A Systematic Approach to Predict System Testing Defects using Prior Phases Metrics for V-Model

Muhammad Dhiauddin Mohamed Suffian[a,], Suhaimi Ibrahim[b]

[a]*Faculty of Computing, Universiti Teknologi Malaysia (UTM), 81310 Skudai, Johor, Malaysia*
[b]*Advanced Informatics School, Universiti Teknologi Malaysia (UTM), 54100 Kuala Lumpur, Malaysia*

Abstract

In order to prevent more defects from escaping to end-users for a V-model development process, independent testing team needs a predicted total number of defects for any software under test at the start of system testing so that defects can be fixed as early as possible. Metrics from requirement to coding phases are required to develop this defect prediction. Thus, this research introduces and explains the systematic approach to predict system testing defects for a V-model by using prior phases metrics. By applying regression analysis as part of the approach, it demonstrates that total number of defects in system testing can be predicted by using requirement, design and coding metrics. The approach produces a mathematical equation which is used to predict defects in system testing. The equation is then verified on new software projects so that it is fit for final implementation and integration into software development process.

Keywords: SYSTEM TESTING, DEFECTS, DEFECT PREDICTION, PRIOR PHASES METRICS, V-MODEL, REGRESSION ANALYSIS

1. Introduction

One of the traditional and well-known process models in software engineering is Waterfall model. This model arranges the software development phases and activities in linear sequence [1]. Concept definition, requirement, design, code, testing and maintenance phases are the core phases involved in Waterfall model [2]. Realizing that early testing is crucial in software development, Waterfall model is extended by incorporating the element of early testing activities and later known as V-model [3]. Rigor verification and validation activities throughout the development phases are the main emphasis of early testing before the software is released to end-users. Verification activities in V-model may involve but not limited to following activities: requirement reviews, design reviews, code inspection, unit testing, integration testing as well as system testing. All activities share the same objectives that are to detect defects and fix them as early as possible in the life cycle. In particular, when the software has

* Corresponding author. Tel.: +60-019-6432654
E-mail address: mdhiauddin2@live.utm.my.

completed coding phase and passed the unit-integration test, it will undergo system testing to validate that the software under test meets system and user requirements.

Specifically in V-model, system testing is carried out by an independent testing team in discovering defects, both functional and non-functional defects as well as ensuring that the software system conforms to the specified user and system requirements. However, the team has to face several challenges in completing the test execution activities within the deadline while making sure all possible defects has been discovered and sent to development for fixing before finally releasing to the recipients. Realizing that it is impossible to have zero defects for software, minimally it is reasonable to have zero known-defects with the understanding that whatever defects that have been found and fixed during system testing should not be re-introduced at the user's site, provided that environment during testing phase and deployment is identical. Thus, early indicator of total defects to be found in system testing is required to serve as a guide or reference for the testing team to execute the test. The ideal scenario is to have this prediction before the system testing starts.

From process improvement perspective particularly on testing, by being able to predict defects for system testing, testing team would be able to contain as many defects as possible within the same phase thus prevent them from escaping to deployment environment. Strategically, testing team can allocate appropriate number of test engineers across multiple test projects and schedule suitable number of days for test execution when prediction of defects is in place. This will improve the management of test execution so that it can be completed within time frame. In terms of test strategy, test engineers could adopt more effective techniques and types of testing to ensure the defects found are significantly close to the prediction. This is because it is impossible to have 100% accurate prediction of defects. Thus, the chances of finding defects in the software under test are higher when the test coverage is high. This will lead to the production of higher quality of software.

For this reason, a systematic approach needs to be developed for predicting system testing defects by considering metrics from prior phases, which include requirement, design, and construction or coding phases. Besides that, testing-related metrics from test activities which go in parallel with those phases, namely test requirement, test planning, and test design are also considered. These metrics are analyzed to determine the ones that can be the suitable predictors for defects.

This research is organized into Section 2 describing related works, Section 3 discussing on the proposed approach for predicting defects in system testing using prior phases metrics, Section 4 explaining the case study while Section 5 concludes the research findings together with the recommendation of future works for improving the approach.

2. Related Work

Numerous works have been done with regard to the area of defect prediction. However, not many of them really focus on developing suitable approach to specifically predict defects for system testing. Works done by [4] has demonstrated that by using

cyclomatic complexity and lines of code (LOC), defects in software can be predicted. Studies by [5] showed that project management, process improvement or work product assessment can be used to predict defects. Product and project metrics that include metrics from review, code testing, code peer review, product release usage and defect validation were used in regression analysis done by [6] to predict defects. [7] used mathematical distributions to predict defects in software. [8] introduced Defect Type Model (DTM) based on Bayesian Network by using defect severity as the method for predicting defects while [9] applied multivariate linear regression for defect inflow prediction. On the other hand, Six Sigma methodology was adopted by [10] as approach to predict defects via Quality Function Deployment (QFD) and transfer function. In another works, [11] used COQUALMO to predict defects density in software to improve software quality. This was based on defect identification and defect removal processes.

