*Mongolian Journal of Engineering and Applied Sciences*, 2023, 5(2)

*Компьютерын ухаан*

# Программ хангамжийн статик загвараас зохиомжийн согогийг илрүүлэх нь

Б.Батням[1], Ц.Лхамролом[1], Б.Наранчимэг[1,*]

[1]МУИС, ХШУИС, Мэдээлэл компьютерын ухааны тэнхим

Хүлээн авсан 2023.05.12; Хянагдсан 2023.06.23; Зөвшөөрөгдсөн 2023.06.25

*Холбоо баригч зохиогч: naranchimeg@seas.num.edu.mn

**Хураангуй**

Программ хангамжийн бүтээгдэхүүн хурдацтай нэмэгдэж буй өнөө үед программ хангамжийн алдаа, согогийг аль болох эрт урьдчилан таамагласнаар төслийн нийт зардлыг бууруулж, төсөл амжилттай хэрэгжихэд чухал нөлөөтэй. Одоо байгаа программ хангамжийн согогийг урьдчилан таамаглах аргууд нь Software Development Life Cycle (SDLC) буюу программ хангамж хөгжүүлэх амьдралын мөчлөгийн хэрэгжүүлэлтийн үе шат эсвэл шалгалтын үе шатанд байгаа эх код дээр тулгуурладаг. Энэ судалгааны ажилд SDLC-ийн зохиомжийн үе шатанд сэжигтэй классыг таамаглах зохиомжийн статик хэмжүүрт суурилсан машин сургалтын аргыг санал болгож байна. Тодруулбал, бид эхлээд жава эх код бүхий PROMISE өгөгдлийн багцаас урвуу инженерчлэлийн аргаар UML статик загвар гарган, түүнээс зохиомжийн статик хэмжүүр бүхий өгөгдлийн багц үүсгэж, сэжигтэй классыг машин сургалтын арга ашиглан таамаглах туршилтыг гүйцэтгэнэ. PROMISE өгөгдлийн багц дээрх туршилтын үр дүнгээс харахад бидний арга нь эх кодоос статик хэмжүүрт суурилсан хувилбараас дутахгүй үр дүн харуулж байгааг батлан харуулна.

**Түлхүүр үг:** программын согог илрүүлэх, статик хэмжүүр, машин сургалт, урвуу инженерчлэл

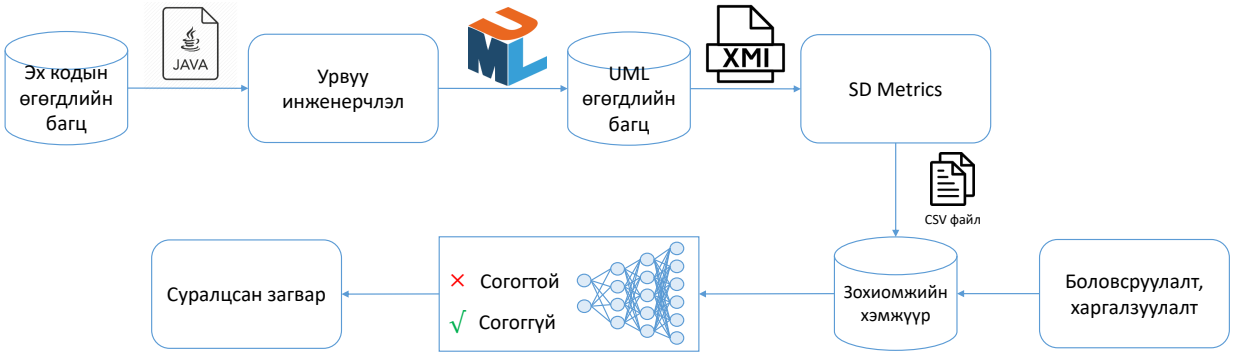
## 1 Удиртгал

Программ хангамжийн согог ялангуяа томоохон төслийн программ хангамж дахь алдааг илрүүлэх, засварлах нь маш төвөгтэй, өртөг өндөр, хүн хүч, хөдөлмөр шаарддаг. Software Development Life Cycle (SDLC) буюу программ хангамж хөгжүүлэх амьдралын мөчлөгийн эхний үе шатанд гарсан алдаа нь дараагийн үе шатуудад тархаж, алдааны тоог улам нэмэгдүүлдэг тул программ хангамжийн алдаа, согогийг аль болох эрт урьдчилан таамагласнаар төслийн нийт зардлыг бууруулж, төсөл амжилттай хэрэгжих боломжийг нэмэгдүүлэх ач холбогдолтой.

