BAB I
PENDAHULUAN

1.1. Latar Belakang

Pengujian perangkat lunak merupakan salah satu fase yang esensial pada tahapan pengembangan perangkat lunak atau *Software Development Life Cycle (SDLC)*. Kualitas dan stabilitas dari perangkat lunak menjadi hal krusial yang perlu diperhatikan karena dari tahapan ini akan diketahui kesalahan, kecacatan, atau kerentanan suatu sistem [1]. Pengujian perangkat lunak memiliki tahapan-tahapan proses yang dilaksanakan secara sistematis atau dikenal dengan *Software Testing Life Cycle (STLC).* STLC mengacu pada tahapan yang spesifik mulai dari analisis kebutuhan, perencanaan pengujian, pembuatan *test case,* penyetingan lingkungan pengujian, dan implementasi pengujian [2]. Pada saat pengujian perangkat lunak, penguji akan mengobservasi jalannya sistem dengan tujuan untuk menemukan kegagalan atau *error.* Kegagalan didefinisikan ketika sistem yang berjalan berbeda dengan kondisi yang diharapkan berdasarkan persyaratan dari sebuah sistem [3].

Terdapat beberapa tipe pengujian perangkat lunak yang paling sering digunakan yakni pengujian fungsional dan struktural. Pengujian fungsional atau yang dikenal dengan pengujian *black-box* merupakan pengujian pada fungsional fitur dengan cara mengamati hasil masukan dan keluaran dari perangkat lunak tanpa pengatahui struktur code perangkat lunak. Sedangkan pengujan struktural atau yang dikenal dengan pengujian *white-box* adalah pengujian perangkat lunak dengan cara menganalisi dan meneliti struktur internal, seperti implementasi kode, alur data, hingga kemungkinan kegagalan dalam perangkat lunak. Pengujian perangkat lunak sangat penting dilakukan karena tidak hanya membawa dampak bagi kinerja perangkat lunak, akan tetapi juga pada reputasi perusahaan hingga kerugian secara finansial [4].

Salah satu insiden kecacatan perangkat lunak pernah terjadi pada aplikasi Uber di Prancis. Bug tersebut menyebabkan data perjalanan pelanggan tetap terlacak meskipun sudah keluar dari aplikasi. Hal ini berujung pada tuntutan hukum sebesar 45 juta dollar terhadap perusahaan Uber. Dalam konteks perngembangan perangkat lunak, biaya untuk memperbaiki (*bug fixing)* sering meningkat secara eksponensial bergantung pada fase pengembangan saat *bug* ditemukan. Menurut prinsip *Cost of Quality,* menemukan dan memperbaiki *bug* pada tahap produksi atau pasca rilis

membutuhkan sumber daya 30-100x lebih besar dibandingkan jika bug tersebut diidentifikasi saat sebelum rilis. Pada tahun 2018, perusahaan perangkat lunak, Tricentis, mengklaim bahwa terdapat 606 laporan *software defects* yang menyebabkan perusahaan merugi sebesar 1.7 juta dollar [5]. Kejadian tersbeut juga menyebabkan 5-20% pengguna mengalami kehilangan kepercayaan terhadap Perusahaan [6]. Dari insiden tersebut mengimplikasikan bahwa *software defects* akan menimbulkan kerugian baik secara finansial maupun reputasi perusahaan. Oleh karena itu, fase pengujian perangkat lunak dianggap fase yang krusial dan sangat penting.

Terdapat beberapa jenis tipe pengujian antara lain pengujian manual dan automasi. Pengujian perangkat lunak secara manual dinilai memerlukan waktu dan sumber daya yang lebih besar, biaya yang tinggi, dan tidak dapat diskalakan karena bergantung pada kapasitas individu [1]. Sedangkan *automation testing* lebih cepat dibandingkan manual, namun kurang adaptif ketika terdapat perubahan sistem. Oleh karena itu, terdapat kebutuhan untuk mengefisiensi proses pengujian dengan mengaplikasikan *automation testing, machine learning* (ML), dan *artificiial intellegence (AI)* [1]. Pendekatan *Machine Learning Based Prediction* diusulkan karena dapat ditingkatkan dalam skala yang besar dan cukup adaptif karena model dilatih secara berkelanjutan dari data *software metrics*. Penggunaan AI dalam pengujian perangkat lunak dinilai lebih hemat, proaktif, efektif, dan ideal untuk sistem kompleks yang membutuhkan prediksi *defects*[7].

