Journal of Information Systems Engineering and Management
2025, 10(62s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

A Neural Approach to Predict the Change Impact in Object-Oriented Systems

BENHABARA Benamar[1] , ABDI Mustapha Kamel[2], DAHANE Miloud[3]
RIIR Laboratory, University Oran1, Oran, Algeria
RIIR Laboratory, University Oran1, Oran, Algeria
RIIR Laboratory, University Oran1, Oran, Algeria
benhabara.benamar@edu.univ-oran1.dz, abdi.mustapha@univ-oran1.dz, dahane.miloud@univ-oran1.dz

ARTICLE INFO

Received: 29 Dec 2024

Revised: 12 Feb 2025

Accepted: 27 Feb 2025

ABSTRACT

Software maintenance is a critical phase in the software lifecycle, often outlasting the development and deployment stages and accounting for the majority of total costs in the industry. These high costs are primarily due to the complexity of implementing changes and managing their potential ripple effects throughout the system, especially in large and evolving codebases with numerous interdependencies and legacy components that are difficult to modify safely. In response to this challenge, numerous studies have aimed to analyze or predict the impact of changes in software systems. However, many existing approaches require extensive input data that is often difficult to collect or maintain, limiting their practical applicability in real-world settings and automated development pipelines. In this paper, we propose a machine learning-based approach using neural classifiers trained on historical change data. Our change impact prediction model leverages two key sources of information: (1) the relationships between classes, characterized by their susceptibility to change, and (2) the structural metrics derived from the change history across successive versions of the software. To evaluate our approach, we applied it to 23 open-source software projects from the PROMISE repository. The results demonstrate that our model outperforms several existing techniques, offering a more efficient and effective solution for change impact prediction in software maintenance, withpromising implications for large-scale systems.

Keywords: Maintenance, analysis / prediction impact of changes, CK metrics, machine learning, neural classifiers.

INTRODUCTION

We are interested in studying the factors that can influence the amplitude of changes in the code during the evolution of object-oriented software. Indeed, software maintenance represents a major expense for the industry[1]. However, despite significant advances in software engineering in recent decades, it is still difficult to estimate maintenance costs. This situation is all the more worrying given that there are far more systems under maintenance than new systems being developed.

Software maintenance refers to the modification of a system to fix a bug, improve performance, or adapt the system to a new environment[2]. It is recognized as the longest and most expensive phase of the software life cycle. It only ends with the software's retirement. According to several studies, including one published in [3], At least 50% of software implementation costs are attributable to maintenance and this figure continues to increase, despite the advantages of the object paradigm.

Changing the software is a necessity. Set of laws that characterize the dynamic evolution of systems. Continuous change is one of the proposed laws. Change introduced to the system is the trigger for any form of maintenance[4]. It is obvious that implementing a change will have different effects on the entire system. To this end, evaluating and understanding the implications of the change helps to better manage maintenance activities, and therefore control its costs.

Journal of Information Systems Engineering and Management
2025, 10(62s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Change impact analysis is the identification of the consequences of a change already introduced or the estimation of the modifications required to accomplish a change[5]. It consists of a set of techniques that determine which subparts of the system are affected by the change. The Year 2000 (Y2K) bug problem is an example that shows the importance of change impact analysis[6]. Contrary to popular belief, the difficulty with this problem was not in identifying date-type fields, but rather in analyzing other parts of the system to determine the impact of changes to date-type fields (derived variables function parameters, calculation methods, etc.).

A concrete way to estimate maintenance costs is to determine a priori, at each stage of the software's evolution (i.e., at each version), the amount of code to be changed to meet a set of needs [7]. Predicting code changes, however, is not an easy task. Indeed, the current state of knowledge does not allow us to identify the factors that can positively or negatively influence the magnitude of changes.

An empirical approach is a good alternative, as it allows us to study these influences based on historical data. Until recently, it was used on a small scale on data from single industrial systems[8]. However, the recent availability of open source software allows this approach to be applied on a larger scale.

This article presents a study of the factors influencing software changeability. According to the ISO9126 standard [9], Changeability is defined as the ease with which software can be modified to meet a given need, including adding functionality, correcting errors, and adapting to a new environment. This ease is closely related to the amount of code that needs to be changed. In other words, the less code that needs to be changed to implement a need, the easier the software is considered to be to change.

There are two families of factors that influence changeability: the need for change and the structure. The greater the need, the more likely a major change in the code is expected [10]. At the same time, a well-structured system should be able to absorb new needs while minimizing changes.