Сүүлийн хорин жилд хийгдсэн программ хангамжийн согогийг илрүүлэх зорилго бүхий судалгааны ажлууд нь программ хангамжийн эх кодод дүн шинжилгээ хийх замаар согогийг таамаглахад чиглэсэн байна. Эдгээр судалгаанууд нь эх кодын статик хэмжүүрүүд [1] эсвэл эх кодоос автоматаар суралцсан онцлог шинж чанарууд [2, 3] дээр төвлөрч, эдгээрийг машин сургалтын алгоритм ашиглан сургах замаар ангилагчийг бий болгох ажлууд байв. SDLC-ийн хэрэгжүүлэлтийн үе шатад алдаа, согогийг урьдчилан таамаглах олон судалгааны ажлууд хийгдэж, амжилтанд хүрсэн хэдий ч SDLC-ийн эхний үе шатууд; ялангуяа зохиомжийн шатанд согогийг илрүүлэх нь программ хангамжийн засвар үйлчилгээний үйл явцыг сайжруулах ирээдүйтэй арга бөгөөд өнөөг хүртэл энэ төрлийн судалгаа маш бага хийгдсэн байна. Энэхүү ажлаар бид SDLC-ийн зохиомжийн үе шатад согогийг урьдчилан таамаглах статик зохиомжийн хэмжүүрт суурилсан аргыг санал болгож байна. Зохиомжийн статик загвар, түүнд харгалзах согог бүхий өгөгдлийн багц хангалттай байдаггүй тул бид эхлээд тус санг үүсгэх зорилгоор эх кодын өгөгдлийн багцаас урвуу инженерчлэл [4] хийж, UML класс диаграмыг гарган авч, түүнээс зохиомжийн үзүүлэлтийг хэмждэг хэрэгсэл ашиглан статик зохиомжийн хэмжүүрийн багцыг бий болгоно. Дараа нь машин сургалтын арга ашиглан алдаатай классыг урьдчилан таамаглах ангилагч бүтээхийг зорив. Зураг 1-т бидний аргын ерөнхий бүдүүвчийг үзүүлэв. Энэ өгүүллийн хоёрдугаар бүлэгт программ хангамжийн алдаа, согог илрүүлэх ижил төстэй зарим ажлын талаар тайлбарласан. Гуравдугаар бүлэгт зохиомжийн статик хэмжүүр, машин сургалтын аргуудын талаар, дөрөвдүгээр бүлэгт хэрэгжүүлэлт болон үр дүнг бичсэн болно. Эцэст нь судалгааны ажлын дүгнэлтийг танилцуулна.

## 2 Судлагдсан байдал

Программын согогийг урьдчилан таамаглах олон тооны судалгаа сүүлийн жилүүдэд хийгдсэн байдаг. Энэ хэсэгт бид энэ салбарын чиг хандлага болон бидний аргын ялгааг онцлох болно.




*Зураг 1: Программ хангамж хөгжүүлэлтийн зохиомжийн шатанд согогийг илрүүлэх аргын ерөнхий бүдүүвч*

### 2.1 Согог урьдчилан таамаглах онцлог шинж чанар

Өмнөх ажлуудад ашиглагдаж буй онцлог шинж чанарууд нь эх кодын статик хэмжүүрүүд [1] ба эх кодоос автоматаар суралцсан онцлог шинж чанарууд [2, 3] гэсэн хоёр төрөлд хуваагдаж байна. Эх кодын статик хэмжүүрүүдэд уламжлалт хэмжүүр (жишээ нь, хэмжээ, нарийн төвөгтэй байдал), объект хандалтат хэмжүүр (жишээ нь, дэд классын тоо, объектын анги хоорондын холболт гэх мэт хэмжүүр), процесст суурилсан хэмжүүр гэх мэт программ хангамжийн алдааг ялгах шинж чанарууд дурьдаж болно. Тухайлбал, Redolf Ferenc нарын [5] судалгаанд 47618 класс бүхий Unified Bug Dataset [6] өгөгдлийн багцад согогийг илрүүлэхдээ 60 гаруй төрлийн статик хэмжүүрийг ашигласан байна.

