BIULETYN INSTYTUTU SYSTEMÓW INFORMATYCZNYCH **10** 19–29 (2012)

Software Reliability Growth Models

R. PEŁKA
radoslaw.pelka@wat.edu.pl

Faculty of Cybernetics, Military University of Technology
Kaliskiego Str. 2, 00-908 Warsaw

History of research on reliability of software began in the early seventies of the last century. A significant progress of the work aimed at the construction of a mathematical model of software reliability growth has been performed since the first publication devoted to this subject was presented. Analysis of existing literature may lead to the conclusion that there is no universal solution which could be applied in every single case. However, it is possible to classify existing models, based on their characteristics such as data domain, the way to describe faults discovered during testing process, the way to express reliability, or other remaining assumptions, including mathematical concepts used in the evaluation process. This article presents an overview of existing solutions related to software reliability modelling, focusing on variety of aspects and methods used within this process.

Keywords: modelling, software, software reliability.

1. Introduction

History of research on the reliability of software began in the early seventies of the last century. Pioneers of the new idea were Jelinski together with Moranda, Shooman and Coutinho, who published results of their work devoted to the subject. In those papers, authors aimed at constructing a mathematical model, which has the ability, based on collected test data, to predict future reliability of software under analysis. Reliability, according to the IEEE/ANSI 982.2 standard, is defined as the probability of fault free program execution in defined time period and runtime environment. It is one of the most important features that software may be characterised. Nowadays, it is even perceived as a key value affecting customer satisfaction, in addition to other important factors, such as functionality or performance.

From the moment when first publications concerning software reliability modelling and analysis were presented, plenty of new papers presenting an entirely new approach to the subject or modifying existing solutions have been prepared. However, it is still not possible to select one particular model, which could be treated as an universal one, applicable in every single case. It is a consequence of not being universal assumptions taken for model construction. Due to this fact, it is required to select the most suitable solution, basing on specific aspects of software under analysis and runtime environment conditions. It is not an easy task. It often requires many time-consuming trials that lead to proper model selection and estimation of its parameters.

2. Background Information

Reliability growth models can be divided into two main groups. Models from the first group introduce metrics for reliability estimation, such as the number of code lines, nested loops, external references or the number of inputs and outputs, that describe the source code. In the second group there are models, which are built on the basis of statistical correlations between data concerning discovered faults and known functions, e.g. exponential function. When such correlations are found, the known function characteristics are used for software behaviour prediction in the future. For this reason also, these kinds of models are called reliability growth models. This approach is much more popular within researchers.

Taking into account the domain, on which software reliability growth models operate, two categories can be specified. Time domain models constitute the widest and also most popular group. In these models reliability, associated with fault intensity, is defined as a function of time. Data domain models constitute the second category. In these models reliability, in turn, is a function of the program execution for defined input data and describes the number of successful executions in relation to the total number of tries. Researchers pay more attention to time domain models. It is worth mentioning that from the perspective of software reliability, three different models of time can be distinguished:

- execution time – time, which the central processing unit (CPU) spends on program execution
- calendar time – time in general (hours, days, weeks, months, years)
- clock time – time, which elapses from the moment of program start, excluding periods when hardware, for instance, is turned off.

The calendar time was initially the only choice made for the proposed models. This fact was put into question by Musa. It turned out that the application of the execution time instead, resulted in the simplification of models and allowed obtaining better results. The superiority of one approach over the other was also proved by Trachtenberg (1985), Musa and Okumoto (1984) and Hecht (1981) in their publications. Nevertheless, there are models where the calendar time is used or the time definition is not explicitly specified. In 1987 Musa described how results received from modelling can be converted depending on different time definitions.

Software reliability growth models define, in general terms, the relation between occurrences of program failures and main factors that have influence on that process. Such factors can be, for instance, fault introduction, fault removal or properties of runtime environment. The main idea behind software reliability modelling is to reflect such a relationship that the number of failures in a time interval decreases or time between consecutive failures increases while faults detection and the removal process proceeds. This is the dependency that every model must take into account. When it is done, statistical methods allow prediction of future behaviours of the process. This kind of knowledge may be used in two different ways. Firstly, it can be an indication for additional time needed to obtain a required reliability level. Secondly, it can be used to determine a future reliability level, at the end of the tests phase, when the current fault detection rate is sustained. By having such information it is possible to verify current test plans and introduce required modifications in order to achieve the intended goal. So, it is possible to estimate the current situation and take proper action.

All measurements, which are made to determine the software reliability involve two concepts: estimation of reliability coefficients and prediction of reliability characteristics. The former determine the current reliability on historical data basis, using statistical methods. The data describe failures from the test phase or past software utilization. The main reason for these kind of calculations is to obtain the current reliability level and to check, if the reliability growth model properly reflects the failure history – whether it has been well-chosen and well-calibrated. Functions, which characterize failure occurrence in software utilization, can be divided according to the way of how they represent these failures:

- mean value function – determines expected, overall number of failures for each moment of time
- failure intensity function – determines rate of mean value function changes
- hazard function – determines probability of failure in time interval [t, $t + D_t$] when there was no failure before t (time expressed in particular units)
- mean time to failure (MTTF) function – determines expected time when the failure occurs; MTTF is also known as mean time between faults (MTBF).