RELATED WORKS

System structure plays an important role in the propagation of change effects. The first study identifying factors influencing the changeability of OO systems comes from Li et al [11]. She showed that a system's structure affects its maintenance costs, measured as changes in lines of code. However, this work did not explain the reasons for this relationship. We proposed the idea that structure allows us to identify the role of certain classes and that this information allows us to predict the magnitude of changes in a system.

In [12], the authors presented object-oriented change metrics. Studying the incremental development of a small system (20 classes), the authors showed that change metrics (design and implementation) and structure metrics are independent. By analyzing several medium-sized systems, we found the opposite: there is an influence relationship when developers used object-oriented mechanisms.

[13] They propose an approach based on analyzing the structure of a system from the perspective of change propagation. To do this, they focus on the influence of each type of link between classes on change propagation. They thus propose, for each type of change, the set of links that are most likely to propagate it. The links considered are those of association, aggregation, inheritance, and invocation. Regarding changes, the authors define 13 types that cover different levels of granularity.

In [14] a method that uses user-provided change queries to determine a list of system entities that could be affected by these changes. This is done by comparing the new query with the old ones. The more similar the queries are, the greater the probability that they will impact the same entities.

Present a probabilistic approach that uses Bayesian networks to assess the potential impact of changing a class based on its integration with other classes. Two types of coupling between classes were chosen: design coupling and implementation coupling. Each type is represented by a set of metrics[15].

In[2] , the authors presented a study in which they evaluated the costs of adapting a library to a new domain. They showed that inheritance is an important factor in encouraging reuse and minimizing changes. This is consistent with our results. In the same vein, combining dependencies between entities and change history,[16] propose a

Bayesian network-type model that predicts co-changes. Both types of information are used to construct Bayesian networks.

Approaches to graph-based software representation consist of extracting a graph based on software artifacts to illustrate the propagation of the impact of change[17], the authors construct graphs that capture software structures, then leverage recent advances in graph topology analysis to better understand software evolution.

In [18], the software is represented by a model based on a graph rewriting system where software components are linked by meaningful relationships.

In this context, [19] propose a generic model of software dependency graphs that synthesizes graphs where the degree distribution is close to that empirically observed in real software systems. This model provides new insights into the potential fundamental rules of software evolution.

In [19], A system dependency graph has been extracted. The underlying idea is to measure the accessibility of different nodes in the graph. This involves computing the transitive closures of the relationships represented by the edges of the graph. Consider that this closure determines the strength of dependency between nodes as linked or strongly linked components. Therefore, this closure can inform about the propagation of the impact of change. The latter is often calculated using a widely applied algorithm called Warshall's algorithm [20].

In [21], Change impact indicators are derived from previous change impact graphs extracted from version history, as well as their associations with various factors determining change propagation.

Metric-based approaches generally deal with object-oriented concepts and primarily concern metrics known as an accurate estimate of maintenance effort. In [22], The values of the analysis metrics show a strong correlation between the coupling metrics and the maintenance effort measure. In [23], The authors established that a lower number of package revisions (REVISIONS) and a smaller number of revised lines of code (RLOC) over the package maintenance history indicate lower package maintenance effort. In [24], The authors propose a global modeling approach to question the structural and qualitative interdependencies of a software during the integration of several modifications.

Model-based approaches primarily aim to predict the impact of change at the conceptual level[25]. In , the various links between the classes of a system are taken into account to calculate or estimate the impact of the change. The impact is therefore expressed by a combination of these links. In the same way [26] propose an approach to change impact analysis in which explicit impact rules capture the consequences of modifying UML class diagrams on other artifacts. The analyzed UML class diagrams typically describe two versions of the system under development. Differences are automatically identified using model differencing. Explicitly formulating the consequences of changes in impact rules allows for the creation of a checklist with precise indications regarding the development steps required to manage the change.

Empirical approaches aim to use the advantages of applying methods and techniques from other fields to software engineering and illustrate how, despite the difficulties, software evolution can be studied empirically. In [27], the authors study the impact of afferent coupling (AC), efferent coupling (EC), and object coupling (OC) on fault prediction by bivariate correlation. The result of this study is a prediction model using these metrics to predict errors.

In [28], the authors divided the source code elements into a group containing the refactored elements and a group of non-refactored elements. The authors analyzed the characteristics of the elements in these groups using correlation analysis, the Mann–Whitney U test, and effect size measures.