Нөгөөтэйгүүр, зарим судалгааны ажлууд эх кодоос гүн мэдрэлийн сүлжээг (Deep neural network буюу DNN) ашиглан суралцсан хийсвэр онцлог шинж чанарууд дээр төвлөрч байсан. Программ хангамжийн инженерчлэлийн салбарт [7, 8] DNN-д суурилсан аргыг эх кодоос согог илрүүлэхэд ашиглахад 89%-95% нарийвчлалтай таньсан үр дүн үзүүлсэн байна. Хэдий тийм боловч DNN-ууд өөр өөр ангилалд хамаарах өгөгдлийн хийсвэр шинж чанарыг сурахын тулд сургалтын их хэмжээний өгөгдөл, үр дүнтэй сургалтын арга, өндөр тооцоолох хүчин чадал шаардагдана.

### 2.2 Согог урьдчилан таамаглах арга

Дээрх онцлог шинж чанарууд ашиглан согог таамаглах загвар бүтээхэд олон төрлийн машин сургалтын алгоритмууд туршигдаж буйг сүүлийн үеийн [9,10] тоймоос харж болно. Ялангуяа Support Vector Machine (SVM) [11, 12], Decision Tree (DT) [13], K Nearest Neighbors (KNN) [11], Linear Regression (LR) [11] аргуудыг ашиглаж байсан байна.

Харин хэрэгжилтийн үе шатаас өмнө UML загвар дээр үндэслэн согогийг урьдчилан таамаглах судалгаанууд цөөхөн хийгдсэн байдаг. Тухайлбал, [13–15] судалгаанууд нь UML хэмжүүрт үндэслэн программ хангамжийн алдаанд өртөмтгий байдлыг урьдчилан таамаглах ажил байна. Харин бидний ажил эх кодоос урвуу шинженерчлэл ашиглан UML хэмжүүр гарган авч, уг хэмжүүрт үндэслэн согогтой классыг урьдчилан таамаглана. Туршилтаар эх кодын статик хэмжүүрт суурилсан үр дүн ба UML хэмжүүрт суурилсан үр дүнг харьцуулан судална.

## 3 Арга зүй

Бидний санал болгож буй аргын бүдүүвчийг Зураг 1-т үзүүлэв. Уг загвар нь эх кодоос урвуу инженерчлэл ашиглан зохиомжийн загвар гаргах, уг зохиомжийн загвараас хэмжсэн үзүүлэлт (хэмжүүр)-ээс классын согогийг таамаглах загвар боловсруулах үндсэн хоёр хэсгээс бүрдэнэ.

Объект хандлагад зохиомжоос классын согогийг таамаглах загварыг үүсгэхийн тулд бидэнд сургалтын өгөгдөл шаардлагатай болсон. Уг сургалтын өгөгдлийг бэлдэхдээ Rudolf Ferenc нарын боловсруулсан Unified Bug Dataset [6]-ыг ашигласан. Уг өгөгдлийн багц нь нээлттэй эхийн нийт 82 программын эх кодоос хэмжсэн зохиомжийн үзүүлэлт болон алдааны мэдээллийг агуулсан нэгдсэн сан юм. Бид зохиомжийн загвараас зохиомжийн үзүүлэлтийг хэмжих шаардлагатай болсон тул дээрх нэгдсэн сангаас эхний удаад PROMISE сангийн нийт 46 программын эх кодоос урвуу инженерчлэлийн аргаар бүтцийн зохиомжийн загвар (класс диаграм, багцын диаграм) гаргаж авсан бөгөөд тус загвараа объект хандлагад загварын META модель хэлбэрээр хадгалсан. Уг моделиос SDMetrics (https://www.sdmetrics.com/) хэрэгслийн тусламж-

*Хүснэгт 1: Өгөгдлийн багцын дэлгэрэнгүй*

| | Түүврийн тоо | Согогтой түүврийн тоо | Хэмжүүрийн тоо |
|---|---|---|---|
| Эх код | 15521 | 5589 | 80 |
| UML статик загвар | 15495 | 5583 | 26 |

тай классын болон интерфейсийн үзүүлэлтийг тус тус хэмжиж гарсан үр дүнг PROMISE сангийн алдааны мэдээлэлтэй харгалзуулсан. Үүний үр дүнд уг сангийн бүтцийн зохиомжийн загвар болон бүтцийн зохиомжоос хэмжсэн үзүүлэлт бүхий алдааны мэдээллийн санг бүрдүүлсэн. Тус сангийн дэлгэрэнгүйг Хүснэгт 1-д үзүүлэв.