Time domain models can be divided into two classes, according to data types they use:

- models, where cumulative number of failures in a time interval is important,
- models, where time between consecutive failures is important.

Despite the fact that these approaches are different, they do not create a permanent division, because there are methods to convert a failure description from one form to another. This gives a chance to use the specified model when we initially do not have data expressed in the required way.

Function, which maps failures history over time can be a concave or s-shaped function. In the latter case, there is an assumption taken that in the beginning of the test process, we are dealing with a learning period, when faults detection rate is much lower than in later phases. Discarding this period of time, functions of both types are similar, which means that the faults detection rate decreases along with the growing number of totally occurred failures (the software becomes more reliable). An asymptote of such graph might be a line, which corresponds to the total number of faults in the program.

A reliability model is a function described by parameters. These parameters must be estimated on a test data basis. Estimation can be done by inserting test data into equations, where wanted parameters exist. The most popular direct method like that is the maximal likelihood method. The indirect method assumes matching test data with the shape of a function and estimation of the parameters through a best fit to the curve. The least squares method is the leading one here. The maximal likelihood method may be very complex because the set of many equations is produced and must be solved.

In 1983 Musa and Okumoto proposed models categorization based on their attributes:

- way to represent time – calendar or execution time
- total number of faults that may appear in an infinite period of time – finite or infinite
- failures distribution in particular time – two most important are Poisson and binominal distributions
- shape of failure intensity function.

The Poisson and binominal distributions play the key role for software reliability growth models classification. The non-homogeneous Poisson process application became the most useful and practical choice for software reliability modelling. This fact come from many experiences. Models based on binominal distribution are finite number of faults type models, which means they assume that a finite number of failures occur in an infinite time. On the other hand, models based on the Poisson distribution can be developed both as a finite or theoretically infinite number of faults models.

Many models base on the Markov process theory. Within this category we can give examples, grouped according to failure occurrence distribution:

a) poisson distribution:
   - Crow (1974)
   - Musa (1975)
   - Moranda (1975, 1979)
   - Schneidewind (1975)
   - Goel and Okumoto (1979)
   - Brooks and Motley (1980)
   - Angus (1980)
   - Yamada et al (1983, 1984)
   - Yamada and Osaki (1984)
   - Ohba (1984)

b) binominal distribution:
   - Jelinski and Moranda (1972)
   - Shooman (1972)
   - Schick and Wolverton (1973, 1978)
   - Wagoner (1973)
   - Goel (1988)
   - Shanthikumar (1981)
   - Littlewood (1981)

c) other distribution type:
   - Shooman and Trivedi (1975)
   - Kim et al (1982)
   - Kremer (1983)
   - Laprie (1984)
   - Shanthikumar and Sumita (1986).

The most popular in literature and playing a key role in studies in the software reliability field are exponential models. According to the Musa–Okumoto classification, this group consists of all finite failure models, where the failure intensity function is expressed as exponential. Models with binominal distribution from this category can be characterized by the fact that the hazard function ($z_T(t) = \mathrm{F}$) is constant in relation to a single fault and it depends on number of faults remaining in the program $(N-(i-1))$. The failure intensity function is exponential $(1(t) = N\,f \cdot \exp(-ft))$. Models with the Poisson distribution from this category can be also characterized by the fact that the hazard function ($z_T(t) = \mathrm{f}$) is constant in relation to a single fault and that time to failure caused by such a fault has an exponential form $(f_X(x) = N\,f \cdot \exp(-fx))$. Both for homogeneous and non-homogeneous Poisson processes, the number of failures that occur in any, defined time interval, is a random variable with the Poisson distribution. For models where time between consecutive failures is crucial, an exponential distribution is used.

In the first software reliability models developed by Jelinski and Moranda or Shooman, time between consecutive failures is described by the exponential distribution with the parameter, which is proportional to the number of faults remaining in the program, e.g. a mean time between failures for t is $1/f(N-(i-1))$, where t is any moment in time between $i-1$ and i-th failure occurrence, f is a proportionality coefficient and N expresses the total number of faults in the software, when analysis was started. When a failure occurs, then the hazard function value decreases by the f – constant value. This fact shows that each fault removal has the same influence on the overall software reliability. In the Musa–Okumoto classification, it is a model of binominal type. In such models the following assumptions are taken:

- all faults existing in the software have an equal influence on the overall reliability. Due to this, in every moment of time, failure intensity is proportional to the number of faults remaining in the program

- failure rate is constant between consecutive failure occurrences
- fault detection is an equivalent to fault removal and it is not possible to introduce a secondary fault at the same occasion
- software under analysis will be utilized in similar conditions to these, where reliability estimation is done
- each fault has the same probability of occurrence and has the same negative impact on the program, as other errors
- failures are independent off each other.