In [29] Code metrics have been widely used as features to build deep learning (DL)-based automatic vulnerability prediction (AVP) models to find vulnerabilities in code using code metrics. The obtained results show that code metrics are very good but not the best to use as features in DL-based AVP.

[10] Is interested in the use of change impact estimation as a primary indicator for maintenance effort estimation by exploiting software measurements or metrics, particularly in object-based systems. They apply a technique called design of experiments, generally used in experimental sciences, with the aim of quantifying the maintenance

Journal of Information Systems Engineering and Management
2025, 10(62s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

effort which would be a measure of the impact of change, in the form of a mathematical equation whose parameters are the values of these metrics.

In [30], a decision support approach to be adopted in case of proposed changes in object-oriented systems is presented. It is based on the use of the ELECTREIII multi-criteria decision support method using coupling metrics as an indicator of the impact of changes.

Our work falls into the category that focuses on using change impact estimation as the primary indicator for estimating maintenance effort using CK measurements or metrics and change history. We are particularly interested in object-oriented systems. We apply a machine learning technique using neural networks to analyze the impact of change, in the form of a Boolean decision class of impacted or not impacted.

PROPOSED APPROACH

Prediction models are used to establish a relationship between two sets of attributes measured at different points in time. On a timeline, two points in time characterize a prediction model: the point in time at which the input attributes are measured and the point in time at which the output attributes can actually be measured [31]. These models can alert a development team early in the development process to the size of a change, allowing them to make corrections if necessary.

PREDICTION MODEL

In this work, we propose a machine learning approach to analyze the impact of changes, based solely on information extracted from the source code. Our approach produces a predictive model (neural classifier) that takes as input the relationships between the classes of the system and produces as output the decision that a class is affected knowing that another has been modified. The use of a machine learning approach, in addition to explicitly managing the uncertainty inherent in the problem, allows analyzing the impact of the change before its introduction into the system. This reduces the cost of the maintenance phase. Indeed, the maintainer only has to analyze the impacted elements. On the other hand, it gives him the possibility to evaluate different change scenarios without having to modify the source code [32].

To be able to carry out this study, we used the famous PROMISE (Predictor Models In Software Engineering) software database.[33], which is an international repository in software engineering. This database offers a collection of publicly available data to serve researchers in the construction of predictive software models and the software engineering community at large.

Our work falls within the framework of software maintenance. In this section, we present the problem related to change impact analysis.

We discuss, in particular, why existing approaches do not provide comprehensive solutions to this problem. Subsequently, we present our contributions in relation to the targeted problem.

Various aspects of software structure, such as coupling and cohesion, can be measured directly from the code, a reliable source of information. This is not the case, however, for changes in requirements. Indeed, the documents describing the requirements are often missing or incomplete. We therefore use the next available source of information: the design. Changes in the design can be measured by comparing the current design to a proposed new design. Figure 1 shows the generation phases of the change impact prediction model.

Journal of Information Systems Engineering and Management
2025, 10(62s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

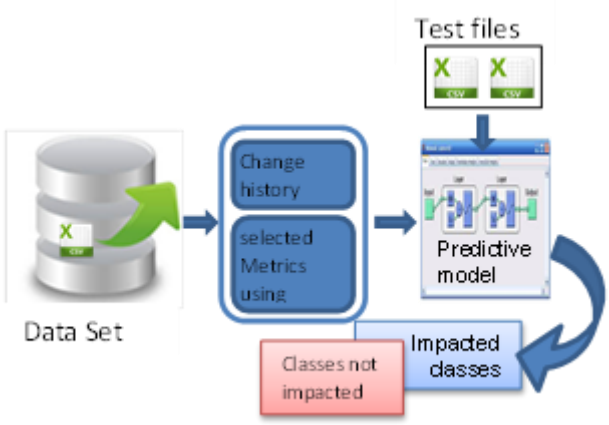

Fig.1 – The generation phases of the change impact prediction model.

EXPERIMENTATION

This section presents the impact analysis of the change using all the data we have collected. It consists of verifying whether a model obtained from this data offers good predictive capacity. The training data is used as input to a learning module to produce neural classifiers. This module is implemented using NetBeans [34], a very powerful Open Source development tool, particularly well-built and adapted to create applications or complex products.

DATA LEARNING

The data learning, extracted from each pair of successive versions of the same considered system Vi, Vi+1, are used for training the classifier. These data contain the attribute to be explained, i.e., impact or not, and the explanatory attributes [35]. Thus, for a pair of successive versions, we retrieve all the classes that have undergone a change. The classes affected by this change are obtained from the change history file.

