Expert Systems with Applications 38 (2011) 2347–2353

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Practical development of an Eclipse-based software fault prediction tool using Naive Bayes algorithm

Cagatay Catal [a,*], Ugur Sevim [a], Banu Diri [b]

[a] The Scientific and Technological Research Council of Turkey (TUBITAK) and the National Research Institute of Electronics and Cryptology (UEKAE), Information Technologies Institute, Kocaeli, Turkey
[b] Yildiz Technical University, Department of Computer Engineering, Istanbul, Turkey

ARTICLE INFO

Keywords:
Machine learning
Naive Bayes
Eclipse technology
Software fault prediction

ABSTRACT

Despite the amount of effort software engineers have been putting into developing fault prediction models, software fault prediction still poses great challenges. This research using machine learning and statistical techniques has been ongoing for 15 years, and yet we still have not had a breakthrough. Unfortunately, none of these prediction models have achieved widespread applicability in the software industry due to a lack of software tools to automate this prediction process. Historical project data, including software faults and a robust software fault prediction tool, can enable quality managers to focus on fault-prone modules. Thus, they can improve the testing process. We developed an Eclipse-based software fault prediction tool for Java programs to simplify the fault prediction process. We also integrated a machine learning algorithm called Naive Bayes into the plug-in because of its proven high-performance for this problem. This article presents a practical view to software fault prediction problem, and it shows how we managed to combine software metrics with software fault data to apply Naive Bayes technique inside an open source platform.

1. Introduction

Over the last 15 years software fault prediction models have received a lot of attention from software engineering researchers and machine learning experts. However, a robust fault prediction model alone is not enough in today's competitive software industry. There is a great need for software tools that make it easier for software quality professionals or project managers to predict faults before they occur. Building and application of a fault prediction model within a software company is time-consuming, detailed, meticulous work and mostly commercial projects do not have enough resources to realize this activity (Ostrand & Weyuker, 2006).

On the other hand, the benefits of this Quality Assurance (QA) activity are impressive. By using such models, one can identify the refactoring candidate modules, improve the software testing process, select the best design from design alternatives with class level metrics, and reach a dependable software system (Catal & Diri, 2008). Hence, a tool for simplifying the prediction process is extremely useful to projects that might not be able to allocate necessary resources for this QA activity (Ostrand & Weyuker, 2006).

Software fault prediction approaches use previous software metrics and fault data to predict the fault-prone modules for the next release of software. If an error is reported during system tests or in field, that module's fault data is marked as 1, otherwise it is marked as 0. For the prediction modeling, software metrics are used as independent variables and fault data (1 or 0) is used as the dependent variable. Therefore, we need a version control system (VCS) such as Subversion to store source code, a change management system (CMS) such as ClearQuest to record fault data, and a tool to collect product metrics (method-level or class-level) from source code. Parameters of the prediction model are calculated using previous software metrics and fault data.

Different version control systems and change management systems may have different kinds of Application Programming Interfaces (APIs) and therefore, we did not aim to use a specific type of VCS or CMS during the development of our software fault prediction tool. In addition, designing a one-size-fits-all tool would not be easy as it seems. We decided to let the prediction tool to calculate the software metrics from a source code directory instead of a VCS. In addition, our prediction tool does not directly read fault data from CMS because sometimes every code change does not necessarily mean a fault. By keeping this strategy in mind, we let the user to add the fault data for each module from an Eclipse editor. Next sections will depict this easy-to-use operation with a figure. For this reason, our prediction tool is not dependent on any

* Corresponding author.
E-mail addresses: cagatay.catal@bte.mam.gov.tr (C. Catal), ugur.sevim@boun.edu.tr (U. Sevim), banu@ce.yildiz.edu.tr (B. Diri).

0957-4174/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2010.08.022

Commercial-Off-The-Shell (COTS) tool. Different project groups will be able to use this tool for their projects in our institute. The only limitation now is the programming language that this tool works on. Our prediction tool can calculate software metrics (Halstead, McCabe and Chidamber–Kemerer) on only Java programs.

We modified the source code of an open source project called Lachesis (Vatkov, 2005) to develop our software fault prediction tool called RUBY. "Lachesis is a software complexity measurement program for Object-Oriented source code" (Vatkov, 2005) and it can analyze both Java source code and byte code. However, Lachesis itself is not a fault prediction tool. Because Lachesis plug-in did not provide information to the user about primitive Halstead metrics such as total operand and unique operator, we changed the source code to let the user see these metrics. Furthermore, dependency metrics (Martin, 1994) such as afferent couplings, abstractness, instability, have been removed from graphical user interface because software fault prediction studies did not take into account these metrics up to now and validation of these metrics for fault prediction have not been done yet. In addition, decision density metric calculated by dividing cyclomatic complexity metric to lines of code metric has been removed because this metric has not been validated for software fault prediction yet. Metrics calculated and shown to the user by our tool are shown in Table 1.

