Optimal Release Time: Numbers or Intuition?

Hans Sassenburg
Software Engineering Institute
An der Welle 4
D-60322 Frankfurt, Germany
+41 (033) 7334682
hanss@sei.cmu.edu

Egon Berghout
Centre for IT Economics Research
P.O. Box 800
NL-9700 AV Groningen, Netherlands
+31 (050) 3633721
e.w.berghout@eco.rug.nl

ABSTRACT

Despite the exponential increase in the demand for software and the increase in our dependence on software, many software manufacturers behave in an unpredictable manner. In such an unpredictable software manufacturer organization, it is difficult to determine the optimal release time. An economic model is presented supporting the evaluation and comparison of different release or market entry alternatives. This model requires information with respect to achieved reliability and maintainability. Existing literature reveals many models to estimate reliability and limited models to estimate maintainability. The practicality of most available models is however criticized. A series of case studies confirmed that software manufacturers struggle with determining the reliability and maintainability of their products prior to releasing them. This leads to a combination of non-analytical methods to decide when a software product is 'good enough' for release: intuition prevails where sharing convincing information is required. Next research steps are put forward to investigate ways increasing the economic reasoning about the optimal release time.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification – *reliability, statistical methods.*

D.2.8 [**Software Engineering**]: Metrics – *process metrics, product metrics.*

K.6.3 [**Management of Computing and Information Systems**]: Software Management – *software development, software maintenance.*

General Terms

Management, Measurement, Economics, Reliability.

Keywords

Optimal release time, software reliability prediction, software reliability estimation, maintainability.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
WoSQ'06, May 21, 2006, Shanghai, China.
Copyright 2006 ACM 1-59593-085-X/06/0005...$5.00.

1. INTRODUCTION

A relatively unexplored area in the field of software management is the release or market entry decision, deciding whether or not a software product can be transferred from its development phase to operational use. As many software manufacturers behave in an unpredictable manner [1], they have difficulty in determining the 'right' moment to release their software products. It is a trade-off between an early release, to capture the benefits of an earlier market introduction, and the deferral of product release, to enhance functionality, or improve quality. A release decision is a trade-off where, in theory, the objective is to maximize the economic value. Inputs into the release decision are expected cash inflows and outflows if the product is released. What is the market window? What are the additional pre-release development costs when continuing testing and the expected post-release maintenance costs when releasing now?

2. ECONOMIC MODEL

A release decision is a trade-off where, in theory, the objective is to maximize the economic value. Inputs into the release decision are expected cash inflows and outflows if the product is released. The determinants of the economic value of a software product are separated into a development and an operations phase, as in Figure 1. A commonly used capital budgeting method to evaluate and compare investment proposals is NPV, being the discounted present value of the difference between total cash inflows and total cash outflows.

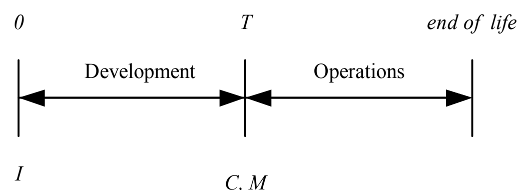

Figure 1: Determinants of Economic Value [5]

Its value can be calculated as the net asset value, equal to $C - M$, from which the cost of development I is deducted, with all cash inflows and outflows expressed in their present value. Equation:

$$NPV = -I + (C - M) / (1 + r)^T \quad (1)$$

With:

- T is the development time or time-to-market, defined as the elapsed time between the commitment to invest in the project and the time the product is released (start of first major cash inflow from revenues or cost savings);

- I is the total present value, at time 0, of all cash outflows from the time the decision to invest is made to the product release date;
- C is the total present value at time T of the cash inflows that the product is expected to generate during its lifetime (revenues, direct cost savings), also called the asset value or revenue;
- M is the total present value at time T of all cash outflows in the operational phase (corrective and adaptive/perfective maintenance), also called operational costs;
- r is the discount rate representing the systematic risk in the software product.

