SOFTWARE RELIABILITY PREDICTION MODEL USING RAYLEIGH FUNCTION

Ana Maria VLADU[1]

Predicția fiabilității produselor software poate determina fiabilitatea prezentă a unui produs, folosind tehnici statistice bazate pe datele de eşec, obținute în timpul testării sau a folosirii sistemului. Scopul acestei lucrări este să studieze evoluția unui produs real de-a lungul a trei variante, folosind funcția Rayleigh pentru a prezice numărul de defecte. Articolul oferă două posibilități pentru calculul parametrilor modelului, iar apoi vom putea decide care model este mai bun şi cum poate fi îmbunătățit. Rezultatele acestui studiu vor fi folosite pentru a stabili cea mai bună abordare. Modelul propus a fost folosit în diferite variante ale aceluiaşi produs, având complexități diferite şi întinzându-se pe perioade diferite de timp.

The prediction of software reliability can determine the current reliability of a product, using statistical techniques based on the failures data, obtained during testing or system usability. The purpose of this paper is to study the evolution of a real-life product over three releases, using the Rayleigh function in order to predict the number of defects. Our paper offers two possibilities for computing the model parameters, and then we should be able to decide which is better and what can be improved. Results from this study will be used to determine which approach is best to be used. The model proposed was used on different releases of the same product, having various complexities in features and expanding over different periods of time.

Keywords: Software reliability, testing, reliability models, defect prediction, defect estimation

1. Introduction

The prediction of software reliability is important to the project management and release management in order to support decision making for the product releases. This method helps understand the current quality of the product, whether or not it can still be tested, and can be used as input for planning the next release. Pham [1] has created an early prediction model, based on lifecycle phases, which divides the development cycle in different phases. This model assumes that the found defects in different phases follow the Rayleigh function, normalized by the number of code lines. It follows the defect statistics over early development stage, in order to predict the defect density in later stages.

[1] PhD. Student, Eng., Dept. of Automatic Control, University POLITEHNICA of Bucharest, Romania, e-mail: ana_vladu@yahoo.com

In this paper we propose to present the comparison of a product having three releases based on real life values, with a simulated defect prediction model suggested by us. For this prediction model, the Rayleigh function has been used.

2. Related Work

One of the purposes of reliability models is to make the prediction of reliability in the early stages of product development. Having a defect prediction model for testing is useful in determining the number of defects that are likely to occur during execution, and thus contributing to no known defects of a software product when it is released. Predicting the total number of defects before testing begins impacts the quality of the product being delivered.

Schneidewind [2] suggests two approaches to defect prediction: time-based and metric-based. A time-based prediction model estimates the number of defects from the number of defects already found in different previous stages of the life cycle. A metric-based approach uses metrics from historical data, applied to a prediction model.

The defects found during the test execution phase are the defects that are prevented from reaching the client, thus increasing the product quality. Although the test engineers cannot give the number of defects that will be found, defect prediction models based on previous release data and experience, can be used. An estimated number of defects can help minimize the number of defects that appear after release, and can even give a good image of the software product quality, in the long run.

The number of releases used is kept at a minimum and usually only one or two recent releases can be used for predicting. According to this model, there can be major changes in the development process along the releases and these can affect the relations between the defects. The lack of data is compensated using as many products as possible, but products that have been developed in an organization at the same time.

3. Theoretical Review

Software reliability has been defined as the probability of a software product to insure operating without failure in a specified environment, for a given amount of time. Based on this definition, software reliability prediction has been defined as a forecast of how reliable a software product will be in the future, based on data available so far. Software reliability tends to improve over testing and operating time, due to errors being removed. This is the reason for the models being named reliability growth models. Our goal is to analyze the early prediction model based on development phases.

For software products, continuous availability is necessary, and reliability is an important component of this. Even so, there can be defects in software products that cause system failure. In order to avoid these situations and to

decrease support expenses, companies want to deliver to clients reliable software. Developing reliable software is one of the hardest problems of the IT industry. Pressure brought by schedule, resource limitations and unrealistic demands can negatively impact the reliability. A difficult issue is knowing the reliability of a delivered product. After reaching the clients, the reliability is indicated by the feedback coming from them, under the form of reports, complaints, or compliments. But, by this time, it is too late to change anything: that is why companies selling software want to know ahead of time the product reliability. Reliability models try to do this.

The most important cause of defects in software is bugs, which mean incorrect implementations. Even the most talented programmers produce software with defects. The software products complexity is too big, at this moment, to be handled by people. With all the progress in programming techniques, such as splitting the programs in small modules, using evolved programming languages and complex developing tools, results are still far away from perfection, and the programming productivity has not increased significantly in the past two decades.