After software metrics are calculated, *MethodLevelTrain.txt* or *ClassLevelTrain.txt* files are created in project tree with respect to RUBY preference page settings. User should add fault-info or fault-free info for each module (class or method) by right clicking to the editor, automatically shown after metrics calculation. Therefore, we contributed two actions, *Add Fault Info* and *Add Fault-Free Info*, on metrics analysis editor. Module class labels initialized with ? character are updated after these actions are activated with 1 value for *Add Fault Info* action and 0 value for the other action. Users can copy this training file into a new software version and then choose *Predict with Naive Bayes* option by right clicking to the project. After this selection, current software metrics are calculated and stored in *MethodLevelTest.txt* or *ClassLevelTest.txt* files. Naive Bayes machine learning algorithm automatically uses train and test files with respect to preference page settings and predicts the fault-prone and faulty-free modules. After the predictions have been made, the results are shown in an Eclipse view called Result view, developed by our project group.

Table 1
RUBY metrics.

Metric type	Metrics groups	Metrics
Method-level metrics	Derived Halstead metrics	Vocabulary
		Implementation length
		Time equation
		Program level
		Program volume
		Program difficulty
		Program effort
		Intelligence content
	Primitive Halstead metrics	Total operand
		Total operator
		Unique operand
		Unique operator
	McCabe metrics	Cyclomatic complexity
		Lines of code
Class-level metrics	Chidamber–Kemerer metrics	Weighted Methods per Class (WMC)
		Depth of Inheritance Tree (DIT)
		Coupling between Object Classes (CBO)
		Number of Children (NOC)
		Response for a Class (RFC)
		Lack of Cohesion of Methods (LCOM)
	Class analysis	Lines of code for class

The architecture of this tool and its reusable parts let us add new machine learning algorithms easily. Training and test files will be used by different algorithms instead of Naive Bayes in that case and other parts of the tool will not be affected. As far as we know, this is the first study to develop an industrial level Eclipse-based software fault prediction tool using machine learning algorithms. The idea and the machine learning method behind this tool is quite influential and any software company can develop their own tool provided that they know the steps we took to develop this prediction tool.

This paper is organized as follows: the following section presents the related work. Section 3 explains Naive Bayes algorithm. Section 4 introduces our Eclipse-based software fault prediction tool. Section 5 presents the conclusions and future works.

2. Related work

Ostrand and Weyuker (2006) discussed the issues in building an automated software fault prediction tool. They had experience working with four industrial software projects and all of these projects used the same VCS and CMS which are fully integrated. Therefore, each Modification Request (MR) to the system is available and the information about each MR is a written description of the change reason. However, it is a crucial problem to determine whether a modification was made because of a fault, or for a functionality enhancement. Ostrand and Weyuker (2006) suggested customizing the MR forms to include an explicit field designating whether or not an MR represents a fault. Otherwise, one should read each MR's natural language description to ensure that it is a software fault or not and it is a time-intensive task for a human to manually read through every MR (Ostrand & Weyuker, 2006). Furthermore, most of the previous projects may not have such a customized field in MR forms as in our case. Because of this problem we aimed to get fault data from the user manually in this version of tool.

Weyuker and Ostrand (2008) described three essential points that should be taken into account when performing research that will finally be transferred to practitioners. Three critical keywords for this kind of research are *aware*, *relevant* and *usable*. Practitioners should aware of what you are doing, they should believe that your research is relevant to their tasks, and they should see that your technique or tool is usable in their environment (Weyuker & Ostrand, 2008). They spoke with practitioners, presented preliminary findings to practitioners, and aimed to build a fully-automated tool requiring no expertise to use. They developed negative binomial regression models to predict the number of faults in each file of the next release.

Different types of learning algorithms such as Genetic Programming (Evett, Khoshgoftaar, Chien, & Allen, 1998), Decision Trees (Khoshgoftaar & Seliya, 2002), Neural Networks (Kanmani, Uthariaraj, Sankaranarayanan, & Thambidurai, 2007; Quah & Thwin, 2004; Thwin and Quah, 2003), Naive Bayes (Menzies, Greenwald, & Frank, 2007), Dempster–Shafer Networks (Guo, Cukic, & Singh, 2003), Case-based Reasoning (El Emam, Benlarbi, Goel, & Rai, 2001), Fuzzy Logic (Yuan, Khoshgoftaar, Allen, & Ganesan, 2000), Artificial Immune Systems (Catal & Diri, 2007), Support Vector Machines (Elish & Elish, 2008; Gondra, 2008), and Ant Colony Optimization (Vandecruys et al., 2008) have been used and their performance has been investigated. Fenton et al. (2007) showed how to model different software lifecycles using a Dynamic Bayesian Network and their approach allowed a Bayesian Network for software fault prediction to be customized to several platforms. Janes et al. (2006) used models such as Poisson regression and zero-inflated negative binomial regression for real-time, telecommunication systems. Tomaszewski, Håkansson, Grahn, and Lundberg

(2007) showed that statistical techniques are better than expert estimations.

