Inf Technol Manag (2009) 10:235–251
DOI 10.1007/s10799-009-0062-5

Antecedents of open source software defects: A data mining approach to model formulation, validation and testing

Uzma Raja · Marietta J. Tretter

Published online: 24 November 2009
© Springer Science+Business Media, LLC 2009

Abstract This paper develops tests and validates a model for the antecedents of open source software (OSS) defects, using Data and Text Mining. The public archives of OSS projects are used to access historical data on over 5,000 active and mature OSS projects. Using domain knowledge and exploratory analysis, a wide range of variables is identified from the process, product, resource, and end-user characteristics of a project to ensure that the model is robust and considers all aspects of the system. Multiple Data Mining techniques are used to refine the model and data is enriched by the use of Text Mining for knowledge discovery from qualitative information. The study demonstrates the suitability of Data Mining and Text Mining for model building. Results indicate that project type, end-user activity, process quality, team size and project popularity have a significant impact on the defect density of operational OSS projects. Since many organizations, both for profit and not for profit, are beginning to use Open Source Software as an economic alternative to commercial software, these results can be used in the process of deciding what software can be reasonably maintained by an organization.

Keywords Data mining · Model building · Open source software · Project performance · Text mining

U. Raja (✉)
Department of Information Systems, Statistics and Management Science, University of Alabama, Tuscaloosa, AL 35487, USA
e-mail: uraja@cba.ua.edu

M. J. Tretter
Department of Information and Operations Management, Texas A&M University, College Station, TX 77847, USA
e-mail: mtretter@mays.tamu.edu

1 Introduction

Organizations spend a major portion of their information technology (IT) budget on maintaining software systems [21, 45]. These tasks typically include detecting and removing defects in the source code, adding new functionality to satisfy the user and enhancing the system to react to changes made in the operational environment. Earlier studies have suggested that maintenance accounts for 60–90% of the software lifecycle costs [8, 22, 75]. It is estimated that companies spent over 100 billion dollars on software maintenance in year 2007 and this number is expected to grow to over 130 billion dollars by the year 2010 [55].

To overcome high software expenses, organizations are looking at alternative approaches to traditional solutions. One such approach is the use of open source software (OSS) systems. Recent reports suggest that the use of OSS systems is increasing and it will continue to make a mark on the software industry [13]. As the use of OSS projects in corporate environment increases, OSS maintenance will become a part of the overall IT costs of an organization. The maintenance process in OSS projects is not contractually mandated. Thus, despite a minimal procurement cost, OSS projects could potentially have a high total cost of ownership when the maintenance related expenses are included. Embedded enterprise use of OSS projects could result in a complex maintenance scenario of applications that are supported by commercial vendors and the ones that are not.

A better understanding of OSS maintenance factors would allow the IT decision makers to consider potential tactical and strategic impacts of using these projects. Investigating OSS defect occurrence and the contributing factors would provide an insight into OSS project

maintenance. Therefore, this research develops a model of defect density in OSS projects that are in maintenance phase (post production), with a particular focus on the predictive power of the resulting model developed using controllable factors. Historical data from post production OSS projects hosted at SourceForge.net (SF) is used to develop the model, using Data and Text Mining techniques.

A review of the existing literature and the framework for this study is presented in Sect. 2. Section 3 discusses the methodology of this research. A comprehensive list of independent variables is created using prior literature and by analyzing the SF data warehouse. The analysis in Sect. 4 leads to a set of hypothesis created using the preliminary analysis of a training data sample. The model is refined using a validation data sample and the resulting final model is tested on a test data sample. The predictive power of the test sample indicates the goodness of the model and its suitability for use over a generalized population of OSS projects. Results are discussed in Sect. 5, followed by implications in Sect. 6 and conclusions in Sect. 7.

2 Previous research and conceptual framework

2.1 Software defect prediction

A significant body of research is focused on predicting rate at which defects occur in an operational software system [30]. Post production defects account for a majority of software maintenance costs and allocating resources to defect reduction can be viewed as an IT investment [67]. Fenton and Neil [25] provided an extensive critique of defect prediction models and the statistical techniques used to develop these models. They suggest creation of holistic models that could help understand software maintenance. This research attempts to provide such a holistic view of OSS maintenance by considering process, product and resource characteristics of a project.

The software reliability approach uses the past defects of a project to predict future defect occurrences. Biyani and Santhanam [9] used the defect volume of previous release to predict the defect density of next release. Li et al. [44] found that naïve time series techniques were inadequate in predicting defect occurrence model parameters from one software release to the next and identified the shortcomings of the time series approach to predicting software defects. Raja et al. [60] proposed a time series model to accurately predict future defects in eight OSS projects. Such studies are focused on estimating defects rather than explaining the factors that impact defect density. They are useful for developing testing strategies and allowing testers to estimate the number of undiscovered defects [54]. This study adds to the body of defect prediction by analyzing the underlying factors that contribute to defect density.

Another stream of research has been focused on developing causal models to relate internal and external software characteristics to defect occurrence and to predict reliability [5], maintenance effort [3] and maintenance costs [67]. These models use metrics that could be product, process, or resource based. Several studies have examined the static characteristics of size and complexity of software to evaluate reliability and number of defects [30, 46]. Khoshgoftaar et al. [42] predicted defects for modules using 11 product metrics. Banker et al. [4] developed a model for software defect density using the traditional static measures as well as external environmental factors. Although several metrics-based predictive models have proven promising, they are limited because of a common lack of available metrics in practice [71].

Process quality metrics have also been studied for their impact on software defects. Graves et al. [29] found that process measures based on change history were better predictors of fault rates than product characteristics. Harter et al. [31] found that the process maturity of an organization had a direct relationship to the number of operational defects found in the software developed by an organization. Mockus et al. [53] found process metrics and configuration metrics to be important predictors of software defects in maintenance phase. Popstajanova and Triveda [41] used architecture, utilization, and control flow information to predict the likelihood of defect occurrences.

Resource based characteristics also impact software defects [1, 7, 10]. Some have supported that large teams have more effort available to handle maintenance tasks [2], while others argue that a large team size can cause a negative effect on performance [11]. Conway [16] suggested that communication patterns of teams are reflected in the products they produce. Herbsleb et al. [33] found evidence of improved maintenance performance due to the use of management tools (e.g. CVS).

2.2 Open source software maintenance

Open source software projects are developed and maintained in cyberspace where users can access project archives. This allows a unique opportunity to examine the historical data on defect detection and removal. Researchers have availed these rich datasets to study OSS development and maintenance processes. Godfrey and Tu [28] conducted one of the first analyses of Linux code evolution and compared it to the existing laws of software evolution. Robles [63] built on their work to test the model on a newer dataset and found evidence of OSS evolution being different than commercial software system (CSS) evolution. Paulson et al. [57] compared the evolution of

OSS and CSS projects using linear approximation techniques. Koch et al. [43] found that the nature of OSS evolution largely depends on the size of the project. Software evolution studies have also facilitated study of software maintenance, since they provide a framework for analyzing OSS projects.

McConnell [48] suggested that OSS project effectiveness can be compared to that of leading-edge organizations that use a combination of practices to produce better quality software at low cost and controlled schedules. Thus the structural code quality should be compared to modern software industry. Mockus, Fielding and Herbsleb [52] developed a series of hypotheses through case studies of two large OSS projects: Apache and Mozilla. Defect Density is one of the aspects they examined. Huntley [36] used the debugging data of the same projects and found evidence of organizational learning in maintenance process. He also discovered the non-uniform nature of evolution of complexity and learning effects across OSS projects. Stamelos et al. [68] studied the code quality of OSS project and found that modularity increased code quality and user satisfaction. Thus these researchers have established a need for a deeper understanding of the structural quality and defect density of OSS projects and an examination into the factors that impact them.

The availability of data archives for OSS projects invites the use of mining techniques. Williams and Hollingsworth [73] demonstrated how data mined from source code repositories can improve static analysis tools. They report that defects reported to databases are more often reported by users. The developers tend to directly report bugs in source code. Thus for operational software where we are interested in the customer usage of software and bug detection, use of bug database is better than use of source code [73]. Jensen and Scacchi used data mining techniques for studying OSS development communities [40] by combining Text Mining and link analysis to discover update patterns. Dinh-Trong and Bieman [20] analyzed the maintenance process by mining repositories of a long-lived open source project, FreeBSD project. Zimmerman et al. [76] developed an approach to use association rule mining on CVS data to recommend source code that is potentially relevant to a given code fragment. Ying et al. [74] developed a mining approach on the change history of source code to identify relevant code for change task.

The majority of the OSS studies have been conducted on small sample of large-scale OSS projects. Given the wide range of variables available in OSS repositories and access to techniques that enable analyzing large samples, this study adds to the body of knowledge by using a large sample of OSS projects and utilizes the power of data mining techniques by using large data archives, building hypothesis from data and then testing the hypothesis on another sample of data. This allows development of a robust model that can be generalized to larger population of OSS communities.