The last three assumptions are common for other basic models. Examples of models, that apart from the notation, are either identical or are very close approximations of the exponential model, can be models developed by Musa (1975), Schneidewind (1975) or Goel and Okumoto (1979).

Another type of models are models using the Bayesian theory. These models differ substantially in terms of certain deterministic findings when compared to models with the Markovian and exponential approach. For instance, in the exponential approach it is assumed, in most cases, that each single fault belongs to one, common severity class and has the same influence on the failure intensity function. The Bayesian approach, however, contests this assumption and says that the fault significance should be adjustable by a certain coefficient in the model (Littlewood, 1981). Previously mentioned models use also the assumption, that any change in reliability estimation shall be done only when a failure occurs. The Bayesian theory introduces a subjective point of view, e.g. in the case when there is no failure at some period of testing, it is allowed to grow one's belief about program correctness and adjust the software reliability estimation accordingly. Therefore, it can be concluded that reliability is the function of both the number of faults detected and the fault-free periods.

In accordance with the belief that different faults have different impact on program reliability, the total number of faults is not so important as their severity is. Such an approach seems to be more adequate from a practical point of view. If there is, for instance, a program, in which few faults are placed in a rarely executed part of code and some other program, in which there is only one fault, but placed in a very often used part of code, then according to the Bayesian theory it is not true to say that the latter program is more reliable than the first one. More important is to look at the entire operation of the code, rather than estimating the total number of faults contained therein. Due to this fact, the mean time to failure (MTTF) value became a very important statistic in this approach. Other unique attribute of this technique is the use of historical data from previous, similar projects to estimate reliability of the current software. It is automatically a significant difficulty to use such data in a smart way. The Littlewood–Verrall (1973, 1974) proposal is probably the best example of solution within this class. This model makes an assumption that the program under testing may become less reliable during the evaluation than it was before. Due to uncertainty of perfect fault removal after each failure, it is possible that the new version can be either better or worse than the previous one. It is reflected in such way that parameters, which define failure distribution over a period of time are selected randomly. The failures, as in the previously described models, are exponentially distributed with known failure intensity, but here the intensity is random, not constant as before. Distribution of the intensity on the basis of historical data, may be a gamma distribution. Variations of such models are models presented by Mazzuchi and Soyer (1988), Musa (1984) or Keiller et al (1983).

Every conventional model may become a Bayesian model, if at least one of its parameters, will be assigned a proper distribution. Most of the models with the Bayesian theory use an exponential model as a starting point, for example Littlewood and Verrall (1974), Goel (1977), Littlewood (1980), Jewell (1985), Langberg and Singpurwalla (1985), Littlewood and Sofer (1987), Becker and Camarinopoulos (1990) or Csenki (1990). There are also completely new models, like Littlewood and Verrall (1973), Thompson and Chelson (1980), Kyparisi and Singpurwalla (1984) or Liu (1987). The main problem with such models is their complexity and difficulty of correct distribution choice for parameters. What is more, most of the software designers have not enough knowledge of statistics to fully understand and use such models. Conventional models are far more often used in practice. What is common for many conventional and Bayesian models is the idea that early fault correction have a bigger influence on failure intensity than those corrections made during later phases.

As it was mentioned before, proper model selection for existing test data is not an obvious and easy task. Different models may give different predictions, basing on the same failure data. It is not a unique phenomenon, typical for software reliability growth models, but it can be observed also in other models, where some changeable in time values are evaluated. What is more, one model may give reasonable results for one failure data set and controversial for other. Searching for the best model of software reliability began in the late 70s and early 80s of the last century. Initial efforts devoted to model comparison, conducted by Schick and Wolverton (1978), also Sukert (1979), did not bring expected results. Basically, the problem was in the lack of proper basis of failure history and overall acceptance on common criteria, which should be used for such purposes. The former deficiency was complemented successively, with a major contribution of Lyu, who in 1996 published fifty useful sets of failure history. These sets were created under special surveillance and represent data related to various applications such as real time command and control systems, commercial systems or military and space systems. A kind of consensus in comparative criteria selection was presented by Iannino et al. In 1984 they proposed the following criteria:

- Usefulness of the model for failure prediction, basing on known or assumed software characteristics, such as code metrics estimation or failure history.
- Usefulness of the model for precise estimation of indicators for planning and maintaining of software development during a project or operation. These indicators can be, for example, current failure intensity, the time when failure intensity reaches a desirable level, amount of resources and cost required to reach a desirable failure intensity.
- Quality of model assumptions, for instance, data availability, clarity and precision.
- Usefulness of the model for the software, which differs in terms of size, structure, function, runtime environment and software development life time phase.
- Model simplicity, for instance, simplicity of used concepts, or simple and inexpensive way to collect data, or implementation in terms of software utility.