When faced with the release or market entry decision, a software manufacturer has to choose between an early release, to capture the benefits of an earlier market introduction, and the deferral of product release, to enhance functionality, or improve quality. If testing, as the last project stage, is stopped too early, significant defects could be released to intended users and the software manufacturer could incur the post-release cost of fixing resultant failures later. If testing proceeds too long, the cost of testing and the opportunity cost could be substantial. At some point in time during product development, two main questions will arise; how long the software will run before it fails; and how expensive the software will be to remove failures? Answers to these questions require knowledge of the reliability and maintainability of the product. The achieved reliability level determines determine how long testing should continue before the product is stable enough to be released. The achieved level of maintainability determines how easily defects can be removed once the product has been released and how easily the software can be further enhanced.

Different alternatives can be evaluated by comparing their NPV values. Erdogmus introduces a method for comparative evaluation of software development strategies based on NPV-calculations, used to compare custom-built systems and systems based on Commercial 'Off the Shelf' (COTS) software [5]. Erdogmus distinguishes comparison metrics for various variables that influence the NPV of a project. This method was used for a similar method to reflect software release decisions [20].

Let V be a variable and let V_a and V_b denote the value of variable V for alternatives A and B respectively. A comparison metric is a function of V_a and V_b and for a specific value of a comparison metric, alternative A is said to be favourable over B if for the value of that metric the project NPV for alternative A is superior to the project *NPV* for alternative B, when everything else is equal. Metrics distinguished are:

Premium: the relative difference of two quantities (if the value of alternative A is 20% more than the value of alternative B, the premium equals 0.2). A negative premium is a penalty.

Advantage: the natural logarithm of the ratio of two quantities (for mathematical convenience and ease of interpretation). A negative advantage is a disadvantage.

Incentive: normalized difference of two quantities to allow comparison of alternatives of variable scale. A negative incentive is a disincentive.

The structure of the NPV model with the breakdown into incentives, advantages and premiums is illustrated in Figure 2.

At the lowest level, two categories of premium metrics are distinguished:

Asset value premiums. Three variables influencing the asset value are considered, namely early market entry (*EEP*), product functionality (*PFP*) and product reliability (*PRP*).

Operational cost premiums. Two variables influencing the operational cost are considered, namely the short-term costs for corrective maintenance (*SMP*) and the long-term costs for adaptive/perfective maintenance (*LMP*).

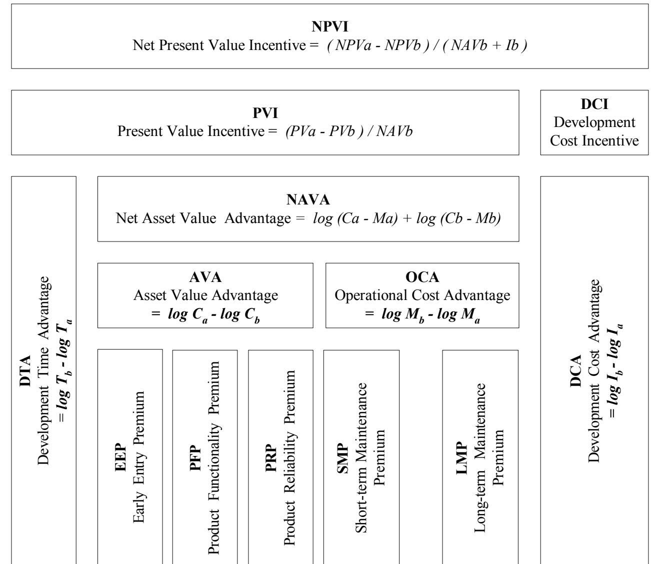

Figure 2: Breakdown of NPV Incentive [20]

The Asset Value Advantage *AVA* is equal to the expected increase in future cash inflows (difference between the two alternatives C_a and C_b) and is the contribution of the Early Entry Premium *EEP*, the Product Functionality Premium *PFP* and the Product Reliability Premium *PRP*.