The most unpredictable defects in software manifest only after a specific combination of values for input data or certain external events that were not predicted by the programmer. Such combinations appear with low probability during normal testing procedures, so they often make it to the operating phase. Also, new versions are built on older versions, fixing defects found and adding new functionality. Even so, the process of fixing defects often introduces new defects, because the effects of a fix have unpredictable consequences.

Defect prediction deals with estimating the number of defects. Although other terms have been used to describe it, such as estimation, fault estimation, we should clarify the difference between the two notions. Defect estimation has been defined by Nayak [3] as a process of identifying different types of defects of a software product, aiming to reach high quality. However, defect prediction helps in estimating the quality of a product before it is released.

Pham [1] has created a prediction model based on lifecycle phases, which divide the development cycle in different phases, such as: requirements review, design, implementation, unit testing, integration, system testing, functional testing etc. This model assumes that the found defects in different phases follow the Rayleigh function. It follows the defect statistics over early development stage, in order to predict the defect density in later stages.

The Rayleigh model is the best suited model for estimating the number of defects logged throughout the testing process, depending on the stage when it was found (unit testing, integration testing, system testing, functional testing etc.). Defect prediction function is the following:

$$F(t) = f(K, t_m, t) \tag{1}$$

Parameter K is the cumulated defect density, t is the actual time unit and t_m is the time at the peak of the curve.

Before representing the Rayleigh curve, the two parameters K and t_m are estimated. At least three points are necessary for estimating this curve. Once these have been calculated, the graph can be represented for the entire time interval.

The Rayleigh function is given by the following formula:

$$f(t) = K\left[\left(\frac{1}{Peak}\right)^2 te^{-\left(\frac{1}{2Peak^2}\right)t^2}\right] \tag{2}$$

, where K represents total number of injected defects, Peak is a function of the time t_{max} where the curve reaches its peak and t is the time value at a specific moment. Parameter Peak is given by:

$$Peak = t_{\max}. \tag{3}$$

The Rayleigh function given by (2) represents the defect arrival rate that is the number of defects to arrive at a specific time t.

The Dalal and Ho model [4] is also known as the development life cycle predictive model. This model is based on a couple of assumption, such as:

1. The defect rates from different products at the same life cycle phase are samples of a statistical universe from the same organization.

2. Different releases of the same product are samples of a statistical universe of product releases.

The first assumption reflects the fact that products developed inside the same organization are more or less homogenous. This model is based on the presumption that products from the same organization behave generally the same, and that releases of the same product behave similar defect rates. Defect density for a given product is related to variables, such as number of lines of code and number of errors in previous life cycles.

4. Method Description

Based on the Rayleigh function and the values of the early stages of testing, we can predict the evolution of the defects number for the second release of the product, following on to compare to the real-life obtained values.

The application we used for this case study is an application on its second version, which means that the testing began from a stabile version of the product. We analyzed the evolution of the defect count across three consecutive releases, being tested over a period of 11, 28 and respectively 15 weeks. The application studied is an automation tool, developed in C#.

The early prediction model proposed by Gaffney and Davis assumes that the faults detected over the life of the project will take the form of the Rayleigh curve (Fig. 1). The algorithm for prediction of defects must begin with obtaining the start date and the estimated duration of the project in months. Also, since this

is a phase-based model, it is important to know the estimated durations for all the phases, which can present itself as an issue at the beginning of the project.

The data collected from the organization showed that the software process is a dynamic process, due to production cycles, client issues and support, resource management (hardware available, testers' rearrangements). Starting from this observation and taking into consideration the evolution of the defect count over the release, we decided to use a model that collects defect data weekly. This model can follow more closely the evolution of the defect rate, doesn't need time scheduling ahead of time, as the phase-based model does, and more easy to represent knowing the entire duration of the testing process.