### 3.1 Объект хандлагат зохиомжийн хэмжүүр

Объект хандлагат зохиомжийг үнэлэхэд түгээмэл ашигладаг хэд хэдэн хэмжүүрүүд байдаг. Тухайлбал,

- MOOSE metrics - Объект хандлагат зохиомжийн эхний гурван алхмын үр дүнг хэмжих Weighted Methods per Class (WMC), Depth of Inheritance Tree (DIT), Number of Children (NOC), Coupling Between Object classes (CBO), Response For a Class (RFC) болон Lack of Cohesion in Methods (LCOM) хэмжүүрүүд [16].
- EMOOSE metrics – MOOSE хэмжүүрийг Message Pass Coupling (MPC), Data Abstraction Coupling (DAC), Number of Methods (NOM), Size1 болон Size2 гэсэн хэмжүүрүүдийг өргөтгөн тодорхойлсон [17].
- MOOD metrics - Бүтцийн загварын объект хандалтад үндсэн шинжүүдийг тодорхойлох битүүмжлэлийн Method Hiding Factor (MHF), Attribute Hiding Factor (AHF), удамшлийг илэрхийлэх Method Inheritance Factor (MIF), Attribute Inheritance Factor (AIF), полиморфизмийг илэрхийлэх Polymorphism Factor (POF), болон зурвас дамжуулалтыг илэрхийлэх Coupling Factor (COF) хэмжигдэхүүн [18].
- MOOD2 Metrics – MOOD хэмжүүрийг Operation Hiding Effectiveness Factor (OHEF), Attribute Hiding Effectiveness Factor (AHEF), Internal inheritance factor (IIF) болон Parametric polymorphism factor (PPF) гэсэн хэмжүүр [18].
- QMOOD metrics - Объект хандлагат зохимжийн дахин ашиглалт, уян хатан байдал, өргөтгөж болохуйц байдал, үр дүнтэй байдал болон ойлгомжтой байдлыг тодорхойлох зорилготой Design Size in classes (DSC), Number of Hierarchies (NOH), Average Number of Ancestors (ANA), Data Access Metric (DAM), Direct Class Coupling (DCC), Class interface Size (CIS), Measure of aggregation (MOA), Cohesion Among Methods of Class (CAM), Measure of Functional Abstraction (MFA), Number of Polymorphic methods (NOP), Number of methods (NOM) хэмжүүр [19].
- Lorenz and Kidd metrics - Зохиомжийн үндсэн гурван үзүүлэлт болох хэмжээ (Number of Public Methods (NPM), Number of Methods (NM), Number of Public Variables per class (NPV), Number of Variables per class (NV), Number of Class Variables (NCV), Number of Class Methods (NCM)), удамшил (Number of Methods Inherited (NMI), Number of Methods Overridden (NMO), Number of New Methods (NNA)) классын нягрталыг (Average parameters per Method (APM), Specialization Index (SIX)) тодорхойлох хэмжүүр [20].
- Coupling metrics - Объект хандлагат классын хоорондох холболтыг тодорхойлох CA (Class-Attribute), CM (Class-Method) болон MM (Method-Method) хэмжүүр [21].

Эдгээр хэмжүүрүүд нь ихэвчлэн тус тусдаа судалгааны ажлуудад тодорхойлогдсон учир зарим нь хоорондоо ижил төстэй үзүүлэлтийг өөр өөрөөр нэрлэсэн байдаг [22]. Тиймээс бид энэхүү судалгааны ажилдаа дээрх зохиомжийн хэмжүүрүүдээс давхардлыг арилган, классын диаграммаас хэмжих боломжтой хэмжүүрүүдийг сонгон ашигласан. Бидний ашигласан зохиомжийн хэмжүүрүүдийг Хүснэгт 2-д үзүүлэв.

