Monitoring the quality of software development and thereby anticipating the defects before release: Examples from the Service Sector

Meer Tauseef Ali[1], Dr .Syed Abdul Sattar[2]

1. Research Scholar,(PP.COMP.SCI.O398)Dept. of Computer Science, Rayalaseema University, Kurnool, A.P.

2. Professor, Principal, Nawab Shah Alam Khan College of Engg. & Tech., Hyderabad.

Abstract: Existing error prediction methods based on change metrics and code must rely on source code and software evaluation information and are not always available. The ability to assess software defects at the beginning of an ongoing project allows the development team to effectively allocate resources and efforts. We assessed the correlation between pre-release defects, funding, integration points and final defect units, i.e., the number of defects. Results at earlier integration points may reflect corresponding defects at later integration points.

A sample size of 86 subsystems and 108 features were analyzed at subsystem and feature granularity at eight integration points. The final results show that the defects found at integration points IP-3 and IP-4 can be used as good predictors of the total number of defects expected in the final version. The correlation of the defects found in the integration points can be examined in order to identify high-risk modules at the beginning of the development cycle.

1. Introduction.

In order to track the quality of a particular product during the development and testing phase, software defects provide a concrete and current indicator that helps maintain quality. It has been observed that in large-scale software development, most defects are found in a small number of subsystems and functions. An early estimate of the number of expected defects in a particular project will not only improve testing efficiency, but also help the team to allocate quality assurance activities and resources to the areas of greatest need. This approach improves the overall quality of the software system under development.

Another theory that needs to be mentioned here is the use of models, i.e. in the case of defect prediction models based on code and change metrics, this use requires access to the source code. However, if the development is done using an outsourcer or if the code is automatically generated by the model, this is model-based development. Another disadvantage of using these models is the uncertainty of what to do if the software under study contains reused code. Models that predict the number of defects at successive integration points in the life cycle of a software project, using only defect data from previous integration points, can magnify many of the defects discussed above.

These models can be used effortlessly and economically to estimate defects at the appropriate level of granularity and timing. Using this approach, project managers can make the right decisions with high quality software. This paper investigates the relationship between defects found during previous integration and defects found during the software project life cycle. The research questions (RQ1 to RQ4) addressed in this paper are

1. Do most of the defects found in the software belong to small modules?

2. Is there a correlation between the errors found in sequential integration?

3. To what extent can faulty input data be used to monitor quality?

4. At what point can defects be predicted before release?

To find a solution to the above research problem, we use correlation and simple regression analysis on the failure data of three large industrial software projects in the service sector, consisting of a total of 86 subsystems and 108 functions at eight integration points.

We find evidence to support the previous observation that most of the defects found in large service impact projects belong to small modules. Our results clearly support the 20-60 rule observed by Fenton and Ohlsson. Regression analysis shows that the number of defects found in the third and fourth integration points can be used as an early indicator to predict the final release defect.

Software can be defined as a problem that occurs as a result of using a software product and leads to its unexpected behaviour. The filtering of which modules are subject to faults and the prediction of the number of faults expected to be found in a software module/project is carried out using software fault prediction methods. Various techniques are used for classification and prediction: expert reports, SRGM regression models and finally ML-based classification and prediction models.

Expert opinion: predicting at project and component level when a bug occurs is a useful approach as it is readily available to experts in such cases, but the disadvantage of this method is that it is a subjective way of predicting software bugs in an ongoing project.

SRGM uses data from defect inflow, development and test phases to model reliability growth in software systems with mathematical equations; SRGM can be used to model reliability growth over the test period and software life cycle using models such as the Rayleigh model for Si can.

Defect prediction by statistical regression using a set of software metrics or code modification attributes as predictors. The method for classifying system modules as prone to defects uses logistic regression, while the method for estimating the expected number of defects for a given software design/model is known as multiple regression. The above methods are based on regression methods. Complexity metrics and source code metrics are common regression-based models in which metrics are used as independent variables in many software processes and products.

Machine learning algorithms are based on statistical methods and data mining concepts used for error classification. There are similarities with regression-based methods, i.e. both use independent variables. The only difference is that ML-based methods are dynamic learning algorithms and their performance tends to improve as more data become available.