2.3 Conceptual framework

Software defect research has traditionally focused on three types of characteristics: Process, Product and Resource. In OSS, however, the end user plays a significant role in software maintenance as well. OSS development is characterized by a close interaction between the end user and the development team [24, 61]. Unresolved defect reports are publically available on project websites and users have the option to participate in the defect resolution process by submitting solutions and patches. This creates a large maintenance community that includes the end user in the defect detection and removal process. Thus the framework adopted for this study identifies relevant constructs from the Product, Process, Resource and End-User characteristics of OSS projects to create a model for defect density. These aspects are discussed below.

Product characteristics refer to the attributes of the software product. Product reliability, maintainability, portability and quality are examples of the product characteristics [26, 59]. Besides the traditional variables, OSS projects provide access to additional attributes that could potentially impact software maintenance. Free availability of OSS projects provides the users with choice of thousands of projects. The popularity of a project is critical in bringing in new users and its eventual success [17]. Popular OSS projects would more likely attract user contributions that would benefit the maintenance process. Prior research in OSS has also analyzed the affects of license choices on the OSS project development [69]. Some licenses are more restrictive than others, thus the extent of upgrades and changes can affect overall maintenance process. Considering the evolving nature of software systems, it is important to consider the ability of software to operate in changing environment. Compatibility refers to the multiple operating environments supported by the software [58]. OSS projects that offer support over multiple platforms would likely attract a diverse audience and would benefit in terms of defect detection and removal.

Process characteristics deal with the development process of software. In the CSS domain, the capability maturity model (CMM) has been proposed as a measure of the software development process. It has been established in prior research that process is directly related to project outcomes [31, 71]. However, in OSS there is a lack of formal definition of organization and process. This by no means indicates absence of organizational structure. Past research has identified the presence of structure and control in OSS communities [39, 65, 72]. The use of project

management methods to control project tasks and resources would affect the maintenance outcomes of OSS projects [39]. OSS projects are developed and maintained by teams that are not physically collocated[1]; the ability to communicate effectively though use of multiple channels would affect the process quality and the maintenance outcomes. In the absence of formal requirements elicitation process, users can request new features through online forums and request tracking tools available through the OSS project hosting sites. The response to implementing these features would vary in projects depending on the internal team processes and would likely affect software maintenance.

Resource characteristics refer to the nature of the team and tools employed in a project. The higher the number of team members the greater is the human effort and contribution in a project [26]. Such effort includes coding, testing, documentation and user support tasks. In OSS projects there is a core group of members who take the responsibility of development and maintenance. The size of this team could impact project maintenance. OSS projects also use tools to manage tasks and resources. Concurrent version system (CVS) keeps records of the modification history of a software project and allows multiple programmers to work on piece of code in parallel [51]. CVS is the most widely used version control system. Its usage allows teams to manage and track changes made to the source code [27].

End-User characteristics refer to the attributes of the end users of the project. Traditionally software projects are developed for a known user with pre-defined requirements [11, 59]. The user does not take a direct part in project development and maintenance. Yet there is evidence that end user participation improves software development and maintenance process in general [32]. OSS projects have the advantage of a more involved end user community throughout their lifecycle [61]. OSS projects are usually initiated by an individual programmer or group of programmers who are trying to solve a problem that is of their own interest [19]. Although these projects do not have predefined clients or users; once a project is launched it is available for public use through the internet and a user community may emerge. Users of OSS projects can be developers themselves or a business entity that adopts OSS projects for corporate use. The nature of the end user and their participation level could impact the maintenance process and the defect density and thus the user characteristics are to be considered in model development.

[1] We acknowledge that there are several large Scale OSS projects that are developed by more traditional teams of software programmers. Such projects provide face to face communication opportunities, user conferences and other avenues of collaboration that are not confined to online development. However, a vast majority of OSS projects, especially the ones hosted by source forge. Net remain small scale projects developed through online collaboration.

3 Methodology

Lack of availability of rich datasets has been a challenge to effective model building in software maintenance studies [25]. However, OSS projects, keeping with the philosophy of free sharing of data, provide public access to their archives. The availability of a large amount of transactional data, make OSS projects an ideal candidate to use exploratory research to hypothesize models that can be validated and tested. If models were to be built and tested based only on existing theories in the CSS domain, there is a chance that some critical variables might be ignored [64]. We therefore use Data Mining techniques for model formulation and testing.

A purely deductive study would rely on predefined constructs and hypothesis. If a domain is new and significant factors are unknown, relying on existing theories can result in missing constructs. Similarly a purely inductive study based on observations could potentially ignore the constructs absent in a sample, but present in a different sample [12]. Data Mining offers the ability to explore large datasets, create knowledge and build valid models through a combination of deductive and inductive methods.

The over view of the model building process through Data Mining methods from a large database is shown in Fig. 1. An initial large data set is split into three

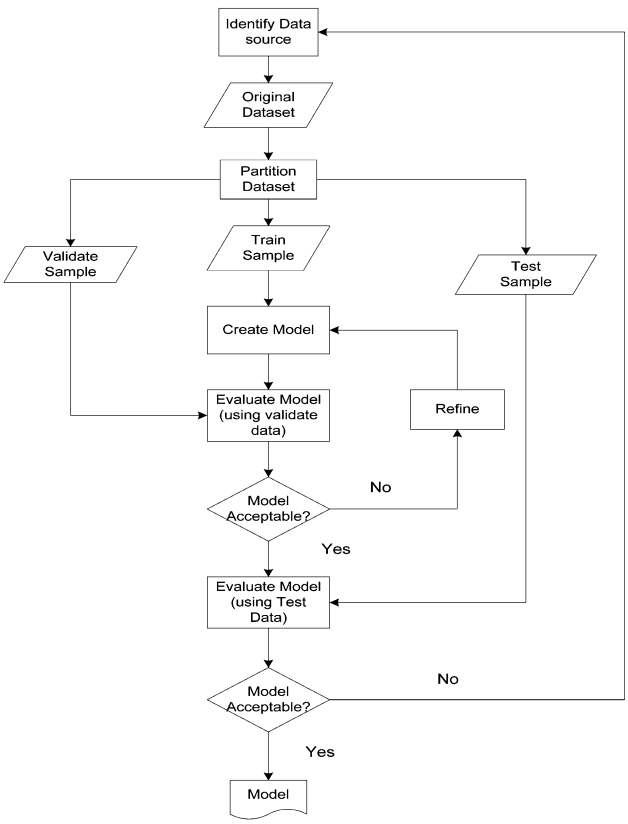

Fig. 1 Model building process through data mining

independent random samples (train, validate, and test) for the purpose of building a final robust model that is not subject to sampling variability within the domain of the original data. The train sample is used to explore various models which are refined with the validation sample and when the final model is determined it is tested on the test sample. Relationships between factors can be examined by using multiple techniques e.g. logistic regression (LR), neural networks (NN) and decision trees (DT) in the training phase of the model building process. Each of these techniques has its strength and weaknesses and their suitability depends on the underlying patterns and relationships in the data which represents the domain being analyzed.

A train sample is used for an initial exploratory analysis to identify variable relationships. The identification of an initial set of variables must be scientific rather than random. Prior research, anecdotal references and study of available data archives are used to identify an extensive list of relevant variables. The tight coupling of existing research and exploration of train samples ensures that multiple dimensions of the problem are explored and that the analysis is relevant to the domain. The process of initial hypothesis development undergoes subsequent validation analysis to establish the rigor or robustness of the model under sampling variability. Using the validation sample the number of variables can be reduced to refine the hypothesis. Once again multiple methods can be used to ensure that the relationships discovered in the first step hold over the entire data set. The fit statistics of the final model and the diagnostic testing is performed on the test sample following the traditional model evaluation process. If the final model does not hold up on the test data then it would be necessary to conclude that a robust model cannot be found for the large data domain [23].

3.1 Data source

SourceForge.net is the world's largest OSS project hosting web site where developers can create, develop, and maintain their OSS projects. SF is a database driven web site, which provides historic and status statistics on over 100,000 projects and records of over 1 million registered users' activities. SF has shared certain data with the research community for the sole purpose of supporting academic and scholarly research on the OSS phenomenon. The SF data archives starting November 1999 through May 2006 were used for this research and were accessed through a research initiative with the University of Norte Dame [47].

The SF warehouse entity relationship diagram (ERD) was studied in detail and the information available in each table was decoded for investigating its usability in this study.[2] To ensure that the projects had been available at SF for some significant time, projects that were less than a year old (from the date of this analysis) were not considered. Some additional transformations (as discussed below) were performed on the data to ensure the validity of the analysis. Extensive SQL queries were used to create the dataset and to generate the variables for each project.

Projects hosted at SF are in various stages of their lifecycle e.g. pre-alpha, alpha, beta, production, mature and inactive. Ignoring the lifecycle phase can result in inconclusive results [26, 48]. To develop a model for operational performance of OSS projects, it is imperative to use projects that are mature or in production. Therefore, projects in all other lifecycle phases were excluded from the sample.