Menzies et al. (2007) showed that Naive Bayes with logNums filter provides the best performance on NASA datasets for software fault prediction problem. Menzies et al. (2008) reported that 50 randomly selected modules (25 defective and 25 non-defective) yield as much information as larger datasets and the simple Naive Bayes recommended in their study (Menzies et al., 2007) again performed as well as anything else. Therefore, they suggested focusing on the information content of datasets rather than complicated machine learning algorithms. As explained by Menzies et al. (2008), very small sample of datasets provided effective fault predictors and ignoring large amount of data was not harmful.

3. Naive Bayes algorithm

Classification is a procedure that assigns a class label to a sample from a given set of samples that have labels (Zhang & Sheng, 2004). Naive Bayesian Classification (aka Simple Bayesian Classifier) is the most known and used classification method. It is not only easy to implement on various kinds of datasets, but also it is quite efficient. Let X be a data sample which has no class label. Let H be a hypothesis such that X belongs to a specified class, C. We aim to ascertain $P(H|X)$, the probability that the hypothesis H corresponds with given the observed X. $P(H|X)$ is the posterior probability representing our confidence in the hypothesis after X is given. The Bayesian Theorem provides a way of calculating the posterior probability $P(H|X)$ using probability $P(H)$, $P(X)$ and $P(X|H)$ (Shatovskaya, Repka, & Good, 2006). The basic relation is formulated in Eq. (1):

$$P(H|X) = \frac{P(X|H) \cdot P(H)}{P(X)} \tag{1}$$

Suppose now that there are a set of m samples $S = \{S_1, S_2, \ldots, S_m\}$ where sample S_i is represented as an n-dimensional feature vector $\{X_1, X_2, \ldots, X_n\}$. This is the training dataset. Values X_i correspond to attributes $A_1, A_2, \ldots, A_n$, respectively. Also, there are k classes $c_1, c_2, \ldots, c_k$ and every sample belongs to one of these classes. In our approach, k value is set to two since only fault-prone and non-fault-prone classes exist. Given an additional data sample X whose class is unknown, it is possible to predict the class for X using the highest conditional probability $P(C_i|X)$, where $i = 1, 2 \ldots, k$.

The basic idea of Naive Bayesian Theorem is formulated in Eq. (2):

$$P(C_i|X) = \frac{P(X|C_i) \cdot P(C_i)}{P(X)} \tag{2}$$

As $P(X)$ is the constant for all classes, only the product $P(X|C_i) \cdot P(C_i)$ needs to be maximized. The prior probabilities of the class are calculated using Eq. (3) as given below:

$$P(C_i) = \frac{\textit{number of training samples of class } C_i}{m(\textit{m is the total number of training samples})} \tag{3}$$

Using the Conditional Independence assumption between attributes, we can express:

$$P(C_i|X) = \prod_{t=1}^{n} P(X_t|C_i)$$

where X_t are values for attributes in the sample X (4)

The probabilities $P(X_t|C_i)$ can be estimated from the training dataset calculating for each attribute columns. In our study, datasets we examined have numeric values for each attribute. In this situation, Bayesian classifier uses some density functions such as normal (aka Gauss), lognormal, gamma, and Poisson distributions in order to calculate $P(X_t|C_i)$ values. We used normal distribution and hence, we will not give details of other equations. When we assume that the data have Gauss distribution, $P(X_t|C_i)$ is calculated with Eq. (5) as given below:

$$P(X_t|C_i) = \frac{1}{\sqrt{2\pi\sigma_i^2}} e^{-\frac{(X_t-\mu_i)^2}{2\sigma_i^2}} \tag{5}$$

where $-\infty < x < +\infty$, $-\infty < \mu < +\infty$, and $\sigma > 0$. μ: mean, σ: standard deviation (calculated according to Gauss distribution).

4. RUBY: Eclipse-based software fault prediction tool

This section introduces our Eclipse-based software fault prediction tool using Naive Bayes machine learning algorithm. Section 4.1 explains Eclipse platform, Section 4.2 presents RUBY plug-ins and preferences. Section 4.3 presents RUBY analysis and prediction operations and Section 4 explains RUBY flow graph.

4.1. Eclipse platform

Several *Free and Open Source Software* (F/OSS) communities such as the Apache Software Foundation, GNU, and the Eclipse Foundation delivered high-quality software (MySQL database server, PHP, Python, Mozilla browser, NetBeans IDE, etc.) using different software development processes until now and yet, there is no agreement for a single set of methods and processes to develop free and open source software (Dueñas, Parada, Cuadrado, Santillán, & Ruiz, 2007).

Even there is lack of agreement for processes, F/OSS communities mostly achieved what many companies or associations have tried for years. One of these successes is Eclipse which provides a robust and industry-level integration platform for tools. Eclipse is a vendor-neutral universal platform which can be used to integrate several software development tools and develop applications or software development tools (Clayberg & Rubel, 2004). IBM, Object Technology International (OTI), and eight companies announced Eclipse platform in November 2001 and Eclipse platform evolved during these seven years. Eclipse is not an extensible application, it is exactly a platform. Extensible applications such as Adobe Photoshop or AutoCAD have a monolithic application, and provide extensibility features such as plug-ins or add-ons to enrich the application. Because every part of extensible applications is not a plug-in, their architectures are not as flexible as platforms.