Those prior works have addressed different areas of software defect prediction. Works by [12] concentrated on the area of remaining defects while testing is on-going while [13] was more concerned about defects in email and web for open source software. Defect could also be predicted for each phase in the software life cycle by using Rayleigh model [14]. [15] and [16] portrayed similar approach of defect prediction with this research via the application of Bemar model and CDM model, respectively.

There were various predictors used and manipulated to predict defects. Defects in software can be predicted by using object-oriented metrics [13] or developer metrics comprising number of developers who made modification prior to release, during the release and all releases [17]. Metrics from historical data could become good predictors and this was used by [18] to predict defects. A model was developed by using detailed requirements and defect potentials derived for each development phase based on historical data [19]. Besides that, by using just three software metrics, effective software defect prediction model can be built with [20].

Every approach of defect prediction needs to be measured for its validity and fitness for use. [21] suggested that the prediction success can be measured by using percent of faults found. Alternatively, a successful prediction can be evaluated in terms of its suitability in assisting in future maintenance of the software [22]. However, a good benchmark for the method and result of the prediction can only be achieved when huge data sets are in place [23].

Recent studies on defect prediction have touched on several areas of concern. [24] grouped the data used for selecting the predictors into quantitative and qualitative data, which could be mapped to the term product and process metrics used in this research. The qualitative one was taken from COQUALMO while the quantitative data consists of software size, team size, test cases and effort. However, the focus was more towards

defects in early stage of development life cycle.

Significant effort has been made to cater for defect prediction in various nature and context of software. A framework for defect prediction in specific context of software was introduced by dividing it into phases: preparation, model creation and model usage [25]. This shall serve as strong guideline in formulating reliable defect prediction model for software. In relation to this, the term "Defect Prediction 2.0" was coined as an emphasis that defect prediction totally work by incorporating finer granularity of metrics, mechanism to deal with noise in the prediction as well as tackling new 'customers' for the prediction [26].

Therefore, this research specifically addresses the development of introducing a systematic approach to predict system testing defects by using metrics in prior phases for any software under test adopting V-model development process.

3. Findings and Discussion

There are many ways of representing V-model development process together with corresponding testing activities. A typical representation of V-model can be seen in Figure 1 below [3]:

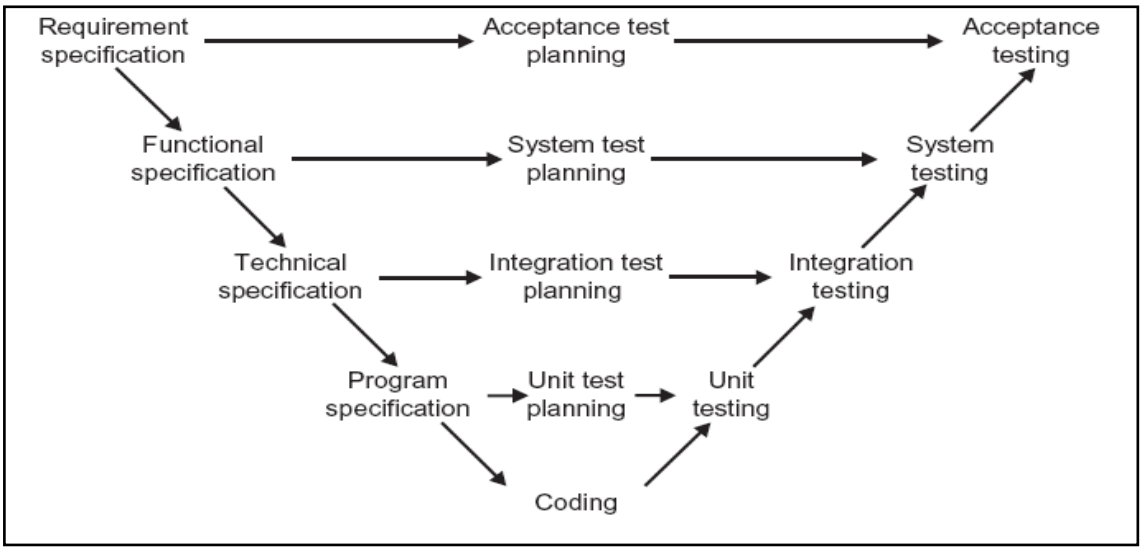

Fig. 1. V-model development process.

Although corresponding test levels for every development phase are indicated in Figure 1, there is no indication on how other kinds of verification activities are mapped into the process. Thus, the diagram above has been revisited and revised by incorporating the activities as well as the area of prediction with regard to V-model. The diagram is presented in Figure 2 below:

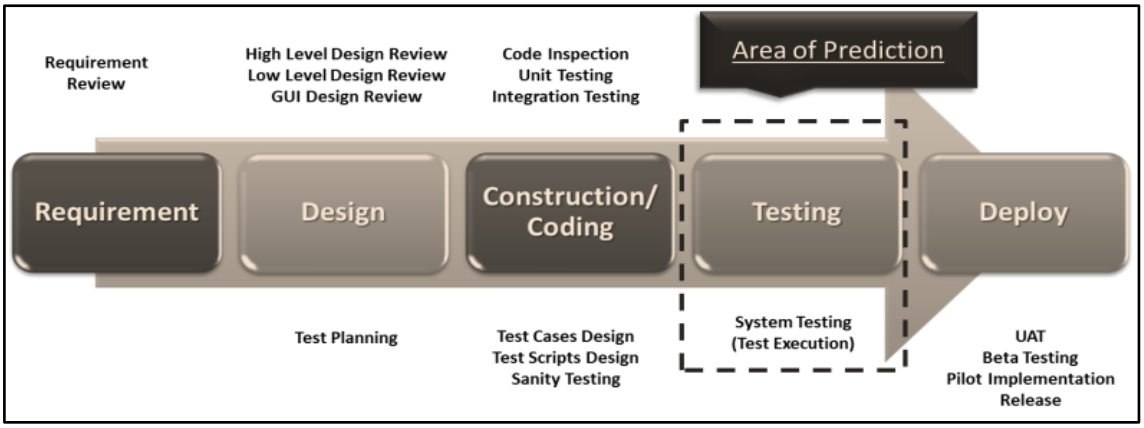

Fig. 2. Verification and validation activities in V-model development process.

Based on Figure 2, the verification and validation (V&V) for V-model could be grouped into two: development-related and test-related activities. Development-related activities may consist of requirement review, high level design review, low level design review, Graphical User Interface (GUI) design review, code inspection, unit test and integration test. On the other hand, test planning, test cases design, test scripts design, sanity test and system testing or system test execution can be grouped into test-related activities. Any V&V activity in deploy phase is not touched in this research as the phase comes after system testing and does not fall in the area of interest. Metrics is used to track and measure each activity. These metrics are categorized into size-related, defects-related and effort-related metrics. The details of each phase, activities and responsibilities are explained below in Table 1:

Table 1: Software development phases and its corresponding activities

Phase	Activity	Responsibilities
Requirement	Requirement analysis and development	Developer
	Requirement review	Developer
Design	Design development	Developer
	Design review (high-level design, low-level design, database design, GUI design)	Developer
	Test plan development	Tester
	Test plan review	Tester
Construction/Coding	Coding	Developer
	Code inspection	Developer
	Unit testing	Developer
	Integration testing	Developer
	Test cases development	Tester
	Test cases review	Tester
Testing	System testing	Tester

Each activity mentioned above is associated with respective metrics which will be discussed later. Thus, metrics collected from requirement to construction/coding phases are used and analyzed to determine which of them could become the significant predictors for defects in system testing. Based on this understanding, the proposed approach is developed by taking into account metrics from requirements to construction phases as independent variables while defects in system testing are treated as dependent variable. This forms the basis of this research, which is to come out with a systematic approach that could predict system testing defects by using prior phases' metrics as presented in Figure 3.

In Figure 3, the metrics collected from the activities in Table 1 are stored in its own logical repository. All repositories form the master repository of product and process metrics. These repositories become the main source for developing the approach for predicting system testing defects. Metrics from the master repository are extracted for analysis so that only accurate data are used for statistical analysis later. This covers metrics data on type of software projects, methodology, effort, defects as well as any other data related to process and product metrics. Data that has been filtered is then selected and put into statistical analysis. As specified earlier, interaction between metrics in prior phases as independent variables and defect metrics in system testing phase as dependent variable can be observed via statistical analysis.

The results produced by the statistical analysis are in the form of several mathematical equations, which later being verified and evaluated by applying them into new projects which are yet entering system testing phase. For each equation applied into new projects, comparison is done between predicted defects and the actual defects. The equation that produces the most significant result of prediction is selected as the final prediction for system testing defects which later incorporated into the whole software development process for actual implementation. In this research context, statistical analysis technique chosen is regression analysis as for demonstrating the suitability of the whole approach.