Introduction of such categorization, based on proposed criteria, in some extent may restrict the scope of models suitable for application. Nevertheless, there are many aspects that may have influence on the process of failure occurrence and which are not taken into consideration by any of the models. The best recommended method of model selection is to examine various possibilities for the same failure data set. Due to the fact that such a process is very time consuming, it has more sense with the help of tools, such as CASRE, SMERFS, SRMP or SREPT, developed to support user in software reliability estimation.

3. Software Reliability Modelling Concept Evolution

The theory of software reliability modelling has been a subject of continuous development over the years. New and various proposals have appeared. They suggest how existing solutions may be enhanced or how new ideas may lead to satisfactory results. The recalibration concept (Abdel-Ghaly and others, 1986), in other words an adaptive prediction, would lead to better results. It is a statistical procedure, which allow model parameters adjustment according to previous failures and, in consequence, giving better predictions – reducing the level of model "corruption". Another idea is a linear combination of models (Lyu i Nikora, 1991, 1992), which, even in the easiest form, is capable of giving more adequate predictions than one, single model. An early prediction model based on phases (Gaffney i Davis (1988)) suggests the use of fault statistics prepared during technical reviews of project requirements or code development and implementation, to reliability prediction for the test phase and future operation. One of the first and well known attempts of software reliability prediction in the early development phase was the method proposed by researchers working for Rome Air Development Center (1987). For their model, they developed a method of fault density prediction, what might be later transformed into other types of reliability measurements, like failure intensity. In another model proposed by Kapur and Garg (1992), dependencies between faults in the fault removal process are taken into consideration.

Reliability growth theory for modelling uses only data related to failures from the system under investigation, while its structure is ignored. A closer look at the system architecture became more interesting when the component--based production became more popular. Evolution of network technologies favoured development of distributed systems. Moreover, the use of object-oriented programming languages, helped logical function separation, what together caused that the modularity philosophy in software engineering became the most popular approach. The essential meaning is fact that in the system under investigation only part of the components have been modified or written from scratch, and the rest of the components remained unchanged. The first model taking into account modularity was the model developed by Littlewood (1979). This idea was further promoted. Smidts and Sova (1999) considered modelling, which for software reliability prediction takes into account the system architecture, based on the requirement decomposition to functions and attributes of the program.

Kuball et al (1999) introduced a hierarchical model with the Bayesian theory for the failure probability prediction of the system based on the components. Lyu et al (2002) formulated the resource requirements for the testing phase of the software based on components as a combinatorial optimization problem with known costs, reliability, incurred effort and other attributes of components. Another papers related to this topic were presented by Kubat (1989), Gokhale (1998), Ledoux (1999), Yamada (2000) or Okamura (2004). Summarizing the introduction of structural parameters into the reliability engineering process gave other opportunities for evolution.

The software reliability model parameterization may also be based on alternative origins of information, such as metrics developed by early prediction models. Other kinds of metrics may concern test cases coverage or system load. Piwowarski et al (1993) proposed a simple software reliability growth model based on test cases coverage application. Malaiya et al (1994) presented a logarithmic model where test team effort is taken into account in relation to test specification coverage, what may have direct influence on coverage of defects present in a program and so the increase of its reliability. Chen et al (2001) included test coverage into modelling by reference to the time of their execution. Fujiwara and Yamada (2001) proposed a model that includes the characteristic of prepared sets of system tests. In particular, it refers to skills and knowledge of testers who prepare test cases. Longer experience leads to the extension of the fault scope that potentially may be discovered with use of a selected test case group. It is worth mentioning that in accordance with software reliability engineering assumptions, test ceases should be prepared so that they reflect the operational profile of the investigated system. The operational profile is defined as a set of software operations, together with the probability of their occurrence. An operation is a set of calls which, in most cases, requires similar computing.

A test case coverage aspect is related with modelling taking into account code coverage. The coverage is understood here as an execution of selected instructions of the program or passing functional paths at least once. Such information is further combined with the failure data. Models introducing such an idea were presented by Fujiwara and Yamada (2002), Malaiya (2002), Pham and Zhang (2003) or Inoue and Yamada (2004).

Another approach is modelling quality metrics (dependent variables) on the basis of their relationship with other, independent variables, such as code size, data size, code complexity, operators, operands etc. They are later used for predictions of software reliability and quality. Agresti and Evanco (1992) attempted to create a model for fault density prediction on the basis of characteristics from the development process for the Ada language. Gokhale and Lyu (1997) used a regression tree analysis technique for relationships establishment between dependent and independent variables. Schneidewind (2000) developed a method for software examination giving the ability to recognize, which modules are fault-prone and which are not. This information may be used for quality controlling purposes and future maintenance. Another example is the use of data in terms of cyclomatic complexity of the code, or number of code lines, in connection with fault data. This idea was used for reliability prediction of software prepared for satellite control (NASA JM1) by NASA. The model used for that analysis was described by Schneidewind (2008).