The Operational Cost Advantage *OCA* is equal to the future cash outflows savings (difference between the two alternatives M_b and M_a) when the product is transferred to the operational phase and is the contribution of the Short-term Maintenance Premium *SMP* (corrective maintenance) and the Long-term Maintenance Premium *LMP* (adaptive/perfective maintenance).

The Asset Value Advantage and the Operational Cost Advantage are combined in the Net Asset Value Advantage *NAVA*.

The Present Value Incentive *PVI* is derived from the Net Asset Value Advantage *NAVA*, taking into account the discount rate r and normalizing it to the base alternative NAV_b.

The Development Cost Incentive *DCI* is the normalized difference of the development cost between the two alternatives I_b and I_a considered.

This leads to the final Net Present Value Incentive *NPVI*, normalized to the project scale:

$$\begin{aligned} NPVI &= (NPV_a - NPV_b) / (NAV_b + I_b) \\ &= (PV_a - I_a - PV_b + I_b) / (NAV_b + I_b) \\ &= (PVI \,.\, NAV_b + DCI \,.\, I_b) / (NAV_b + I) \end{aligned} \quad (2)$$

This *NPVI-method* enables a software manufacturer to evaluate and compare different release alternatives and therefore to determine the optimal release or market entry time. It requires however the availability of as complete and reliable as possible

information regarding the market window on one hand (asset value premium) and the product reliability and maintainability on the other hand (operational cost premium). In this paper, focus is on available models to make quantitative statements about the operational cost premium. This requires the capability of assessing reliability, influencing the short-term corrective maintenance cost, and maintainability, influencing both the short-term corrective maintenance cost and the long-term adaptive/perfective maintenance cost.

3. RELIABILITY

The crucial question during the testing phase of a product is: when can testing be stopped so the product can be released? Reliability, defined as the probability that a product will operate without failure under given conditions for a given time interval, is an important non-functional requirement to take into account when this question is raised. If testing, as the last project stage, is stopped too early, significant defects could be released to intended users and the software manufacturer could incur the post-release cost of fixing resultant failures later. In literature, two types of software reliability models are described, supporting a software manufacturer to make quantitative statements about reliability prior to a release decision [19]:

Software reliability prediction models (also referred to as quality management models) address the reliability of the software early in the life-cycle, at the requirements, design or coding level, using historical data. The reliability is, for example, predicted using fault density models and uses code characteristics, such as lines of code and nesting of loops, to estimate the number of faults in the software. Examples of such models are *Orthogonal Defect Classification* or *ODC* [2] and *COQUALMO* [3].

Software reliability estimation models (also referred to as reliability growth models) evaluate current and future reliability from faults, beginning with the integration, or system testing, of the software. The estimation is based on test data. These models attempt to statistically correlate defect detection data with known functions, such as an exponential function.

Although software reliability prediction models can be applied during the entire product development process, software reliability estimation models have been formulated to find the optimal release time for software products. These models have in common the support of the trade-off between three dimensions cost, time and quality during the test phase, i.e. when the project is nearing the release date. Most literature focuses on software reliability estimation models, evaluating current and future reliability from faults, beginning with the integration, or system testing, of the software. The estimation is based on test data. These models attempt to statistically correlate defect detection data with known functions, such as an exponential function.

These models take the general form [21]:

$$C(t) = c_1 . m(t) + c_2 . t + c_3 . [m(\infty) - m(t)] \quad (3)$$

With:

m(t): expected mean number of faults detected in time (0,t]

The usefulness of the software reliability estimation models is heavily criticized. Criticism is twofold:

Most models assume a way of working that does not reflect reality [16], meaning that the quality of assumptions is low. As a result, several models can produce dramatically different results for the same data set meaning that the predictive validity is limited [9] [6].