## 4 Хэрэгжүүлэлт ба Үр дүн

Энэ хэсэгт эх кодын хэмжүүр болон зохиомжийн хэмжүүрийг машин сургалтын алгоритм ашиглан ангилсан үр дүнг харьцуулан тайлагнах болно.

Туршилтад бид LR, SVM, DT, RF болон KNN алгоритмын загварыг scikit-learn [23] санг ашиглан хэрэгжүүлсэн. Эдгээр алгоритмын параметрүүдийг сургалтын багц дээр (нийт өгөгдлийн 75%) 10-fold хөндлөн баталгаажуулалтаар тохируулсан. Бүх алгоритмуудын гүйцэтгэлийг үнэлэхдээ туршилтын багц дээр (нийт өгөгдлийн 25%) accuracy, precision, recall болон F1 оноо гэсэн хэмжүүрээр хэмжив.

Хүснэгт 3 болон 4-т PROMISE өгөгдлийн багцын эх кодоос гаргасан хэмжүүр болон зохиомжийн хэмжүүр дээрх машин сургалтын аргуудын үр дүнг харуулав. Харьцуулсан үр дүнгээс харахад зохиомжийн загвараас суралцсан хамгийн сайн арга RF нь 70.5% accuracy буюу нарийвчлалтай байгаа бол кодоос суралцсан хамгийн сайн арга болох SVM нь 72.4% нарийвчлалтай байна. Энэ нь зохиомжийн шатны арга нь хэрэгжүүлэлтийн шатны аргаас нарийвчлалаар харьцангуй бага буюу 1.9% байна. Үүнээс харахад зохиомжийн шатанд алдааг илрүүлж буй бидний арга нь хэрэгжүүлэлтийн шатанд алдааг илрүүлж буй аргаас дутахгүй сайн үзүүлэлт үзүүлж буйг илтгэж байна.

Эх кодоос гаргаж авсан 80 хэмжүүр, зохиомжийн загвараас гаргаж авсан 26 хэмжүүр ойролцоо үр дүн үзүүлж буй учир бид эдгээр онцлог шинж чанарууд нь эцсийн үр дүнд хэрхэн нөлөөлж буй нөлөөллийг тооцоолж үзсэн. Үр дүнг Зураг 2 болон Зураг

*Хүснэгт 2: Зохиомжийн хэмжүүр*

| Бүлэг | Хэмжүүр | Тайлбар |
|---|---|---|
| Хэмжээ | NumAttr | Классын шинжийн тоо |
| | NumOps | Классын үйлдлийн тоо |
| | NumPubOps | Классын нийтийн хандалттай үйлдлийн тоо |
| | Setters | 'set' түлхүүр үгээр эхэлсэн үйлдлийн тоо |
| | Getters | 'get' түлхүүр үгээр эхэлсэн үйлдлийн тоо |
| Удамшил | IFImpI | Классын хэрэгжүүлсэн интерфейсийн тоо |
| | NOC | Тухайн классын дэд классын тоо |
| | NumDesc | Тухайн классын нийт үр удмын тоо (дэд классууд, тэдгээрээс удамшсан нийт класс) |
| | NumAnc | Тухайн классын нийт өвгийн тоо (эцэг, эцэг классын эцэг, тэдгээрийн эцэг классууд) |
| | DIT | Тухайн классаас түүнийг агуулж буй удамшлын модны үндэс хүртэлх зай (тухайн классын түвшин) |
| | CLD | Тухайн классаас түүнийг агуулж буй удамшлын модны хамгийн хол навч хүртэлх зай |
| | OpsInh | Нийт удамшиж ирсэн үйлдлийн тоо |
| | AttInh | Нийт удамшиж ирсэн шинжийн тоо |
| | NumDirClients | Тухайн интерфейсийг шууд хэрэгжүүлсэн элементийн тоо |
| | NumIndClients | Тухайн интерфейсийг шууд болон шууд бусаар хэрэгжүүлсэн нийт элементийн тоо |
| Холболт | Dep_Out | Тухайн классыг ашиглаж буй нийт классын тоо |
| | Dep_In | Тухайн классд ашиглаж буй нийт классын тоо |
| | NumAssEl_ssc | Нэг үйлчлэх хүрээнд орших тухайн класстай холбогдсон классын тоо |
| | NumAssEl_sb | Нэг үйлчлэх хүрээ болон түүний дэд хүрээнд орших тухайн класстай холбогдсон классын тоо |
| | NumAssEl_nsb | Тухайн классаас ялгаатай үйлчлэх хүрээнд орших тухайн класстай холбогдсон классын тоо |
| | EC_Attr | Тухайн классыг шинжийн төрөл болгосон нийт тоо |
| | IC_Attr | Тухайн класс дахь бусад класс төрлийн шинжийн нийт тоо |
| | EC_Par | Тухайн класс төрлийн параметр ашигласан нийт тоо |
| | IC_Par | Тухайн классд бусад класс төрлийн параметр ашигласан нийт тоо |
| | Assoc | Тухайн интерфейстэй холбогдсон элементийн тоо |
| | Nesting | Дотоод классын хувьд түүнийг багтааж буй түвшин |