Fig. 1 shows Eclipse platform including three layers: IDE, Rich Client Platform, and Java Virtual Machine. Rich Client Platform includes (Daum, 2005):

- Platform runtime: This runtime is based on OSGi (Open Services Gateway Initiative) and OSGi technology provides Java component and service model for developing embedded software on devices (McAffer & Lemieux, 2006).
- SWT (Standard Widget Toolkit): SWT is a graphical library such as SWING and provides user interface controls such as buttons, menus, and colors (McAffer & Lemieux, 2006). Because SWT uses native widgets, the look and feel of SWT application is similar with host system. SWT uses GTK library in Linux, Motif library in AIX, and Carbon library in MacOS X.
- JFace: JFace is a user interface framework that provides classes for common user interface programming tasks. Dialog windows, wizards, and preference pages can be developed with this framework.
- Help: Help component simplifies providing help support to tools or plug-ins. Developers create several html and xml files to provide static content or context-sensitive help.

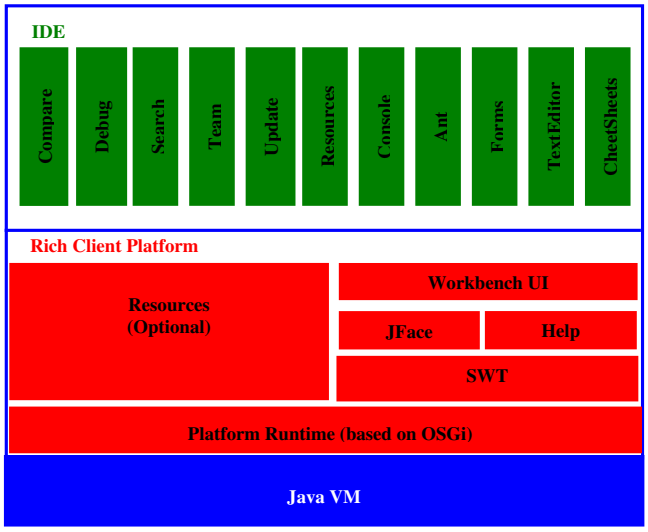

Fig. 1. Eclipse platform.

- Workbench User Interface (UI): Editors, views, and perspectives locate in Workbench UI. If user can edit data in a window, it is called editor and otherwise called view. A perspective includes several editors and views for a specific purpose. For example, debugging perspective may include breakpoints and variables views.
- Resources: Resources plug-in introduces a workspace and files, folders, and projects can be created inside this workspace. This plug-in is not only for tools and sophisticated file manipulations can be managed with this plug-in support.

IDE layer includes several features such as debug, search, team, and update. Team provides access to CVS repositories. Update sites are created using Update framework by plug-in developers and accessed from Software Updates section of Eclipse. Furthermore, Java Virtual Machine (VM) layer is necessary for Eclipse to work.

4.2. RUBY plug-ins and preferences

Our Eclipse-based software fault prediction tool includes the following plug-ins:

- *gov.mam.bte.ruby.eclipse.plugin*: This plug-in calculates the metrics and provides interaction mechanisms with the user. This is the main plug-in of our tool.
- *gov.mam.bte.ruby.thirdPartyLibraries*: This plug-in has been created from existing third party jar achieves. These jar files belong to SableCC and log4j projects. One of them is used for logging and the other one, SableCC, is used for Java lexical analysis and Java parsing operations.
- *gov.mam.bte.ruby.help*: This plug-in provides static html files to the user.

User can customize RUBY tool settings from Eclipse Preference Pages and we developed *RUBY Metrics* preference page for this purpose. This page includes five sub-pages:

- *RUBY Metrics*: This page determines whether metrics will be stored in a training file or not. "*Create Train Files*" option should be selected if there is not any train file yet. Otherwise, "*do not Create Train Files*" option should be selected.
- *Additional Settings*: This page represents metrics with names or abbreviations regarding to the user selection in this page.
- *Analysis Packages*: Metrics which will be used for software fault prediction can be customized from this page. Table 1 lists these metrics and Fig. 2 shows this page.
- *Class Metric Limits*: Software fault prediction models require previous fault data, but sometimes there may not be any fault data. In this situation, modules are marked with red color if any metric for that module exceeds its threshold level. Red label is propagated until package node is reached. This page customizes class metric thresholds and currently our tool uses same thresholds explained in McCabeIQ tool documentation.