These models provide little support for determining the reliability of a software product due to many shortcomings. Studies show for instance that the number of pre-release faults is not a reliable indicator of the number of post-release failures [8]. The problem is that many software manufacturers use the pre-release fault count as a measure for the number of post-release failures, e.g. the reliability of the released product.

The lack of practical applicability of traditional verification approaches for non reliability, has led to the exploration of new approaches. Fenton and Neil argue that *Bayesian nets* offer a model that takes into account the crucial concepts missing from classical approaches [7] [17]. The nodes in the net represent uncertain variables and the arcs in the net represent causal/relevance relationships between the variables. Traditional approaches do not take these relationships into account, but focus on correlation between variables (e.g. size and defects). Although positive results have been reported [17], its practical application is assumed still to be limited for large and complex software products due to the multitude of interdependent variables and the excessive assessment burden, which might lead to informal, and indefensible, quantification of the modeled variables. Further research in this area is required to obtain more evidence.

Another relatively new approach to construct and present well reasoned arguments that a system achieves acceptable levels of safety, is the development of safety cases, where arguments are structured using a technique called *Goal Structuring Notation* or *GSN* [13]. This approach focuses on creating and documenting structured rationales that convincingly show how evidence gathered during system design and test, supports claims regarding not only safety but also other non-functional requirements like dependability, real-time performance, reliability and maintainability. Ongoing research is required here as well to investigate the practical application.

It is concluded that determining the reliability of a product using software reliability estimation models is difficult due to the lack of practically applicable models. A favourable choice should be to use software reliability prediction models instead, using historical data to make predictions of the expected defects densities in the different development phases. It requires however the availability of such historical data.

4. MAINTAINABILITY

Software reliability estimation models have received criticism from different angles. Two higher-order limitations regarding these models exist as well [20]:

Focus is on cash outflows, not on profit. The models only take into account cash outflows, assuming that minimizing total cash outflows is the main objective. However, in profit-oriented environments, for example, where software manufacturers sell products to their customers, the expected cash inflows should also be taken into account. In this case the optimal release time would not be determined by minimizing the total cash outflows but by maximizing the difference between cash inflows and cash outflows.

Focus is on pre-release testing versus post-release corrective cash outflows, not total cash outflows. Considering the total life-cycle cost of a software product, focus should not only be on the short-term operational cost for repairing failures (corrective maintenance cost), but also on the expected future cost for extending the product with additional functionality (adaptive and perfective maintenance cost). Important factors influencing the long-term maintenance cost are, for example, the quality of the product design (the extent to which maintainability requirements are addressed), the quality of the product realization (the extent to which maintainability requirements are correctly implemented), and the quality of the documentation supporting the product (the extent to which the product is documented in an accessible way: e.g. specifications, design, code, test cases, build procedures).

The *Maintainability Index* or *MI*, defined by Oman and Hagemeister, gives an indication of how maintainable a software product is [18]. Two equations are available; the second one takes into account the availability of comment in the code (assuming it has a positive influence on maintainability):

$$MI = 171 - 3.42 \ln(aveV) - 0.23\, aveV(g') - 16.2 \ln(aveLOC) \quad (4)$$

$$MI' = MI + 50 \sin \sqrt{(2.46\, perCM)} \quad (5)$$

With:

aveV: *average Halstead Volume per module (related to number of operators and operands used)*
aveV(g´): *average extended cyclomatic complexity per module (number of linearly independent test paths)*
aveLOC: *average lines of code per module*
perCM: *average percent of lines of comment per module*

However, one of the general problems is the lack of reliable metrics for software complexity – one of the main input drivers for estimation. Inputs like lines of code, function points and cyclomatic complexity all have severe limitations [14].