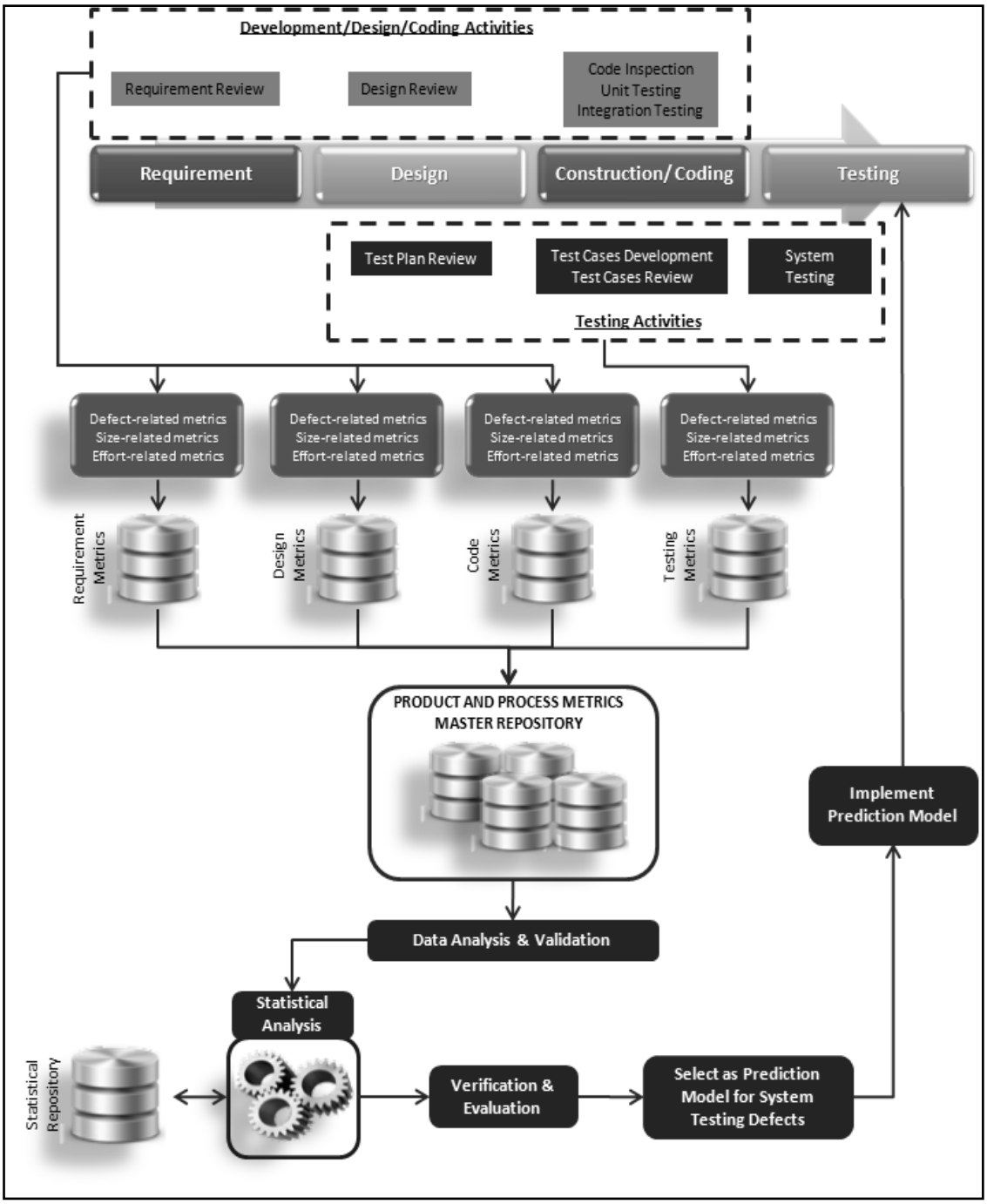

Fig. 3. Proposed approach for predicting system testing defects using prior phases metrics

Figure 4 outlines the statistical analysis process as depicted in Figure 3. The explanation of the steps presented in Figure 4:

- It starts with identifying the metrics to be collected from phases prior to system testing that involve development-related activities and testing-related activities as explained previously. The metrics are treated as the independent and dependent variables.
- Once the right metrics have been identified, they are collected from both development and testing repositories, which also include metrics on historical defects.
- The metrics are filtered and validated to ensure only correct and accurate metrics data are used.
- The validated metrics are then used interchangeably in regression analysis. Interchangeably here means dependent variable and independent variables are interchanged between each other to observe the interaction between them and to see which set of interactions produces the best prediction equation.
- The acceptance criteria for any mathematical equation to be considered as prediction equation candidate I is based on R-squared and R-squared (adjusted) values of at least 90% as well as P-value of less than 0.05. If the prediction equation achieve the values of equal or more than 90%, for both R-squared and R-squared (adjusted) while then P-value for each metric (predictor) used is less than 0.05, then that equation could be selected as the right candidate for defect prediction equation.
- If both R-squared and R-squared (adjusted) values are less than 90%, then the selection of metrics (predictors) need to revised and put into regression analysis again. Same process takes place if P-value of each predictor is more than 0.05.

The candidate (s) for prediction equation is applied into new testing projects. During the verification of the equation, comparison is made between the predicted defects before testing starts against the total number of defects found after test execution completed. The equation is selected as the final prediction model equation for system testing defects when the actual defects found fall within the 95% of Prediction Interval (PI). If the defects found fall out of the PI range, then the metrics need to be revisited and revised following the same process.

