The Institute for Integrating Statistics in Decision Sciences

Technical Report TR-2016-9

Game and Decision Models for Software Testing

Fabrizio Ruggeri
CNR-IMATI, Italy

Refik Soyer
Department of Decision Sciences
The George Washington University, USA

Game and Decision Models for Software Testing

Fabrizio Ruggeri

CNR-IMATI

Refik Soyer

George Washington University

April 3, 2017

Synopsis

In this chapter we consider decision problems that arise during software testing/debugging process. We present game and decision-theoretic frameworks to develop optimal software testing strategies. We discuss optimal choice of release time for software under different scenarios: minimization of an objective function based on testing and failure costs as well as software reliability at release time, minimization of costs subject to the achievement of an acceptable reliability level and constrained simultaneous optimization of costs and reliability. We also present a game theoretic approach where a software implementer and a tester involved in an adversarial setting with both interested in producing quality software as well as maximizing their respective rewards.

1 Introduction

As described in the chapters by Rigdon, Zacks and Pena, software reliability models are often used for inference and prediction. Singpurwalla and Wilson (1999), among others, stated that software reliability models can also be used for decision making. The chapter by Wilson and

O'Riordain is an example: the authors are interested in determining the optimal time between software (Mozilla Firefox, in their case) releases based on bug detection data. The development of optimal testing strategies is an important component of software reliability modeling. In particular, it is important to find those strategies determining how long software should be tested and which test cases should be used, as pointed out by Singpurwalla and Wilson (1999). In this chapter we consider different approaches, starting from the most common one based on Decision Theory. In Section 2 we first present a decision model based on the optimization of an objective function which combines testing and repair costs (that is, costs before and after release), as well as software reliability at its release time. This is followed by a sequential decision model where corrections and modifications are made to the software at the end of each test stage with the hope of increasing its reliability. In Section 3 we present a game theoretic approach where two players, namely an Implementer and a Tester, are involved in an adversarial testing. The two parties are in adversaries, where maximization of individual payoffs are not necessarily leading to the best quality software. In Section 4 a multi-objective optimization approach is presented, where the interest of the software producer is not represented by a unique objective function as in Section 2 but the simultaneous optimization of costs and reliability. Finally, few concluding remarks are presented in Section 5, mostly based on a new paradigm, Adversarial Risk Analysis, which has been an area of increasing interest for researchers in recent years. The interested readers can find thorough illustration of Decision and Game Theory in many books, like French and Rios Insua (2000) and Barron (2013), respectively.

2 Decision Models in Software Testing

The choice of an optimal release time to stop testing of software has been addressed in many papers. We will provide a review of some of these contributions in the literature, including their merits and shortcomings. The interested reader is encouraged to refer to those works for

a thorough mathematical development and illustration.

The first works addressing the issue of stopping the testing process and releasing the software were not based on a formal decision-theoretic approach. Such works include Forman and Singpurwalla (1977, 1979), Okumoto and Goel (1980), Yamada, Narihisa and Osaki (1984) and Ross (1985). Dalal and Mallows (1986) were among the first who considered a decision theoretic approach to the problem, providing an exact but complicated solution as well as an asymptotic one. Singpurwalla (1989, 1991) also followed a Bayesian decision theoretic approach, considering a two-stage problem where the preposterior analysis requires complex computations. Later work by Ozekici and Catkan (1993) provided characterizations of the optimal release policy whereas McDaid and Wilson (2001) considered, from a Bayesian viewpoint, the case of a single stage testing using a nonhomogeneous Poisson process model. The latter authors also considered a sequential testing problem but the solution was not analytically tractable. Ozekici and Soyer (2001) considered optimal testing strategies for software with an operational profile as discussed in Musa (1993). It was assumed that the software was tested sequentially for a given time duration under each of the operations and optimal testing time for each operational profile was obtained by the authors using a Bayesian approach. Boland and Singh (2003) considered a geometric Poisson model for the release time, whereas Morali and Soyer (2003) considered the testing process as sequential decision problem in a Bayesian framework. provide characterizations of the optimal release policy. The work of the latter will be presented in Subsection 2.2.

2.1 Minimization of Expected Cost

The most common decision model in software testing considers the minimization of an objective function, i.e., an expected cost, which combines (sometimes conflicting) losses due to testing and repair costs as well as software reliability at release time. A particular, but a detailed

representation of such expected cost function at time t is provided by Li, Xie and Ng (2012)

$$E[C(t)] = c_0 + c_1 t^{\kappa} + c_2 \mu_y m(t) + c_3 \mu_w [m(t + t_w) - m(t)] + c_4 [[1 - R(x|t)], \tag{1}$$

where

- c_0 is the set-up cost for software testing
- c_1 is the cost of testing per unit testing time
- κ, $0 < \kappa \leq 1$, is the discount rate of testing cost over time
- c_2 is the cost of removing a fault per unit time during the testing phase
- c_3 is the cost of removing a fault per unit time during the warranty phase
- μ_y is the expected time to remove a fault during the testing phase
- μ_w is the expected time to remove a fault during the warranty phase
- t_w is the warranty period
- c_4 is the cost due to software failure
- $m(t)$ is the mean value function of the nonhomogeneous Poisson process (NHPP) describing the failure process
- $R(x|t) = e^{-[m(t+x)-m(t)]}$ is the software reliability at time t

Since fixing a fault during the warranty period is more expensive than during testing, $c_3 > c_2$. Furthermore the parameter κ models the learning process of the testing team. Many NHPPs can be chosen as illustrated in the chapter by Rigdon and Zacks.