DATA COLLECTION

We used the evolutions of 23 open-source systems written in Java. For each of these 23 systems, a series of more than twenty metrics is calculated using Metrics. The main ones are the structural metrics of CK (Chidamber and Kemerer). These are the metrics that will be used to build the training and test databases. We seek to evaluate our approach with systems of different types, which justify the choice of these systems. Furthermore, these systems vary in size. Table I presents the number of versions for each system, as well as the size of each system, illustrated by the number of classes in the latest version. Grouping the systems by their versions together results in a large dataset of 87,000 to 100,000 classes, a large-scale system required for neural networks.

Table I. Systems used

system	version used	Number of Versions	Numbers of Classes
Ant	1.3--1.7	5	745
Arc	1	1	234
Camel	1.0--1.6	4	965
E-learning	1	1	64
Forrest	0.6--0.8	3	32
Intercafe	1	1	27
Ivy	1.1--2.0	3	352
Jedit	3.2--4.3	5	492
Kalkulator	1	1	27
Log4j	1.0--1.2	3	205

Journal of Information Systems Engineering and Management
2025, 10(62s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Lucene	2.0--2.4	3	340
Nieruchomsci	1	1	27
Pbeans	1.0--2.0	2	51
Pdftranslator	1	1	33
Poi	1.5--3.0	4	442
Prop	1.0--6.0	6	660
Redaktor	1	1	176
synapse-1.0	1.0--1.2	3	256
tomcat	1	1	858
Velocity	1.4--1.6	3	229
Xalan	2.4--2.7	4	909
Xerces	1.2--1.4	3	588
Zuzel	1	1	29
Total number of classes		87047 classes	

CLASSIFIER CREATION

The training data is used as input to a learning module to produce neural classifiers. This module is implemented using advanced supervised learning techniques. Neural classifiers predict the propagation of change between two classes. The Ck metrics serve as input data for the classifier. The module classifies the classes to be analyzed into two groups: impacted class and non-impacted class. The architecture of the neural classifier is shown in Figure 2.

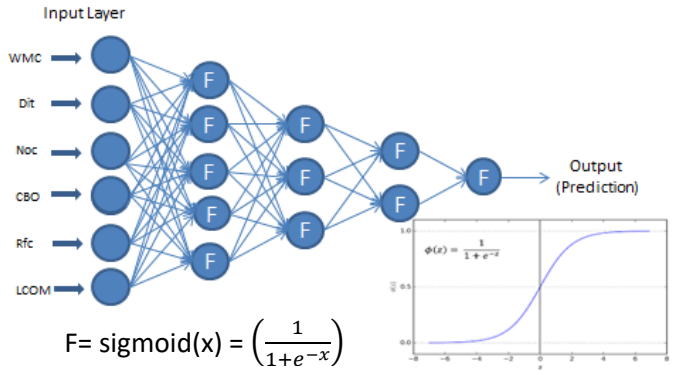

Fig.2. Neural classifier

MEASURING THE PERFORMANCE OF CLASSIFIERS

The models were built with Artificial Neural Networks (ANNs) consist of input and output layers, and (in most cases) one or more hidden layers composed of units that transform the input into something the output layer can use. The goal of the ANN is to determine a set of weights θ (between the input, hidden, and output nodes) that minimize the total sum of errors. The calculation is done using the sum of the inputs weighted by the weights θ (a function of the strength of the connections). During training, the weights θi are adjusted according to a learning parameter $\lambda_l \epsilon$ [0, 1] until the outputs become consistent with the output. Each neuron has an activation function that will calculate the value of the neuron's signal. The neuron then compares the weighted sum of the inputs to a threshold value and then provides an output response [36].

Our approach is a so-called supervised learning technique that will be possible to predict the class or category of a new object submitted to it for each class in the system (impacted class / non-impacted class). To present our results, we made the assumption that the resources to inspect the impacted classes are not limited. From the

Journal of Information Systems Engineering and Management
2025, 10(62s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

training set, or base of examples, the algorithm estimates the parameters of the prediction model that are the most efficient possible, that is to say, those that produce the least errors in prediction. From this model that was built with the optimal parameters, it will be possible to predict the class or category of the classes of the system (impacted class / non-impacted class) following a change made to another class.