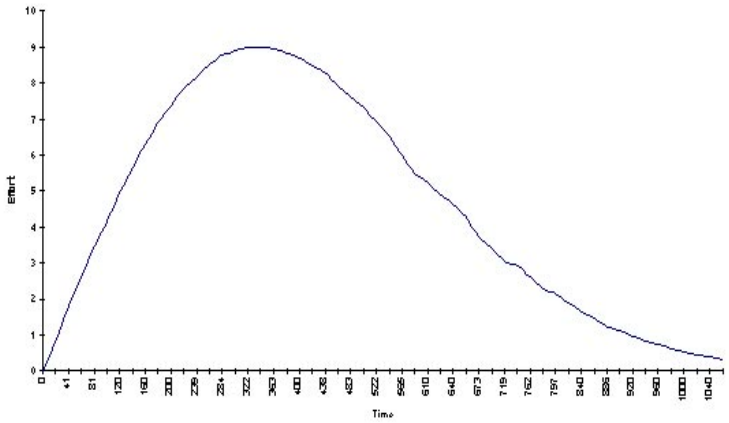

Fig. 1. Example of a Rayleigh curve

We followed the steps suggested by Dhiauddin [5] for collecting the data and using it for prediction model are described as following:

1. Gather defect data from past releases, using especially the total number of defects found.

2. Use Linear Regression to estimate the total number of defects that may appear after release.

3. Compute the number of latent defects.

4. Calculate estimated defect rate using the Rayleigh function, for each week.

5. Calculate the estimated defect rate by phase, based on the project schedule.

6. Plot the Rayleigh function for the defect prediction pattern.

7. Compare Rayleigh curve and actual data.

A Rayleigh model uses six phases to predict the defect rate, in chronological order: high level design, low level design, coding, unit testing, integration testing, and system testing, according to Qian [6].

The algorithm used by us was the following: first, given the data points, we plotted them and determined t_{max}, the time at which the Rayleigh function is at a maximum. According to Laird [7], by the time t_{max}, almost 40% of the defects have been found which can help us calculate the total number of predicted defects. Mathematically, we can determine the curve to predict the defect rate as long as we hit the maximum value.

The simulation method used for representing the proposed software reliability prediction model is created using MATLAB®. Starting from the known value of the parameter K, the time t, we could simulate and plot the function (2) over the period of time of the lifecycle. On the same plot, we have represented the real numerical values, which were introduced as arrays. For the second approach, in which an estimation of the K parameter was attempted, the method also uses MATLAB® program, as following: assuming the peak time and the peak value are two known numerical values, the parameter K could be computer, also using equation (2). The two approaches to the model, together with real data are plotted together.

5. Results

For the first release, at maximum time, the total number of defects is 149 (Fig. 2), which gives us the predicted number of defects for the entire release 372.5. Knowing K = 372.5 and the time to hit the maximum, we can determine $f(t)$. We know that the time when the function reaches the maximum is at week 3, which is part of the first phase of the product lifecycle. Instead of using the classical phase model, we used t_{max} as the number of the week to reach the peak.

Substituting the numerical values, we can determine that for the first release:

$$f(t) = 372.5 * \left[\left(\frac{1}{3} \right)^2 t e^{-\left(\frac{1}{18}\right)t^2} \right] \tag{4}$$

Another approach we followed was to try a better estimation of the K parameter, for this using the first set of available data. Having (2) and the number of defects obtained at the time when the peak is reached, we can calculate the value for K. For the same release – release 1 – the number of defects at t=3 is 69. Therefore, the value of the parameter is $K \sim 341.285$.

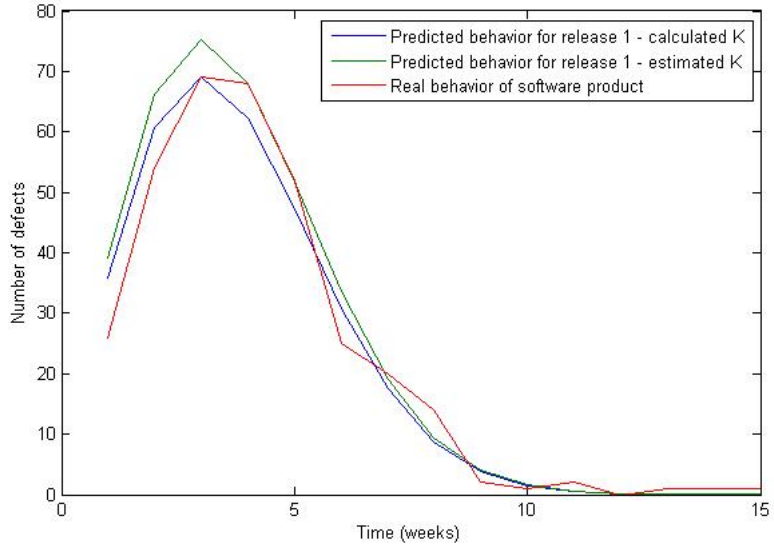

Fig. 2. Comparison of the predicted reliability models proposed and the real defect curve for the first release