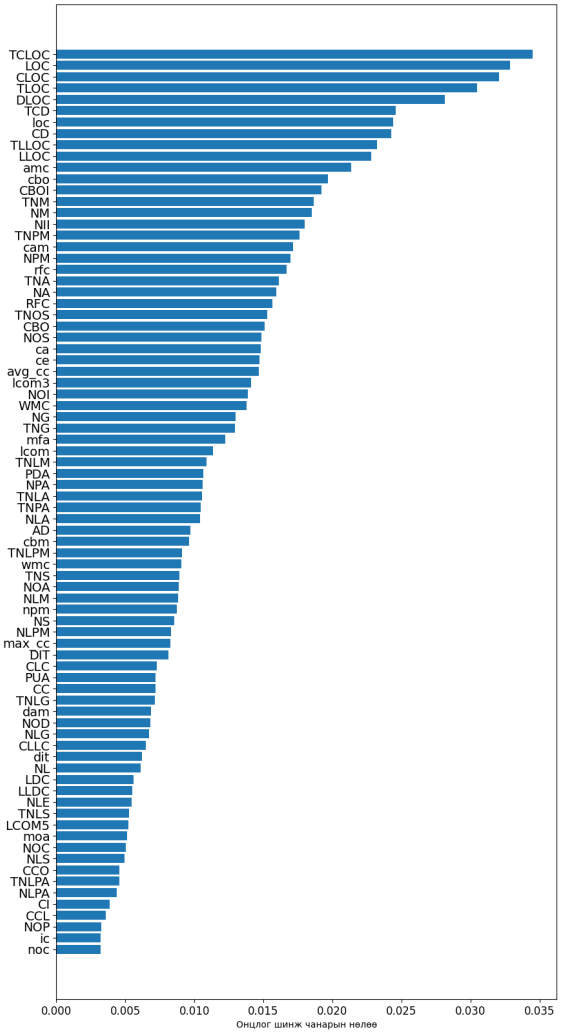
*Хүснэгт 3: Программын кодоос гаргасан хэмжүүр дээрх машин сургалтын аргуудын үр дүн*

| Арга | Хамгийн сайн параметер | Accuracy | Precision | Recall | F1 score |
|---|---|---|---|---|---|
| LR | C=10, solver=newton-cg, penalty=l2 | 0.695 | 0.681 | 0.612 | 0.609 |
| SVM | kernel=RBF, gamma=0.01, C=10 | **0.724** | **0.712** | 0.659 | 0.666 |
| DT | criterion=gini, max-depth=8 | 0.713 | 0.688 | **0.663** | **0.670** |
| RF | max-features = sqrt, n-estimators =1000 | 0.710 | 0.684 | 0.664 | **0.670** |
| KNN | n-neighbors = 11 | 0.701 | 0.675 | 0.643 | 0.648 |