SourceForge.net allows free registration and project hosting that leads to instances where projects are registered, but never activated. Moreover, many projects do not maintain data on all aspects and that results in missing values. Inclusion of such projects would impact the reliability of the results. Therefore, projects that were never operational and had no data available on development, download or defects were excluded from the analysis. The identified variables were collected for the remaining projects. To ensure integrity of the data collection process, the resulting dataset was validated against an independent data extraction performed by another analyst.

3.2 Variable identification

Identification of the initial set of variables is critical in the Data Mining process. Selection of too few variables can result in an incomplete analysis and may result in excluding significant factors from the final model [64]. On the other hand, inclusion of irrelevant variables can adversely affect the model building process and the correct identification of the significant factors. The process of identifying the initial set of variables for use in the Data Mining process requires domain knowledge and an understanding of the available data archives [50]. As discussed in Sect. 2.3, process, product, resource and end-user characteristics play a role in OSS maintenance. Therefore, a range of variables is identified to reflect these constructs by studying the data definitions of the SF data warehouse. A listing of the variables along with the references is given in Tables 1, 2, 3 and 4. All the identified variables have been selected because of their use in prior literature or on their apparent connection to software maintenance. Data Mining is a technique that provides the ability to develop hypothesis

[2] Since the data was used from a third party, independent validation of data was performed by random verification of variables with the actual SF dataset. The results were also compared with another independent extraction of variables from the same warehouse, to verify the queries used for data extraction.

Table 1 Product related variable measurement and sources identified for analysis

Variable	Summary	Measure	Source
Functionality	Functionality refers to the number of functions being offered by the software. Functionality has been used in CSS models. Increase in product functionality is attained through new releases over project lifecycle [39]	Increase in features	Count feature request closed
		New modules	Count file release
Maintainability	Maintainability refers to the extent to which software is maintainable. Maintainable software should not be dependent upon a small group of people who understand it. In OSS the ability of users to be able to understand and maintain code is very critical [68]. If an OSS project is not maintainable, then detecting and removing defects can be a problem	Number of distinct members reporting the defects	Count distinct submitted_by
		Number of distinct members fixing the defects	Count distinct user_id closed_by
Compatibility	The compatibility of a software project indicates the flexibility of project use. Software that runs on multiple platforms offers more flexibility to the user compared to platform dependent software [37]. Similarly support for multiple programming languages can attract contributions from a larger user community	Number of platforms supported	Count operating systems
		Number of prog; languages supported	Count prog; languages
License type	OSS projects are launched under various licenses. The most common license is the OSI license. Prior research indicates that license choice can affect the development performance of OSS projects [69]	License type	OSI (Y/N)
Project type	SF classifies projects based on various aspects, e.g., games, application file transfer protocols, desktop applications, operating system, etc. This variable was used to check some types of projects are more suitable for development in OSS community compared to others	The text description of the project was used to create categorization for project type, using text analysis	200 word, textual description of the project
Popularity	OSS projects are free for download. Users have the choice of thousands of OSS projects. The actual download of a specific project depends upon a variety of factors, one of them being the relevance of the product to the users. Popularity determines how much interest there is in a particular project. Not all downloads translate to actual adoption. However, popularity has been used as a measure of project success in past studies	Downloads	Count downloads
		Page views	Count page views

Table 2 Process related variable measurement and sources identified for analysis

Variable	Summary	Measure	Source
Project management	Use of traditional project management methods include a designated project manager and centralized task allocations. These activities can affect the outcomes of a software project. Though OSS projects are developed in a more informal environment, yet projects may use PM [39]	Whether an OSS project decides to have a project manager or not	Use PM (Y/N)
Process quality	The overall quality of development and maintenance process can affect the performance of a project. In CSS, there is evidence that process quality of a project has a positive impact reducing defects [5]	The response time to fix an error	Mean time to fix a defect (MTTR)
Communication channel	Availability of various methods of communication between the developers and the users can improve the maintenance process in OSS projects [34]	Number of forums	Count
		Use of mail messaging	Use mail (Y/N)
		Use of news groups	Use news groups (Y/N)
Requirement implementation	CSS project performance is associated to its ability to conform to user requirements. In OSS, there are no predefined requirements, yet the end users can make requests for implementing new features to the projects	Response time to feature requests	Time to implement a feature

Table 3 Resource related variable measurement and sources identified for analysis

Variable	Summary	Measure	Source
Effort	The size of the project team will be indicative of how much effort is available for the maintenance process. There are conflicting views on the size of a team. Some researchers support the view that a large team size will have more effort available to the development and maintenance process [1], while others argue that a large team size can cause a negative effect [11]	Number of registered developers for the project	Team size
Team communication	Conway suggested that communication patterns of teams are reflected in the products they produce [16]. OSS teams communicate mostly online. The frequency of this communication can reflect on the maintenance outcomes	Frequency of development team communication	Messages posted at development forums
Version control	In CSS literature, use of configuration management (CM) techniques has been linked to better performance of software [33]. OSS projects use configuration management tools during development and maintenance. There is extensive use of version control tools. The effects of CM on project performance will be investigated	Use of CVS	Use CVS (Y/N)
		Number of CVS commits	Count CVS commits

Table 4 User related variable measurement and sources identified for analysis

Variable	Summary	Measure	Source
User type	OSS projects are developed for various types of end users. Some projects are developed purely for a development community. Others are developed as end user applications that can be used by non-programmers too. Considering the nature of OSS development, the type of audience would affect the extent of end user involvement. If the end users were programmers, they would be able to effectively modify the code	Nature of the end user	Audience programmer (Y/N)
Activity level of user	OSS user can be an active member of the development and maintenance community. They can also contribute to the source code and participate in the detection and removal of defects. Anecdotal references to active user community have been made in literature, but no empirical testing has been performed to investigate its effects on project performance [66]. This research uses the activity level of the user in project development and maintenance performance. This variable has not been used in CSS models	End user interaction	Forum posts by users
		Number of users interacting	Number of distinct individuals posting messages, defects or feature requests
		User involvement in the maintenance process	Defects reported by users who are not a part of the development team
Community size	Using the argument that the active user has an impact on the project performance, the effects of the size of the user community will also be used for the model. A larger community will imply more effort going into the development and maintenance process. This will also test the Linus law, which states, "Given enough eye balls, all bugs are shallow" [62]	Number of active users	Number of distinct senders of messages

from a wide range of *relevant* constructs. This by no means undermines the necessity for correct identification of the relevancy of the initial list based.

3.3 'Project type' variable creation

The projects available at SF are diverse in nature. SF provides a categorization of project, but a single project is allowed to have multiple categories. To overcome this dilemma, the text data describing each project was used to create a new variable called "project type" through Text Mining.

Text Mining allows the numeric interpretation of textual data. A stop list contains words that are to be ignored while performing Text analysis. Such a list typically contains words that are used often and do not contain any useful information. An initial run was performed using a stop list containing such words e.g. on, of, the etc. This run provided with a word frequency table. A new start list was created by removing the words that were not considered

Table 5 Description terms, frequency and percentage of each cluster for project type

Cluster	Percentage	Freq	Descriptive terms
1	0.150252	746	+ Library, C++, + class, python, + support
2	0.035851	178	+ Game, + player, + play, game, + base
3	0.091037	452	+ File, + program, + image
4	0.082578	410	php, + easy, mysql, web, + database
5	0.090634	450	+ Framework, development, + application, java, web
6	0.338771	1,682	+ Server, + client, + allow, + tool
7	0.210876	1,047	+ Code, + source, + project, java

project descriptors or added no usefulness to the analysis e.g. frequent words like where, upon or abbreviation like en, dl etc. Clustering was then performed based on software generated singular value decomposition (SVD) terms [14]. A maximum number of 40 clusters were allowed. Word stemming was used to address the occurrence of the same word in multiple forms e.g. walk, walking etc. The logarithmic method was used for frequency weighing, and the Entropy method was used for term weighing. The expectation maximization algorithm was used for clustering. This algorithm is best suited in cases where the expected number of categories is unknown [18].

The observations were classified into seven clusters. The resulting clusters and the descriptive terms along with percentages and frequencies are shown in Table 5.

The project descriptions split into seven clusters. Cluster #1 contains terms associated with programming languages e.g. C++ and python. Cluster #2 is associated with games, Cluster #3 contains terms related to image programs, Cluster #4 has terms related to databases, Cluster #5 had terms related to JAVA and web applications, Cluster #6 has terms related to networking while Cluster #7 has terms related to general OSS projects. Once the Text Mining results were merged with the original dataset, the new data was ready to be used for model building.