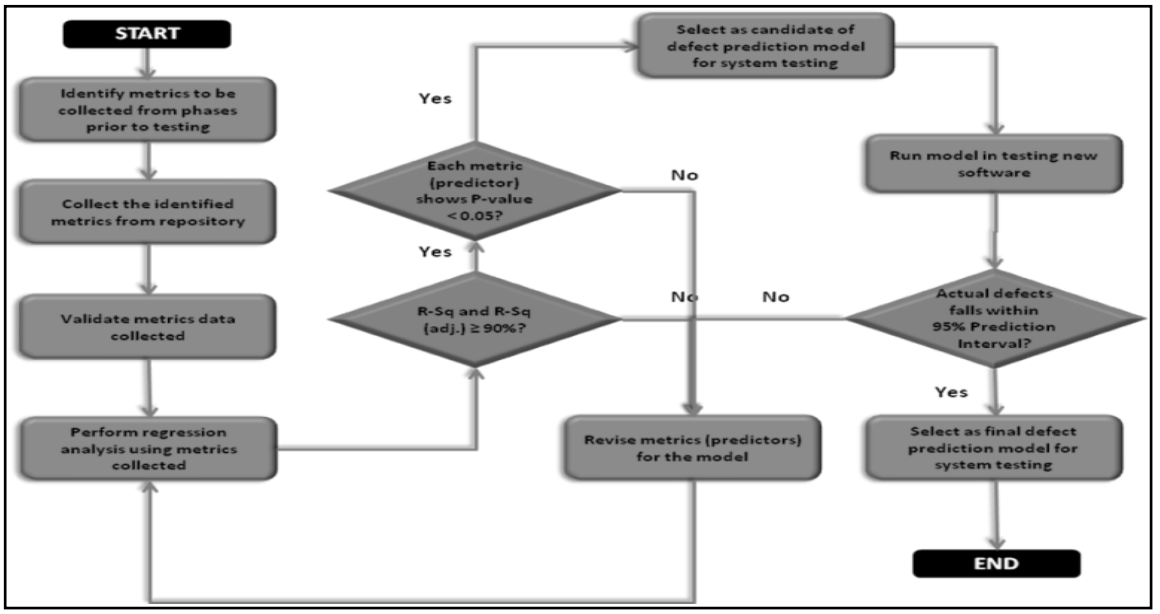

Fig. 4. Regression analysis process.

The final prediction model is then integrated back into the software development process as part of continuous improvement activities. As the prediction could be higher or lower than the actual defects found, it is subjected for further refinement. Figure 5 depicts how the final prediction is implemented in the overall V-model process.

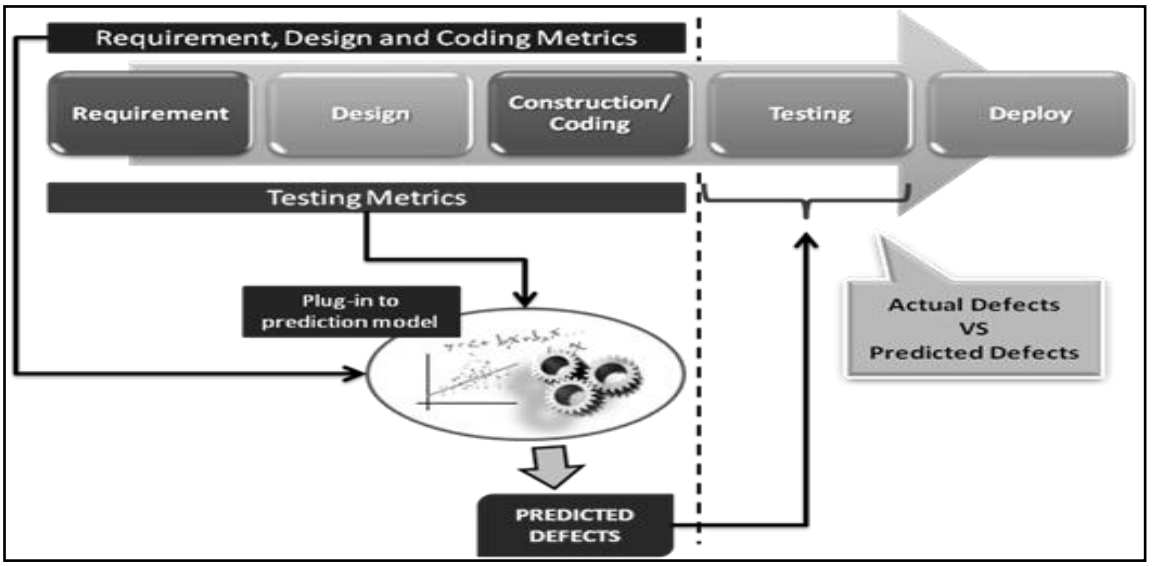

Fig. 5. Implementation of defect prediction for system testing.

Next section presents the case study that demonstrates the suitability of the proposed approach in predicting defects for system testing in V-model.

4. Case Study

Several software projects that have completed testing are selected. The nature of the projects comprise of web-based, standalone application and web service. This involves various programming languages including PHP, Java and .NET. The list of metrics collected from these project's repositories for further analysis is listed below:

- Number of requirement pages
- Number of design pages
- Code size in a form of kilo lines of code (KLOC)
- Total test cases
- Total effort in test case design
- Total effort in phases prior to system testing
- Requirement error
- Design error
- Code and unit testing (CUT) error
- Test cases error
- Total defects logged in a form of all defects and functional defects
-

Data for that metrics are collected and validated for accuracy. After validation, the following set of data as in Table 2 is obtained and later used for regression analysis:

Table 2: Data set used

	Req. Error	Design Error	Coding Error	KLOC	Req. Page	Design Page	Total Test Cases	Test Cases Error	Total Effort	Test Design Effort	Funct. Defects	All Defects
Project A	5.00	22.00	12.00	28.80	81.00	121.00	224.00	34.00	16.79	15.20	19.00	19.00
Project B	0.00	0.00	1.00	6.80	171.00	14.00	17.00	6.00	45.69	40.91	1.00	1.00
Project C	9.00	10.00	14.00	5.40	23.00	42.00	24.00	6.00	13.44	13.44	4.00	4.00
Project D	7.00	12.00	2.00	1.10	23.00	42.00	25.00	9.00	4.90	4.90	0.00	0.00
Project E	11.00	29.00	3.00	1.20	23.00	54.00	28.00	12.00	4.72	4.59	3.00	3.00
Project F	0.00	2.00	7.00	6.80	20.00	70.00	66.00	7.00	32.69	16.00	16.00	27.00
Project G	3.00	25.00	11.00	4.00	38.00	131.00	149.00	0.00	64.00	53.50	3.00	3.00

