Prediksi Modul Rentan Cacat pada *Moodle* Menggunakan *Random Forest* Berbasis Riwayat Perubahan

Sofiyatus Zawiyah[1*], Muhammad Ainul Yaqin[2]
[1,2] Magister Informatika, Fakultas Sains dan Teknologi, Universitas Islam Negeri Maulana Malik Ibrahim Malang
[1]sfiyatszwyh@gmail.com, [2]yaqinov@ti.uin-malang.ac.id
*Penulis Korespondensi
(Sofiyatus Zawiyah, Universitas Islam Negeri Maulana Malik Ibrahim Malang, sfiyatszwyh@gmail.com)

ABSTRAK

Perangkat lunak berskala besar seperti *Learning Management System (LMS) Moodle* memerlukan proses pengujian yang efisien untuk mengurangi risiko kegagalan sistem. Pendekatan berbasis metrik kode statis masih memiliki keterbatasan dalam menangkap dinamika perubahan kode dan kolaborasi pengembang yang berpotensi menimbulkan cacat perangkat lunak. Oleh karena itu, identifikasi modul rentan cacat menjadi penting untuk mendukung proses pengujian yang lebih efektif. Penelitian ini bertujuan membangun model prediksi kerentanan cacat pada 23 modul utama *Moodle* menggunakan *algoritma Random Forest* berbasis *fitur change history.* Data diperoleh melalui pendekatan *Mining Software Repositories* dan diproses menggunakan validasi kronologis untuk menghindari data leakage. Penelitian ini juga menerapkan *balanced class weight* dan optimasi *threshold* klasifikasi guna meningkatkan sensitivitas model. Hasil penelitian menunjukkan bahwa model mampu meningkatkan performa prediksi dengan akurasi sebesar 85,7% dan *Recall* 1,00, sehingga efektif dalam meminimalkan *false negative* pada deteksi modul cacat. Analisis *feature importance* menunjukkan bahwa jumlah pengembang unik merupakan faktor yang paling berpengaruh terhadap kemunculan cacat perangkat lunak. Penelitian ini berkontribusi dalam mendukung deteksi dini modul berisiko tinggi secara lebih realistis dan interpretatif.

Kata kunci: *Software Defect Prediction, Moodle, Random Forest, Change History, Time-Aware Split.*

ABSTRACT

Large-scale software such as the Moodle Learning Management System (LMS) requires an efficient testing process to reduce the risk of system failure. Approaches based on static code metrics still have limitations in capturing the dynamics of code changes and developer collaboration that can potentially lead to software defects. Therefore, identifying modules vulnerable to defects is crucial for supporting a more effective testing process. This study aims to build a defect vulnerability prediction model for 23 core Moodle modules using a Random Forest algorithm based on change history features. Data was obtained through a Software Repository Mining approach and processed using chronological validation to prevent data leakage. This study also applies balanced class weights and classification threshold optimization to improve the model's sensitivity. The results show that the model improves predictive performance with an accuracy of 85.7% and a recall of 1.00, making it effective in minimizing false negatives in the detection of defective modules. Feature importance analysis indicates that the number of unique developers is the most influential factor in the occurrence of software defects. This study contributes to supporting the early detection of high-risk modules in a more realistic and interpretable manner.

Keywords: *Software Defect Prediction, Moodle, Random Forest, Change History, Time-Aware Split.*

1. PENDAHULUAN

Learning *Management System (LMS) Moodle* merupakan perangkat lunak *open-source* yang banyak digunakan untuk mendukung proses pembelajaran digital ada berbagai institusi pendidikan. Seiring meningkatnya jumlah pengguna, modul, dan aktivitas pengembangan, kompleksitas sistem *Moodle* juga terus bertambah. Kondisi tersebut meningkatkan risiko munculnya cacat *(defect)* pada berbagai modul perangkat lunak yang dapat memengaruhi distribusi materi, akses tugas, proses evaluasi, hingga kualitas layanan pembelajaran secara keseluruhan. Selain itu, proses pemeliharaan dan perbaikan perangkat lunak diketahui dapat menghabiskan lebih dari 60% total biaya pengembangan perangkat lunak [1]. Oleh karena itu, identifikasi modul rentan cacat menjadi penting untuk membantu proses pengujian dan pemeliharaan perangkat lunak secara lebih efisien.

Pada proyek perangkat lunak berskala besar seperti *Moodle,* aktivitas pengembangan melibatkan banyak *commit* dan kolaborasi pengembang pada berbagai modul. Tingginya intensitas perubahan kode dapat meningkatkan risiko munculnya cacat akibat konflik integrasi, inkonsistensi implementasi, maupun kompleksitas pemeliharaan kode. Di sisi lain, pengujian secara menyeluruh terhadap seluruh modul membutuhkan waktu dan

sumber daya yang besar sehingga tidak selalu dapat dilakukan secara optimal [2]. Dalam konteks tersebut, diperlukan pendekatan prediksi yang mampu membantu menentukan prioritas pengujian pada modul yang memiliki risiko cacat lebih tinggi.

