**ROMÂNIA**

**MINISTERUL APĂRĂRII NAŢIONALE**

**ACADEMIA TEHNICĂ MILITARĂ**

**„FERDINAND I”**

**Facultatea de Sisteme Informatice şi Securitate Cibernetică**

**Specializarea: Calculatoare și Securitate Cibernetică**



Metode de automatizare a notării sarcinilor de programare

Realizat de către:

Std.Plt. Pavel Ionuţ

Std.Plt. Adj. Bursuc Alex

Std. Plt. Maj. Cazamir Teodor

Grupa: C114C

Cuprins

[1. Introducere 3](#_Toc94043697)

[2. Metode de notare folosind Tehnici de Machine Learning 4](#_Toc94043698)

[3. Metode de notare folosind analizatoare statice de cod 5](#_Toc94043699)

[4. Bibliografie 6](#_Toc94043700)

# Introducere

Sarcinile de programare sunt un element esenţial în cadrul modulelor de programare a calculatoarelor predate la universitate. Odată cu creşterea numărului de elevi, evaluarea temelor devine o provocare, iar cercetătorii explorează metode noi de automatizare a evaluarilor. Automatizarea evaluării are avantaje precum viteza, consecvența, reducerea nevoii de post-marcare moderare, o mai bună utilizare a orelor de lucru ale oamenilor și eliminarea favoritismului și a prejudecăților din procesul de notare. Ca un exemplu ilustrativ, în cazul unei universităţi care are 10 module de programare în programele de licență, fiecare dintre acestea având câte 2 teme, cu o medie de 1000 de studenți, rezultă 20.000 de sarcini de programare pe

semestru. Este estimat un timp mediu de 20 de minute pentru corectarea fiecărei sarcini de programare ceea ce înseamnă 400 000 de ore de lucru petrecute pentru corectare în fiecare semestru academic. Abordările de marcare automată pot fi împărțite în două categorii categorii mari – testarea:

* Blackbox;
* Whitebox.

Testarea Blackbox, conform definiției sale, se concentrează pe programul care produce rezultatul așteptat pentru o intrare dată. O serie de teste Blackbox, cum ar fi testarea unitară, există și este utilizată într-o o serie de instrumente comerciale și necomerciale. Dezavantajul major al Blackbox de testare este faptul că programele ar trebui să fie dezvoltate cu o interfață sau API pentru a furniza intrări și a obține ieșiri (prin intermediul consolei, fișiere, argumente ale metodelor și returnări valori ale metodelor, etc.) și pentru a produce o ieșire, programul trebuie să fie corect din punct de vedere sintactic și să ruleze fără erori - ambele aspecte nu pot fi garantate prin simpla submisie a unui student.

Pe de altă parte, Whitebox Testing atribuie o notă prin citirea codului sursă prezentat, indiferent dacă testarea Blackbox este posibilă sau nu. De obicei, se obișnuiește ca practică în universități să se includă o rubrică de notare care are atât o componentă Blackbox, cât și una Whitebox, de exemplu, 50 puncte pentru programul care produce ieșiri corecte

pentru un set dat de intrări și 50 de puncte prin citirea codul sursă - căutând corectitudinea, curățenia și bunele practici de codare.

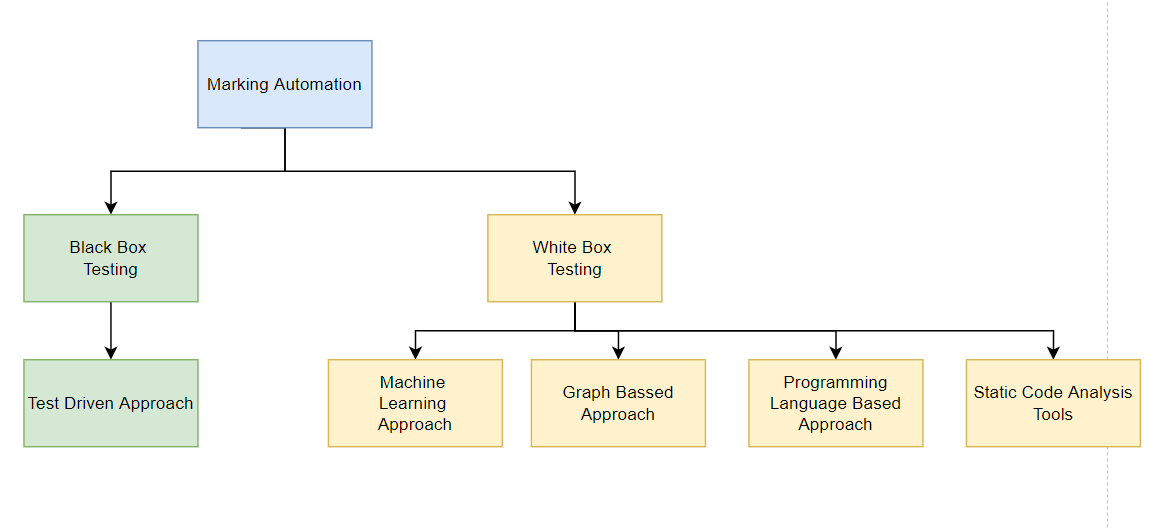


Fig. 1 – Tipuri de testare pentru notarea proiectelor

# Metode de notare folosind Tehnici de Machine Learning

ML este una dintre principalele abordări utilizate pentru a evalua marcarea codului sursă. În continuare vor fi prezentate diferite articole ştinţifice care fac referire la utilizarea tehnicilor de Machine Learning pentru notarea automată a codurilor sursă de programare:

* 1. În studiul de “[Srikant & Aggarwal, 2014](http://research.aspiringminds.com/wp-content/uploads/2019/11/mlProgramKDD2014.pdf)” sunt folosite tehnici de regresie precum Linear Ridge Regression şi Support Vector Machine (SVM) combinate cu diferite nuclee bazate pe rubrică şi predicţii evaluate manual. Studiile au utilizat modelarea cu o singură clasă ca modelul de predicție. Conform rezultatelor, Linear Ridge Regression a arătat o validare încrucișată și rezultatele validării erorilor mai bune decât regresia SVM. Aceasta a arătat că mai mult de 80% din notele prezise se încadrează în notele corespunzătoare evaluate de experți. În plus, metoda Regresiei față de cea a evaluatorilor umani poate oferi o notare mult mai bună decât notarea omniprezentă bazată pe teste de trecere (folosind fişiere de output).
  2. În studiul de “([Singh, Gulwani, & Solar-Lezama, 2013](https://people.csail.mit.edu/rishabh/papers/autograderPLDI13.pdf))” a fost folosită o soluţie de translatare în două faze pentru a găsi corecţiile minime. Eficacitatea instrumentului a fost de 64% de feedback pentru toate încercările incorecte în termen de 10 secunde în medie.
  3. În studiul de “([Takhar & Aggarwal, 2019](https://ojs.aaai.org/index.php/AAAI/article/view/4987))” a fost utilizat un sistem bazat pe reguli şi Regresie Liniară pentru a prezice poziţia erorilor de compilare şi evaluarea codurilor necompilabile. Acest lucru a fost realizat cu ajutorul abordarii de predicție a token-urilor bazată pe N-gram, care este numită Make Compliable (MC), Rule Relaxation (RR). Apoi a fost dezvoltat un model ML prin combinarea MC și RR ca RRMC. Măsurătorile de performanță au fost media pentru MC este de 0.73, iar pentru RR de 0.69. Abordarea RRMC a avut ca rezultat o medie de 0,71 corelații care oferă cele mai bune rezultate cu timp și efort reduse.

Temele de programare pot fi evaluate în mai multe moduri, printre acestea, metodele de Machine Learning depășeşte celelalte abordări. Motivul care stă la baza acestui lucru este capacitatea de "învățare" şi adaptare a modelului. Acesta poate analiza noile inputuri și să „învețe”, prin urmare structura de notare va fi actualizată și mai precisă.

Linear Ridge Regression, Random Forests și SVM bazat pe kernel, Naive Bayes (Naive Bayes multinomial și Gaussian Naive Bayes), pot fi identificați ca fiind cei mai populari și eficienți algoritmi utilizați în literatura de specialitate. Linear Ridge Regression arată în mare măsură rezultate mai bune la validarea încrucișată și la validarea erorilor decât Random Forests și SVM bazat pe kernel. În plus, regresia poate oferi o notare mult mai bună decât cea a testelor bazate pe fişiere de output, atunci când codul trimis pentru verificare nu poate fi compilat . În acest caz, Linear Ridge Regression au fost cele mai bune modele. Cu toate acestea, rezultatele arată, de asemenea, că modelele construite folosind atât caracteristici semantice, cât și cazuri de testare prezintă rezultate mai bune.

În plus, dezvoltarea tehnicilor de notare independente de problemă care pot fi facilitată de modelarea eficientă a unei singure clase eficiente, se dovedește a fi o necesitate. Pe de altă parte, algoritmii propuși trebuie să se antreneze pentru a fi utilizați pe mai multe probleme, ceea ce reprezintă o necesitate în viitor.

# Metode de notare folosind analizatoare statice de cod

Printre cele mai des întâlnite analizatoare de cod statice întâlnim Abstract Syntax Tree (AST) şi Abstract Syntax Graph (ASG). Aceste 2 abordări sunt sugerate în lucrarea “Striewe & Goedicke (2014)”. Numărul de instrumente SCA (Static Code Analysis) crește exponențial odată cu avansarea tehnologiei. Pentru a analiza codului sursă există două abordări care pot fi utilizate ca AST și ASG, iar ASG este o versiune îmbogățită a arborelui, dar care cuprinde arce suplimentare. În comparație cu AST, ASG conține câteva avantaje, deoarece poate detecta cu ușurință fragmente de cod irelevante, de exemplu, bucăți de cod neutilizate pot fi identificate cu ajutorul nodurile de declarare a metodelor, iar ASG este capabil să identifice cu ușurință

metodele recursive pe baza arcelor de dependență ciclică între declarația metodei și nodurile de apelare a metodei.

În consecință, majoritatea instrumentelor SCA tind să utilizeze ASG ca instrument de principala tehnică analitică. Unele studii sugerează generarea ASG din AST pentru a crește acuratețea ca o etapă valoroasă de preprocesare înainte de evaluarea codului sursă în sarcinile de programare.

# Bibliografie

* <https://www.scitepress.org/Papers/2021/104005/104005.pdf>
* <http://research.aspiringminds.com/wp-content/uploads/2019/11/mlProgramKDD2014.pdf>
* <https://people.csail.mit.edu/rishabh/papers/autograderPLDI13.pdf>
* <https://ojs.aaai.org/index.php/AAAI/article/view/4987>