An interesting area for scientific researches devoted to software reliability estimation are also simulation techniques. Von Mayrhauser et al (1993) conducted an experiment to verify the nature of relations between program faults and its structure and they proposed a technique for reliability prediction. Tausworthe and Lyu (1996) proved that the Monte Carlo simulation technique is useful for software reliability prediction. They established a simulation technique allowing the modelling of a complete cycle of software development, including fault and failures life time cycle. Gokhale et al (1998) enhanced this technique further for reliability analysis of systems based on components, for various architectures and configurations.

The simplest assumption of research on the process of faults detection and removal is that the similar scale of effort and test strategy are required for various faults detection. Practically, this might be untrue. To reflect that differentiation, faults might be classified into separate categories, each group of faults of different complexity. In this way, fault detection and removal rate of different categories vary. In the modified exponential model Yamada (1983) assumed two categories of faults. Pham (1993) proposed a model with many types of faults. Kapur (1995) also introduced such assumptions into his model, where time between failures and time required for fault removal is dependent on fault category. The same author, in another paper (2000), proposed the classification of faults to different categories based on the time of a particular fault detection. Another issue is whether the fault detection rate should remain constant throughout the testing process. It turns out that many factors, such as test strategy, changes in a testing environment or in test team personnel assignments, including tester motivation, have direct impact on faults detection and removal process. Any deviation may be analyzed using the concept of testing time division into intervals, where during each interval test strategy and environment remain, more or less, unchanged and differ from other intervals. The fault detection rate (alternatively fault removal rate) in such an interval is constant or is a function of testing time and differs from analogous quantities in other intervals. This concept was initiated by Zhao (1993), and later developed by other authors: Chang (2001), Chen (2001), Shyur (2003), Zou (2003), Kapur (2006) and Gupta (2008). Examples of other models introducing modifiable level of testing effort are models proposed by Yamada (1991, 1993), Bokhari (2006), Kapur (2004), Kuo (2001) or Huang (2007).

Moving away from a total number of faults, as an indicator of software reliability, was proposed by Sawada and Sandoh (1999). According to their vision, testing of software is concluded as a series of demonstrative tests where information about the number of revealed faults and their negative impacts on software is collected. It is a sort of prototype testing when functionality of the code grows from stage to stage. Before each presentation, a fixed limit of faults and their consequences is assumed and according to received results, the decision about prototype acceptance is taken. Some additional figures related to risk level of the vendor and of the customer are also calculated by applying statistical analysis. The concept proposed by Japanese researchers is useful, especially from the perspective of new techniques of software development. An example could be the SCRUM, where each stage, called a sprint, aim to produce complete, up to some point, program version, which may be presented to a customer.

For typical software reliability growth models, failure history, collected in past test phases, is very important. This information might not be representative due to changes in the test environment, differentiated test strategy etc. Xie, Hong and Wohlin (1997) presented a method of ”exponential smoothing” a technique application for reliability prediction. In their approach, much higher importance for estimations is the current information. This allows elimination of negative influence of the premature test phase, when the system is not stable yet. Additional advantages of this model are intuitive parameters and low requirements on time consuming calculations.

Typical reliability growth models are used for failure process modelling under the assumption, that fault removal is immediate and reliable, which means that imperfect fault correction and new faults introduction is not taken into account. Such an assumption is far away from reality when it comes to software development. That is why researchers started to look into this phenomenon, introducing elements of faulty correction. Initiators of the idea of imperfect debugging were Goel (1985) and Kapur with Garg (1990). Initiators, who introduced fault generation (perfect fault correction but, at the same time introducing new faults) were Ohba and Chou (1989). Later, this approach was further developed by Yamada (1992), Kapur (1996), Pham (2000, 2006), Shneidewind (2001), Shyur (2003) or Chatterjee (2004). In practice, there are two reasons of imperfect debugging. First, it is the impossibility of perfect fault removal, because a mistake probability always exists (Goel-Okumoto – 1979, Yamada – 1993). Second, by analogy, it is the introduction of new faults, because it is likely to happen. It can also be assumed, that the total number of program faults is an increasing function of time (Ohba-Chou – 1989, Yamada – 1992, Pham-Zhang – 1997). There are examples of researches linking both reasons mentioned above (Zeephongsekul – 1994, Pham – 1995). Time and effect analysis related to debugging creates another process. In this case, the model should take into account data from fault detection and fault removal process. The process of fault removal can be regarded as a delayed process of fault detection, since the fault can be fixed only after its detection, so it can be e.g. the non-homogeneous Poisson process. The delay reflects a time required for fault correction, which can be a constant or random value.