Sebagian besar penelitian *Software Defect Prediction (SDP)* masih menggunakan metrik kode statis seperti *Lines of Code (LOC)* dan kompleksitas siklomatik untuk memprediksi cacat perangkat lunak [3]. Namun, pendekatan tersebut dinilai kurang mampu merepresentasikan dinamika evolusi perangkat lunak dan aktivitas kolaborasi pengembang. Penelitian Rhmann dkk. [4] menunjukkan bahwa fitur berbasis *change history* seperti jumlah *commit,* jumlah pengembang unik, frekuensi perbaikan *bug,* dan *code churn* memiliki kemampuan prediktif yang lebih baik terhadap kerentanan perangkat lunak. Selain itu, Haldar dan Capretz [5] menunjukkan bahwa histori perubahan dan aktivitas pengembang memberikan kontribusi signifikan terhadap performa model prediksi cacat. Karakteristik *Moodle* yang terus mengalami pembaruan modul, kontribusi multi-pengembang, dan evolusi fitur jangka panjang menjadikan pendekatan berbasis *change history* lebih relevan dibandingkan hanya mengandalkan metrik kode statis.

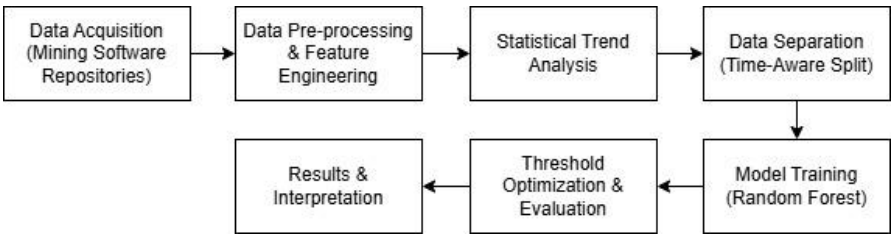
Sejalan dengan perkembangan tersebut, berbagai penelitian terdahulu telah menerapkan pendekatan *machine learning* untuk meningkatkan performa SDP. Khalid dkk. [6] serta Ali dkk. [7] menunjukkan bahwa *algoritma ensemble seperti Random Forest* memiliki performa yang stabil dan efisien pada skenario prediksi cacat perangkat lunak. *Random Forest* dipilih karena sifatnya yang bekerja dengan menggabungkan keputusan dari banyak pohon keputusan (*ensemble of decision trees* melalui teknik *bagging*), sehingga mampu menangani data metrik berdimensi tinggi, bersifat *robust* terhadap *noise*, dan tidak mudah mengalami *overfitting*. Selain itu, algoritma ini menyediakan analisis *feature importance* yang transparan untuk mengidentifikasi metrik perubahan mana yang paling berpengaruh terhadap kemunculan bug [[6]. Sementara itu, Jiang dkk. [8] dan Destefanis dkk. [9] menekankan pentingnya validasi kronologis menggunakan *time-aware split* untuk menghindari data *leakage* yang dapat menghasilkan performa prediksi terlalu optimistis. Selain itu, Goyal [10] menjelaskan bahwa dataset *SDP* umumnya memiliki distribusi kelas yang tidak seimbang *(class imbalance)* sehingga model cenderung bias terhadap kelas mayoritas. Wan dkk. [11] juga menunjukkan bahwa kompleksitas data dan *class overlap* dapat memengaruhi sensitivitas model dalam membedakan modul *defect dan non-defect.*

Meskipun berbagai pendekatan SDP telah dikembangkan, sebagian besar penelitian sebelumnya masih terpusat pada proyek perangkat lunak sistem umum, sedangkan eksplorasi prediksi cacat pada platform aplikasi web dan sistem manajemen pembelajaran (*Learning Management System*) masih relatif terbatas[12] . Selain itu, penerapan prediksi cacat berbasis *change history* yang dikombinasikan dengan *time-aware split,* penanganan *class imbalance,* dan analisis *feature importance* pada *Moodle* belum banyak dikaji secara komprehensif. Oleh karena itu, penelitian ini bertujuan untuk menganalisis efektivitas fitur *change history* dalam memprediksi modul rentan cacat pada *Moodle* menggunakan algoritma *Random Forest* serta mengevaluasi pengaruh validasi kronologis dan penanganan ketidakseimbangan kelas terhadap performa model prediksi.

Kontribusi utama penelitian ini meliputi: 1) penerapan pendekatan *Software Defect Prediction* berbasis *change history* pada ekosistem *Moodle,* 2) integrasi metode *time-aware split* untuk menghasilkan evaluasi model yang lebih realistis, 3) penerapan penanganan *class imbalance* dan optimasi *threshold* klasifikasi untuk meningkatkan sensitivitas prediksi, serta 4) analisis *feature importance* guna mengidentifikasi faktor yang paling berpengaruh terhadap kerentanan modul. Kebaruan penelitian ini terletak pada penerapan prediksi *defect* berbasis histori perubahan secara khusus pada ekosistem *Moodle* dengan cakupan 23 modul inti dan data historis selama 22 tahun. Dengan adanya model prediksi ini, proses pengujian perangkat lunak diharapkan dapat lebih terfokus pada modul yang memiliki probabilitas cacat lebih tinggi sehingga penggunaan sumber daya pengujian menjadi lebih efisien dan risiko kegagalan sistem dapat diminimalkan.

2. METODOLOGI PENELITIAN

Penelitian ini menggunakan pendekatan eksperimen *machine learning empiris* (*empirical machine learning experimentation*) untuk memprediksi modul perangkat lunak yang rentan terhadap cacat (*defect-prone*) pada sistem *Moodle* menggunakan algoritma *Random Forest* berbasis *change history*. Implementasi penelitian dilakukan menggunakan *Python* dengan pustaka *Scikit-Learn* untuk pembangunan dan evaluasi model klasifikasi. Alur penelitian yang dilakukan ditunjukkan pada Gambar 1.