Saat ini penelitian tentang pengujian berbasis AI semakin berkembang baik dari kalangan akademisi maupun industri. Pendekatan prediksi *defect* mulai berfokus pada prediksi dibandingkan deteksi. Pendekatan predisksi ini bertujuan untuk mengindentifikasi potensi *defect* pada tahpa awal fase SDLC. Dengan kapabilitas model untuk memprediksi kecacatan perangkat lunak, penguji dapat lebih efisien untuk mengelola sumber daya, memprioritaskan pengujian, dan memperbaiki kualitas dari produk perangkat lunak [1]. Dengan adanya prediksi kecacatan perangkat lunak, sumber daya pengujian bisa lebih optimal dengan mengarahkan fokus ke area yang rawan *defect*. Dengan demikian akan meminimalisir biaya perbaikan dan modifikasi perangkat lunak setelah rilis.

Terdapat beberapa penelitian mengenai *Software Defect Prediction* yang revelan dengan skripsi ini. Salah satunya penelitian dengan judul *Software Defect Prediction Using Ensemble Learning: An ANP Based Evaluation Method*, yang

membahas prediksi *software defects* menggunakan pendekatan *ensemble learning* [8]. Penelitian ini bertujuan untuk mengevaluasi performa algoritma klasifikasi dalam *Software Defect Prediction* (SDP) dengan membandingkan performa *single classifier* (SMO, MLP, KNN, dan Decision Tree) dengan *Ensemble Methods* (*Bagging, Boosting, Stacking,* dan *Voting*). Penelitian ini menggunakan 11 dataset *software defect* project perangkat lunak berbahasa Java dan C++ yang diambil dari repositori publik. Data ini mencakup matriks analitik perangkat lunak seperti kompleksitas dan ukuran code.

Salah satu pendektaan model *machine learning* yang digunakan untuk memprediksi *software defects* adalah klasifikasi atau regresi menggunakan *source code metrics.* Pendekatan ini bertujuan untuk mengidentifikasi modul perangkat lunak yang rentan terhadap kesalaahan berdasarkan analisis *source code metrics*. *Software metrics* yang dihasilkan dari ekstraksi *source code* digunakan untuk membangun model prediktif. *Software metrics* yang umumnya digunakan pada prediksi *software defects* adalah McCabe Metrics, Halstead Metrics, dan *Static Code Metrics* [9]. Pendekatan ini juga dilakukan menggunakan *Chidamber and Kemerer Metrics* atau matriks CK untuk membangun model prediksi *defects* pada *software* [5].

Beberapa penelitian yang membahas mengenai *software defects prediction* menggunakan project berbasis Java sebagai objek penelitian karena memiliki *software metrics* berbasis OOP. Java merupakan bahasa pemrograman berorientasi objek yang memiliki dokumentasi standar pengembangan yang memudahkan untuk pemeliharan produk perangkat lunak berbasis Java. Java memiliki skalabilitas solusi perangkat lunak yang mampu memberikan kinerja dan memiliki skalabilitas yang kuat bagi perusahaan. Popularitas Java dibuktikan dengan fakta bahwa 90% perushaan Fortune menggunakan Java [10].