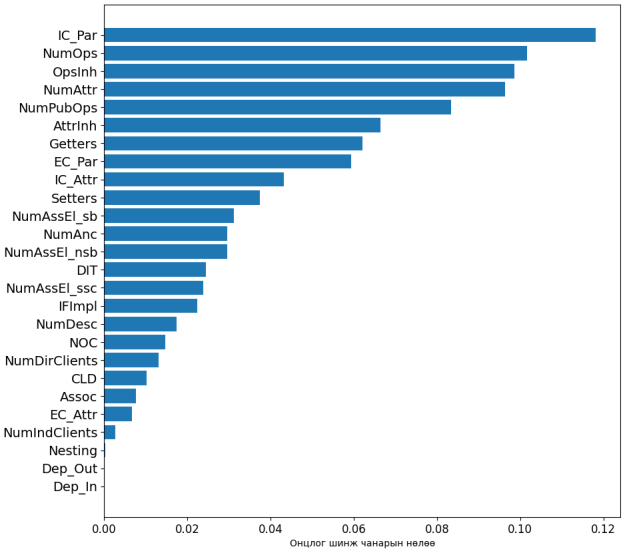
*Хүснэгт 4: Зохиомжийн статик загвараас гаргасан хэмжүүр дээрх машин сургалтын аргуудын үр дүн*

| Арга | Best parameters | Accuracy | Precision | Recall | F1 score |
|---|---|---|---|---|---|
| LR | C=0.01, solver=liblinear, penalty=l2 | 0.675 | 0.666 | 0.571 | 0.548 |
| SVM | kernel=RBF, gamma=0.1, C=50 | 0.700 | 0.676 | 0.632 | 0.66 |
| DT | criterion=gini, max-depth=12 | 0.696 | 0.666 | 0.641 | 0.646 |
| RF | max-features = log2, n-estimators =1000 | **0.705** | 0.676 | **0.662** | **0.667** |
| KNN | n-neighbors = 14 | 0.699 | **0.677** | 0.624 | 0.627 |

3 дээр үзүүлэв. Зургаас харахад эх кодын хэмжүүрээс TCLOC (Total Comment Lines of Code), LOC (Lines of Code) гэх мэт кодын мөрийн тоотой хол-




*Зураг 2: Эх кодын онцлог шинж чанаруудын нөлөө*




*Зураг 3: Зохиомжийн онцлог шинж чанаруудын нөлөө*

боотой онцлог шинж чанарууд өндөр нөлөөтэй байгаа бол зохиомжийн хэмжүүрээс IC_Par (Тухайн классд бусад класс төрлийн параметр ашигласан нийт тоо), NumOps (Классын үйлдлийн тоо), зэрэг онцлог шинж чанарууд өндөр нөлөөтэй байна.

## 5 Дүгнэлт

Энэхүү ажлаар программ хангамж хөгжүүлэлтийн зохиомжийн шатанд согогийг урьдчилан таамаглах статик зохиомжийн хэмжүүрт суурилсан аргыг санал болгосон. Ингэхдээ эх кодын өгөгдлийн багцаас статик зохиомжийн хэмжүүрийн өгөгдлийн багц бий болгох үр дүнтэй аргыг ашигласан гэдгээрээ онцлогтой. Туршилтын үр дүнгээс харахад зохиомжийн шатанд алдааг илрүүлж буй бидний арга нь хэрэгжүүлэлтийн шатанд алдааг илрүүлж буй аргаас дутахгүй сайн үзүүлэлт үзүүлсэн. Цаашид

энэ аргыг улам сайжруулахын тулд өгөгдлийн багцыг тэнцвэртэй болгох, сургалтын өгөгдлийн багцыг нэмэгдүүлэх, гүн сургалтын аргуудыг турших боломжтой гэж үзэж байна.

## Зохиогчийн оролцоо

Б.Батням нь энэхүү өгүүллийн ерөнхий санааг гаргаж, арга зүйн зөвлөгөө өгч, Ц.Лхамролом нь өгөгдөл бэлтгэж, Б.Наранчимэг нь туршилтыг гүйцэтгэж, үр дүнг нэгтгэв. Үр дүнг дүгнэх, өгүүллийн бичилтийг бүх зохиогчид хамтран гүйцэтгэв.

## Санхүүжилт

Энэхүү судалгааны ажлыг Монгол улсын их сургуулийн P2020-3971 дугаартай “Мэдрэлийн гүн сүлжээг ашиглан олон төрлийн өгөгдлийг нэгтгэх нь” төслөөр санхүүжүүлсэн болно.

## Ашиг сонирхлын зөрчилгүйн баталгаа

Ашиг сонирхлын зөрчилгүй болно.