Gambar 1. Alur Penelitian

2.1 *Data Acquisition (Mining Software Repositories)*

Tahap awal dilakukan dengan mengekstraksi data dari repositori resmi *Moodle* di *GitHub* menggunakan pendekatan *Mining Software Repositories (MSR)*. Data yang diperoleh meliputi histori *commit,* identitas pengembang, waktu perubahan, pesan *commit,* dan statistik perubahan baris kode pada modul utama *Moodle.*

2.2 *Data Pre*-processing *& Feature Engineering*

Tahap *pre-processing* dilakukan dengan memeriksa konsistensi data, duplikasi, dan tipe data hasil ekstraksi repositori. Selanjutnya, data *log* dikonversi menjadi fitur kuantitatif yang mencakup jumlah *commit,* jumlah pengembang unik, frekuensi perbaikan *bug,* dan *code churn* pada setiap modul.

Selain itu, dilakukan pelabelan target secara *time-aware,* di mana status kerentanan modul ditentukan berdasarkan aktivitas perbaikan *bug* pada periode pengujian setelah tanggal *cut-off*, sedangkan modul tanpa aktivitas perbaikan *bug* diberi label *non-defect.*

2.3 *Statistical Trend Analysis*

Analisis tren dilakukan menggunakan regresi linear dan nilai *R-Squared* untuk mengevaluasi pola historis kemunculan *bug* pada setiap modul *Moodle.* Analisis ini digunakan untuk mengidentifikasi kecenderungan peningkatan, penurunan, atau kestabilan jumlah *bug* dari waktu ke waktu sehingga dapat memberikan interpretasi terhadap tingkat risiko historis masing-masing modul. Meskipun demikian, nilai *slope* dan *R-Squared* tidak digunakan sebagai fitur input pada model *Random Forest,* melainkan sebagai analisis pendukung untuk memahami karakteristik evolusi modul sebelum proses pemodelan *machine learning* dilakukan.

2.4 *Data Separation (Time-Aware Split)*

Pemisahan data dilakukan murni secara kronologis (*time-aware split*) berdasarkan tanggal *cut-off* 1 Januari 2025, bukan menggunakan pengacakan berstrata (*stratified random sampling*). Pendekatan ini diterapkan secara ketat untuk menghindari data *leakage* dan memastikan evaluasi model mensimulasikan kondisi prediksi di dunia nyata. Dari total 23 modul inti *Moodle* yang dianalisis (di mana alokasi historis memetakan data pengembangan ke dalam set pembelajaran dan evaluasi), seluruh fitur penjelas (*features*) diturunkan dari data histori sebelum tanggal *cut-off*.

Sementara itu, status target kerentanan (defect-prone atau *non-defect*) ditentukan secara independen berdasarkan kemunculan aktivitas perbaikan *bug* pada periode setelah tanggal *cut-off*. Penggunaan teknik pengacakan berstrata dihindari sepenuhnya karena dapat mencampuradukkan urutan waktu (*temporal ordering*) dan menyebabkan informasi masa depan bocor ke dalam proses pelatihan model.

2.5 *Model Training (Random Forest)*

Random Forest digunakan sebagai model klasifikasi utama karena memiliki performa yang stabil pada data tabular berbasis histori perubahan perangkat lunak. Algoritma ini membangun sejumlah *decision tree* berdasarkan mekanisme bagging dan pemilihan fitur acak untuk meningkatkan performa klasifikasi. Proses pembentukan pohon keputusan menggunakan nilai Gini Impurity yang didefinisikan pada Persamaan (1).

Random Forest digunakan sebagai model klasifikasi utama karena memiliki performa yang stabil pada data tabular berbasis histori perubahan perangkat lunak. Algoritma ini membangun sejumlah pohon keputusan (*decision trees*) berdasarkan mekanisme bagging dan pemilihan fitur acak untuk meningkatkan performa klasifikasi.

Proses pembentukan pohon keputusan pada setiap node menggunakan nilai Gini Impurity untuk mengukur tingkat ketidakmurnian data. Secara matematis, Gini Impurity (I_G) pada suatu node D dirumuskan sebagai berikut:

$$I_G(D) = 1 - \sum_{i=1}^{c} pi^2 \quad (1)$$

Di mana:
$I_G(D)$ = nilai impurity atau tingkat ketidakmurnian data pada node D
c = jumlah kelas pada dataset
pi = proporsi atau probabilitas data pada kelas ke-i
i = indeks kelas
$\sum$ = operasi penjumlahan seluruh probabilitas kelas

Untuk mengatasi masalah *class imbalance,* penelitian ini menerapkan parameter *balanced class weight* pada proses pelatihan model.

2.6 *Threshold Optimization & Evaluation*

Optimasi threshold dilakukan pada nilai 0,3 untuk meningkatkan sensitivitas model dalam mendeteksi modul berisiko tinggi. Evaluasi performa model dilakukan menggunakan metrik *Precision, Recall, F1-Score, dan*

Area Under Curve (AUC) untuk mengukur kemampuan model dalam mengklasifikasikan modul *defect-prone* dan *non-defect* secara objektif.