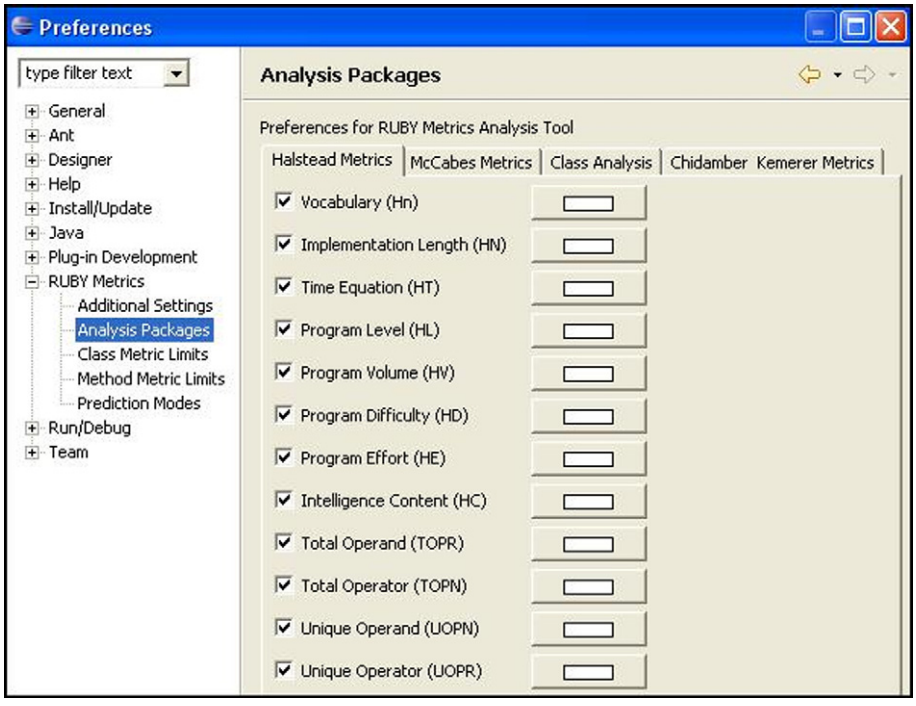

Fig. 2. Analysis packages.

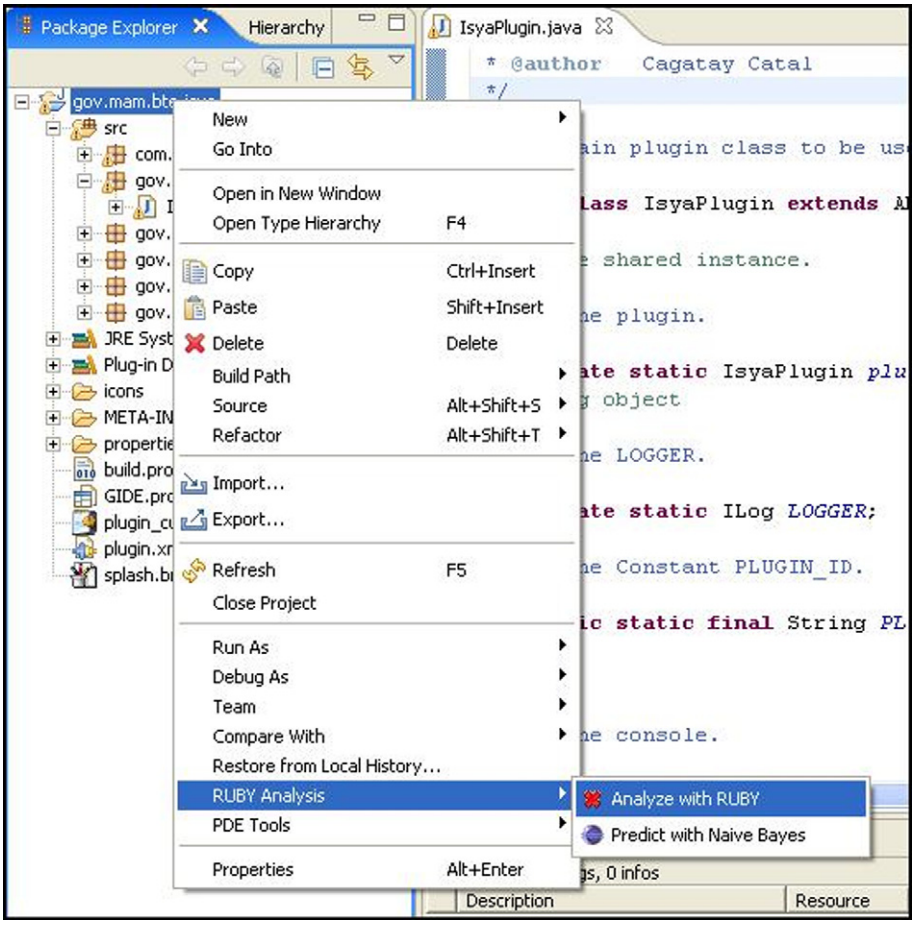

Fig. 3. RUBY analysis.