Project H	4.00	9.00	2.00	0.20	26.00	26.00	24.00	0.00	5.63	5.63	0.00	0.00
Project K	17.00	0.00	3.00	1.40	15.00	28.00	13.00	4.00	9.13	7.88	1.00	1.00
Project N	61.00	34.00	24.00	36.00	57.00	156.00	306.00	16.00	89.42	76.16	25.00	28.00
Project O	32.00	16.00	19.00	12.30	162.00	384.00	142.00	0.00	7.00	7.00	12.00	12.00
Project P	0.00	2.00	3.00	3.80	35.00	33.00	40.00	3.00	8.86	8.86	6.00	6.00
Project Q	15.00	18.00	10.00	26.10	88.00	211.00	151.00	22.00	30.99	28.61	39.00	57.00
Project R	0.00	4.00	0.00	24.20	102.00	11.00	157.00	0.00	41.13	28.13	20.00	33.00

Different sets of regression analysis have been carried out to get the most significant result. In the case of dependent variable, defects are interchanged between "All Defects" and "Functional Defects", while for independent variables, "EffortDays" and "EffortTestDesign" are interchanged accordingly. As a note, "EffortDays" refers to metrics on the effort spent by testers in all phases prior to test execution while "EffortTestDesign" refers to metric spent by testers in creating test cases during test design. Figure 6, Figure 7, Figure 8 and Figure 9 present the results of regression analysis. The details of each regression set are explained below:

Figure 6
- Independent variables (predictors): Requirement error, CUT error, KLOC, requirement pages, design pages, total test cases, effort days
- Dependent variable (target): Functional defects

Figure 7
- Independent variables (predictors): Requirement error, CUT error, KLOC, requirement pages, design pages, total test cases, effort days
- Dependent variable (target): All defects

Figure 8
- Independent variables (predictors): Requirement error, CUT error, KLOC, requirement pages, design pages, total test cases, effort test design
- Dependent variable: Functional defects

Figure 9
- Independent variables (predictors): Requirement error, CUT error, KLOC, requirement pages, design pages, total test cases, effort test design
- Dependent variable: All defects

```
The regression equation is
Functional_Defects = 4.00 - 0.204 Req Error - 0.631 CUT error + 1.90 KLOC
                     - 0.140 ReqPage + 0.125 DesPage - 0.169 TotalTC
                     + 0.221 EffortDays


Predictor       Coef  SE Coef      T      P
Constant      3.9969   0.9435   4.24  0.005
Req Error   -0.20389  0.05001  -4.08  0.007
CUT error    -0.6309   0.1566  -4.03  0.007
KLOC          1.9050   0.1249  15.25  0.000
ReqPage     -0.14043  0.01515  -9.27  0.000
DesPage      0.12459  0.01151  10.82  0.000
TotalTC     -0.16911  0.02055  -8.23  0.000
EffortDays   0.22127  0.03439   6.43  0.001


S = 1.80987   R-Sq = 98.9%   R-Sq(adj) = 97.6%
```

Fig. 6. Result A of regression analysis

```
The regression equation is
All Defects = 7.20 - 0.374 Req Error - 1.37 CUT error + 3.07 KLOC
              - 0.245 ReqPage + 0.217 DesPage - 0.310 TotalTC + 0.449 EffortDays


Predictor       Coef  SE Coef      T      P
Constant       7.203    1.958   3.68  0.010
Req Error    -0.3744   0.1038  -3.61  0.011
CUT error    -1.3689   0.3250  -4.21  0.006
KLOC          3.0732   0.2593  11.85  0.000
ReqPage     -0.24534  0.03143  -7.81  0.000
DesPage      0.21707  0.02389   9.08  0.000
TotalTC     -0.30951  0.04263  -7.26  0.000
EffortDays   0.44933  0.07137   6.30  0.001


S = 3.75562   R-Sq = 97.7%   R-Sq(adj) = 95.1%
```

Fig. 7. Result B of regression analysis

```
The regression equation is
Functional_Defects = 4.37 - 0.222 Req Error - 0.658 CUT error + 1.89 KLOC
                     - 0.144 ReqPage + 0.124 DesPage - 0.161 TotalTC
                     + 0.254 EffortTestDesign


Predictor             Coef  SE Coef      T      P
Constant             4.367    1.352   3.23  0.018
Req Error          -0.22167  0.07337  -3.02  0.023
CUT error           -0.6585   0.2284  -2.88  0.028
KLOC                 1.8915   0.1814  10.43  0.000
ReqPage            -0.14358  0.02269  -6.33  0.001
DesPage             0.12364  0.01702   7.26  0.000
TotalTC            -0.16110  0.02936  -5.49  0.002
EffortTestDesign    0.25354  0.06172   4.11  0.006


S = 2.60493   R-Sq = 97.7%   R-Sq(adj) = 95.1%
```