Tabel 1. 1 Popularitas Bahasa Pemrograman

Nov 2024	Nov 2023	Status	Bahasa	Persentase	Perubahan
1	1	-	Python	22.85%	+8.69%
2	3	∧	C++	10.64%	+0.29%
3	4	∧	Java	9.60%	+1.26%
4	2	∨	C	9.01%	-2.76%
5	5	-	C#	4.96%	-2.67%
6	6	-	JavaScript	3.71%	+0.50%
7	13	∧	Go	2.35%	+1.16%
8	12	∧	Fortran	1.97%	+0.67%

Nov 2024	Nov 2023	Status	Bahasa	Persentase	Perubahan
9	8	∨	Visual Basic	1.95%	-0.15%
10	9	∨	SQL	1.94%	+0.05%

Berdasarkan Tabel 1.1, survei yang dilakukan TIOBE Programming Community Index tahun 2024, Java menduduki peringkat ke-3 dalam bahasa pemrograman terpopular dengan peringkat sebesar 9.6%. Angka popularitas Java pada bulan November tahun 2024 meningkat sebanyak 1.26% dibandingkan pada bulan November tahun 2023 [11]. Menurut survei yang dilakukan oleh Eclipse pada tahun 2019, java juga menjadi bahasa pemrograman terpenting dalam ranah *Artificial Intellegence (AI), Internet of Things (IoT),* dan big data. Dalam bidang AI, Java digunakan untuk pengembangan solusi *Machine Learning, Neural Network,* pemrograman genetik, dan sistem multirobot. Oleh karena itu, dalam skripsi ini Java dipilih sebagai fokus objek karena memiliki tingkat penggunaan yang tinggi di industri. Java juga memiliki relevansi historis dalam penelitian sebelumnya mengenai *software defects.* Dengan menggunakan project berbasis bahasa Java sebagai objek, skripsi ini akan memanfaatkan metriks perangkat lunak berbasis *Object Oriented Programming* pada Java untuk menghasilkan model prediksi *software defects*.

Berdasarkan penelitian sebelumnya, salah satu metode yang digunakan untuk melakukan prediksi *defect* pada perangkat lunak adalah menggunakan algoritma klasifikasi yang dioptimasi dengan metode *ensemble learning.* Metode e*nsemble learning* merupakan salah satu teknik yang banyak digunakan untuk menaikkan performa model dengan menggabungkan beberapa base model. Teknik ini terbukti mampu meningkatkan akurasi model, dibandingkan dengan pendekatan yang hanya menggunakan satu algoritma saja (*Single Classifier)* [12]. Teknik *ensemble* yang digunakan dalam skripsi ini adalah teknik *Stacking,* sebuah metode yang menggabungkan hasil prediksi dari beberapa model dasar (*base classifiers)* untuk menghasilkan model yang memiliki tingkat akurasi lebih tinggi dengan *meta-classifier*. Pendekatan dengan metode *ensemble* untuk memprediksi *software defect* juga pernah dilakukan menggunakan 10 dataset pulik NASA MDP dan menggunakan 13 pengukuran performa yang berbeda [13]. Metode *ensemble* yang digunakan pada penelitian tersebut adalah *stacking* yang menghasilkan nilai roc_auc sebesar 0.7560.

Pendekatan metode *ensemble* untuk prediksi *software defect* juga pernah dilakukan dengan menggunakan pendekatan *Nested-Stacking*. Pada penelitian tersebut

menggunakan dataset Kamei dan Promise dengan atribut *software metrics* Java. Adapun algoritma yang digunakan untuk layer pertama adalah LightGBM, CatBoost, AdaBoost, layer meta model menggunakan MLP, Random Forest, Gradient Boosting Decision Tree, serta menggunakan Logistic Regression sebagai *meta-classifier*. Hasilnya nilai rata-rata ROC-AUC dari metode Nested-Stacking pada dataset Promise sebesar 0.7988 dengan menggunakan Stratified Cross Validation sebanyak 10 [14]. Kekurangan dari penelitian ini adalah nilai performa yang rendah serta kombinasi baseline model dari eksperimen Nested Stacking membuat model menjadi terlalu kompleks dan proses menjadi kurang efisien.