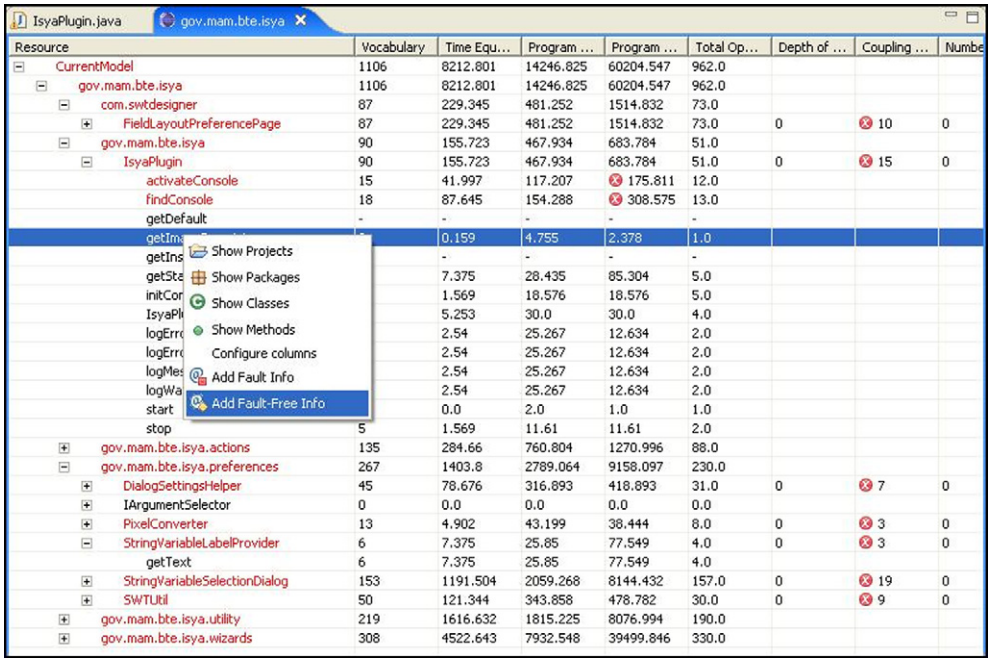

Fig. 4. Adding fault information.

- *Method Metric Limits*: This page customizes method metric thresholds and currently our tool uses same thresholds explained in Predictive tool documentation.
- *Prediction Modes*: This page determines metrics type which will be used for software fault prediction. Currently class and method level prediction modes exist in this tool.

Problems | Javadoc | Declaration | Result View

Result for Naive Bayes Prediction

Module Name	Class Result
gov.mam.bte.isya.actions.InToOut	Low
gov.mam.bte.isya.actions.BuildJobAction	Low
gov.mam.bte.isya.actions.ConfigureJobAction	Low
gov.mam.bte.isya.actions.CleanJobAction	High
gov.mam.bte.isya.actions.UserInformationSingleton	Low
gov.mam.bte.isya.preferences.SWTUtil	Low
gov.mam.bte.isya.preferences.PixelConverter	Low
gov.mam.bte.isya.preferences.StringVariableSelectionDialog	High
gov.mam.bte.isya.preferences.StringVariableLabelProvider	Low
gov.mam.bte.isya.preferences.DialogSettingsHelper	Low
com.swtdesigner.FieldLayoutPreferencePage	High
gov.mam.bte.isya.utility.LinkedResourceUtil	High
gov.mam.bte.isya.utility.FileTypeDetector	Low
gov.mam.bte.isya.utility.FileUtility	High
gov.mam.bte.isya.IsyaPlugin	High
gov.mam.bte.isya.wizards.ProjectNameWizardPage	High
gov.mam.bte.isya.wizards.SomeRunnable	Low
gov.mam.bte.isya.wizards.KernelSourcesWizardPage	High
gov.mam.bte.isya.wizards.InitialRamDiskWizardPage	High

Fig. 5. Naive Bayes results.

4.3. RUBY analysis and prediction

RUBY analysis and prediction is quite straightforward from the user perspective. Right clicking on the project in *Package Explorer* view and selecting *Analyze with RUBY* lets us see metric values and Fig. 3 shows this operation.

After this step, a training file including software metrics for each module is created. Class labels are shown with ? character as explained in previous sections. User should add fault information and fault-free information into metric editor shown in Fig. 4 for each module.

Class label for each module is updated with 1 or 0 value depending on the selected action (Add Fault Info or Add Fault-Free Info). Thus, training file gets ready for the prediction phase and later, user can copy this training file from project tree into any project tree which needs fault prediction. When user selects *Predict with Naive Bayes* action shown in Fig. 3, a test file is created with the current software metrics and training file is used to predict the class labels of modules locating in test file. It is also possible to choose the new version modules which will be used for the prediction step. Prediction is done with Naive Bayes algorithm and results are shown in an Eclipse view called *Result View* developed by our developer group. Fig. 5 shows this result view.