IEEE defines the *Software Maturity Index* or *SMI*, which provides an indication of the stability of a software product and can be used as a metric for planning software maintenance activities [10] [11]. As SMI approaches *1*, the product begins to stabilise. In a formula:

$$SMI = [\, M_t - (F_a + F_c + F_d) \,] / M_t \quad (6)$$

With:

M_t: *number of modules in the current release*
F_c: *number of changed mules in the current release*
F_a: *number of deleted modules in the current release*
F_d: *number of deleted modules in the current release*

This index cannot provide an accurate estimate of operational costs, and its main purpose is to demonstrate the evolution of a product over time.

5. CASE STUDIES

5.1 Introduction

The conclusion of the previous two sections is that proven models to determine the reliability and maintainability of a software product are limited. It was found that collecting and analyzing historical data from similar projects is probably a better instrument. With regard to reliability, it will support the use of software reliability prediction models to estimate pre-release development costs for further testing and the number of residual faults after product release. With respect to maintainability, it will support the estimation of expected post-release maintenance costs. The limited availability raises the question how software manufacturers make their release decisions in a practical context. How are estimated values for reliability and maintainability obtained in practice? Seven case studies were conducted. The selected environments varied with respect to the software manufacturer types (custom system written in-house versus commercial software), geographical locations (The Netherlands and Switzerland), the product version developed (new product versus new version of existing product), and the process maturity level (ranging from CMMI level 1 to 3). The obtained results are discussed in the next subsections (see [20] for a broader and more detailed overview and discussion). The presented results show to which extent reliability and maintainability are addressed and quantified during the:

specification phase as part of the (non-functional) product requirements;

design phase (deployment or breakdown of the specified requirements to the different subsystems and lower level components), and

testing phase (evaluation of the specified requirements).

5.2 Reliability

Specification phase: In all cases, reliability was addressed in the specifications as an important project objective. Only in some of the cases was reliability defined in quantitative terms.

Design phase: Only in case G was evidence found that, during the design or architecture phase, time and effort was spent to deploy reliability to identified components. This was however not done in quantitative terms. In case C, although not explicitly addressed and quantified during the design phase, very detailed design and coding rules were available with the objective of implicitly contributing to high reliability.

Testing phase: In all the cases was reliability evaluated prior to the release decision. Software reliability prediction and estimation models were not used, although one organization was investigating the application of ODC at that time. In none of the cases, the achieved reliability could be quantified.

Table 1. Case Study Results – Reliability [20]

Reliability	Spec.		Design		Testing	
Case	A	Q	A	Q	A	Q
A	+	-	-	-	+	-
B	+	+	-	-	+	-
C	+	+	-/+	-	+	-
D	+	-	-	-	+	-
E	+	+	-	-	+	-
F	+	+	-	-	+	-
G	+	+	+	-	+	-

Legend: A = addressed; Q = quantified

The results are summarized in Table 1.

5.3 Maintainability

Specification phase: In all cases, maintainability was addressed in the specifications as an important project objective. However, in none of the cases was maintainability defined in quantitative terms.

Design phase: Only in case G was evidence found that, during the design or architecture phase, time and effort was spent to deploy maintainability in identified components. This was however not done in quantitative terms. In case C, although not explicitly addressed and quantified during the design phase, very detailed design and coding rules were available with the objective of implicitly contributing to high product maintainability.

Testing phase: In none of the cases was product maintainability evaluated prior to the release decision. It was not addressed at all and was not expressed in quantitative terms. Only in case C was it verified that the detailed design and coding rules were followed, implicitly contributing to high product maintainability.

The results are summarized in Table 2.

Table 2. Case Study Results – Maintainability [20]

Maintainability	Spec.		Design		Testing	
Case	**A**	**Q**	**A**	**Q**	**A**	**Q**
A	+	-	-	-	-	-
B	+	-	-	-	-	-
C	+	-	-/+	-	-	-
D	+	-	-	-	-	-
E	+	-	-	-	-	-
F	+	-	-	-	-	-
G	+	-	+	-	-	-

Another important observation here is that the information regarding the availability of relevant documentation and the quality of this documentation was limited in several cases (A, B, E). This is expected to undermine the efficiency and effectiveness of correcting defects, or giving the product additional quality, especially when this discrepancy occurs during initial product development [4].