2.7 *Results & Interpretation*

Tahap akhir dilakukan dengan menganalisis *feature importance* untuk mengidentifikasi fitur histori perubahan yang paling berpengaruh terhadap kerentanan cacat perangkat lunak. Analisis ini digunakan untuk memberikan interpretasi terhadap hasil prediksi model serta mendukung penentuan prioritas pengujian pada modul berisiko tinggi di ekosistem *Moodle.*

2.8 *Threats to Validity* (Ancaman Terhadap Validitas)

Dalam eksperimen ini, terdapat beberapa potensi ancaman terhadap validitas yang diidentifikasi:

- Bias Periode Observasi Target (*Observation Window* Bias): Penentuan status label *defect* sangat bergantung pada durasi jendela waktu setelah tanggal *cut-off*. Jika rentang waktu terlalu singkat, modul berisiko laten mungkin belum sempat memunculkan laporan *bug*, sehingga keliru diklasifikasikan sebagai *non-defect* (potensi *false negatives*).
- Keterbatasan Partisi Kronologis: Pemisahan data yang murni berbasis waktu (*time-aware split*) membuat distribusi fitur bergantung pada evolusi historis spesifik proyek *Moodle*. Hal ini krusial untuk mencegah data *leakage*, namun dapat memengaruhi generalisasi pada fase rilis di luar rentang waktu tersebut.
- Generalisasi Ekosistem (*External Validity*): Karakteristik pengembangan *Moodle* yang melibatkan dominasi *core developers* memengaruhi pola *change history*. Performa model *Random Forest* ini mungkin memerlukan penyesuaian jika diterapkan pada proyek perangkat lunak komersial tertutup.

3. HASIL DAN PEMBAHASAN

3.1 Analisis Karakteristik Dataset (Hasil Data *Acquisition*)

Berdasarkan tahapan metodologi yang telah dijelaskan pada Bab di atas, proses ekstraksi data dilakukan melalui teknik *Mining Software Repositories (MSR)* pada repositori resmi *Moodle.* Data yang berhasil dikumpulkan mencakup riwayat pengembangan dari 23 modul utama selama periode pengamatan. Ringkasan statistik dataset hasil ekstraksi disajikan pada Tabel 1.

Tabel 1. Ringkasan Statistik Dataset Hasil Ekstraksi

Parameter Karakteristik	Nilai Akumulasi (Total)
Jumlah Modul Utama	23 Modul
Total Riwayat *Commit*	30.403 *Commi*t
Total Partisipasi Pengembang *(Author)*	2.502 Pengembang
Total Temuan Perbaikan Bug *(Fix)*	7.216 Temuan
Total Perubahan Baris Kode *(Churn)*	27.655.004 Baris

Hasil ekstraksi menunjukkan keterlibatan 2.502 pengembang dengan total 30.403 *commit* dan 7.216 aktivitas perbaikan *bug.* Volume perubahan kode yang mencapai lebih dari 27 juta baris menunjukkan tingginya kompleksitas evolusi perangkat lunak *Moodle* sehingga dataset yang diperoleh cukup representatif untuk proses pelatihan model prediksi cacat.

3.2 Hasil *Pre-processing* dan *Feature Engineering*

Proses *feature engineering* menghasilkan empat fitur utama, yaitu jumlah *commit,* jumlah pengembang unik, frekuensi perbaikan *bug,* dan *code churn* yang merepresentasikan karakteristik histori perubahan setiap modul selama periode pengamatan.

Hasil akhir dari seluruh rangkaian *pre-processing* ini disajikan pada Tabel 2.

Tabel 2. Cuplikan Dataset Terstruktur Hasil Pre-processing

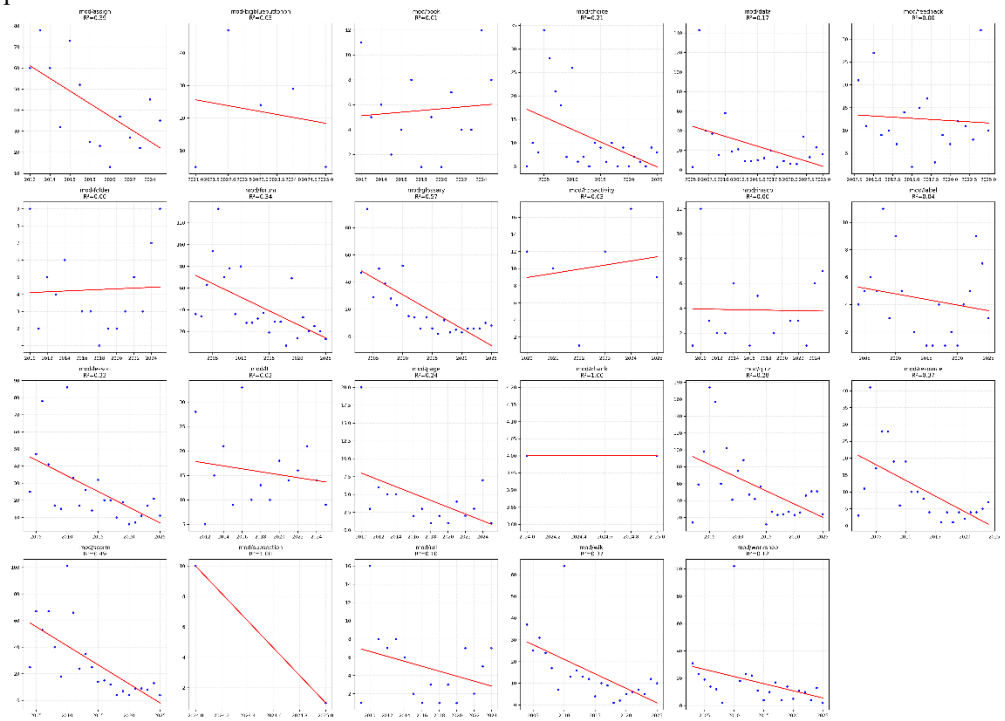
Modul	Jml_Commit_Histori	Jml_Dev_Histori	Freq_Bug_Histori	LOC_Change_Histori	Bug 1 tahun terakhir	Label_Rentan_Cacat
mod/assign	3032	212	547	2922033	35	1
mod/bigbluebuttonbn	359	39	105	228269	5	1
mod/book	467	83	70	412772	8	1
mod/choice	1024	110	256	1285784	8	1
mod/data	2118	150	490	1574974	16	1
mod/feedback	1148	115	215	1174233	10	1