Oleh karena itu, berdasarkan uraian latar belakang tersebut, skripsi ini bertujuan untuk membuat model prediksi *software defects* dengan performa yang lebih tinggi dan proses yang lebih sederhana. Model prediksi *software defects* ini memiliki urgensi untuk meminimalisir kerugian atau dampak dari ditemukannya *software defects* pasca rilis. Dengan melakukan prediksi bug sebelum perangkat lunak dirilis, maka biaya untuk melakukan perbaikan (*bug fixing*) juga akan berkurang. Terlebih dengan adanya alat untuk menganalisis *software metrics* suatu project, teknik seperti ini bisa diimplementasikan diberbagai pengembangan project berbasis *Object Oriented Programming*. Berdasarkan latar belakang tersebut, judul "Prediksi *Software Defect* pada Level *Package* Project Java Menggunakan Metode *Ensemble Learning*" dipilih untuk mengembangkan dan meningkatkan performa model prediksi *software defects* dengan menggunakan metode *ensemble learning*. Luaran yang dihasilkan dalam skripsi ini yakni model prediksi *software defects* dengan performa yang lebih tinggi dari penelitian sebelumnya. Selain itu, model yang dihasilkan akan diimplementasikan pada aplikasi berbasis desktop menggunakan *framework Robotic Process Automation (RPA)* untuk memprediksi *software defects* pada level *package* project Java.

1.2. Rumusan Masalah

Berdasarkan uraian latar belakang di atas, rumusan masalah dalam skripsi ini adalah bagaimana meningkatkan nilai performa model prediksi *software defects* dengan menggunakan metode *Stacking Ensemble Learning* dan mengimplementasikannya pada aplikasi berbasis desktop?

1.3. Tujuan Penelitian

Adapun tujuan dari skripsi ini adalah untuk membangun model prediksi *software defects* dengan akurasi yang lebih tinggi dari penelitian sebelumnya (ROC-AUC 0.7988) dengan menggunakan metode *Stacking Ensemble Learning* dan mengimplementasikannya ke dalam aplikasi berbasis desktop.

1.4. Manfaat Penelitian

Skripsi yang dilakukan memiliki manfaat baik bagi akademisi maupun bagiindustri. Manfaat dari segi akademisi yakni mengembangkan penelitian sebelumnya terkait dengan prediksi *software defects* dengan berbagai pendekatan *ML-Based Prediction* sehingga diharapkan bisa menghasilkan model prediksi dengan performa yang lebih baik. Adapun manfaat dari skripsi ini bagi industri adalah menghasilkan alat pengujian yang mampu melakukan prediksi *software defects* sehingga dapat diimplementasikan oleh *software tester* untuk memprediksi *package* yang memiliki kemungkinan *defects.* Sehingga penguji dan pengembang bisa berfokus untuk melakukan identifikasi dan perbaikan pada *package* terkait.

1.5. Batasan Masalah

Berdasarkan uraian latar belakang dan rumusan masalah diatas, berikut merupakan hal-hal yang menjadi batasan dalam skripsi ini.

1. Dataset yang digunakan pada skripsi ini adalah *software metrics* project yang berasal dari *open source* repositori publik (Github) dengan prasyarat memiliki harsu berbahasa Java dan jumlah *commit* sebanyak 1000.
2. Jumlah baris yang digunakan didataset ini sebanyak 8924 baris yang berasal dari lima project java *open source.*
3. Algoritma yang digunakan sebagai *base learner* pada skripsi ini adalah Random Forest (RF), Extra Trees (ET), Adaptive Boosting (ADA), Gradient Boosting (GB), Histogram-based Gradient Boosting (HGB), XGBoost (XGB), dan CatBoost (CAT)
4. Metode *ensemble learning* yang digunakan pada skripsi ini adalah stacking
5. Evaluasi performa model yang digunakan pada skripsi ini menggunakan nilai ROC-AUC, akurasi, presisi, *recall*, dan f1score

1.6. Sistematika Penulisan

Sistematika penulisan berikut memberikan gambaran dan kerangka skripsi ini dengan pokok bahasan setiap bab sebagai berikut:

BAB I PENDAHULUAN

Pada bab ini akan diuraikan permasalahan yang menjadi latar belakang dan urgensi skripsi prediksi *software defects*. Terdapat juga tinjauan dari penelitian sebelumnya yang menjadi acuan pengembangan skripsi. Kemudian terdapat batasan dan ruang lingkup skripsi pada tahap awal seperti ketentuan pengambilan dataset, jumlah dataset, penentuan level package. Selanjutnya diuraikan juga pemilihan metode dan algoritma yang digunakan dalam memprediksi *software defects.* Pada bab ini juga dijelaskan tujuan skripsi dan batasan masalah pada prediksi *Software Defects*, serta terdapat sistematika penulisan yang akan menjadi acuan dalam penulisan skripsi ini.

BAB II TINJAUAN PUSTAKA

Pada bab ini akan diuraikan landasan teori yang mencakup studi literatur dari penelitian terdahulu mengenai prediksi *Software Defects* yang mencakup rangkuman tujuan, metode, dan hasil penelitian. Terdapat beberapa landasan teori yang relevan dengan objek skripsi yakni tentang pengujian perangkat lunak, prediksi *software defects*, matriks *software defects*, metode dan algoritma yang digunakan, evaluasi model, hingga *deployment.* Diuraikan juga penelitian-penelitian yang mendukung metode ensemble dalam konteks prediksi *software defects.*

BAB III DESAIN DAN IMPLEMENTASI SISTEM

Pada bab ini akan diuraikan metodologi atau tahapan-tahapan yang digunakan dalam skripsi prediksi *software defect* pada project Java menggunakan metode *Ensemble Learning*. Tahapan pertama adalah studi literatur untuk memahami teori dan penelitian terdahulu yang relevan dengan skripsi. Selanjutnya, terdapat tahapan pengumpulan data *software metrics* dan *bugs documentation* dari lima project Java yang dikumpulkan menjadi dataset. Kemudian terdapat tahapan preprocessing data yang merupakan tahap pembersihan data, normalisasi data, dan pembagian data. Tahapan keempat adalah pelatihan data *base learner* (Level-0) dan pelatihan *model learner*

(Level-1). Selanjutnya akan dilakukan pengujian dan evaluasi model. Tahapan terakhir dalam skripsi ini adalah *deployment model.*

BAB IV PEMBAHASAN

Pada bab ini akan diuraikan hasil dari setiap tahapan sesuai dengan tahapan yang ada pada bab metodologi penelitian. Dimulai dengan tahapan pengumpulan data *software metrics* dari project Java dan *bugs documentation* yang akan digunakan pada proses labeling data. Sebelum melalui tahap labeling data, dilakukan tahapan preprocessing data yang merupakan tahap pembersihan data, normalisasi data, dan pembagian data. Data yang tersisi setelah *preprocessing* berjumlah 8.924 baris data. Diterapkan juga beberapa metode untuk menangani ketidakseimbangan kelas data dengan menggunakan teknik SMOTE, SMOTE Tomek-Links, dan *Stratified-Undersampling.* Kemudian dari beberepa skenario tersebut diterapkan pada pelatihan data *base learner* (Level-0). Pada pelatihan base learner diterapkan pula beberapa skenario hiperparameter. Setiap *base m*odel dengan hiperparameter yang memiliki performa terbaik akan dipilih untuk *training* pada *model learner.* Pada bab ini juga dijelaskan hasil dari pengujian dan evaluasi model. Serta terdapat juga pembahasan mengenai hasil implementasi model pada aplikasi RPA berbasi desktop.

BAB V PENUTUP

Pada bab ini akan diuraikan rangkuman yang menyimpulkan proses-proses dan temuan yang didapatkan dari skripsi ini. Terdapat juga rekomendasi dan saran yang dapat dilakukan pada penelitian selanjutnya.

DAFTAR PUSTAKA

Berisi tinjauan literatur yang menjadi pedoman dan teori pendukung penyusunan skripsi