Models, which assume a probabilistic character of the fault detection process were criticized by Cai (1991). He claimed, that software uniqueness should determine applicability of fuzzy logic for reliability modelling, because such a process is fuzzy due to its nature. The argument for this theory is that a debugging process cannot be recurrent in terms of probability theory and none, even large, set of samples does not guarantee possessing enough amount of information for good prediction. The model introducing this new approach, the model of Cai-Wen-Zhang, was proposed in 1993 Elements of fuzzy sets theory were used later also by Utkin and Gurov (2002). They also take into account the imperfect debugging and removal of faults, which are related. Cai, in another of his papers (2006), put into question the non-homogeneous Poisson process (NHPP) applicability as a method for fault detection process description. In the case when applicability of Poisson process is assumed, the expected value and variance of the number of failures up to time *t* are equal, according to Poisson distribution properties. Cai presented results from the experiment conducted on software with known number of faults. It turned out that the estimation for the expected value and variance were far different, what can be regarded as proof against assumption about NHPP applicability. In the same paper the Markovian approach, used almost as often as Poisson theory, was also criticized. According to this approach, passing between two consecutive states during program execution, where in case of reliability analysis the transition is done at the moment of failure, is a Markov process realization. On an experiment basis, some gaps in this hypothesis were found. Revealed inaccuracies do not cause the need for abandoning of a popular techniques, but it only show, that described techniques do not have a universal character.

In models using NHPP, the failure intensity function is a continuous function of time. The general argument for NHPP applicability is its simplicity. The main information here is an expected value of software failures. Continuity assumption is unreal, because the debugging process causes time gaps. Moreover, software is not wearing out, so when it is not modified, then its failure susceptibility does not change. That is why failure intensity for periods between consecutive debugging sessions should be constant. Imperfection of models with finite number of faults and the non-homogeneous Poisson process theory features for some failure data sets, resulted in the need for finding models with some other theory application. That was the background for, inter alia, models using loglogistic (Gokhale and Trivedi), or hypergeometrical distribution (Hou – 1995, Tohma – 1989).

Another emerging element in the construction of software reliability growth models are neural networks. The first, who introduced this idea was Karunanithi (1991). Later, other researching results on the same field were presented by Khoshgoftar (1992, 1993, 1996), Guo and Lyu (2000), Cai (2001), Tiang (2004), Su (2005, 2007) or Kapur (2008). Neural network models were used to determine software quality attributes, such as reliability, or to detect fault-prone code. Some authors also introduced in their research fault severity classes or imperfect debugging and they also applied various architecture of the neural network. A neural network has the capability to give realistic results, basing on sophisticated and not precise input information. It is a perfect mechanism for failure process analysis, especially when simplifying assumptions are eliminated and the process becomes unintuitive for human perception. Experience showed that this is a good alternative for typical parametric models.

Markovian models were created on the basis of the assumption that detection of a new fault during the debugging session is dependent on the current software state. Previous states, related to already detected faults, may not be taken into account, so it is a memory-less process. Various assumptions taken for model parameters led to different models development: Jeliński-Moranda (1972), Moranda (1979), Littlewood-Verral (1973) or Gaudion (1994, 1999). Reliability related figures usually concern consecutive times between failures or cumulative numbers of failures in a particular moment. The information, whether the fault correction took place just after fault detection or if correction was trivial or complex, is usually not mentioned, while it might be a useful input for the reliability modelling process. This point of view is adapted in models where theory of hidden Markov models is used, e.g. the model proposed by Durand and Gaudoin (2003, 2005).

A relatively new approach is the use of the mixed Poisson distribution for software reliability modelling. In general, well known models with NHPP theory (Goel-Okumoto, Yamada, Ohba-Osaki) are based on stochastic counting processes, describing defects discovered during the test phase. However, the debugging process is not so simple to describe it in such way, due to the fact of fault dependencies. In this context, Markovian processes are an alternative, but from the standpoint of statistical calculations, it is a real disadvantage. The modified NHPP model might give better results.

4. Summary

Software reliability growth models are generally used for establishing the current reliability and predicting future reliability of software. Information obtained from software reliability modelling might be useful for many cases where decision related to cost analysis, resource allocation or release date, must be taken during software development phases. Among techniques having the greatest influence on software reliability modelling, the non-homogeneous Poisson process should be highlighted as the one playing an important role. Especially the model created by Goel and Okumoto (1979) must be mentioned here. Some researches proved that models with simple implementation can be as good as those, which are complex, and which include many important, additional aspects.

Today some opinions are that reliability growth models developed at the time when the waterfall philosophy was the dominated one in software development, are not so useful for new, agile techniques. It is caused by the fact that it is difficult in the estimation of model parameters, mainly due to lack of suitable data.

A natural conclusion from observing the reliability growth during the testing phase is that the longer the software is being tested the better the quality can be assured. However, overzealous testing is pointless, because it causes project costs to grow and postpones the moment of product release. Short time-to-market is a very important indicator for customers today. In dynamically changing conditions and needs of the market, producing even the most reliable software, but not following scheduled time frames, is not worth spending money, because customers start to locate their point of interest somewhere else. It is a big challenge for software production companies, where, for instance, old processes of development must be replaced by new processes that reflect the current market requirements more. On the other hand, releasing software with major faults in functionality, which are discovered too late, because in the customer environment, is a serious loss for company. Such faults are much more expensive because, apart from money, they lower the level of trust in the vendor and spoil company's reputation. It is difficult to make the correct decision on the time of software release and its readiness for the market. Researches devoted to the optimal release time problem were conducted by Yamada (1985), Brown (1989), Ohtera and Yamada (1990), Ehrlich (1993), Hou (1997), Pham (1999, 2004), Rinsaka (2004) or Huang (2006).