3.4 Dependent variable

Occurrence of defects in operational software systems is an area of concern for IT managers. Larger projects, with more modules and interfaces would encounter more defects, but also provide increased functionality. Therefore, a common measure used in study of post deployment software defects, is the occurrence of defects normalized for the scale of project. Defect density is the number of defects per size of the software. This definition controls for variation in the size of software and allows comparing operational defect occurrence [31].

A defect is defined as a reported bug on the SF data warehouse. Registered SF users can report defects they encounter in a project. The same system is used by the project team members, thus all the defect data is maintained at a central location. The defect dataset includes additional information regarding the member reporting and rectifying the defect, time of report and resolution time. For the dependent variable, we use defect density in terms of number of defects per thousand lines of code (KSLOC). Thus higher number of defect reports, for larger projects compared to smaller projects is accommodated by controlling for size.[3]

4 Analysis

Operational OSS projects in mature or production phase were included in the analysis. The life cycle phases of projects can be tracked using the "*Trove_category*" maintained for every project. Any changes to the lifecycle phase, along with the date of change are also recorded. Using SQL queries, we extracted the projects that were in mature or production phase, thus limiting our dataset to operational OSS projects. Using the criteria of: life cycle phase, active defect repository, and registration date of at least 1 year prior to analysis date, a dataset of *4,965* observations was produced. Text analysis generated variables were merged with the dataset using the project identification number.

The resulting data was then randomly sampled into three independent datasets: train, validate and test: 40% training, 30% validation and 30% test sample. Since the number of low defect density projects is small compared to high defect density, we used stratified random sampling to ensure that every sample contained a proportionate number of high and low defect density projects. The training sample is used for exploring the relationships between the variables. The validation sample is used to refine the model and develop hypothesis. The test sample is used to test the hypothesis and access the fit of the final model.

The process flow diagram of the analysis is shown in Fig. 2. SAS Enterprise Miner 5.2 was used for Text and Data Mining.

[3] We use the raw KSLOC as an indicator of project size, with the acknowledgement that program size can be language dependent. However, KSLOC remains one of the most commonly used measures of software size, especially when the purpose is to control for project scale [28].

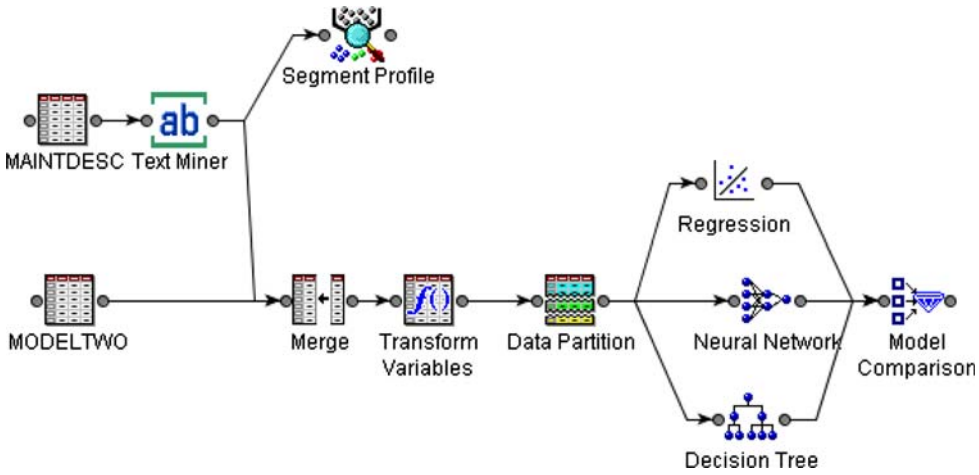

Fig. 2 Process flow diagram for the analysis

4.1 Building the model

LR, NN and DT techniques were used in parallel for model formulation. The stepwise method for variable selection was used in the LR analysis. Stepwise LR is a recommended method of variable selection for exploratory research [35]. The results of all three methods were analyzed to devise a strategy for model improvement. Initial runs indicated that the DT model was the best model. The initial runs of the LR method did not produce the best model. Further analysis of the DT model revealed a persistent splitting of the nodes based on defects reported by end users and the type of project. Based on this added insight from the DT, interaction terms were introduced in the LR model. With the addition of the interaction terms, the LR model improved significantly. Once an acceptable training model was built, the validation set was used to evaluate the model. A comparison was then made with specific diagnostics e.g. lift charts, to check how well the training model holds for the validation sample. At times, there were several iterations of re-training before a reasonable model was selected. Once a model was selected, the validation dataset can no longer be used to test the accuracy of this model. To create a robust model, the final hypothesized model was applied to the new test data. The accuracy of the model on the test data gives a realistic evaluation of the performance of the model for OSS projects in general.

Various factors are considered in the decision of the best model. One such criterion is the receiver operating characteristics (ROC) chart. A ROC chart is traditionally used in signal detection theory to show how the receiver operates on the existence of signal in the presence of noise. It is a plot of the probability of detecting the true signal (sensitivity) and the false signal (1-sensitivity) for an entire range of possible cutoff points. In Data Mining, ROC curves are used to plot the true positive responses against the false positives (identified by the model). The closer the curve follows the left-hand border and the top border of the ROC space, the more accurate the test. The closer the test comes to the 45 degrees line, the less accurate the test is. The area under the ROC curve measures the accuracy of the test. If the area under curve is greater than 0.7 (The entire area being 1.00), the resulting model is considered to be acceptable.

Based on these criteria, the LR model with interaction terms was the best model. The ROC for the train model was .83 which indicates excellent predictive power. The model was evaluated performing diagnostic testing and LR evaluation criteria. The ROC curves for the LR, DT and NN model are shown in Fig. 3. Test sample results are shown here for purpose of final comparison and will be discussed in the results.

4.2 The hypothesized model

The defect density (DDen) will be affected by the product, process, resource and end-user characteristics. Using the significances of the variables in the exploratory analysis and using the prior framework discussed in Sect. 2.3, hypotheses were developed using the train sample. The significance values and the signs reveal the relationships between the independent variables and the dependent variable. Several hypothesis are developed using the initial analysis and are refined using the validate sample. Finally the test data sample is used to determine the predictive power and goodness of the model. These steps ensure that the model holds well across all samples of data and those hypotheses are created and tested on different samples. Following is a brief discussion of the hypotheses.

Open source software domain offers a wide variety of projects to choose from. However, some projects enjoy significant popularity compared to the rest. Projects attract a wide range of users. The reasons of this popularity can be the nature of the project. The popularity of OSS projects

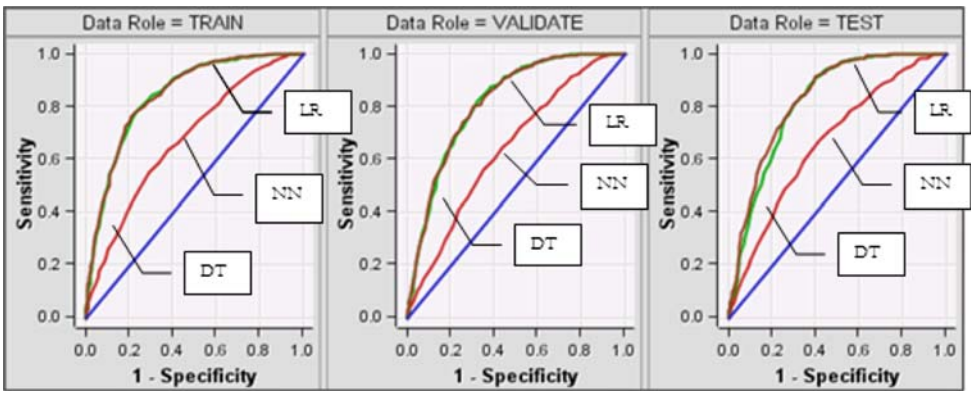

Fig. 3 ROC curves for LR, DT and NN

can be determined by the interest of the user and developer community. The survival of OSS projects depends upon attracting new developers, users and in some cases sponsors who can provide monitory donations. Popular projects would attract traffic and downloads. Popular projects would attract more interest in use and contributions to maintenance process. Prior literature in OSS has focused on the popularity of a project in terms of downloads and page views [70]. In fact it has been used as a measure of success of OSS projects, high popularity indicating that there is a high demand for the project and it provides superior software solutions. Projects with more visibility would attract contributions and downloads. Similarly in the event of a defect, a larger user and developer base would be available to detect and remove defects. Therefore, we analyze the affects of popularity on defect density and hypothesize that:

H1 Popular OSS projects will have a lower defect density.

In terms of process features, we explore the affects of process quality. Prior research in CSS indicates that if the process quality is high, there can be a significant reduction in defect density [31]. Occurrence of defects in software is inevitable. However, their detection and removal efficiency can significantly impact the ability of a project to retain customers and reduce operational costs. When a defect is detected, the team needs to resolve it effectively and swiftly. Delays in resolution of operational defects can increase the operational costs of software systems and reduce customer satisfaction [15]. A project with a high process quality would have effective maintenance resolution practices. Poor process quality can also result in ripple effect i.e. defects leading to additional defects [49]. Therefore a high process quality reflects lower defect density. The training session supports the significance of process quality in defect density. Therefore it is hypothesized that:

H2 OSS projects that have a high process quality will have a low defect density.

