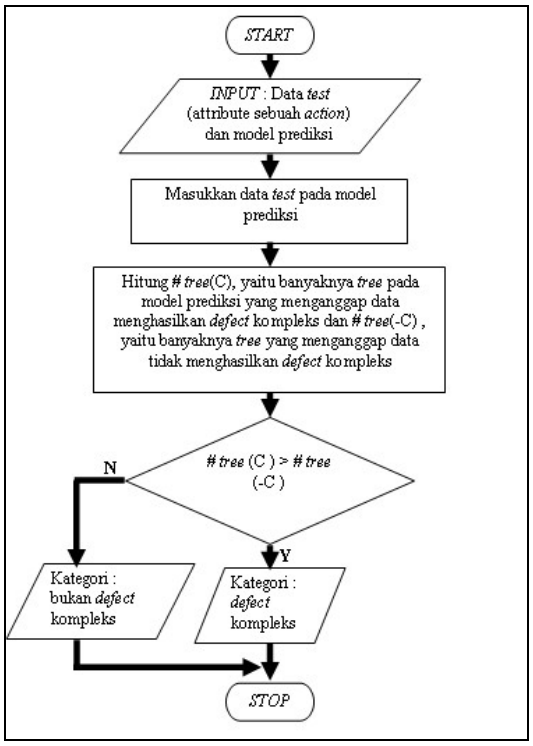
```
1  treeSet = cell(nTree, 1);
2  for i = 1 : nTree
3      sampling = randsample(trainingPos,
         nDataTraining,   true);
4      newAttributes = attributes(sampling, :);
5      for j = 1 : nDataTraining
6          newComplexity{j} =
           complexity{sampling(j)};
7      end
8      treeSet{i} = classregtree(newAttributes,
       newComplexity, 'categorical', [1 2 3 4 5 7 8 10
       11], 'cost', Cost);
9  end
```

C. Tahap prediksi

Gambar 7 menunjukkan *flowchart* dari proses klasifikasi sebuah data yang berasal dari *test set*. Untuk proses prediksi, input yang dibutuhkan adalah attribut - attribut dari *action* (attribut *antecedent* pada dataset) yang akan diprediksi dan model prediksi yang telah didapatkan dari tahap sebelumnya.

Model prediksi terdiri atas beberapa *classification tree* yang bisa menghasilkan prediksi berbeda – beda. Untuk memberi keputusan akhir apakah sebuah data test akan menghasilkan *high defect* atau tidak, dilakukan pemungutan suara. Data test akan dikategorikan menghasilkan high defect jika

$$NT\ (H) \geq NT\ (L) \qquad (1)$$



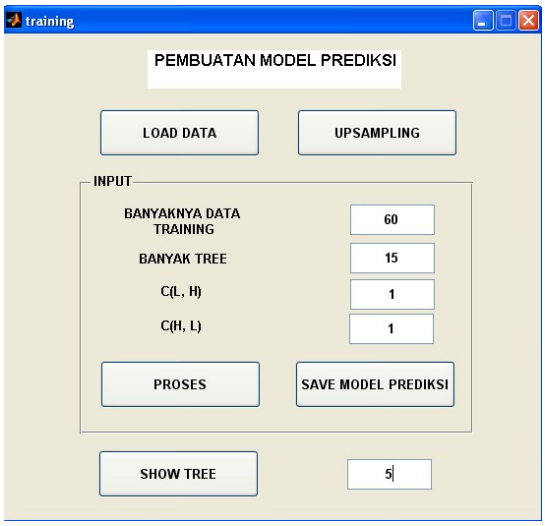
Gambar. 7. Proses prediksi

Dan dikategorikan menghasilkan low defect jika selain itu. NT(H) adalah banyaknya classification tree pada model prediksi yang menganggap data test sebagai *high defect* dan N(L) adalah banyaknya *classification tree* pada model prediksi yang menganggap data test sebagai *low defect.*

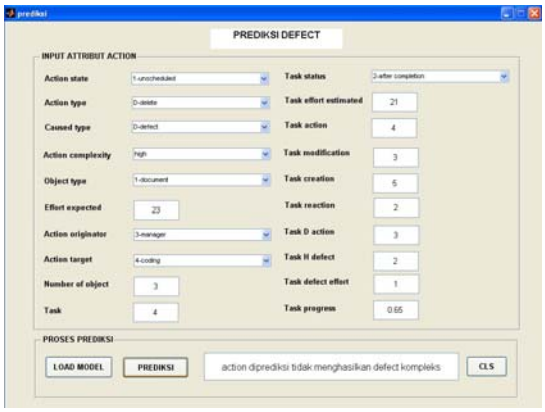
TABEL 4 KODE MATLAB UNTUK MELAKUKAN PROSES PREDIKSI

```
1   L = 0;
2   H = 0;
3   voteResult = 'H';
4   for i = 1 : nTree
5       yfit = eval(treeSet{i}, attribut);
6       if strcmp(yfit, 'L')
7           L = L + 1;
8       else
9           H = H + 1;
10      end
11  end
12  if L > H
13      voteResult = 'L';
14  end
```

Tabel 4 menunjukkan implementasi dari *flowchart* pada gambar 7. Pada baris 4 sampai 11 dilakukan proses prediksi oleh setiap *tree* pada model prediksi. Dengan demikian dapat diketahui berapa banyak *tree* yang menganggap *action* menghasilkan *high defect* (H) dan berapa banyak yang tidak menganggap demikian (L). Pada baris 12 sampai 14 diambil keputusan apakah *action* menghasilkan *high defect* atau tidak berdasar suara terbanyak.