To evaluate the performance of our method, we calculate precision and recall. Precision allows us to ensure the accuracy of predictions. On the other hand, recall allows us to determine the proportion of classes actually affected by the change that are detected with a probability at least equal to the threshold.

To calculate precision and recall, we use the concept of true positives (TP), which is the number of correct predictions, and the concept of false positives (FP), which is the number of classes, predicted as impacted when they are not in fact. Classes that are actually impacted when they are not predicted as such are false negatives (FN). The formulas for precision and recall are as follows:

$$\text{precision} = \frac{\text{TP}}{\text{TP+FP}} \quad , \quad \text{recall} = \frac{\text{TP}}{\text{TP+FN}}$$

We used other measures to evaluate the performance of our approach. The first is the F-measure [37] which is an aggregation of precision and recall.

Formally: $$\text{F} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision+recall}}$$

RESULTS AND INTERPRETATION AND VALIDATION

In this section, we present the implementation of our approach. Tables III and IV present the results for each system. We will interpret the results based on the models performance metrics and compare these results with other related work. Figure 3 and 4 illustrates the perfect curves for precision and recall and accuracy for the 23 systems studied. The general classifiers are tested on the changes from all systems. CK metrics provide enough evidence to enable good prediction of code changes. This is clearly seen in Table III, with an average accuracy of p = 88% and an F-measure of F = 93%. The neural classifiers demonstrated very good ability to predict change propagation for small systems. This is observed in the E-learning, Intercafe, and Kalkulator systems, where the accuracy of p = 1 and an F-measure of F = 97%. This provides good prediction for large systems, such as the Prop system, where the accuracy of p = 88% and an F-measure of F = 99% .

Table III. Classifier results by system (precision, recall, F-measure)

System	Precision	Recall	F-measure	Accuracy
Ant	0.85	0.96	0.90	0.84
Arc	0.92	1	0.96	0.93
Camel	0.83	0.94	0.88	0.8
E-learning	1	0.98	0.99	0.98
Forrest	0.94	1	0.97	0.95
Intercafe	1	0.95	0.97	0.96
Ivy	0.86	0.98	0.91	0.85
Jedit	0.87	0.96	0.91	0.85
Kalkulator	1	0.95	0.97	0.96
Log4j	0.56	0.92	0.7	0.67
Lucene	0.67	0.69	0.68	0.71
Nieruchomsci	0.94	1	0.97	0.96
Pbeans	0.78	0.95	0.86	0.81
Pdftranslator	0.94	0.94	0.94	0.94
Pio	0.68	0.66	0.67	0.68
Prop	0.88	0.99	0.93	0.88
Redaktor	0.93	0.99	0.96	0.94
Synapse	0.78	0.98	0.87	0.78

Journal of Information Systems Engineering and Management
2025, 10(62s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Tomcat	0.93	0.99	0.96	0.92
Velocity	0.63	0.64	0.55	0.64
Xalan	0.61	0.48	0.54	0.62
Xerces	0.66	0.91	0.76	0.66
Zuzel	0.94	1	0.96	0.96
All système	0.83	0.99	0.90	0.83

Table IV. Validation of results (T.C Approach, Proposed Approach)

System	Number of classes	Number of classes impacted	T.C. approach		proposed approach	
			Number of predicted classes	%	Number of predicted classes	%
Ant	1692	350	100	29%	261	75%
Arc	234	27	20	74%	28	104%
Camel	2784	562	80	14%	555	99%
E-learning	64	5	20	400%	5	100%
Forrest	67	8	60	750%	5	63%
Intercafe	27	4	20	500%	4	100%
Ivy	704	119	60	50%	101	85%
Jedit	1749	303	100	33%	247	82%
Kalkulator	27	6	20	333%	6	100%
Log4j	449	260	60	23%	148	57%
Lucene	782	438	40	9%	223	51%
Nieruchomsci	27	10	20	200%	9	90%
Pbeans	77	30	40	133%	20	67%
Pdftranslator	33	15	20	133%	14	93%
Pio	1378	707	80	11%	723	102%
Prop	69653	8854	120	1%	8373	95%
Redaktor	176	27	20	74%	26	96%
Synapse	635	162	60	37%	136	84%
Tomcat	858	77	20	26%	81	105%
Velocity	639	367	60	16%	164	45%
Xalan	3320	1806	80	4%	1240	69%
Xerces	1643	654	80	12%	549	84%
Zuzel	29	13	20	154%	12	92%
All systems	**87047**	**14804**	**1200**	**8%**	**12930**	**87%**