An active user community is a unique characteristic of OSS projects. There is a high level of transparency in the development and maintenance process. The end users have access to source code and maintenance archives. There is anecdotal evidence that this characteristic of OSS projects is a potential reason for the success of projects. Users who take ownership of the project and take an active part in the maintenance process can effectively become stakeholders in the maintenance outcomes. Support for process improvement due to end user participation has been studied in CSS as well [6, 38]. Due to the voluntary nature of the maintenance process, an active user community becomes vital for project success. We therefore consider the end user activity in the maintenance process in our model. The train sessions support the suggestion that an involved community of end users could significantly reduce the defect density of OSS projects. This leads to the hypothesis:

H3a OSS projects with high end user involvement will have a low defect density.

Open source software projects differ in nature. Some projects are large scale applications while others are small size plug-ins. The role and nature of user participation also varies from project to project. In a few domains, end user involvement in maintenance can be indicative of a confounding role of end user activity. Complex projects require a certain level of understanding of system modules and their interactions. Interactive user driven applications can benefit from user involvement. This was supported by the train sessions, which indicate that that the effects of project participation varied with the type of projects. Different domains have a different expectation from the end user. Thus leading to the hypothesis:

H3b The relationship between end-user activity and defect density will be moderated by project type

Once a defect is detected in a project, its effective and timely resolution can depend on the availability or lack of availability of manpower. OSS projects thrive in online communities that can promote defect detection; however, in operational projects a degree of understanding of the software itself is required to resolve defects. Effort has been traditionally operationalized as the availability of team members. OSS teams vary in size from a single developer project to several hundred developers. Team size is significant in operational performance because larger teams can not only work on improvement of existing software, but also enable effective post deployment defect resolution. Many OSS teams use subject matter experts (SME) to manage maintenance process and have dedicated members for various parts of the software system. Prior research in OSS indicates that team members become part of the development teams for a wide variety of motivations; however, once committed to a project the additional resource can improve project performance [62]. Analysis of the train sample indicates a significant relationship between team size and DDen. This leads to developing the hypothesis that:

H4 OSS projects with larger teams will have lower defect density.

4.3 Variable operationalization

Section 2.3 provided an overview of the variables used in initial analysis. In this section we discuss the detailed operationalization of the constructs tested in the hypothesis. Table 6 presents a list of the independent variables used, their symbol and an operationalization summary.

The partial, simplified Entity relationship diagram of the SF repository is shown in Fig. 4.

The total number of reported defects for every project is extracted using the project identifier and the defect artifact identifier. The defect density was computed by dividing the total defects by KSLOC. The distribution of defect density reveals that the cutoff of 10 defects per KSLOC (thousand lines of code) was a reasonable point of discrimination between high and low defect density. Therefore, we binned the projects as a 0 or 1 based on the defect density. Projects with less than 10 defects per KSLOC are coded as "low" defect density (DDen = 0) and the project with greater than 10 defects per KSLOC are coded as "High" defect density (DDen = 1). Thus we get a dichotomous or binary dependent variable, with a one indicating a high defect density and a zero indicating a low defect density. The frequency plot of defect density is shown in Fig. 5 and the summary statistics is shown in Table 7.

The popularity of a project is measured in terms of the number of downloads. The SF data warehouse contains the actual count of number of downloads for every project. We used the project identification numbers to extract the number of downloads for each project. The frequency plot of the number of downloads is shown in Fig. 6. SF itself is an OSS project and provides information on its own development and maintenance at the repository. It has the highest number of downloads.

The process quality is operationalized as the time taken to resolve reported defects and other artifacts. For every project there can be several artifacts including defect reports. The time taken by the team to resolve an issue is computed as the difference between the defect report time and the defect resolution time. Since the defect repository

Table 6 Hypothesis, measures and symbols used in the model

Construct	Hypothesis	Measure	Symbol
Defect density		The defect density is measured in terms of number of defects per KLOC. If the number of defects per release are low, the defect density is coded as 0 and vice versa	(DDen)
Popularity	*H1*	The number of downloads (Dwnlds) are used as a surrogate of project popularity. It is expected to have a significantly improve maintenance and thus reduce the defect density	(Dwnlds)
Quality	*H2*	Mean Time to Remove (MTTR) a defect is used to operationalize process quality. A high (MTTR) reflects low quality, thus, resulting in a high defect density	(MTTR)
User activity	*H3a*	The user involvement in the maintenance process is measured as the defects reported by the end users (BNU). OSS success has been attributed to a user community actively engaged in the process of defect detection. A high number of user detected defects is expected to have a significantly negative relationship to defect density	(UsrAct)
	H3b	The user activity will vary for various types of projects. Some domains make effective use of user participation while others are more suited for internal defect tracking. The type of project is ascertained by text analysis of project description. The interaction of user activity and project type is expected to be significant to defect density	(UsrAct) (PrjType)
Team Size	*H4*	A large team can allocate resources on defect resolution effectively because of the availability of more maintainers. A larger team size is expected to have a significantly positive impact on maintenance outcomes and reduce the defect density	(TSize)