Gambar. 8 Antarmuka pembuatan model prediksi



Gambar. 9 Antarmuka pembuatan proses prediksi

Gambar 8 menunjukkan antarmuka untuk membuat model prediksi. Pada proses pembuatan model prediksi, mula – mula diinputkan dataset menggunakan tombol load data. Untuk menyeimbangkan banyaknya data yang berjenis low defect dengan high defect dilakukan proses upsampling dengan menggunakan tombol upsampling. Setelah menginputkan data – data yang diperlukan, pembuatan model prediksi dapat dieksekusi dengan menekan tombol proses. Model prediksi yang telah dibuat dapat disimpan menggunakan tombol *save* model prediksi.

Antarmuka yang mengimplementasikan proses prediksi ditunjukkan pada Gambar 9. Pada Gambar 9 ditunjukkan bahwa aplikasi membutuhkan *input* berupa nilai dari attribut – attribut *action* yang akan diprediksi dan model prediksi yang telah dibuat mengggunakan

form pada Gambar 8. Setelah semua data *input* diisikan, proses prediksi dilakukan dengan menekan tombol proses. Hasil prediksi akan ditunjukkan pada *static text.*

IV. UJI COBA

Untuk mengetahui performa model prediksi, maka uji coba dilakukan dalam beberapa skenario. Beberapa metrik yang digunakan untuk mengukur performa model prediksi antara lain

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (2)$$

$$recall = \frac{TP}{TP + FN} \quad (3)$$

TP adalah banyaknya data test yang sebenarnya *high defect* dan diprediksi *high defect* oleh model prediksi. TN adalah banyaknya data test yang *low defect* dan diprediksi *low defect* oleh model prediksi. FP merupakan banyaknya data test yang sebenarnya *low defect* namun diprediksi *high defect* oleh model prediksi. FN merupakan banyaknya data test yang *high defect* namun diprediksi *low defect* oleh model prediksi.

Accuracy merupakan metrik untuk mengukur performa model prediksi dalam memprediksi jenis defect yang dihasilkan data test secara keseluruhan Recall merupakan metrik untuk mengukur performa model prediksi dalam mendeteksi action yang sebenarnya *high defect*. Penggunaan *ensemble classification tree* bertujuan untuk meningkatkan *accuracy* model prediksi, sedangkan penggunaan *cost sensitive learning* bertujuan untuk meningkatkan *recall* model prediksi.

Uji coba dilakukan pada cuplikan dataset AMS-COMFT *project* yang didapatkan dari [3]. Cuplikan dataset tersebut tidak imbang. Banyaknya data berjenis *high defect* sangat sedikit dibanding dengan data berjenis *low defect*. Oleh karena itu, sebelum digunakan untuk membuat model prediksi dan uji coba, dataset yang berjenis *high defect* di-*upsampling* terlebih dulu agar jumlahnya imbang.

Uji coba dilakukan dalam beberapa skenario. Pada setiap skenario dibandingkan nilai *accuracy* dan *recall* antara metode ABDP dengan metode yang diajukan. Pada setiap uji coba, data *training* dan data *testing* yang digunakan diambil secara acak dari dataset. Karena itu, uji coba ulang menggunakan data dan inputan yang sama dapat memberikan hasil berbeda

A. Skenario 1

Uji coba dilakukan dengan mengubah - ubah banyaknya data *training* dan data *testing* yang digunakan. Untuk setiap pengujian diperhatikan hubungan antara banyaknya data *training* yang digunakan dengan nilai *accuracy* dan *recall*

Pada tiap – tiap uji coba skenario 1, nilai c(l, h), c(h,l) dan banyak *tree* yang digunakan pada setiap uji coba besarnya tetap, yaitu 1, 1 dan 25.

Tabel 5 menunjukkan hasil uji coba skenario 1. Pada tabel 5 ditunjukkan bahwa pada uji coba 1 sampai 4 nilai *accuracy* metode yang diajukan lebih baik dari metode ABDP. Sedangkan pada uji coba 5, *accuracy*-nya sama. Dapat disimpulkan bahwa penggunaan *ensemble classification tree* dapat meningkatkan *accuracy* dibanding menggunakan satu *tree* saja

Namun pada tabel 5 juga ditunjukkan bahwa nilai *recall* metode ABDP dan metode yang diajukan tidak terdapat perbedaan, sama – sama 100%. Diduga bahwa hal ini terjadi karena dataset yang digunakan kurang representatif. Tidak dapat ditarik kesimpulan metode mana yang memberikan recall lebih baik.