Journal of Information Systems Engineering and Management
2025, 10(62s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

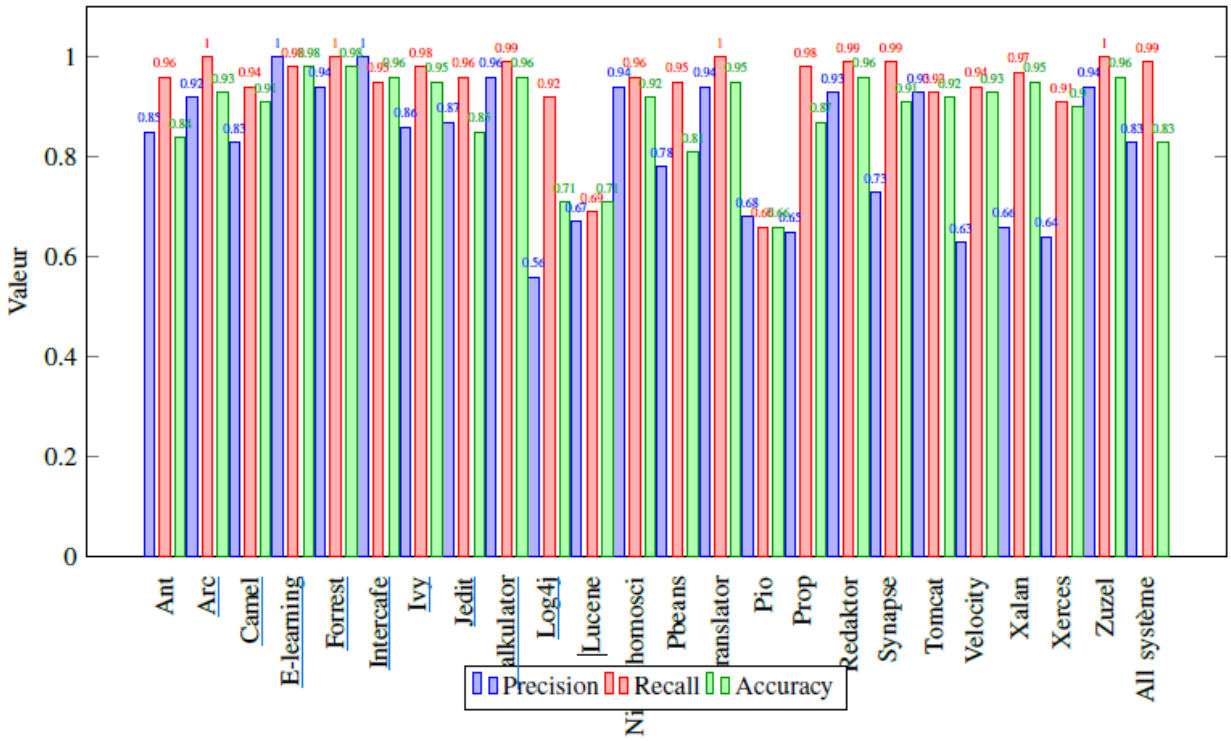

Figure 3. Precision, recall, and accuracy by system.