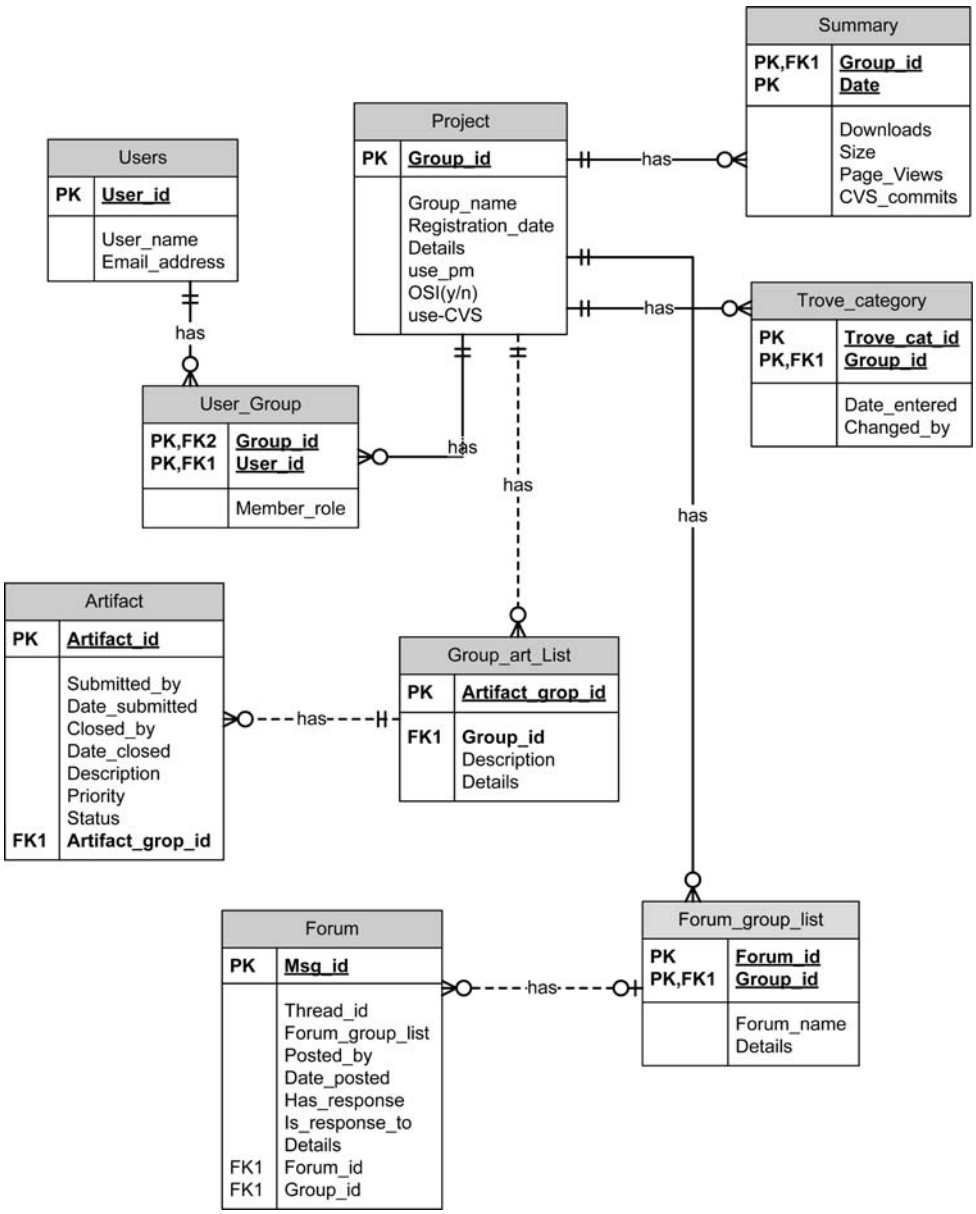

Fig. 4 Partial, simplified ERD of the data source

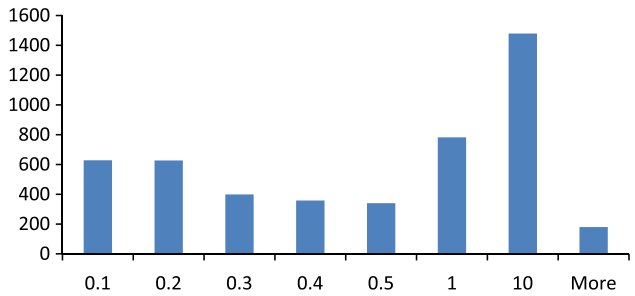

Fig. 5 Frequency plot of defect density

is used for reporting and tracking, the data on time automated and we consider it to be precise. However, lack of precision is actual report time by the team members is a limitation that is beyond the control of the study. The time stamps are available as UNIX epoch time and are transformed into days to produce MTTR in the units days. The frequency plot of the MTTR is shown in Fig. 7. As can be seen a majority of the projects have resolution times in the range of 1–10 days. Defect that takes longer to resolve have a wider spread and have been combined for the ease of scaling the graph. The summary statistics of MTTR is shown in Table 7.

End user activity is measured in terms of user participation in the maintenance process. The user identification numbers of the team members can be identified through their group affiliations. Defects reported by SF registered members who do not belong to the project team are considered user reported. We used queries to identify the count

Table 7 Summary statistics and pair-wise correlation coefficient of the variables

	Mean	SD	Min	Max	DDen	Dwnlds	MTTR	UsrAct	Cluster
Dden	2.215386	11.311	0.0024	572.428					
Dwnlds	108,837.9	1,733,675.12	0	100,177,172	0.0228				
MTTR	84.15856	0	1,342	129.684	−0.0006	0.0501			
UsrAct	56.06593	261.845	1	10,140	0.3200	0.0704	−0.02424		
PrjType	3.983518	1.808	1	6	−0.01584	−0.02036	−0.05873	−0.0042	
TSize	5.005844	9.796	1	302	0.0986	0.0506	0.02598	0.3818	0.01427

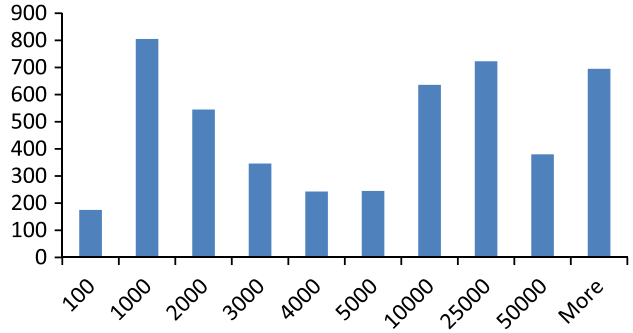

Fig. 6 Frequency plot of the number of downloads

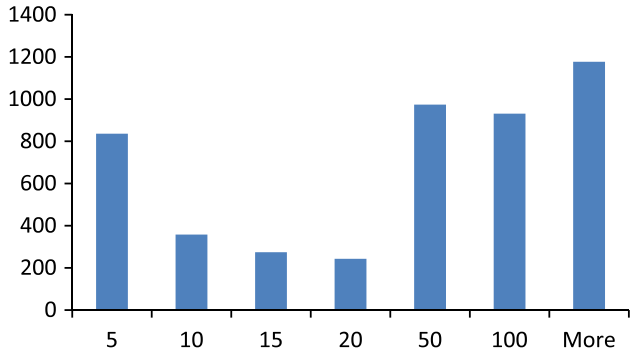

Fig. 7 Frequency plot of the MTTR

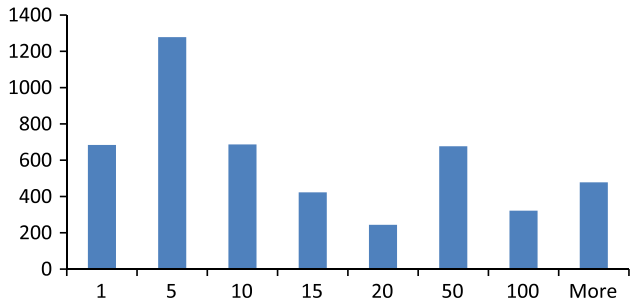

Fig. 8 Frequency plot of the number of user reported defects

of user reported defects using nested queries.[4] The frequency plot of end user activity is shown in Fig. 8 and the summary statistics in Table 7.

[4] All SQL queries are available from the first author upon request.

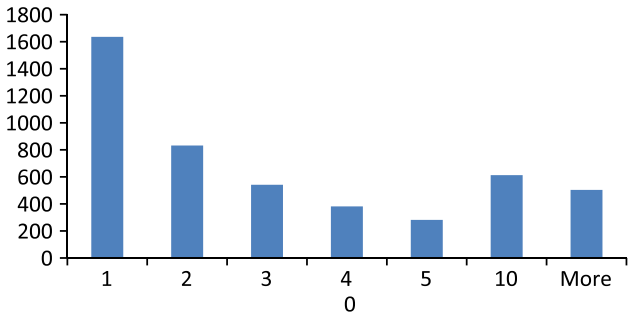

Fig. 9 Frequency plot of team size

Each SF registered member is assigned a user identification number. A registered member can become member of multiple projects. The user identifier and the project identifier together can be used to compute the team size for an individual project. The frequency plot of team size is shown in Fig. 9. The summary statistics of the variable is shown in Table 7.

Operationalization of the variable PrjType was discussed in Sect. 3.3. The frequency plot of the variable is shown in Fig. 10.

The final form of the model can be written as:

$$\begin{aligned}\text{Logit}(\text{DDen}) = {} & \beta_0 + \beta_1(\text{dwnlds}) + \beta_2(\text{MTTR}) \\ & + \beta_3(\text{UsrAct}) + \beta_4(\text{UsrAct } x \text{ PrjType}) \\ & + \beta_5(\text{Tsize})\end{aligned}$$

Figure 11 shows the final model that is to be tested along with the hypothesis numbers. This final model was tested on the independent test sample data. The results are discussed in the next section.

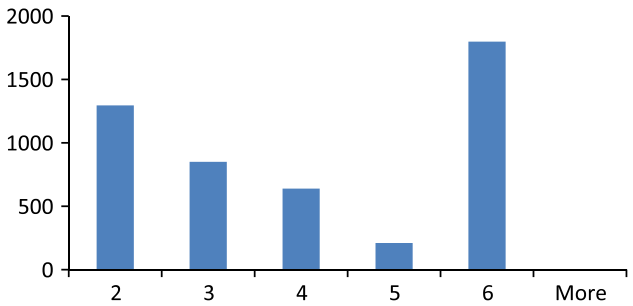

Fig. 10 Frequency plot of project type

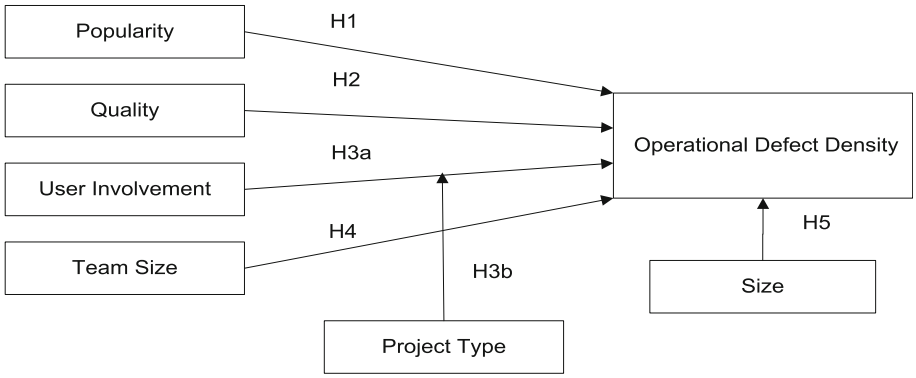

Fig. 11 Model of OSS defect density

5 Results

To test the fit of the final model, the first step is to ensure that the model contains all required variables, entered in the correct functional form. Next, the goodness-of-fit is used to evaluate the effectively of the model. This ensures that knowing the values of all the independent variables in the model allows an accurate prediction of defect density, better than the case of no information in the independent variables. The next step is to evaluate how well the group of independent variables explains the defect density. In LR models, the Log Likelihood (LL) criteria are used to select model parameters. The values of −2LL of the model with and without the independent variables were used to check the model fit. The fit of the model is determined by the reduction in the value of −2LL with and without the covariates. The results of this test are shown in Table 8. The results showed that the overall model is significant ($p < 0.0001$).