4.4. RUBY flow graph

During the development of RUBY tool, we modified the source code of Lachesis plug-in to enrich it with fault prediction capabilities. Java grammar is built using open source SableCC library and Lachesis plug-in includes metrics extraction features. "SableCC is a parser generator which generates fully featured object-oriented frameworks for building compilers, interpreters and other text parsers" (www.sablecc.org). Generated frameworks consist of Abstract Syntax Trees and tree walkers, simplifying parsing operation. Overview of RUBY flow graph is shown in Fig. 6.

Java Lexer component in Fig. 6 converts a sequence of characters into Java tokens and *Java Parser* component makes a syntactic analysis with respect to a formal Java grammar by using SableCC library. *Java CST/AST* (Concrete/Abstract Syntax Tree) component provides the Abstract Syntax Tree that we work with. Metrics are calculated inside *org.bog.lachesis.impl* package and *org.bog.lachesis.eclipse.plugin.editors* package is used to interact with the user. Fault data taken from the user are written into a train file by using editor package. Naive Bayes implementation evaluates train and test files, and finally provides the result in ResultViewPart shown in Fig. 6.

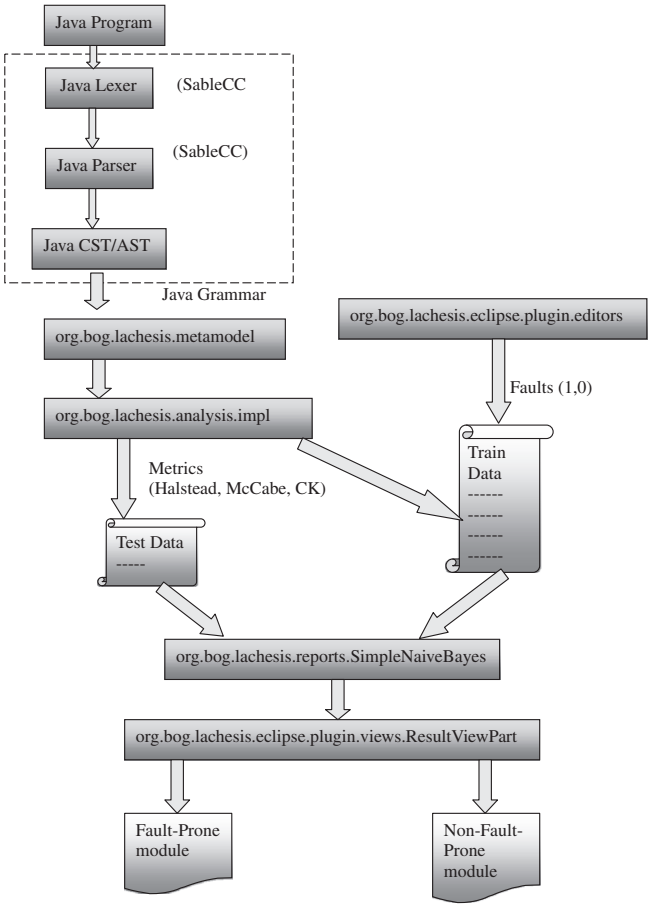

Fig. 6. RUBY flow graph.

5. Conclusions and future works

We developed an Eclipse-based software fault prediction tool for Java programs to simplify the fault prediction process and we also integrated a machine learning algorithm called Naive Bayes into this plug-in because of its proven high-performance for this problem. As this tool proves, machine learning algorithms can be easily used for real world applications and problems. We examined the probabilistic results of our Naive Bayes implementation and noticed that we have same results with WEKA machine learning tool's Naive Bayes implementation. This step helped us to validate our implementation. Because Naive Bayes is one of the most powerful machine learning algorithms for software fault prediction problem, we implemented and integrated it into our Eclipse based plug-in. New machine learning algorithms such as Random Forests and J48 can easily be integrated into this fault prediction plug-in. For the near future, we plan to make testers adapt to this platform and provide presentations or trainings in our research institute.

Acknowledgement

This project is supported by the Scientific and Technological Research Council of TURKEY (TUBITAK) under Grant 107E213. The findings and opinions in this study belong solely to the authors, and are not necessarily those of the sponsor.

References

Catal, C., & Diri, B. (2007). Software fault prediction with object-oriented metrics based artificial immune recognition system. In *Proceedings of the 8th international conference on product focused software process improvement. Lecture notes in computer science* (pp. 300–314, Vol. 4589). Riga, Latvia.

Catal, C., & Diri, B. (2008). A fault prediction model with limited fault data to improve test process. In *9th international conference on product focused software process improvement. Lecture notes in computer science* (pp. 244–257, Vol. 5089). Monte Porzio Catone, Italy.

Clayberg, E., & Rubel, D. (2004). *Eclipse: Building commercial quality plug-ins*. Addison-Wesley.

Daum, B. (2005). *Professional Eclipse 3 for Java developers*. Wrox.

Dueñas, J. C., Parada, H. A. G., Cuadrado, F., Santillán, M., & Ruiz, J. L. (2007). Apache and Eclipse: Comparing open source project incubators. *IEEE Software, 24*(6), 90–98.