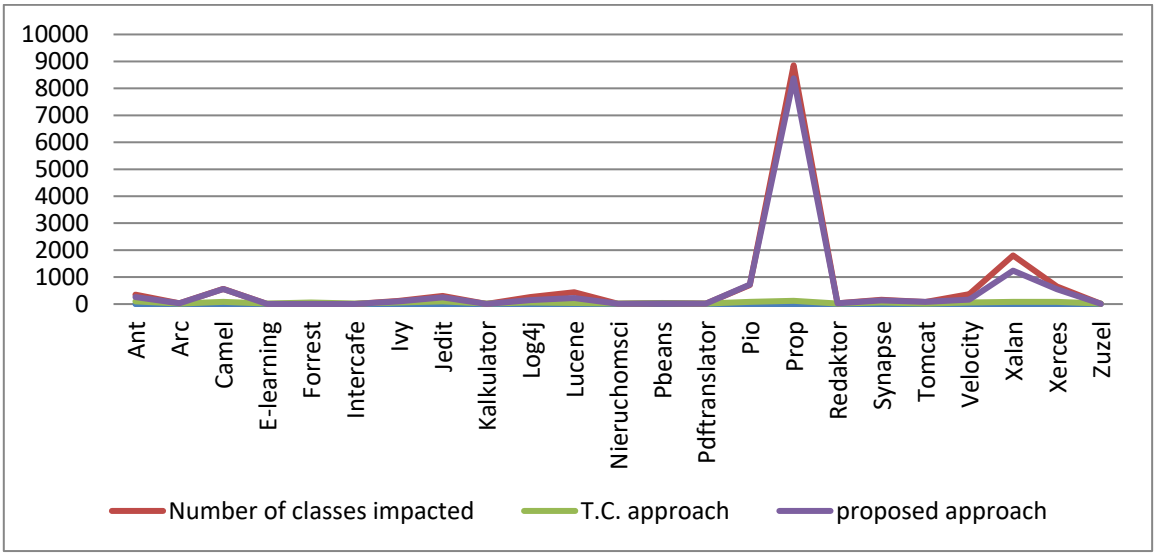

Figure 4. Neural classifier

CONCLUSIONS

In this paper, we presented a comprehensive study aimed at identifying key factors affecting the changeability of object-oriented software. We used a supervised machine learning technique to build robust prediction models that were evaluated on 23 object-oriented systems. In this study, the coupling metrics between classes are used to predict the potential impact of changes in the source code with reasonable accuracy. The results can be summarized as follows: These models perform well regardless of the nature of the system, but their predictive power is limited by the simplicity of the metrics used and the absence of semantic or contextual data. This is evident when observing a class whose interface has not changed; nonetheless, hidden dependencies and internal logic shifts can still affect the systems behavior. - Structural metrics contribute to predicting changes, provided the system follows the

principles of the object-oriented paradigm. This has been observed in Elearning, Forrest, Kalkulator, Nieruchomosci, Pdftranslator, Redaktor, Tomcat, and Zuzel, which all exhibit strong adherences to OO principles. - In some cases, general models yield better results than individual system models. We believe it may be possible to use a general model to address a new system in the absence of historical data, particularly in early development stages. Several factors can jeopardize the validity of studies and limit the generalizability of results. First, there is the choice of systems.

The systems were selected based on their nature (library vs. application), popularity, and maturity. However, the domain was not considered, as three of the four systems are development tools, which may bias the behavior of the metrics. The same claims have already been made by previous related works [38]. All these works have shown the importance of coupling as the most precise and reliable measure for estimating the impact of change, especially in large, complex software architectures.

REFERENCES

[1] Vaucher, S. and H.A. Sahraoui. Étude de la changeabilité des systèmes orientés objet. in LMO. 2008.
[2] Kaur, U. and G. Singh, A review on software maintenance issues and how to reduce maintenance efforts. International Journal of Computer Applications, 2015. 118(1): p. 6-11.
[3] Lee, M., A.J. Offutt, and R.T. Alexander. Algorithmic analysis of the impacts of changes to object-oriented software. in Proceedings. 34th International Conference on Technology of Object-Oriented Languages and Systems-TOOLS 34. 2000. IEEE.
[4] Fitzgerald, B. and K.-J. Stol, Continuous software engineering: A roadmap and agenda. Journal of Systems and Software, 2017. 123: p. 176-189.
[5] Jayatilleke, S. and R. Lai, A systematic review of requirements change management. Information and Software Technology, 2018. 93: p. 163-185.
[6] Jönsson, P. and M. Lindvall, Impact analysis. Engineering and managing software requirements, 2005: p. 117-142.
[7] Nagappan, N. and T. Ball. Use of relative code churn measures to predict system defect density. in Proceedings of the 27th international conference on Software engineering. 2005.
[8] Zhang, L., et al., Empirical research in software engineering—a literature survey. Journal of Computer Science and Technology, 2018. 33: p. 876-899.
[9] Haboush, A., et al., Investigating software maintainability development: A case for ISO 9126. International Journal of Computer Science Issues (IJCSI), 2014. 11(2): p. 18.
[10] Dahane, M., et al., Using design of experiments to analyze open source software metrics for change impact estimation, in Research Anthology on Usage and Development of Open Source Software2021, IGI Global. p. 762-781.
[11] Li, W. and S. Henry, Object-oriented metrics that predict maintainability. Journal of Systems and Software, 1993. 23(2): p. 111-122.
[12] Li, W., et al., An empirical study of object-oriented system evolution. Information and Software Technology, 2000. 42(6): p. 373-381.
[13] Chaumun, M.A., et al., A change impact model for changeability assessment in object-oriented software systems. Science of Computer Programming, 2002. 45(2-3): p. 155-174.
[14] Canfora, G. and L. Cerulo. Fine grained indexing of software repositories to support impact analysis. in Proceedings of the 2006 international workshop on Mining software repositories. 2006.
[15] Abdi, M.K., Analyse et prédiction de l'impact de changement dans un système à objets, 2007, Université d'Oran1-Ahmed Ben Bella.
[16] Zhou, Y., et al. A bayesian network based approach for change coupling prediction. in 2008 15th Working Conference on Reverse Engineering. 2008. IEEE.
[17] Bhattacharya, P., et al. Graph-based analysis and prediction for software evolution. in 2012 34th International conference on software engineering (ICSE). 2012. IEEE.
[18] Bouneffa, M. and A. Ahmad. The change impact analysis in BPM based software applications: A graph rewriting and ontology based approach. in International Conference on Enterprise Information Systems. 2013. Springer.
[19] Musco, V., M. Monperrus, and P. Preux, A generative model of software dependency graphs to better understand software evolution. arXiv preprint arXiv:1410.7921, 2014.
[20] Floyd, R.W., Algorithm 97: shortest path. Communications of the ACM, 1962. 5(6): p. 345-345.
[21] Abdeen, H., et al., Learning dependency-based change impact predictors using independent change histories. Information and Software Technology, 2015. 67: p. 220-235.