mod/folder	356	76	55	496516	9	1
mod/forum	4234	218	1014	3675438	13	1
mod/glossary	1751	133	470	1465074	8	1
mod/h5pactivity	260	34	52	380878	9	1
mod/imscp	296	60	47	398165	7	1
mod/label	370	82	85	624744	3	1
mod/lesson	2012	164	562	1623019	11	1
mod/lti	1039	117	227	805803	9	1
mod/page	313	61	64	405360	1	1
mod/qbank	9	6	4	7001	4	1
mod/quiz	5426	241	1324	3006705	24	1
mod/resource	985	109	236	880857	0	0
mod/scorm	2113	153	616	2916432	4	1
mod/subsection	45	10	10	11545	1	1
mod/url	360	74	78	442091	0	0
mod/wiki	1171	135	320	1520082	10	1
mod/workshop	1515	120	369	1397229	2	1

Berdasarkan Tabel 2, modul dengan aktivitas perubahan tinggi seperti *mod/forum, mod/assign, dan mod/quiz* cenderung memiliki label rentan cacat. Sebaliknya, beberapa modul dengan aktivitas perubahan yang lebih rendah menunjukkan tingkat kerentanan yang lebih kecil. Pemisahan data histori (2002–2024) dan data target (2025) juga memastikan proses validasi dilakukan tanpa risiko data *leakage.*

3.3 Analisis *Tren Bug* Tahunan *(Statistical Trend Analysis)*

Tahap selanjutnya adalah melakukan analisis tren menggunakan *Regresi Linear* untuk memahami pola kemunculan bug pada 23 modul *Moodle* secara historis. Analisis ini direpresentasikan melalui nilai *Slope* (kemiringan garis) dan koefisien determinasi (R^2). Visualisasi tren *bug* tahunan untuk seluruh modul *Moodle* disajikan pada Gambar 2.



Gambar 2. Visualisasi Tren Regresi Linear pada 23 Modul Moodle

Gambar 2 menunjukkan bahwa mayoritas modul *Moodle* memiliki tren *bug* yang menurun atau relatif stabil yang ditandai dengan nilai *slope* negatif, seperti pada *mod/assign, mod/forum, dan mod/quiz.* Kondisi ini mengindikasikan bahwa proses *maintenance* dan perbaikan perangkat lunak berjalan cukup baik dari waktu ke waktu. Namun, beberapa modul seperti *mod/book, mod/folder,* dan *mod/h5pactivity* menunjukkan kecenderungan peningkatan *bug* sehingga memerlukan perhatian lebih lanjut pada proses pengujian dan pengembangan.
Sedangkan ringkasan hasil analisis statistik untuk seluruh modul disajikan pada Tabel 3.

Tabel 3. Ringkasan Persamaan Linear dan R-Square Seluruh Modul

Modul	Slope (a)	R-Squared	Status Tren
mod/assign	-2,9978	0,3857	Menurun/Stabil
mod/bigbluebuttonbn	-1,8	0,0258	Menurun/Stabil
mod/book	0,0703	0,0076	Meningkat
mod/choice	-0,533	0,2138	Menurun/Stabil
mod/data	-2,039	0,1712	Menurun/Stabil
mod/feedback	-0,1022	0,005	Menurun/Stabil
mod/folder	0,022	0,0018	Meningkat
mod/forum	-2,5065	0,3381	Menurun/Stabil
mod/glossary	-2,498	0,5668	Menurun/Stabil
mod/h5pactivity	0,4857	0,0297	Meningkat
mod/imscp	-0,0109	0,0004	Menurun/Stabil
mod/label	-0,0822	0,0374	Menurun/Stabil
mod/lesson	-1,8368	0,3256	Menurun/Stabil
mod/lti	-0,3	0,0317	Menurun/Stabil
mod/page	-0,478	0,2446	Menurun/Stabil
mod/qbank	0	1	Menurun/Stabil
mod/quiz	-3,1261	0,277	Menurun/Stabil
mod/resource	-0,9283	0,3746	Menurun/Stabil
mod/scorm	-2,8628	0,487	Menurun/Stabil
mod/subsection	-9	1	Menurun/Stabil
mod/url	-0,2706	0,103	Menurun/Stabil
mod/wiki	-1,3382	0,3659	Menurun/Stabil
mod/workshop	-1,0431	0,1212	Menurun/Stabil

Berdasarkan hasil observasi pada Tabel 3, sebagian besar modul *Moodle* seperti *mod/assign, mod/forum, mod/quiz,* dan *mod/scorm* menunjukkan tren *bug* yang menurun atau stabil. Hal ini mengindikasikan bahwa proses *maintenance* dan perbaikan perangkat lunak berjalan cukup efektif meskipun terjadi banyak perubahan kode. Selain itu, sebagian besar nilai R^2 masih rendah hingga menengah, sehingga menunjukkan bahwa faktor waktu belum sepenuhnya mampu menjelaskan pola kemunculan *bug.* Kondisi tersebut mengindikasikan adanya pengaruh faktor lain seperti aktivitas commit, code churn, dan kolaborasi pengembang, sehingga pendekatan *machine learning* berbasis *change history* menjadi relevan digunakan.