5.4 Conclusions

In the first place, it is concluded that software manufacturers face difficulties when deploying non-functional requirements to the level of components during the design phase and evaluating them once implemented. Available quality models like ISO/IEC 9126 [12] are of limited support here, a problem also recognized by for instance Kitchenham and Pfleeger [15]. Existing quality models share certain common problems:

They lack a rationale for determining the hierarchy (between for instance characteristics and sub-characteristics in ISO/IEC 9126) making it impossible to use the model as a reference to define all non-functional requirements.

There is no description of how the lowest level metrics (indicators in ISO/IEC 9126) can be used to evaluate non-functional requirements at a higher level.

Secondly, it is concluded that the estimate of post-release operational cost for short-term corrective activities and long-term product enhancements, prior to the release decision, is a difficult task due to problems in determining exact levels for the reliability level obtained, and the maintainability of the software product.

Thirdly, as a consequence, in none of the cases studied the expected post-release maintenance effort or operational cost premium could be quantified:

The reliability level was uncertain, making it difficult to (accurately) estimate the expected number of post-release defects.

The average effort or cost for correcting a defect was hardly known. This means that even when the reliability level could be quantified, the corrective maintenance would difficult to quantify.

The maintainability of the product was basically unknown, making it difficult, if not impossible, to state the extent to which a product can be further adapted, or perfected, in the future and the associated costs.

This often leads to situations where software is released prematurely with serious post-release problems. The case studies revealed (a combination of) the following non-analytical methods to decide when a software product is 'good enough' for release:

A 'sufficient' percentage of test cases run successfully.

Statistics are gathered about what code is exercised during the execution of a test suite.

Defects are classified and numbers and trends are analysed.

Real users conduct beta testing and report problems that are analysed.

Developers analyse the number of reported problems in a certain period of time. When the number stabilizes, or remains below a certain threshold, the software is considered 'good enough'.

Intuition seems to prevails, where as economic reasoning by sharing convincing information is required. Intuition on its own is not sufficient for software release decisions, especially in cases where large prospective financial loss outcomes to a software manufacturer and its customers/users are present.

6. NEXT STEPS

The NPVI-method presented in section 2 offers an instrument to evaluate and compare different release alternatives. However, to make it a good candidate for determining the optimal release time, information is required for the market window and the operational cost premium. This study revealed that software manufacturers are confronted with serious problems when trying to report the pre-release level of product reliability obtained and the expected post-release maintenance cost, based on the level of reliability and the maintainability of the resulting product. This hampers the economic reasoning about the optimal release time, where the decision-making process is characterized by sharing of convincing information. Further research is required into the following issues:

Deployment of non-functional requirements. It was concluded that existing quality models lack a rationale for determining the hierarchy and a description of how the lowest level metrics can be used to evaluate non-functional requirements at a higher level. This makes it difficult to address the deployment of non-functional requirements like reliability and maintainability requirements during the design phase. The

case studies confirmed that deployment is something software manufacturers either ignore or struggle with. It is therefore recommended to pursuit research in finding better ways to support this deployment process.

Evaluation of reliability requirements. It was concluded that traditional software reliability estimation models lack practical applicability. Most models assume a way of working that does not reflect reality. In none of the cases studied, such models were used. Therefore, future research should concentrate on revisiting the applicability of existing software reliability prediction models like COQUALMO and ODC and enforcing the development of new software reliability estimation approaches like Bayesian Nets and GSN. In addition, successful application of such approaches will require software manufacturers to collect and analyze historical data from different projects, thus enforcing the successful implementation of metrics programs as well.

Evaluation of maintainability requirements. It was concluded that there is a lack of models supporting the evaluation of the achieved level of maintainability. In none of the cases studied, models were found supporting these estimates. This hampers software manufacturers in making strong statements about the post-release maintenance costs (operational cost premium), both and the short-term (corrective) and the long-term (adaptive/perfective). It is recommended to look into ways to support the evaluation of the level of maintainability achieved.