El Emam, K., Benlarbi, S., Goel, N., & Rai, S. (2001). Comparing case-based reasoning classifiers for predicting high risk software components. *Journal of Systems and Software, 55*(3), 301–320.

Elish, K. O., & Elish, M. O. (2008). Predicting defect-prone software modules using support vector machines. *Journal of Systems and Software, 81*(5), 649–660.

Evett, M., Khoshgoftaar, T., Chien, P., & Allen, E. (1998). GP-based software quality prediction. In *Proceedings of the 3rd annual genetic programming conference* (pp. 60–65). San Francisco, CA.

Fenton, N., Neil, M., Marsh, W., Hearty, P., Marquez, D., Krause, P., et al. (2007). Predicting software defects in varying development lifecycles using Bayesian nets. *Information and Software Technology, 49*(1), 32–43.

Gondra, I. (2008). Applying machine learning to software fault-proneness prediction. *Journal of Systems and Software, 81*(2), 186–195.

Guo, L., Cukic, B., & Singh, H. (2003). Predicting fault prone modules by the Dempster-Shafer belief networks. In *Proceedings of the 18th IEEE international conference on automated software engineering* (pp. 249–252). Montreal, Canada: IEEE Computer Society.

Janes, A., Scotto, M., Pedrycz, W., Russo, B., Stefanovic, M., & Succi, G. (2006). Identification of defect-prone classes in telecommunication software systems using design metrics. *Information Sciences, 176*(24), 3711–3734.

Kanmani, S., Uthariaraj, V. R., Sankaranarayanan, V., & Thambidurai, P. (2007). Object-oriented software fault prediction using neural networks. *Information and Software Technology, 49*(5), 483–492.

Khoshgoftaar, T. M., & Seliya, N. (2002). Software quality classification modeling using the SPRINT decision tree algorithm. In *Proceedings of the 4th IEEE international conference on tools with artificial intelligence* (pp. 365–374). Washington, DC.

Martin, R. C. (1994). Object-oriented design quality metrics – an analysis of dependencies. In *Position paper, proc. workshop pragmatic and theoretical directions in object-oriented software metrics, OOPSLA'94*.

McAffer, J., & Lemieux, J. M. (2006). *Eclipse rich client platform*. Addison–Wesley.

Menzies, T., Turhan, B., Bener, A., Gay, G., Cukic, B., & Jiang, Y. (2008). Implications of ceiling effects in defect predictors. In *4th international workshop on predictor models in software engineering* (pp. 47–54). Leipzig, Germany.

Menzies, T., Greenwald, J., & Frank, A. (2007). Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering, 33*(1), 2–13.

Ostrand, T. J., & Weyuker, E. J. (2006). On the automation of software fault prediction. *Testing: Academia and Industry Conference - Practice And Research Techniques (TAIC PART 2006)* (pp. 41–48). Windsor, United Kingdom: IEEE Computer Society.

Quah, T., & Thwin, M. M. T. (2004). Prediction of software development faults in PL/SQL files using neural network models. *Information and Software Technology, 46*(8), 519–523.

Shatovskaya, T., Repka, V., & Good, A. (2006). Application of the Bayesian Networks in the informational modeling. *International conference: Modern problems of radio engineering, telecommunications, and computer science, international conference* (p. 108). Lviv-Slavsko, Ukraine.

Thwin, M. M., & Quah, T. (2003). Application of neural networks for software quality prediction using object-oriented metrics. In *Proceedings of the 19th international conference on software maintenance* (pp. 113–122). Amsterdam, The Netherlands.

Tomaszewski, P., Håkansson, J., Grahn, H., & Lundberg, L. (2007). Statistical models vs. expert estimation for fault prediction in modified code – An industrial case study. *Journal of Systems and Software, 80*(8), 1227–1238.

Vandecruys, O., Martens, D., Baesens, B., Mues, C., De Backer, M., & Haesen, R. (2008). Mining software repositories for comprehensible software fault prediction models. *Journal of Systems and Software, 81*(5), 823–839.

Vatkov, B. I. (2005). *Program for automatic analysis of software Metrics. Diploma thesis*. Faculty of Computer Systems and Control, Technical University of Sofia, Sofia, Bulgaria. http://lachesis.sourceforge.net.

Weyuker, E. J., & Ostrand, T. J. (2008). What can fault prediction do for you? In *Tests and proofs, 2nd international conference, TAP 2008. Lecture notes in computer science* (pp. 18–29, Vol. 4966). Prato, Italy.

www.sablecc.org.

Yuan, X., Khoshgoftaar, T. M., Allen, E. B., & Ganesan, K. (2000). An application of fuzzy clustering to software quality prediction. In *Proceedings of the 3rd IEEE symp. on application-specific systems and software eng. technology* (pp. 85–90). Washington, DC: IEEE Computer Society.

Zhang, H., & Sheng, S. (2004). Learning weighted Naive Bayes with accurate ranking. *Fourth IEEE international conference on data mining* (pp. 567–570). Brighton, UK.