In conclusion, it is worth mentioning a quote of one statistician, who had a significant contribution to reliability analysis. George E.P. Box once said that "Generally, all models are wrong, but some of them are useful."

5. Bibliography

[1] M.R. Lyu, ”Handbook of software reliability”, *IEEE computer society press*, (1996).

[2] Ch.H. Lee, Y.T. Kim, D.H. Park, ”S-shaped software reliability growth models derived from stochastic differential equations”, *IIE transactions,* (2004).

[3] K-Y. Cai, ”Software Reliability Experimentation and Control”, *J. Comput. Sci. & Technol*., (2006).

[4] W. Everett, S. Keene, A. Nikora, ”Applying Software Reliability Engineering in the 1990s”, *IEEE Transactions on Reliability,* (1998).

[5] P.K. Kapur, A. Kumar, K. Yadav, S.K. Khatri, ”Software reliability growth modelling for errors of different severity using change point”, *International Journal of Reliability, Quality and Safety Engineering*, (2007).

[6] K. Sawada, H. Sandoh, ”Software Reliability Demonstration Testing with Consideration of Damage Size of Software Failures”, *Electronics and Communications in Japan*, (1999).

[7] M. Xie, G.Y. Hong, C. Wohlin, ”A study of the exponential smoothing technique in software reliability growth prediction”, *Quality and Reliability Engineering International*, (1997).

[8] M. Xie, Q.P. Hu, Y.P. Wu, S.H. Ng, ”A Study of the Modeling and Analysis of Software Fault-detection and Fault-correction Processes”, *Quality*

and Reliability Engineering International, (2007).

[9] L.V. Utkin, S.V. Gurov, ”A fuzzy software reliability model with multiple-error introduction and removal”, *International Journal of Reliability, Quality and Safety Engineering*, (2002).

[10] A. Yadav, R.A. Khan, ”Critical Review on Software Reliability Models”, *International Journal of Recent Trends in Engineering*, (2009).

[11] S. Yamada, K. Sera, ”Imperfect Debugging Models with Two Kinds of Software Hazard Rate and Their Bayesian Formulation”, *Electronics and Communications in Japan*, (2001).

[12] S. Yamada, ”Software Reliability Growth Models Incorporating Imperfect Debugging with Introduced Faults”, *Electronics and Communications in Japan*, (1998).

[13] J-Y. Park, ”Integration of imperfect debugging in general testing-domain dependent NHPP SRGM”, *International Journal of Reliability, Quality and Safety Engineering*, (2005).

[14] P. Zeephongsekul, W. Bodhisuwan, ”On a generalized dual process software reliability growth model”, *International Journal of Reliability, Quality and Safety Engineering*, (1999).

[15] K. Esaki, M. Takahashi, ”A model for program error prediction based on testing characteristics and its evaluation”, *International Journal of Reliability*, *Quality and Safety Engineering*, (1999).

[16] P.K. Kapur, O. Singh, R. Mittal, ”Software reliability growth and innovation diffusion models: an interface”, *International Journal of Reliability*, *Quality and Safety Engineering*, (2004).

[17] F. Padberg, ”Maximum likelihood estimates for the hypergeometric software reliability model”, *International Journal of Reliability, Quality and Safety Engineering*, (2003).

[18] P.K. Kapur, S.K. Khatri, M. Basirzadech, ”Software reliability assessment using artificial neural network based flexible model incorporating faults of different complexity”, *International Journal of Reliability, Quality and Safety Engineering*, (2008).

[19] L. Tian, A. Noore, ”Software reliability prediction using recurrent neural network with Bayesian resularization”, *International Journal of Neural Systems*, (2004).

[20] J. Zheng, ”Predicting software reliability with neural network ensembles”, *Expert Systems with Applications*, (2009).

[21] S.S. Gokhale, ”Software failure intensity, reliability and optimal stopping time incorporating repair policies”, *International Journal of Reliability*, *Quality and Safety Engineering,* (2006).

[22] P.J. Boland, H. Singh, ”Determining the optimal release time for software in the geometric Poisson reliability model”, *International Journal of Reliability*, *Quality and Safety Engineering*, (2002).

[23] X. Zhang, H. Pham, ”Comparison of nonhomogeneous Poisson process software reliability models and its application”, *International Journal of System Science*, (2000).

[24] K. Worwa, *Modelowanie i ocena wzrostu niezawodności oprogramowania w procesie testowania*, Wojskowa Akademia Techniczna, (2005).