Fig. 8. Result C of regression analysis

```
The regression equation is
All Defects = 7.98 - 0.403 Req Error - 1.40 CUT error + 3.01 KLOC
              - 0.247 ReqPage + 0.211 DesPage - 0.286 TotalTC
              + 0.491 EffortTestDesign


Predictor             Coef  SE Coef      T      P
Constant             7.983    3.097   2.58  0.042
Req Error          -0.4026   0.1681  -2.39  0.054
CUT error          -1.4000   0.5234  -2.67  0.037
KLOC                3.0148   0.4156   7.25  0.000
ReqPage            -0.24701  0.05198  -4.75  0.003
DesPage             0.21086  0.03900   5.41  0.002
TotalTC            -0.28641  0.06727  -4.26  0.005
EffortTestDesign    0.4913   0.1414   3.47  0.013


S = 5.96836   R-Sq = 94.3%   R-Sq(adj) = 87.6%
```

Fig. 9. Result D of regression analysis

The rationale behind subtituting "AllDefects" and "Functional Defects" as dependent variable is to observe whether the model equation could effectively predict all defects or functional defects in. The same thing goes for the independent variables when substituting "EffortDays" and "EffortTestDesign". This is to examine which of the two metrics could become the most significant predictor for system testing defects, as part of other independent variables used in generating the equation for predicting the defects.

The results for regression analysis demonstrate that all prediction model equations have both R-Squared and R-Squared (adjusted) values of more than 90%. Besides that, P-value of each predictor or independent variable in every equation is less than 0.05. Thus, no decision can be made on which equation should be selected as the final prediction model for system testing defects. For this reason, all four equations are verified by applying them three new projects that have yet entered system testing phase. This is to determine which equation could closely predict total defects in system testing. The results of verification are shown in Table 3 below:

Table 3. Verification results

Target	Effort Predictors	Project	Predicted Defects	Actual Defects	95% PI (min, max)
Functional Defects	All Tester Effort Prior to System Testing	Project 1	182	187	(155, 210)
		Project 2	6	1	(0, 14)
		Project 3	1	1	(0, 6)
All Defects	All Tester Effort Prior to System Testing	Project 1	298	230	(241, 356)
		Project 2	9	9	(0, 24)
		Project 3	2	1	(0, 12)
Functional Defects	All Tester Effort In Test Design Prior to System Testing	Project 1	183	187	(201, 392)
		Project 2	8	1	(0, 19)
		Project 3	2	1	(0, 9)
All Defects	All Tester Effort in Test Design Prior to System Testing	Project 1	296	230	(142, 225)
		Project 2	11	9	(0, 37)
		Project 3	3	1	(0, 19)

Prediction Interval column represented as 95% PI (min, max) describes the minimum and maximum number of predicted total defects for the project for particular equation used. For example, for predicting defects in Project 1, the range of prediction should fall between 155 to 210, in which "EffortDays" is used as one of the predictors and "FunctionalDefects" is used as dependent variable. After looking at all prediction results, it is apparent that first equation produces the most significant result since the prediction falls very close to the specified 95% PI. In other words by applying that equation, requirement error, coding error, kilo lines of code (KLOC), requirement page, design page, total test cases as well as total effort days spent by test engineers are the most significant metrics from prior phases that serve as predictors to predict system testing defects in V-model. The case study also shows that due to several limitations such as size of data set, number of projects and type of metrics collected, the proposed

approach could produce mathematical equation that can only predict functional defects. Therefore, more things need to be put in place in terms of improving the prediction so that in the future, this approach could predict different type of defects as well as dynamically produce prediction model equation for different nature and type of software.

5. Conclusion

This research has successfully established a systematic approach to predict system testing defects by exploiting prior phases metrics, specifically for V-model development process. This is achieved by adopting statistical analysis via regression analysis technique as part of the proposed approach, which is able to serve as great tool in determining right predictors for testing defects and measuring accuracy of the prediction. It is also recognized that it is a challenge to have an absolute prediction of defects in system testing since there are many other aspects that need to be considered. Therefore, having a maximum and minimum range for defect prediction can provide sufficient room for the independent testing team to have a control plan on what to do should the prediction does not fall within the specified range.

Moving forward, it is essential to consider more metrics to construct the prediction, particularly product-related metrics as well as the level of granularity of the metrics. Furthermore, the improved version of the prediction should take into account the capability of the approach to formulate prediction model that could forecast non-functional defects as well as severity of the defects. Importantly, the prediction in this research is treated as generalized solution and does not consider prediction of defects for specific type and nature of software. Therefore, any future work is needed to strengthen the approach by being able to have specific prediction for different types of software, so that prediction of defects, particularly for system testing is more useful and practical.

Acknowledgements

The authors wish to thank to the representatives from the R&D organization that provided the data and input for making the research a successful one.

References

[1] Jalote, P.A Concise Introduction to Software Engineering. Springer, 1st ed. New York, 2008.

[2] Laplante, P.A. What Every Engineer Should Know About Software Engineering. Taylor & Francis Group. Boca Raton, FL, 2007.