Software for the service industry is diverse and complex. The main reason for the complexity is, among other things, the following

Because of the long life of the system, it is expected that the system will often continue to operate without updates. Reliability and trustworthiness are required. States and events are the elements for expected behaviour in real time.

In the case of service domain software, it follows the V model, i.e. the concept of left and right branches. As the system designer designs the system, each assigned number of logical components to implement the required functions.

A combination of technologies and programming languages is used in the development of software for the service sector. The behavioural model includes both inheritance and handwritten code. The production code that is executed can contain both and is automatically generated by the behavioural model. There are challenges in determining accurate and precise complexity metrics. Where should I get the complexity metrics (from the generated code or the behavioral model)? Is it possible to compare the complexity metrics of self-generated and handwritten code?

According to the above, the success of a failed unit at one accumulation point is evaluated as an early prediction of a failed unit at the next accumulation point and as a prediction of all previous failures.

This document makes the following important contributions. Technology for early detection of defects that may require more attention. As a first predictor of the defect count at the next integration point, we assess the functionality of the defect count at a given integration point and evaluate the total defect count predicted for each release. A new approach to defect prediction that does not require code or modification criteria.

2. Related services

Previous empirical studies have supported the Pareto principle of error distribution: according to Fenton and Ohlsson's 20-60 rule, it was observed that around 20% of software modules were responsible for more than 60% of the defects detected in pre-release tests. Similar observations were made in the replication study by Andersson and Runeson and in other studies.

Three major software projects in the service sector strengthen the rule because they provide complementary evidence to support the Pareto principle of defect distribution. More methods are needed to predict the predictable nature of the system without defects, or at a lower granularity. Most predictions are based on expert opinion and evaluation of various SRGMs. wood industrial inflow data [1], since SRGMs show a significant correlation between defects before and after release.

According to Staron and Meding, the model based on operating averages shows good predictability in predicting short-term failures and concludes that model-based prediction is better than prediction based on expert opinion. The regression-based SDP approach proved to be more effective. To classify modules as error-prone or not, Khoshgoftaar and Allen [2] used logistic regression; a study by Zimmermann etal [3] also classified files/packages in an Eclipse project using logistics Regression was also used to classify files/packages in the Eclipse project in a study by Zimmermann et al [3].

To monitor software changes, a multiple regression model was used with a set of software complexity metrics as an independent variable. A model based on a set of code complexity metrics and the number of changes in a given module was studied by Khoshgoftaar [2], using linear regression to predict the dependent variable and to predict program defects.

Weyuker et al [4] compared four abstraction methods, including negative binomial regression, random forest, recursive partitioning and Bayesian additive regression trees for such predictive models and we compared four modeling methods and found Weyuker et al [4].

Weyuker et al [4] compared between four modeling methods using negative regression, binomial regression, random forests, recursive partitioning and Bayesian additive regression trees for such predictive models, and the source that binomial regression and random forests outperform recursive partitioning and Bayesian additive regression. This study complements work based on regression methods using linear regression models.

3. Research methods and data.

The main objective is to assess the relationship between the defects found at different integration points of a complex software design. The case study discussed here is an interpretative study using fixed design principles. It is structured as a study involving the investigation (evaluation) of subsystems and functions. The only objective is to find correlations in the evaluation of the subsystem level and characteristics.

For convenience, subsystems and functions have been used as trap units. We need to find a level at which both the project manager and the quality manager can manage the progress of the project and assess the quality and reliability characteristics of the software system to be developed. In addition, these predictions should help the software engineer to implement correlation measures. The initial level of prediction provides information to these partners. These partners can then use this information to modify their projects.

3.2 Software development process

The AIS project basically follows the V model, with a long-term platform design, divided into several integration points: IP1, IP2,…. IP8, which are divided into a number of integration points. Here, new functionalities are designed, developed, tested, validated and released in the latest system version.

We start with the requirements, and then design for functional implementation at each integration point, and finally at each integration point we rigorously test the newly developed software. Until integration 3-4 we focus on adding functionality and then we shift our attention to system testing and acceptance testing. The main activity here is optimization and all bugs found in software testing are removed as soon as they are found in software updates to maintain the final product.

4. Methods of data collection and analysis

4.1 Collect and analyses data.

In this case, the shortcomings of data collected from three major software development projects for the service sector, i.e. (p1, p2 and p3), AIS deals with the integration of various software functionalities, because these projects are managed through integration, complete hardware and are responsible for the final evaluation of software systems.