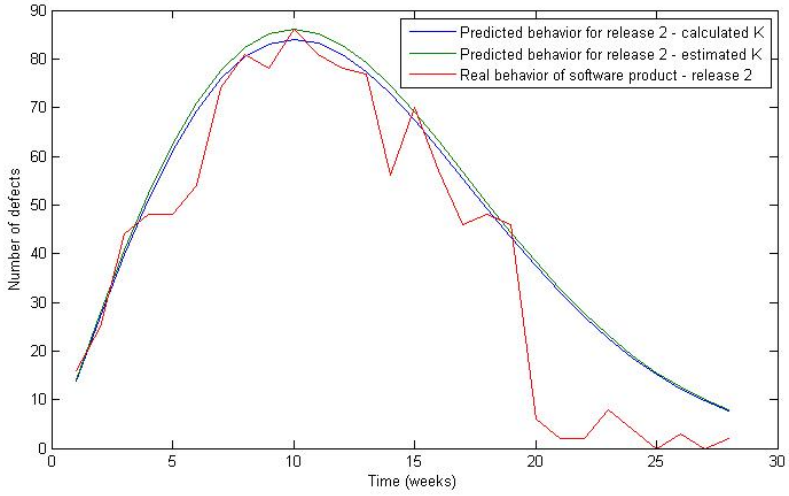

Fig. 3. Comparison of the predicted reliability models proposed and the real defect curve for the second release

For the first release, the comparison between the prediction and the real life case is show in Fig. 2. As we can notice from the plot, estimating the value of K based on the defect rate from t_m and using the time estimated in weeks, gives a

Rayleigh curve that is close to the real defect rate curve. Since the peak value was the starting point, the curves reach the value at the same time, but continue closely together after this point. While in the first two weeks, there is a slight difference between the real curve and the predicted one, after reaching the maximum value at a close point, the predicted curve follows closely the real data. This behavior could be due to the fact that the first weeks of the process life, the process and product timelines were not definite and the testing team was not at a full percent functionality.

The same algorithm was used on the following two releases. Fig. 3 shows the curve predicted using the weekly defect rate model, the curve predicted by computing parameter *K* and the curve given by the real number of defects logged for the second release of the same application. Having the total number of defect at the peak time, this gives us $K = 1385$, and the Rayleigh function:

$$f(t) = 1385 * \left[\left(\frac{1}{10} \right)^2 te^{-\left(\frac{1}{200}\right)t^2} \right] \tag{5}$$

Using the number of defects at the peak time, we computed the parameter *K*, and obtained $K \sim 1417.9$. As noticed from Fig. 3, the Rayleigh prediction model fits the real data curve well for the entire duration of the project. The major fall in defect detection, close to the 21st week of testing, is the only difference noticeable in the comparison. This is due to process decisions, such as dropping a feature that raise issues, supplementing the number of developers assigned to difficult modules or features.

For the third release, we followed the same logic in plotting the predicted defect arrival rate and the model curve using estimated *K* against the real defect arrival rate. We calculated $K = 225$ and so we have:

$$f(t) = 225 * \left[\left(\frac{1}{4} \right)^2 te^{-\left(\frac{1}{32}\right)t^2} \right] \tag{6}$$

Also starting from the number of defects found at the peak moment of time, the parameter *K* was computed ($K \sim 593.539$) and the curve was represented in Fig. 4. We noticed that the three curves are fitted closely together, especially after the peak value is reached. There are small differences between the two ways to calculate the parameter *K* using the time to reach the *Peak* expressed in week number. As we can notice from the figure, the second release expanded over a long period of time, with feature being added over time, some features removed in time, and software versions being dropped for testing, which had an impact on the defect rate. This is best seen in the real behavior as the high variations between the number of defects found from week to week.