A variety of specification tests recommended for LR models were performed on the final model [35, 50]. The Pearson Residuals and Deviance Residuals were examined and no violations were detected. The highest condition number of the model was 1.56, which is well within the recommended cutoff limit. The variance inflation factor (VIF), of the independent variables was well below 5, suggesting that multicollinearity was not affecting the estimates [35, 50].

As previously discussed, the accuracy of LR model can be judged by the area under the ROC curve. As a rule, area under the curve indicates how well the model provides discrimination between the high and low values of the target variable. For the final model the area under the test data ROC curve was .827, which implies that the model provides excellent discrimination between the projects of high and low performance. This test model ROC curve also compares favorably with the Training and Validation curves in Fig. 3. The estimates of the parameters are shown in Table 9. The significances confirm that all the hypotheses developed in the previous section are supported in the final model testing.

The popularity of an OSS project has a significant negative relationship to defect density and significantly reduces its defect density; therefore *H1* is supported by the final analysis. Higher downloads of a project indicate that the project has an interested group of users who are downloading the project for evaluation or use. Thus while evaluating the maintenance outcomes in terms of defect detection and removal for an OSS project, the past data on downloads can be useful.

The process quality results in a significant reduction in defect density, reflected by a positive coefficient of MTTR being significant in the model; therefore *H2* is supported by the final analysis. Projects that take a longer time to recover

Table 8 Likelihood ratio test for global null hypothesis: beta = 0

−2 Log likelihood		Likelihood ratio χ^2	*df*	Pr > χ^2
Intercept only	Intercept and covariates			
904.912	649.183	217.4447	15	<0.0001

Table 9 Model parameters and estimates

Parameter	Estimate	Error	Wald χ^2	Pr > χ^2
Intercept	3.5859	0.2930	14.78	<.0001
Dwnlds	−1.548	0.0694	4.97	0.003
MTTR	0.0125	0.00731	2.93	0.008
UsrAct	−0.004	0.00197	28.02	<.0001
UsrAct × PrjType _1	−.01357	0.00230	4.29	0.00384
UsrAct × PrjType _2	0.0388	0.00178	4.75	0.0029
UsrAct × PrjType _3	0.0171	0.00191	0.8	0.0371
UsrAct × PrjType _4	−0.0129	0.00236	0.3	0.0518
UsrAct × PrjType _5	−0.0139	0.00655	4.49	0.0348
TSize	−0.6219	0.0135	7.91	0.004

from the defects that occur, exhibit low performance and a high defect density. A prompt response to the defects would also translate into low operational downtime and thus help manage IT expenses.

If the end users are active in participating in defect detection, the project exhibits low defect density. This indicates that the user population of the project displays a sense of citizenship and becomes a productive stakeholder in the maintenance process. The affects do, however, vary according to the nature of the project. Both *H3a* and *H3b* are supported by the results of the final analysis.

Team size has a significant negative result on defect density. This indicates that even though the participation is voluntary, the commitment to serve on a project team provides increased effort for the maintenance process. H4 was supported by the results of the final analysis. Commercial companies that use OSS projects, at times dedicate manpower resources to these projects. The research results indicate that such increased resource allocation could improve the OSS project and help in managing the defect density.

6 Discussion

6.1 Implications

This study is the first of its kind to develop a predictive model for OSS maintenance outcomes utilizing the complete SF dataset and using multiple Data Mining techniques. The use of multiple Data Mining techniques on a large database for model development offers a fertile avenue for future research. The use of separate data for model creation, validation and testing gives some assurance that the results are robust across samples from the SF OSS data set. At the same time Data Mining sampling techniques provide the ability to explore a wide range of factors. Such methods can be used in other domains (where large transactional data is available) to improve the existing theories by supplementing the choice of available constructs and powerful computation.

The use of Text Mining to create new variables, provides an avenue for using qualitative research methods along with quantitative methods to extract useful knowledge from the rich textual archives created and available in the new age of cyber communication.

The affects of end-use involvement in project performance calls for more work in the area of potential use in CSS projects. This role varies by the nature of the project that indicates that the role of end user changes as the nature of project changes. It would be worthwhile to study CSS projects and see if the pattern is across domains i.e. CSS or OSS or is across project types.

For practitioners, this study provides a framework for evaluation OSS projects. It gives the development team factors identified in the study that can be monitored and improved to control maintenance tasks. The model can also be used as an alert system to signal upcoming potential performance issues to users and development teams. End users can plan better for budget and system downtime. The development teams can factor in model indicators to estimate manpower requirements and plan for upgrades.

6.2 Limitations and future research

This research exclusively uses project data from the largest OSS project hosting community, SF. There is a possibility of the results not being generalizable over other OSS communities. Given the fact that SF is the largest and the most popular community, this threat is not deemed critical. However, future replication, validation and enhancement of this model requires similar analysis to be performed on datasets from other OSS communities e.g. Freshmeat.org.

The computation of defect density relies on the assumption that all reported defects are legitimate, accurate and complete. However, it is possible that some defect reports may be invalid, e.g. an erroneous or duplicate report submitted by a user. In such a case, the reliability of the measure becomes doubtful. To mitigate this issue, the defect reports that had the same report and removal time were excluded from the analysis.

The total defects were normalized for project size in KSLOC, when computing the DDen. Raw KSLOC measure includes comments in the source code. The variation in density of comments across different projects can introduce some unnecessary noise. This variation might cause inflation in project size for well commented projects and vice verse. The size of program in terms of lines of code also varies with languages. Some programming languages produce more lines of code than others. Despite these limitations, KSLOC is a useful and consistent measure of size especially, when used to normalize quality indicators [56].

To overcome the threats to validity, data collection was carefully designed. Two researchers extracted data independently and compared the counts to verify the queries used for data extraction. Multiple methods and multiple samples ensure that the errors in establishing relationships are minimized. Some OSS projects have internal defect reporting and communication channels. Such data is not available through the SF warehouse and its inclusion in the analysis is beyond the scope of this work.

Another study (already in progress) uses the defect intensity categorization to create a model that predicts major defects vs. minor defects. Such model could be used by IT professionals to develop response plans for

addressing major maintenance blackouts and develop better mechanisms for evaluating OSS projects.

7 Conclusions

The study develops a model to predict the defect density in post production OSS projects and identifies the factors that affect maintenance outcome. Data Mining and Text Mining methods were used on archival data of SF projects in the operational phase. After investigating a wide range of variables using multiple techniques, the resulting model reveals that the end user activity in the process of maintenance is critical to OSS defect density. The number of downloads and the type of the project are also important in evaluating OSS projects in terms of maintenance. The study also confirms that Data Mining techniques can contribute to effective predictive model building in OSS domain.