[22] de AG Saraiva, J., et al., Classifying metrics for assessing object-oriented software maintainability: A family of metrics' catalogs. Journal of Systems and Software, 2015. 103: p. 85-101.
[23] Almugrin, S., W. Albattah, and A. Melton, Using indirect coupling metrics to predict package maintainability and testability. Journal of Systems and Software, 2016. 121: p. 298-310.
[24] Ahmad, A., H. Basson, and M. Bouneffa. Analyzing and modeling the structural and qualitative interdependencies of software evolution. in 7th International Workshop on Computer Science and Engineering, workshop of, WCSE 2017, International Conference on Software Engineering (ICOSE). 2017.
[25] Mohagheghi, P., V. Dehlen, and T. Neple, Definitions and approaches to model quality in model-based software development–A review of literature. Information and Software Technology, 2009. 51(12): p. 1646-1669.
[26] Müller, K. and B. Rumpe, A model-based approach to impact analysis using model differencing. arXiv preprint arXiv:1406.6834, 2014.
[27] Anwer, S., et al. Effect of coupling on software faults: An empirical study. in 2017 International Conference on Communication, Computing and Digital Systems (C-CODE). 2017. IEEE.
[28] Hegedűs, P., et al., Empirical evaluation of software maintainability based on a manually validated refactoring dataset. Information and Software Technology, 2018. 95: p. 313-327.
[29] Zagane, M., M.K. Abdi, and M. Alenezi, Deep learning for software vulnerabilities detection using code metrics. IEEE Access, 2020. 8: p. 74562-74570.
[30] Zoheir, D.M. and A.M. Kamel, Towards Group Decision Support in the Software Maintenance Process. International Journal of Decision Support System Technology (IJDSST), 2022. 14(1): p. 1-22.
[31] Bouslama, M. and M.K. Abdi, Towards a Formal Approach for Assessing the Design Quality of Object-Oriented Systems. International Journal of Open Source Software and Processes (IJOSSP), 2021. 12(3): p. 1-16.
[32] Judge, V., Apport de l'apprentissage automatique pour la modélisation et l'analyse des changements d'occupation du sol, 2019, Université Bourgogne Franche-Comté.
[33] Cheikhi, L. and A. Abran, An Analysis of the PROMISE and ISBSG Software Engineering Data Repositories. Int. Journal of Computers and Technology, 2014. 13(5).
[34] Mendoza González, G., Herramienta de Desarrollo Netbeans. Herramienta de Desarrollo Netbeans, pág, 2015. 1.
[35] Bali, K., Analyse de changements multiples: une approche probabiliste utilisant les réseaux bayésiens. 2014.
[36] Haykin, S. and N. Network, A comprehensive foundation. Neural networks, 2004. 2(2004): p. 41.
[37] Pan, D., et al. Using dempster-shafer's evidence theory for query expansion based on freebase knowledge. in Asia Information Retrieval Symposium. 2013. Springer.
[38] Abdi, M.K. and D.M. Zoheir, Change Impact Identification in Object-Oriented System: Dependence Graph Approach. International Journal of Education and Management Engineering, 2015. 5(3): p. 1.