TABEL 5. HASIL UJI COBA SKENARIO 1

(# data training, # data testing)	ABDP		Metode yang diusulkan	
	Accuracy (%)	*Recall* (%)	*Accuracy* (%)	*Recall* (%)
(30, 64)	83	100	86	100
(40, 54)	96	100	100	100
(50, 44)	98	100	100	100
(60, 34)	94	100	97	100
(70, 24)	100	100	100	100

B. Skenario 2

Pada skenario 2, banyaknya *tree* yang digunakan pada setiap uji coba berubah – ubah. Sedangkan banyaknya data *training*, banyaknya data *testing*, c(l, h) dan c(h, l) yang digunakan tetap, yaitu 50, 44, 1 dan 1.

Tabel 6 menunjukkan hasil uji coba skenario 1. Pada Tabel 6 ditunjukkan bahwa nilai *accuracy* metode yang diajukan sama dengan metode ABDP pada uji coba 1, 3 dan 5. Sedang pada uji coba 2 dan 4, metode yang diajukan memberikan *accuracy* yang lebih baik. Dapat disimpulkan bahwa metode yang diajukan memberikan *accuracy* yang lebih baik dari metode ABDP

Namun pada Tabel 6 juga ditunjukkan bahwa nilai *recall* metode ABDP dan metode yang diajukan tidak terdapat perbedaan, sama – sama 100%. Diduga bahwa hal ini terjadi karena dataset yang digunakan kurang representatif. Tidak dapat ditarik kesimpulan metode mana yang memberikan *recall* lebih baik.

TABEL 6. HASIL UJI COBA SKENARIO 2

Banyak tree	ABDP		Metode yang diusulkan	
	Accuracy (%)	*Recall* (%)	*Accuracy* (%)	*Recall* (%)
10	80	100	80	100
20	98	100	100	100
30	93	100	93	100
40	95	100	98	100
50	98	100	98	100

C. Skenario 3

Uji coba dilakukan dengan mengubah – ubah nilai c(l, h) dan c(h,l). Sedangkan banyaknya data training, data testing dan banyak tree yang digunakan tetap yaitu 60, 34 dan 20.

Tabel 7 menunjukkan hasil uji coba skenario 1. Pada Tabel 7 ditunjukkan bahwa nilai *accuracy* metode yang diajukan sama dengan metode ABDP pada uji coba 2, 4 dan 5. Sedang pada uji coba 1 dan 3, metode yang

diajukan memberikan *accuracy* yang lebih baik. Dapat disimpulkan bahwa metode yang diajukan memberikan *accuracy* yang lebih baik dari metode ABDP

Namun pada Tabel 7 juga ditunjukkan bahwa nilai *recall* metode ABDP dan metode yang diajukan tidak terdapat perbedaan, sama – sama 100%. Diduga bahwa hal ini terjadi karena dataset yang digunakan kurang representatif. Tidak dapat ditarik kesimpulan metode mana yang memberikan *recall* lebih baik.

V. KESIMPULAN

1. Metode prediksi *defect* yang diusulkan (*ensemble tree* dengan *cost sensitive learning*) memberikan *accuracy* yang lebih baik dari metode ABDP. Hal ini ditunjukkan pada uji coba yang telah dilakukan, metode yang diajukan secara konsisten memberikan *accuracy* yang sama atau lebih baik dari metode ABDP. Dapat disimpulkan untuk *data testing* secara keseluruhan, metode yang diusulkan memiliki performa yang lebih baik
2. Metode yang diusulkan memberikan *recall* yang sama dengan metode ABDP, yaitu 100% di semua uji coba. Karena nilai 100% merupakan batas atas dari nilai *recall*, tidak dapat diambil kesimpulan apakah metode yang diajukan menghasilkan *recall* yang sama atau lebih baik dari metode ABDP. Diduga hal ini disebabkan karena *dataset* yang digunakan kurang representatif.

DAFTAR PUSTAKA

[1] Chang*, Chingpao-Pao, Chu, Chih-Ping, 2007, Defect prevention in software processes: An action-based approach approach, Journal of Systems and Software 80 (4)

[2] Chang, Ching-Pao, Chu, Chih-Ping, Yeh, Yu-Fang, 2008, Integrating in-process software defect prediction with association mining to discover defect pattern, Journal of information and software technology

[3] Chang, Chingpao-Pao, 2008, Software process improvement using multivariate statistical process control and action based defect prediction, Departement of computer science and information engineering, National Chen Kung University, Taiwan

[4] Tan, P.N., Steinbach, M., Kumar, V., 2006, Introduction to Data Mining, Pearson Education, Inc., Boston.