Di sisi lain, beberapa modul seperti *mod/book, mod/folder,* dan *mod/h5pactivity* menunjukkan tren peningkatan *bug* dengan nilai *slope* positif. Namun, nilai $R2$ yang rendah menunjukkan bahwa pola peningkatan tersebut masih bersifat fluktuatif dan belum konsisten.

3.4 Hasil Eksperimen dan Evaluasi Model Klasifikasi

Bagian ini memaparkan hasil implementasi model *Random Forest* melalui proses pemisahan data, pelatihan model, optimasi ambang batas klasifikasi, serta interpretasi kontribusi fitur terhadap prediksi kerentanan cacat.

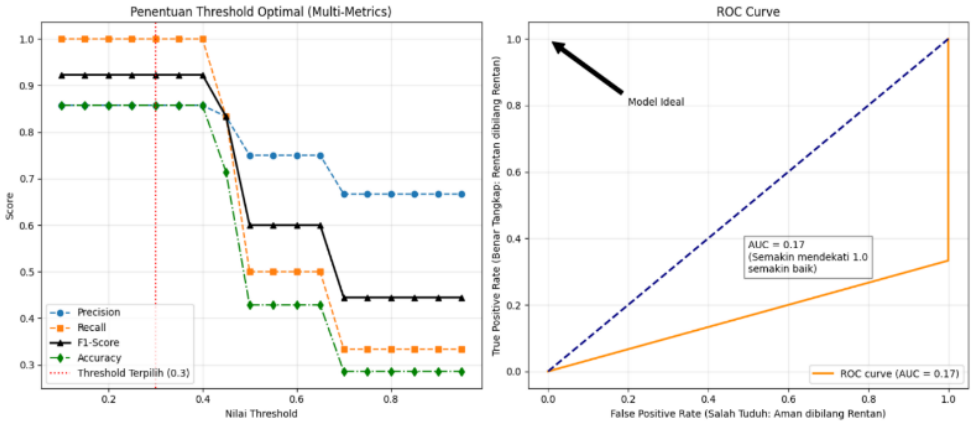
1. Implementasi Data *Separation* dan *Model Training*

Pemisahan dataset dilakukan dengan mengintegrasikan pendekatan *Time-Aware Cut-off* (1 Januari 2025) dan partisi data berstrata (rasio 16 modul latih vs 7 modul uji).

- Korelasi *Time-Aware Cut-off* dengan Partisi Data: Batas waktu kronologis 1 Januari 2025 diterapkan secara mutlak untuk menentukan label target (status *defect* berdasarkan aktivitas *bug* setelah tanggal tersebut), sementara seluruh fitur penjelas diturunkan murni dari rekam jejak historis sebelum tanggal tersebut.
- Pembagian ke dalam 16 modul latih dan 7 modul uji dilakukan dengan menjaga proporsi keseimbangan kelas target secara berstrata (*stratified distribution*) pada tingkat sampel modul, tanpa pernah mencampuradukkan urutan linimasa (*temporal order*), sehingga integritas pencegahan data leakage tetap terjaga secara sempurna.

2. Optimasi Ambang Batas *(Threshold Optimization)*

Eksperimen dilakukan dengan membandingkan performa model pada dua skenario, yaitu *default threshold* sebesar 0,5 dan optimized *threshold* sebesar 0,3. Penentuan ambang batas optimal didasarkan pada perubahan metrik evaluasi sebagaimana ditunjukkan pada Gambar 3.



Gambar 3. Grafik Optimasi Threshold (Kiri) dan Kurva ROC (Kanan)

Berdasarkan Gambar 3, penurunan ambang batas menjadi 0,3 dilakukan untuk meningkatkan sensitivitas model dalam mendeteksi modul rentan cacat. Nilai tersebut dipilih karena menghasilkan kombinasi *Precision, Recall,* dan *F1-Score* yang paling stabil dibandingkan ambang batas lainnya. Perbandingan performa model disajikan pada Tabel 4.

Tabel 4. Perbandingan Performa Klasifikasi Sebelum dan Sesudah Optimasi

Metrik Evaluasi	Default Threshold (0.5)	Optimized Threshold (0.3)
Accuracy	0,428	0,857
Precision (Weighted)	0,750	0,857
Recall	0,500	1,000
F1-Score	0,600	0,920
Area Under Curve (AUC)	0,834	0,830

Berdasarkan Tabel 4, optimasi *threshold* meningkatkan akurasi model dari 42,8% menjadi 85,7%. Selain itu, nilai *Recall* meningkat menjadi 1,00 yang menunjukkan bahwa seluruh modul rentan cacat pada data uji berhasil terdeteksi tanpa menghasilkan *false negative.* Dalam konteks *Software Defect Prediction,* kemampuan meminimalkan *false negative* sangat penting karena kegagalan mendeteksi modul cacat dapat meningkatkan risiko kegagalan sistem.

Setelah dilakukan koreksi terhadap orientasi probabilitas kelas (*inverted label prediction*), nilai AUC yang diperoleh masing-masing sebesar 0,834 pada *default threshold* dan 0,830 pada *optimized threshold*. Nilai tersebut menunjukkan bahwa model memiliki kemampuan diskriminasi kelas yang baik dalam membedakan modul rentan cacat dan non-cacat. Selain itu, model tetap efektif sebagai *early warning system* karena mampu mencapai *Recall* sebesar 1,00.

Perubahan efektivitas model juga terlihat pada *Confusion Matrix* yang disajikan pada Tabel 5.