[25] S.H. Khan, *Metryki i modele w inżynierii jakości oprogramowania*, Wydawnictwo Naukowe PWN SA, (2006).

[26] M.R. Lyu, ”Software Reliability Engineering: A Roadmap”, *IEEE Computer Society*, (2007).

[27] S. Chatterjee, S.S. Alam, R.B. Misra, ”Sequential Bayesian technique: An alternative approach for software reliability estimation”, *Sadhana*, Vol. 34, Part 2, (2009).

[28] T.M. Khoshgoftaar, T.G. Woodcook, ”Software reliability model selection”, *Quality and Reliability Engineering International*, (1992).

[29] M.R. Lyu, ”Software Reliability Theory”, *John Wiley & Sons, Inc.*, (2002).

[30] R.I. Zequeira, ”A model for Bayesian software reliability analysis”, *Quality and Reliability Engineering international*, (2000).

[31] M. Kimura, S. Yamada, S. Osaki, ”Statistical Software Reliability Prediction and Its Applicability Based on Mean Time between Failures”, *Elsevier Science Ltd.*, (1995).

[32] K. Sawada, H. Sandoh, ”A summary of software reliability demonstration testing models”, *International Journal of Reliability, Quality and Safety Engineering*, (1999).

[33] S. Ramani, S.S. Gokhale, K.S. Trivedi, ”SREPT: software reliability estimation and prediction tool”, *Performance evaluation 39*, (2000).

[34] T. Fujiwara, S. Yamada, "Software Reliability Growth Modeling Based on Testing-Skill Characteristics: Model and Application", *Electronics and Communication in Japan*, 2001.
[35] S. Yamada, Y. Tamura, M. Kimura, "A Software Reliability Growth Model for a Distributed Development Environment", *Electronics and Communications in Japan*, (2000).
[36] H. Okamura, S. Kuroki, T. Dohi, S. Osaki, "A Reliability Growth Model for Modular Software", *Electronics and Communications in Japan*, (2004).
[37] H. Tanaka, S. Yamada, S. Osaki, "Software Reliability Growth Model with Continuous Error Domain – Application of a Linear Stochastic Differential Equation", *Electronics and Communications in Japan*, (1992).
[38] A. Wood, "Software Reliability Growth Models", *Tandem Computers*, (1996).
[39] H. Okamura, T. Dohi, "Software reliability modeling based on mixed Poisson distribution", *International Journal of Reliability, Quality and Safety Engineering*, (2008).
[40] J-B. Durand, O. Gaudoin, "Software reliability modeling and prediction with hidden Markov chains", *Statistical Modeling*, (2005).
[41] S. Inoue, S. Yamada, "Testing-coverage dependent software reliability growth modeling", *International Journal of Reliability, Quality and Safety Engineering*, (2004).
[42] S. Yamada, T. Fujiwara, "Testing-domain dependent software reliability growth models and their comparison of goodness-of-fit", *International Journal of Reliability, Quality and Safety Engineering*, (2001).
[43] T. Fujiwara, S. Yamada, "A Testing-Domain-Dependent Software Reliability Growth Model for Imperfect Debugging Environment and Its Evaluation of Goodness-of-Fit", *Electronics and Communications in Japan*, (2003).
[44] A. Gupta, R. Kapur, P.C. Jha, "Considering testing efficiency and testing resource consumption variations in estimating software reliability", *International Journal of Reliability, Quality and Safety Engineering*, (2008).
[45] O. Gaudion, "Software reliability models with two debugging rates", *International Journal of Reliability*, *Quality and Safety Engineering*, (1999).
[46] N. Schneidewind, "Complexity-driven reliability model", *International Journal of Reliability*, *Quality and Safety Engineering*, (2008).

Modele wzrostu niezawodności oprogramowania

R. PEŁKA

Historia badań nad niezawodnością oprogramowania sięga lat 70. ubiegłego wieku. Od momentu pojawienia się pierwszych publikacji poświęconych tej tematyce nastąpił znaczący rozwój i postęp prac mających na celu między innymi budowę matematycznego modelu umożliwiającego badanie wzrostu niezawodności oprogramowania w procesie jego testowania. Analizując dostępną literaturę, można dojść do wniosku, że nie istnieje rozwiązanie uniwersalne, które dałoby się zastosować w każdym przypadku. Możliwa jest natomiast klasyfikacja dostępnych modeli ze względu na cechy charakterystyczne poszczególnych rozwiązań, takie jak dziedzina danych, sposób opisu błędów pojawiających się w procesie testowania, sposób opisu niezawodności czy też pozostałych założeń, w tym narzędzi matematycznych wykorzystywanych w procesie ewaluacji. Artykuł ten przedstawia przegląd istniejących rozwiązań modelowania niezawodności oprogramowania, kładąc nacisk na różnorodność aspektów oraz metod wykorzystywanych w tym procesie.

Słowa kluczowe: modelowanie, oprogramowanie, niezawodność oprogramowania.