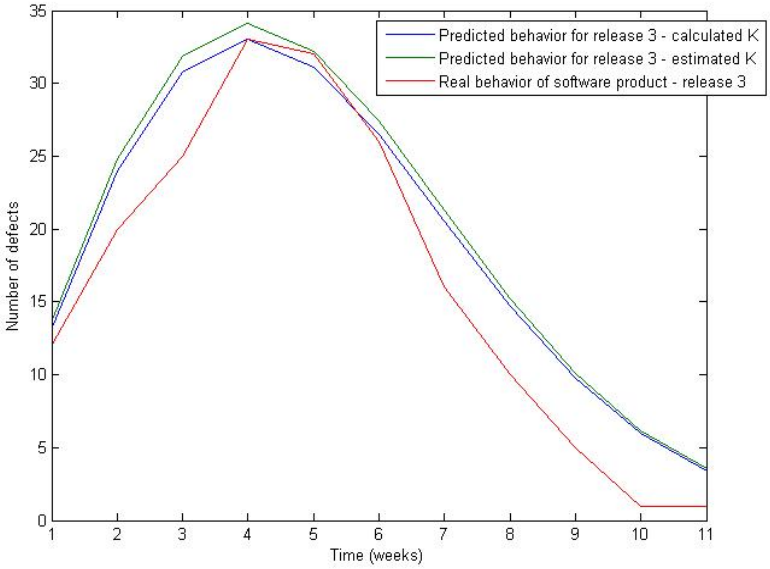

Fig. 4. Comparison of the predicted reliability models proposed and the real defect curve for the third release

Over the duration of a project the high points will tend to compensate for the low ones, so the behavior predicted by the Rayleigh model is a good approximation. Instead of using the classical approach of the phase-based Rayleigh model, we calculated all the variables of the model considering the number of the weeks for the entire release.

6. Conclusions

The most commonly used and useful defect prediction model is the model given by the Rayleigh function. The number of defects obtained per phase can vary from release to release, depending on the number of features implemented, their complexity, as well as other variables part of the product life cycle. Many of these variables can be hard to predict at the beginning of the testing process.

This paper uses the existing phase-based model as a starting point in developing a new approach to predicting the defect rate of a software product. We have considered changing this model to better suit the testing process, and thus the defect rate was calculated weekly. The most important aspect of the model was to have a very good estimation of the total number of defects, represented in the function by the K parameter, and for this reason, we have developed two approaches used in computing it. The first approach starts from the observation that by t_{max} almost 40% of the defects have been reported, while the second approach uses the Rayleigh function at the t_{max} moment of time. Further on, the

model in the two different cases was validated using data sets of the software product, provided by a company that develops software applications.

This comparison shows the real data often has some variation when compared to the original theoretical Rayleigh model. In some phases the data points will be higher or lower than the model would predict. It can also occur that real data will not follow an exact curve shape. This behavior has improved significantly using a model based on time evolution of a project, which doesn't take into consideration the phases of the process.

One aspect to notice is the importance of a correct time when the maximum rate is reached, in both cases we studied. Another suggested approach to compute K is to have its value estimated for the first three phases. Consequently, as perspectives, we are considering finding better algorithms to calculate the parameters of the Rayleigh function, in order to improve the prediction of defect rate, as well as eliminate as much as possible the empirical aspects of the reliability prediction models. Another aspect to follow in research is to find the causes for the differences between prediction and reality, and find an algorithm to obtain a better estimation.

Our approach was to determine the number of defects reported weekly and estimate the total number of defects starting from this point. It is easier and more useful to predict the evolution of the defect rate starting from the previous weeks of defect reporting, as it follows the Rayleigh curve better. It can also help with the project scheduling, giving a better estimation of the number of weeks involved in testing.

R E F E R E N C E S

[1] *H. Pham*, Handbook of Reliability Engineering, Springer, XXXI, 663 p., ISBN: 978-1-85233-453-6, 2003, pp 201-210

[2] *N.F. Schneidewind*, "Body of Knowledge for Software Quality Measurement", IEEE Computer, **vol. 25**, 1999, pp. 675 - 689

[3] *V. Nayak and D. Naidya,* Defect Estimation Strategies, Patni Computer Systems Limited, Mumbai, 2003

[4] *D. Kumar, J. Crocker, T. Chitra, and H. Saranga*, Reliability and Six Sigma, Chapter 8 – Software Reliability, Springer, XX, 386 p., ISBN: 978-0-387-30255-3, 2006

[5] *M. Dhiauddin*, Defect Prediction Model for Testing Phase, Universiti Teknologi Malaysia, May 2009, pp. 16 – 25

[6] *L. Qian, Q. Yao, T.M. Khoshgoftaar,* Dynamic Two-phase Truncated Rayleigh Model for Release Date Prediction of Software, J. Software Engineering & Applications, 2010, pp. 603-609

[7] *L. Laird*, In Praise of Defects, Stevens Institute of Technology, http://www.njspin.org/present/Linda%20Laird%20March%202005.pdf, 2006

[8] *K. Naik, P. Tripathy*, Software Testing and Quality Assurance. Theory and Practice, John Wiley and Sons, Inc., Hoboken, New Jersey, 661 p., ISBN: 978-0-471-78911-6, 2008, pp 471-479.