## Ашигласан ном

[1] Radjenović D, Heričko M, Torkar R, Živkovič A. Software fault prediction metrics: A systematic literature review. Information and software technology. 2013;55(8):1397-418.

[2] Wang S, Liu T, Tan L. Automatically learning semantic features for defect prediction. In: Proceedings of the 38th International Conference on Software Engineering; 2016. p. 297-308.

[3] Dam HK, Tran T, Pham T, Ng SW, Grundy J, Ghose A. Automatic feature learning for vulnerability prediction. arXiv preprint arXiv:170802368. 2017.

[4] Canfora G, Di Penta M, Cerulo L. Achievements and challenges in software reverse engineering. Communications of the ACM. 2011;54(4):142-51.

[5] Ferenc R, Bán D, Grósz T, Gyimóthy T. Deep learning in static, metric-based bug prediction. Array. 2020;6:100021.

[6] Ferenc R, Tóth Z, Ladányi G, Siket I, Gyimóthy T. A public unified bug dataset for java. In: Proceedings of the 14th international conference on predictive models and data analytics in software engineering; 2018. p. 12-21.

[7] Pradel M, Sen K. Deepbugs: A learning approach to name-based bug detection. Proceedings of the ACM on Programming Languages. 2018;2(OOPSLA):1-25.

[8] Shi K, Lu Y, Chang J, Wei Z. PathPair2Vec: An AST path pair-based code representation method for defect prediction. Journal of Computer Languages. 2020;59:100979.

[9] Karim S, Warnars HLHS, Gaol FL, Abdurachman E, Soewito B, et al. Software metrics for fault prediction using machine learning approaches: A literature review with PROMISE repository dataset. In: 2017 IEEE international conference on cybernetics and computational intelligence (CyberneticsCom). IEEE; 2017. p. 19-23.

[10] Thota MK, Shajin FH, Rajesh P, et al. Survey on software defect prediction techniques. International Journal of Applied Science and Engineering. 2020;17(4):331-44.

[11] Wahono RS, Herman NS, Ahmad S. A comparison framework of classification models for software defect prediction. Advanced Science Letters. 2014;20(10-11):1945-50.

[12] Shuai B, Li H, Li M, Zhang Q, Tang C. Software defect prediction using dynamic support vector machine. In: 2013 Ninth International Conference on Computational Intelligence and Security. IEEE; 2013. p. 260-3.

[13] Maddeh M, Ayouni S, Alyahya S, Hajjej F. Decision tree-based design defects detection. IEEE Access. 2021;9:71606-14.

[14] Cruz AEC, Ochimizu K. A UML approximation of three Chidamber-Kemerer metrics and their ability to predict faulty code across software projects. IEICE TRANSACTIONS on Information and Systems. 2010;93(11):3038-50.

[15] Han AR, Jeon SU, Bae DH, Hong JE. Measuring behavioral dependency for improving change-proneness prediction in UML-based design models. Journal of Systems and Software. 2010;83(2):222-34.

[16] Chidamber SR, Kemerer CF. A metrics suite for object oriented design. IEEE Transactions on Software Engineering. 1994;20(6):476-93.

[17] El Emam K, Benlarbi S, Goel N, Rai SN. The confounding effect of class size on the validity of object-oriented metrics. IEEE Transactions on Software Engineering. 2001;27(7):630-50.

[18] Abreu FB, Carapuça R. Object-oriented software engineering: Measuring and controlling the development process. In: Proceedings of the

4th international conference on software quality. vol. 186; 1994. .

[19] Bansiya J, Davis CG. A hierarchical model for object-oriented design quality assessment. IEEE Transactions on Software Engineering. 2002;28(1):4-17.

[20] Lorenz M, Kidd J. Object-oriented software metrics: a practical guide. Prentice-Hall, Inc.; 1994.

[21] Lionel Briand WM Prem Devanbu. An Investigation into coupling measures for object-oriented designs; 1997. .

[22] Oyun-Erdene N, Batnyam B, Erdenetuya N, Lkhamrolom T. Predicting class fault proneness using data mining classification techniques. Journal of Advanced Research in Dynamical and Control Systems. 2018;10(2):58-590.

[23] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research. 2011;12:2825-30.