As the project consists of several modules, it is developed by different teams and tested within the development team. The integration department requires a separate team to perform integration and acceptance tests. Defects found during integration tests, system tests and acceptance tests are reported to one of the central defect databases managed by the integration department (collection of source data for investigations).

The data used in this study were collected in cooperation with our partners and were generated over the last eight years, so they are complete in all dimensions.

Table 1.1: Percentage of modules with 80% of reported defects and defect distribution summary

Module	Project	N, total number of modules	%age of defects in top 20% modules
Sub-systems	Proj?1	30	75.88%
	Proj?2	20	68.10%
	Proj?3	16	84.96%
	Proj?4	20	90.06%
Features	Proj?1	26	84.90%
	Proj?2	28	79.96%
	Proj?3	30	56.88%
	Proj?4	24	65.90%
Total sub-systems		86	80.92%
Total Features		108	82.86%

Considering the development time, the average project extends over more than two or three years. The main focus in the early stages is on the development of full functionality, while the subsequent stages are dedicated to systems and acceptance testing. If integration 3 & 4 is covered, the project is about 55-60% complete. Conversely, the time dimension between integration points 3 & 4 and pre-release is about 35%-25%, with most of the use of test resources being about one year in a calendar year.

The practical importance of predicting pre-release defect laws through 3 & 4 integration is high because it enables project managers and quality users to effectively allocate and distribute test resources according to the expected needs of the project under development.

4.2 Analysis

This is a brief overview of the analysis to address the questions of this study.

1. Do most of the defects found in the software belong to small modules?

2. Is there a correlation between the errors found in sequential integration?

3. To what extent can faulty input data be used to monitor quality?

4. At what point can defects be predicted before release?

(R1) In order to find the solution to the above problem, it is necessary to assess what percentage of the 20% of subsystems and main features are error-free. In order to obtain a detailed scenario, the analysis has to be carried out both at module and individual project level.

(R2) The next problem is to measure the strength of the relationship between the numbers of defects between the different integration points, after assessing the correlation coefficient between the two variables. In our case, the correlation is considered weak for $r < 0.6$, moderate for $0.6 < r < 0.8$, strong for $0.8 < r < 0.9$, and finally very strong for $r < 0.9$.

Table 1.2: shows regression results for the prediction model

Module	Scope	Sample	Coefficient ( )	Standard Error	Correlation Coefficient, r	Coefficient Of Determination R2
Sub- systems	Top 15	80	0.683	0.038	0.68	0.4624
	Top 10	51	0.674	0.045	0.67	0.4489
	Top 5	26	0.653	0.060	0.65	0.4225
Features	Top 15	88	0.717	0.036	0.72	0.5184
	Top 10	58	0.704	0.045	0.70	0.4900
All sub-systems		204	0.708	0.024	0.71	0.5041
All Features		262	0.753	0.018	0.75	0.5625

Table 1.3: Correlation coefficient between defects found across **IP_1 to IP_8**

Module	Scope	Sample Size	IP_1-IP_2	IP_2-IP_3	IP_3-IP_4	IP_4-IP_5	IP_5-IP_6	IP_6-IP_7	IP_7-IP_8
Sub-systems	Top 15	43	-0.10	0.370	0.905	0.815	0.620	0.680	0.870
	Top 10	19	-0.12	0.355	0.905	0.800	0.580	0.650	0.865
Features	Top 15	32	0.765	0.780	0.795	0.740	0.830	0.910	0.905
	Top 10	22	0.755	0.770	0.785	0.725	0.820	0.905	0.920
All sub-systems		86	-0.06	0.390	0.905	0.840	0.675	0.720	0.880
All Features		108	0.775	0.800	0.825	0.785	0.86	0.920	0.915

To find the dependence, i.e. what proportion of the variance of one variable is predictable from the other, i.e. to find the size of the regression line, we construct a simple linear regression model with a dependent variable (x) and an independent variable.

(R3) To what extent can quality be monitored using faulty inflow data?

For this purpose it is necessary to find a correlation between the defects found and the subsequent integration points. By applying a linear regression model between the defects found at subsequent integration points, where s is the SD for a given coefficient value and X is the number of defects at a given integration point.