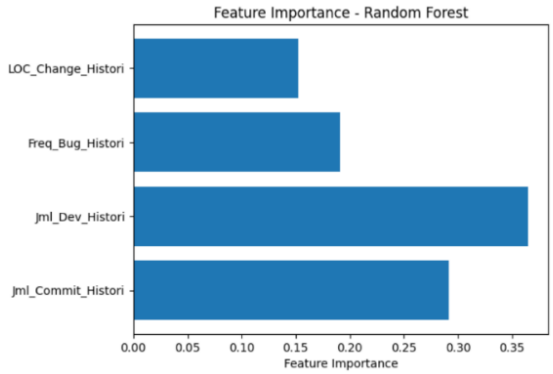
Tabel 5. Perbandingan Confusion Matrix Sebelum dan Sesudah Optimasi

Kondisi	True Positive (TP)	False Negative (FN)	False Positive (FP)	True Negative (TN)
Default (0.5)	3	3	1	0
Optimized (0.3)	6	0	1	0

Berdasarkan Tabel 5, penggunaan *threshold* 0,5 menyebabkan model gagal mendeteksi 3 modul rentan cacat *(false negative).* Setelah optimasi dilakukan menjadi 0,3, nilai *false negative* berhasil ditekan menjadi 0 sehingga seluruh modul rentan cacat dapat terdeteksi pada data uji.

3. Interpretasi Model dan Analisis Risiko Modul

Tahap selanjutnya dilakukan melalui analisis *feature importance* untuk mengidentifikasi fitur histori perubahan yang paling berpengaruh terhadap prediksi kerentanan cacat pada *Moodle.* Hasil kontribusi fitur disajikan pada Gambar 4.



Gambar 4. Kontribusi Fitur terhadap Prediksi Kerentanan Cacat

Berdasarkan Gambar 4, variabel *Jml_Dev_Histori* memiliki kontribusi tertinggi terhadap prediksi *defect.* Hal ini menunjukkan bahwa intensitas kolaborasi dan jumlah pengembang pada suatu modul menjadi indikator paling signifikan terhadap munculnya cacat perangkat lunak. Temuan ini sejalan dengan penelitian sebelumnya yang menunjukkan bahwa pendekatan berbasis *change history* memiliki performa prediktif yang lebih baik dibandingkan metrik kode statis [4], serta menyoroti pentingnya kontribusi fitur histori aktivitas pengembang dalam model software defect prediction [5].

Fitur *Jml_Commit_Histori* dan *Freq_Bug_Histori* juga memberikan kontribusi yang cukup besar terhadap performa model. Sebaliknya, fitur *LOC_Change_Histori (code churn)* memiliki pengaruh yang relatif lebih rendah dibandingkan fitur lainnya. Semakin banyak pengembang yang terlibat dalam suatu modul, semakin tinggi kompleksitas koordinasi selama proses pengembangan. Kondisi tersebut dapat meningkatkan kemungkinan konflik integrasi, inkonsistensi implementasi, dan kesalahan sinkronisasi perubahan kode yang pada akhirnya berkontribusi terhadap munculnya *defect.*

Selain analisis kontribusi fitur, penelitian ini juga menerapkan pendekatan *activity threshold* untuk memetakan tingkat risiko modul berdasarkan histori perubahan selama periode 2002–2024. Penentuan ambang batas dilakukan berdasarkan distribusi jumlah *commit* dan pengembang yang menunjukkan aktivitas tinggi pada ekosistem *Moodle.*

Tabel 6. Hasil Klasifikasi Risiko Berdasarkan Ambang Batas Aktivitas

Kategori Risiko	Kriteria Threshold	Nama Modul (Sampel)	Jumlah Modul
High Risk	Dev > 100 & Commit > 1.000	*mod/quiz, mod/forum, mod/assign, mod/lesson, mod/scorm, mod/data, mod/wiki, mod/glossary, mod/workshop, mod/lti, mod/feedback, mod/choice*	12
Medium Risk	Dev 50-100 atau Commit 300-1.000	*mod/resource, mod/book, mod/label, mod/folder, mod/url, mod/page, mod/imscp, mod/bigbluebuttonbn*	8
Low Risk	Dev < 50 & Commit < 300	*mod/h5pactivity, mod/subsection, mod/qbank*	3

Berdasarkan Tabel 6, sebagian besar modul *Moodle* berada pada kategori *High Risk,* yang menunjukkan adanya hubungan antara intensitas histori perubahan dan potensi kerentanan modul. Modul dengan jumlah pengembang dan aktivitas *commit* yang tinggi cenderung memiliki risiko *defect* lebih besar dibandingkan modul dengan aktivitas yang lebih rendah.

Selain itu, tingginya nilai *code churn* dan frekuensi perbaikan *bug* pada kelompok *High Risk* memperkuat bahwa volume perubahan kode dan histori kegagalan merupakan indikator penting dalam prediksi kerentanan perangkat lunak. Namun demikian, beberapa modul yang telah matang menunjukkan tingkat stabilitas yang lebih baik meskipun memiliki histori perubahan yang cukup tinggi.

4. Keterbatasan dan Pengembangan Penelitian

Penelitian ini masih memiliki keterbatasan pada ukuran dataset dan penggunaan data yang hanya berasal dari repositori *Moodle.* Jumlah modul yang digunakan pada penelitian ini mengikuti struktur modul inti *Moodle* yang tersedia pada repositori resmi sehingga ukuran dataset bersifat *domain-specific* terhadap ekosistem *LMS Moodle.* Penelitian selanjutnya dapat menggunakan dataset yang lebih besar, metode penanganan *class imbalance* seperti *SMOTE,* serta melakukan perbandingan dengan *algoritma* lain seperti *Support Vector Machine (SVM), Logistic Regression,* atau *XGBoost* untuk mengevaluasi performa model secara lebih komprehensif.