[3] Hambling, B. Software Testing: An ISTQB-ISEB Foundation Guide, Second Edition. British Information Society Limited. North Star Avenue, Swindon, 2010.

[4] Fenton, N.E. and Neil, M. A Critique of Software Defect Prediction Models. IEEE Transactions On Software Engineering. vol. 25 (5), pp.675-689, 1999.

[5] Clark, B. and Zubrow,D. How Good is the Software: A Review of Defect Prediction Techniques. Software Engineering Symposium. Carnegie Mellon University, 2001.

[6] Wahyudin, D., Schatten, A., Winkler, D., Tjoa, A.M. and Biffl, S. Defect Prediction using Combined Product and Project Metrics: A Case Study from the Open Source "Apache" MyFaces Project Family. In Proceedings of Software Engineering and Advanced Applications (SEAA '08), 34th Euromicro Conference, pp. 207-215, 2008.

[7] Sinovcic, I. and Hribar, L. How to Improve Software Development Process using Mathematical Models for Quality Prediction and Element of Six Sigma Methodology. In Proceedings of the 33rd International Conventionions 2010 (MIPRO 2010), pp. 388-395, 2010.

[8] RadliRski, L. Predicting Defect Type in Software Projects. Polish Journal of Environmental Studies, vol.18 (3B), pp. 311-315, 2009.

[9] Staron, M. and Meding, W. Defect Inflow Prediction in Large Software Projects. e-Informatica Software Engineering Journal, vol. 4 (1), pp. 1-23, 2010.

[10] Fehlmann, T. Defect Density Prediction with Six Sigma. Presentation in Software Measurement European Forum, 2009.

[11] Mittal, A. and Dubey, S.K. Defect Handling in Software Metrics. International Journal of Advanced Research in Computer and Communication Engineering, vol. 1(3), pp. 167-170, 2012.

[12] Haider, S.W. , Cangussu, J.W. , Cooper, K.M.L. and Dantu, R. Estimation of Defects Based on Defect Decay Model: ED3M. IEEE Transactions on Software Engineering, vol. 34 (3), pp. 336-356, 2008.

[13] Gyimothy, T., Ferenc, R. and Siket, I. Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction. IEEE Transactions on Software Engineering, vol. 31 (10), pp. 897-910, 2005.

[14] Thangarajan, M. and Biswas, B. Software Reliability Prediction Model. Tata Elxsi Whitepaper,2002.

[15] Bertolino, A. and Marchetti, E. A Simple Model to Predict How Many More Failures will appear in Testing. Proceedings of Quality Week Europe. Brussel, Belgium. 1998.

[16] Karcich, R.M., Cangussu, J.W. and Earl, A. System Testing Process Behavior Prediction at Sun Microsystem. 14th International Symposium on Software Reliability Engineering 2003 (ISSRE 2003).

[17] Weyuker, E.J. , Ostrand, T.J. and Bell, R.M. Using Developer Information as a Factor for Fault Prediction. In Proceedings of the Third International Workshop on Predictor Models in Software Engineering (PROMISE'07), pp.8, 2007.

[18] Zimmermann, T., Nagappan, N. and Zeller, A. Predicting Bugs from History. In Software Evolution (Software Evolution), pp. 69-88, February 2008.

[19] Zawadzki, L. and Orlova, T. Building and Using a Defect Prediction Model. Presentation in Chicago Software Process Improvement Network, 2012.

[20] Wang, H., Khoshgoftaar, T.M. and Seliya, N. How Many Software Metrics Should be Selected for Defect Prediction?. Proceedings of the Twenty-Fourth International Florida Artificial Intelligence Research Society Conference, pp. 69-74, 2011.

[21] Ostrand, T.J. and Weyuker, E.J. How to Measure Success of Fault Prediction Models. In Proceedings of Fourth International Workshop on Software Quality Assurance, 2007 (SOQUA ’07), pp. 25-30, 2007

[22] Li, L.P. , Shaw, M. and Herbsleb, J. Selecting a Defect Prediction Model for Maintenance Resource Planning and Software Insurance. In Proceedings of 5th Workshop on Economics-Driven Software Engineering Research (EDSER '03), pp. 32-37, 2003.

[23] D’aq, M., Lanza, M. and Robbes, R. Evaluating Defect Prediction Approaches: A Benchmark and an Extensive Comparison. Journal of Empirical Software Engineering, vol. 17, no. 4-5, pp. 531-577, 2012

[24] Wang, D., Wang, Q., Hong, Z., Chen, X., Zhang, L. and Yang, Y. Incorporating Qualitative and Quantitative Factors for Software Defect Prediction. In Proceedings of the 2nd International Workshop on Evidential Assessment of Software Technologies (EAST’12), pp. 61-66, 2012.

[25] Wahyudin, D., Ramler, R. and Biffl, S. A Framework for Defect Prediction in Specific Software Project Contexts. Book Section, Software Engineering Techniques, Lecture Notes in Computer Science, vo. 4980, pp. 261-274, 2011.

[26] Kim, S. Defect, Defect, Defect: Defect Prediction 2.0. In Proceedings of the 8th International Conference on Predictive Models in Software Engineering (PROMISE’12), pp. 1-2, 2012.