If the error count observed at the next integration point is outside the upper or lower limit, this may indicate that the respective module is more prone to errors than expected. Modules whose error count is external should then be checked further by the project manager.

(R4) At what point in the project timeline is it best to predict the correct number of errors? or

When can the number of defects in a project plan be correctly predicted?

The purpose of this study request is to determine the level at which the total number of expected defects will be found before the project is released.

To predict defects before release, we need to find the earliest time of the project using the correlation between the number of defects found at a given integration point and the total number of defects found at lower levels before release. To test the predictive capability of such a prediction model, we must build a model based on a simple linear regression as shown in the equation (1).

Table 1.4: shows pre-release defect count of Correlation matrix & defect count in given integration point

Correlation Coefficient	Sub-systems				Features		
	All	Top 15	Top 10	Top 5	All	Top 15	Top 10
IP_1	0.343	0.298	0.265	0.320	0.444	0.399	0.374
IP_2	0.505	0.478	0.458	0.449	0.716	0.683	0.667
IP_3	**0.830**	**0.821**	**0.813**	**0.847**	**0.843**	**0.820**	**0.814**
IP_4	**0.927**	**0.922**	**0.919**	**0.922**	**0.897**	**0.877**	**0.875**
IP_5	0.884	0.861	0.845	0.801	0.826	0.782	0.765
IP_6	0.690	0.629	0.578	0.417	0.721	0.651	0.621
IP_7	0.497	0.423	0.361	0.173	0.688	0.616	0.594
IP_8	0.395	0.333	0.284	0.148	0.657	0.593	0.590
Pre_Rel	1	1	1	1	1	1	1

Table 1.5: shows Correlation matrix for cumulative defect count

Correlation Coefficient	Sub-systems				Features		
	All	Top 15	Top 10	Top 5	All	Top 15	Top 10
IP_1	0.410	0.359	0.323	0.369	0.428	0.381	0.349
IP_2	0.597	0.559	0.533	0.503	0.681	0.642	0.613
IP_3	0.853	0.838	0.825	0.848	0.825	0.799	0.786
IP_4	0.923	0.916	0.910	0.914	0.898	0.881	0.877
IP_5	0.950	0.944	0.939	0.938	0.946	0.935	0.934
IP_6	0.971	0.966	0.963	0.958	0.974	0.968	0.968
IP_7	0.987	0.984	0.982	0.979	0.991	0.989	0.989
IP_8	0.997	0.996	0.996	0.995	0.997	0.997	0.997
Pre_Rel	1	1	1	1	1	1	1

To access the accuracy of the forecast, we use the relative absolute error. Therefore, prior to publication, a final sample of 86 subsystems and 106 characteristics and data for eight integration points was available.

Sample size observations are analyzed for all subsystems and features in queries R2, R3 and R4. A smaller subset, consisting of the top 15, top 10 and top 5 systems and the top 15 and top 10 for features, may be better correlated with a smaller subset.

The above ideas are very important in this study because they are important for large projects and systems under development and managers need information to manage and control them. Predicting the number of defects in a subordinate system can be very useful to optimize the allocation of testing resources and, in addition, early intervention in design and implementation can provide the best conditions for high quality performance.

5 Results.

For defect prediction, it is necessary to construct a simple regression model. This model can be used after assessing the robustness of the ratio of the number of defects between the integration points. In the following, the distribution of the number of defects in the eight integration points of the three projects is presented and finally the results are organized according to the four hypotheses tested in this study.

Reference:

[1] A. Wood, "Predicting software reliability," Computer, vol. 29, no. 11, pp. 69– 77, 1996.

[2] T. M. Khoshgoftaar and E. B. Allen, "Logistic regression modeling of software quality," International Journal of Reliability, Quality and Safety Engineering, vol. 6, no. 04, pp. 303–317, 1999.

[3] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007. International Workshop on, 2007, pp. 9–9.

[4] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Where the bugs are," in ACM SIGSOFT Software Engineering Notes, 2004, vol. 29, pp. 86–96.

1. 

Meer Tauseef Ali,
Research Scholar,
(PP.COMP.SCI.O398)
Dept. of Computer Science, Rayalaseema University, Kurnool, A.P.

2. 

Dr .Syed Abdul Sattar,
Professor, Principal,
Nawab Shah Alam Khan College of Engg. & Tech., Hyderabad.