In parallel, further research is planned regarding the applicability of the NPVI-method to determine its potential benefits in a practical context. This may possibly lead to further extensions and/or refinements of the method. It is assumed that in more mature environments information is increasingly perfect. Organizations interested in participation and/or contributions are invited to contact the authors.

7. REFERENCES

[1] Boehm, B.W., Sullivan, K.J., *Software Economics: A Roadmap.* ACM Press, 2000.

[2] Chillarege, R., et al., Orthogonal Defect Classification – A Concept for In-Process Measurements. *IEEE Transactions on Software Engineering, 18,* 11, 1992.

[3] Chulani, S., COQUALMO (COnstructive QUALity MOdel): A Software Defect Density Prediction Model. In *Project Control for Software Quality*, Kusters et al., (Eds.), Shaker Publishing, 1999.

[4] Cook, C., Visconti, M., New and improved documentation process model. *Proceedings of the 14th Pacific Northwest Software Quality Conference, Portland*, 1996, 364-380.

[5] Erdogmus, H., Comparative evaluation of software development strategies based on Net Present Value. *Proceedings of the First International Workshop on Economics-driven Software Engineering Research*, Toronto (Canada), 1999.

[6] Fenton, N.E., Pfleeger, S.L., *Software Metrics: A Rigorous & Practical Approach.* PWS Publishing Company, 1997.

[7] Fenton, N.E., et al., Assessing Dependability of Safety Critical Systems using Diverse Evidence. *IEEE Proceedings Software Engineering, 145,* 1, 1998, 35-39.

[8] Fenton, N.E., Neil, M., A Critique of Software Defect Prediction Research. *IEEE Transactions on Software Engineering, 25,* 5, 1999.

[9] Gokhale, S.S., et al., Important Milestones in Software Reliability Modeling. *Communications in Reliability, Maintainability and Serviceability,* SAE International, 1996.

[10] IEEE, IEEE Standard Dictionary of Measures to Produce Reliable Software. *IEEE Std. 982.1,* The Institute of Electrical and Electronics Engineers, 1988.

[11] IEEE, IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software. *IEEE Std. 982.1,* The Institute of Electrical and Electronics Engineers, 1988.

[12] ISO, *ISO/IEC 9126-1:2001 Software Engineering - Product Quality - Part 1: Quality model.* International Organization for Standardization, 2001.

[13] Kelly, T.P., *Arguing Safety.* PhD thesis, University of York (UK), 1998.

[14] Kemerer, C., Software complexity and software maintenance: a survey of empirical research. *Annals of Software Engineering I,* J.C.Baltzer AG, Science Publishers, 1995, 1-22.

[15] Kitchenham, B., Pfleeger, S.L., Software Quality: The Elusive Target. *IEEE Software, 13,* 1, 1996, 12-21.

[16] Li, P.L., et al., Selecting a defect prediction model for maintenance resource planning and software insurance. *Proceedings of the Fifth International Workshop on Economics-driven Software Engineering Research*, Oregon (USA), 2003.

[17] Neil, M., Fenton N., *Improved Software Defect Prediction.* European SEPG Conference, London (UK), 2005.

[18] Oman, P., Hagemeister, J., Constructing and testing of Polynomials Predicting Software Maintainability. *Journal of Systems and Software, 24,* 3, March, 1994.

[19] Reliability Analysis Center, *Introduction to Software Reliability: A State of the Art Review.* Reliability Analysis Center (RAC), 1996.

[20] Sassenburg, H., *Design of a Methodology to Support Software Release Decisions: Do the Numbers really Matter?* PhD thesis, University of Groningen (The Netherlands), 2006.

[21] Xie, M., *Software Reliability Modeling.* Singapore. World Scientific, 1991.