4. KESIMPULAN

Penelitian ini berhasil membangun model prediksi modul rentan cacat pada *Moodle* menggunakan *algoritma Random Forest* berbasis fitur *change history.* Pendekatan *time-aware split* berhasil diterapkan untuk menghindari data *leakage* sehingga proses validasi model menjadi lebih realistis sesuai skenario evolusi perangkat lunak. Hasil pengujian menunjukkan bahwa optimasi *threshold* sebesar 0,3 mampu meningkatkan performa model dengan akurasi sebesar 85,7% dan *Recall* sebesar 1,00 sehingga efektif dalam meminimalkan *false negative* pada deteksi modul cacat. Selain itu, analisis *feature importance* menunjukkan bahwa jumlah pengembang unik dan aktivitas *commit* menjadi faktor paling berpengaruh terhadap kemunculan *defect* pada modul *Moodle.*

Hasil penelitian ini menunjukkan bahwa pendekatan berbasis *change history* memiliki potensi yang baik untuk mendukung *Software Defect Prediction* pada ekosistem *LMS Moodle* yang memiliki karakteristik evolusi dan kolaborasi pengembang yang kompleks. Model yang dihasilkan dapat dimanfaatkan sebagai *early warning system* untuk membantu optimalisasi prioritas pengujian dan pemeliharaan perangkat lunak pada modul berisiko tinggi. Meskipun demikian, penelitian ini masih memiliki keterbatasan pada ukuran dataset dan ruang lingkup data yang hanya berasal dari repositori *Moodle.* Oleh karena itu, penelitian selanjutnya dapat menggunakan dataset yang lebih besar, menerapkan teknik penanganan class imbalance seperti *SMOTE,* serta membandingkan performa *Random Forest* dengan *algoritma* lain seperti *XGBoost, Support Vector Machine (SVM),* atau *Logistic Regression* untuk memperoleh hasil prediksi yang lebih komprehensif.

DAFTAR PUSTAKA

[1] S. Stradowski and L. Madeyski, “Machine learning in software defect prediction: A business-driven systematic mapping study,” Mar. 01, 2023, *Elsevier B.V.* doi: 10.1016/j.infsof.2022.107128.

[2] A. Dwi Permatasari, N. Rahmatina, and M. Ainul Yaqin, “Evaluasi Teknik Elisitasi pada Software Requirement dalam Menentukan Efektivitas Kebutuhan Perangkat Lunak,” *Jurnal Pustaka Data (Pusat Akses Kajian Database, Analisa Teknologi, dan Arsitektur Komputer)*, vol. 5, no. 1, pp. 183–193, Jun. 2025, doi: 10.55382/jurnalpustakadata.v5i1.993.

[3] M. Jorayeva, A. Akbulut, C. Catal, and A. Mishra, “Machine Learning-Based Software Defect Prediction for Mobile Applications: A Systematic Literature Review,” Apr. 01, 2022, *MDPI.* doi: 10.3390/s22072551.

[4] W. Rhmann, B. Pandey, G. Ansari, and D. K. Pandey, “Software fault prediction based on change metrics using hybrid algorithms: An empirical study,” *Journal of King Saud University - Computer and Information Sciences*, vol. 32, no. 4, pp. 419–424, May 2020, doi: 10.1016/j.jksuci.2019.03.006.

[5] S. Haldar and L. F. Capretz, “Feature Importance in the Context of Traditional and Just-In-Time Software Defect Prediction Models,” in *2024 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2024, pp. 818–822. doi: 10.1109/CCECE59415.2024.10667167.

[6] A. Khalid, G. Badshah, N. Ayub, M. Shiraz, and M. Ghouse, “Software Defect Prediction Analysis Using Machine Learning Techniques,” *Sustainability (Switzerland)* , vol. 15, no. 6, Mar. 2023, doi: 10.3390/su15065517.

[7] M. Ali, T. Mazhar, A. Al-Rasheed, T. Shahzad, Y. Y. Ghadi, and M. A. Khan, “Enhancing software defect prediction: a framework with improved feature selection and ensemble machine learning,” *PeerJ Comput. Sci.*, vol. 10, 2024, doi: 10.7717/peerj-cs.1860.

[8] Y. Jiang, B. Shen, and X. Gu, “Just-In-Time Software Defect Prediction via Bi-modal Change Representation Learning,” Oct. 2024, doi: 10.1016/j.jss.2024.112253.

[9] G. Destefanis *et al.*, “An audit of machine learning experiments on software defect prediction,” *Empir. Softw. Eng.*, vol. 31, no. 4, Jul. 2026, doi: 10.1007/s10664-025-10797-w.

[10] S. R. Goyal, “Current Trends in Class Imbalance Learning for Software Defect Prediction,” *IEEE Access*, vol. 13, pp. 16896–16917, 2025, doi: 10.1109/ACCESS.2025.3532250.

[11] X. Wan, Z. Zheng, F. Qin, and X. Lu, “Data Complexity: A New Perspective for Analyzing the Difficulty of Defect Prediction Tasks,” *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 6, Jun. 2024, doi: 10.1145/3649596.

[12] M. Ali *et al.*, “Software Defect Prediction Using an Intelligent Ensemble-Based Model,” *IEEE Access*, vol. 12, pp. 20376–20395, 2024, doi: 10.1109/ACCESS.2024.3358201.