References

1. Abdel-Hamid T (1989) The dynamics of software project staffing: a system dynamics based simulation approach. IEEE Trans Softw Eng 15(2):109–119
2. Abdel-Hamid T (1992) Investigating the impacts of managerial turnover/succession on software project performance. J Manage Inf Syst 9(2):127–144
3. Banker RD, Datar SM, Kemerer CF (1991) A model to evaluate variables impacting the productivity of software maintenance projects. Manage Sci 37(1):1–18
4. Banker RD, Datar SM, Kemerer CF, Zweig D (2002) Software errors and software maintenance management. Inf Technol Manage 3(1–2):25–41
5. Banker RD, Davis GB, Slaughter SA (1998) Software development practices, software complexity and software maintenance performance. Manage Sci 44(4):433–450
6. Barki H, Hartwick J (1994) Measuring user participation, user involvement, and user attitude. MIS Q 18(1):59–82
7. Barry EJ, Kemerer CF, Slaughter SA (2006) Environmental volatility, development decisions, and software volatility: a longitudinal analysis. Manage Sci 52(3):448–464
8. Bennett KH, Rajlich VT (2002) Software maintenance and evolution: a roadmap. In: 22nd ICSE, Limrick, Ireland, pp 75–87
9. Biyani S, Santhanam P (1998) Exploring defect data from development and customer usage on software modules over multiple releases. IBM T.J. Watson Research Center, Yorktown Heights
10. Boehm B (1987) Improving software productivity. IEEE Comput 20(1):43–57
11. Brooks FJ (1995) The mythical man-month. Addison-Wesley, Reading
12. Bryant A, Charmez K (2007) Grounded theory in historical perspective: an epistemological account. In: Bryant A, Charmez K (eds) The Sage handbook of grounded theory. Sage, Los Angeles, pp 31–57
13. Cearley DW, Fenn J, Plummer DC (2005) Gartner's positions on the five hottest IT topics and trends in 2005. Gartner Inc, Stamford
14. Chiarini-Tremblay M, Berndt DJ, Foulis P, Luther S (2005) Utilizing text mining techniques to identify fall related injuries. In: Eleventh Americas conference on information systems, Omaha, NE, 11–14 August 2005, pp 1497–1504
15. Chulani S, Ray B, Santhanam P, Leszkowicz R (2003) Metrics for managing customer view of software quality. In: Proceedings of ninth international software metrics symposium, pp 189–198
16. Conway ME (1968) How do committees invent? Datamation 14(4):28–31
17. Crowston K, Annabi H, Howison J (2003) Defining open source project success. In: International conference of information systems, Seattle, WA, 14–17 December 2003
18. Deerwester EA (1990) Indexing by latent semantic analysis. J Am Soc Inf Sci Technol 41(6):391–401
19. Dempsey BJ, Weiss D, Jones P, Greenberg J (2002) Who is an open source software developer? Commun ACM 45(2):67–72
20. Dinh-Trong TT, Bieman JM (2005) The FreeBSD project: a replication case study of open source development. IEEE Trans Softw Eng 31(6):481–495
21. Eastwood A (1993) Firm fires shots at legacy systems. Comput Canada 19(2):17
22. Erlich L (2000) Leveraging legacy system dollars for e-business. IT Pro 17–23
23. Fayyad U, Piatetsky-Shapiro G, Smyth P (1996) Data mining and knowledge discovery in databases. Commun ACM 39(11):37–54
24. Feller J, Fitzgerald B (2002) Understanding open source software development. Addison-Wesley, London
25. Fenton NE, Neil M (1999) A critique of software defect prediction models. IEEE Trans Softw Eng 25(5):675–689
26. Fenton NE, Pfleeger S (1991) Software metrics: a rigorous approach. Chapman & Hall, New York
27. German DM (2006) An empirical study of fine-grained software modifications. Empir Softw Eng 11:369–393
28. Godfrey M, Tu Q (2001) Growth, evolution and structural change in open source software. In: International workshop on principles of software evolution. ACM, Vienna, Austria
29. Graves TL, Karr AK, Marron JS, Siy H (2000) Predicting fault incidence using software change history. IEEE Trans Softw Eng 26(7):653–661
30. Gremillion LL (1984) Determinants of program repair maintenance requirements. Commun ACM 27(8):826–832
31. Harter DE, Krishnan MS, Slaughter SA (2000) Effects of process maturity on quality, cycle time, and effort in software product development. Manage Sci 46(4):451–466
32. Hartwick J, Barki H (1994) Explaining the role of user participation in information system use. Manage Sci 40(4):440–465
33. Herbsleb J, Zubrow D, Goldenson D, Hayes W, Paulk M (1997) Software quality and the capability maturity model. Commun ACM 40(6):30–40
34. Herbsleb JD, Moitra D (2001) Global software development. IEEE Softw 18(2):16–20
35. Hosmer DW, Lemeshow S (2000) Applied logistic regression. Wiley, New York
36. Huntley CL (2003) Organizational learning in open-source software projects: an analysis of debugging data. IEEE Trans Softw Eng 50(4):485–493
37. IEEE-STD-1061 (1993) IEEE standard for a software quality metrics methodology. Institute of Electrical and Electronics Engineers, Inc, New York
38. Ives B, Olson MH (1984) User involvement and MIS success: a review of research. Manage Sci 30(5):586–603
39. Jensen C, Scacchi W (2005) Collaboration, leadership, control, and conflict negotiation and the Netbeans.org open source software development community. In: Proceedings of the 38th annual Hawaii international conference on system sciences, HICSS '05, Hawaii, p 196.2

40. Jensen C, Scacchi W (2004) Data mining for software process discovery in open source software development communities. In: Proceedings of workshop on mining software repositories, Edinburgh, Scotland, pp 449–462
41. Popstajanova K, Trivedi K (2001) Architecture based approach to reliability assessment of software systems. Perform Eval 45:179–204
42. Khoshgoftaar T, Munson J, Lanning D (1993) A comparative study of predictive models for program changes during system testing and maintenance. In: International conference on software maintenance
43. Koch S (2007) Software evolution in open source projects—a large-scale investigation. J Softw Maint Evol Res Prac 19(6): 361–382
44. Li PL, Shaw M, Herbsleb J, Ray B, Santhanam P (2004) Empirical evaluation of defect projection models for widely-deployed production software systems. In: Proceedings of the 12th ACM SIGSOFT twelfth international symposium on foundations of software engineering, Newport Beach, CA, pp 263–272
45. Lientz BP, Swanson EB (1980) Software maintenance management. Addison-Wesley, Reading
46. Lind RK, Vairavan K (1989) An experimental investigation of software metrics and their relationship to software development effort. IEEE Trans Softw Eng 15(5):649–653
47. Madey G, SourceForge.net Research Data Archive, http://www.nd.edu/~oss/Data/data.html. Accessed May 2007
48. McConnell S (1999) Open source methodology: ready for prime time? IEEE Softw 16(4):6–8
49. Melouk S, Raja U, Keskin B (forthcoming) Managing resource allocation and task prioritization in a large scale virtual development project. Inform Res Manage J
50. Menard SW (2002) Applied logistic regression analysis. Sage, Thousand Oaks
51. Michlmayr M, Senyard A (2006) A statistical analysis of defects in Debian and strategies for improving quality in free software projects. In: Bitzer J, Schröder PJH (eds) The economics of open source software development, pp 131–148
52. Mockus A, Fielding RT, Herbsleb J (2002) Two case studies of open source software development: Apache and Mozilla. ACM Trans Softw Eng Methodol (TOSEM) 11(3):309–346
53. Mockus A, Weiss D, Zhang P (2003) Understanding and predicting effort in software projects. In: International conference on software engineering
54. Musa J, Iannino A, Okimoto K (1990) Software reliability. McGraw-Hill, New York
55. Pang A (2008) Worldwide Enterprise Applications 2008–2012 Forecast Update and 2007 Vendor Shares, IDC, Editor, p 55
56. Park RM (1992) Software size measurement: a framework for counting source statements. CMU-SEI-92-TR-2. Software Engineering Institute, Pittsburg
57. Paulson JW, Succi G, Eberlein A (2004) An empirical study of open-source and closed-source software products. IEEE Trans Softw Eng 30(4):246–256
58. Plekhanova V (1999) Capability and compatibility measurement in software process improvement. In: 2nd European software measurement conference (FESMA'99), Amsterdam, The Netherlands
59. Pressman R (2004) Software engineering: a practitioner's approach with bonus chapter on agile development. McGraw-Hill Science/Engineering/Math, New York
60. Raja U, Hale DP, Hale JE (2009) Modeling software evolution defects: a time series approach. J Softw Maint Evol Res Pract 21(1):49–71
61. Raymond ES, O'Reilly T (2001) Cathedral and the Bazaar, http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar
62. Roberts JA, Hann I-H, Slaughter S (2006) Understanding the motivations, participation, and performance of open source software developers: a longitudinal study of the apache projects. Manage Sci 52(7):984–999
63. Robles G, Amor JJ, Gonzalez-Barahona JM, Herraiz I (2005) Evolution and growth in large libre software projects. In: Eighth international workshop on principles of software evolution (IWPSE'05). IEEE Computer Society, pp 165–174
64. Sabherwal R, Jeyaraj A, Chowa C (2006) Information system success: individual and organizational determinants. Manage Sci 52(12):1849–1864
65. Scacchi W (2004) Free and open source development practices in the game community. IEEE Softw 21(1):59–66
66. Scacchi W (2004) Understanding free/open source software evolution: applying, breaking and rethinking the laws of software evolution. In: Madhavji NH et al (eds) Software evolution. Wiley, New York
67. Slaughter S, Harter DE, Krishnan MS (1998) Evaluating the cost of software quality. Commun ACM 41(8):67–73
68. Stamelos I, Angelis L, Oikonomou A, Bleris GL (2002) Code quality analysis in open source software development. Inf Syst J 12:43–60
69. Stewart KJ, Ammeter AP (2006) Impacts of license choice and organizational sponsorship on user interest and development activity in open source software projects. Inf Syst Res 17(2): 126–144
70. Subramaniam C, Sen R, Nelson ML (2009) Determinants of open source software project success: a longitudinal study. Decis Support Syst 46(2):576–585
71. Swanson EB, Dans E (2000) System life expectancy and the maintenance effort: exploring their equilibration. MIS Q 24(2): 277–297
72. von Krogh G, Spaeth S, Lakhani KR (2003) Community, joining, and specialization in open source software innovation: a case study. Res Policy 32(7):1217–1241
73. Williams CC, Hollingsworth JK (2005) Automatic mining of source code repositories to improve bug finding techniques. IEEE Trans Softw Eng 31(6):466–480
74. Ying ATT, Murphy GC, Ng R, Chu-Carroll MC (2004) Predicting source code changes by mining change history. IEEE Trans Softw Eng 30(9):574–586
75. Zelkowitz MV, Shaw AC, Gannon JD (1979) Principles of software engineering and design. Prentice Hall Inc, Englewood Cliffs
76. Zimmermann T, Zeller A, Weissgerber P, Diehl S (2005) Mining version histories to guide software changes. IEEE Trans Softw Eng 31(6):429–445