The actual function to be minimized is obtained following a frequentist approach, by replacing parameters with their maximum likelihood estimators, whereas integration with respect to the posterior distribution of the parameters is needed when following the Bayesian paradigm.

Minimization of equations like (1), or its expectation in a Bayesian framework, leads to determination of the optimal release time. Although very practical, such approach is a cause of concerns for some authors since uncertainty could arise from both statistical estimation errors and misspecification of costs. The problem has been addressed, e.g., in Li, Xie and Ng (2010) who investigated sensitivity of the software release time through various methods, like one-factor-at-a-time approach, design of experiments and global sensitivity analysis. As an alternative, construction of credible (confidence, in a frequentist setup) intervals about optimal release times could be pursued like in Okamura, Dohi and Osaki (2011).

A constrained optimization problem arises when considering minimization of costs subject to the achievement of a minimal reliability level R_0. An example of such problem is given by the minimization of

$$E[C(t)] = c_0 + c_1 t^{\kappa} + c_2 \mu_y m(t) + c_3 \mu_w [m(t + t_w) - m(t)], \tag{2}$$

subject to $R(x|t) \geq R_0$.

The constrained optimization could also arise as a consequence of converting the problem to maximizing the reliability $R(x|t)$ subject to a constraint $E[C(t)] \leq C^*$, where C^* is the maximum allowable cost level.

Simultaneous optimization of multiple objectives will be discussed in Section 4.

2.2 A Sequential Decision Model

We now consider the protocol where testing is done sequentially in stages up to a fault detection or a pre-specified testing time, the one which occurs first. Software is then corrected and modified at the end of each test stage, aiming to increase its reliability. Let $T_i, i = 1, 2, \ldots,$ denote the (possibly censored) life-length of the software during the i-th testing stage, i.e. after the $(i-1)$-st modification has been made to it. Each T_i follows the same distribution but,

in general, with different parameters. In particular, we consider, for illustrative purposes, an exponential model with failure rate λ_i which changes from one stage to another as a result of the modifications made to it after each stage.

At the end of each stage, after modifying the software, a decision is taken about termination of the debugging process, based on the accumulated information $T^{(i)} = (T_i, T^{(i-1)})$, where $T^{(0)}$ is the available information before testing. Morali and Soyer (2003) assumed that the evolution of λ_i's was described by a Markovian model and considered, after each stage i, the loss function

$$\mathcal{L}_i(T^{(i)},\, \lambda_{i+1}) = \sum_{j=1}^{i} \mathcal{L}_{\mathrm{T}}(T_j) + \ \mathcal{L}_{\mathrm{S}}(\lambda_{i+1}), \tag{3}$$

where $\mathcal{L}_{\mathrm{T}}(\cdot)$ is the loss related to the life-length for each individual stage, and $\mathcal{L}_{\mathrm{S}}(\cdot)$ is the loss associated with stopping and releasing the software after the stage. The latter loss depends on the current value of the parameter (a proxy for the current software reliability) and it is an increasing function of this since the parameter is proportional to the number of bugs present in the software. It should be observed that the loss due to releasing software before any testing, i.e., $\mathcal{L}_0$, is just a function of λ_1.

Morali and Soyer (2003) presented the stopping problem as a sequential decision problem described by the m-stage decision tree given in Figure 1. The solution of the decision problem requires dynamic programming, taking expectation at random nodes and minimizing the expected loss at the decision nodes. At each decision node i, the additional expected loss associated with the STOP and the TEST decisions are given by $\mathrm{E}[\mathcal{L}_{\mathrm{S}}(\lambda_{i+1})\big|T^{(i)}]$ and $\mathrm{E}[\mathcal{L}_{\mathrm{T}}(T_{i+1})\big|T^{(\mathrm{i})}] + \mathrm{L}^*_{i+1}$, respectively, where

$$\mathrm{L}^*_i = \ \min\Big\{\mathrm{E}[\mathcal{L}_{\mathrm{S}}(\lambda_{i+1}) \mid T^{(i)}]\ ,\ \mathrm{E}[\mathcal{L}_{\mathrm{T}}(T_{i+1}) \mid T^{(i)}] + \mathrm{L}^*_{i+1}\ \Big\} \tag{23}$$

for $i = 0, 1, \ldots,$ It can be shown that the optimal decision at decision node i is the one associated

with L_i^*.

In Figure 1, the maximum number of testing stages, m, can be considered infinite with $L_{m+1}^* = \infty$. It is worth mentioning that even for the case of finite m the calculation of L_i^* in (23) is not trivial as it involves implicit computation of expectations and minimizations at each stage. Morali and Soyer (2003) studied the possibility of developing one-stage ahead optimal stopping rules by using results from van Dorp, Mazzuchi and Soyer (1997) and illustrated implementation of their approach using simulated as well as actual software failure data.

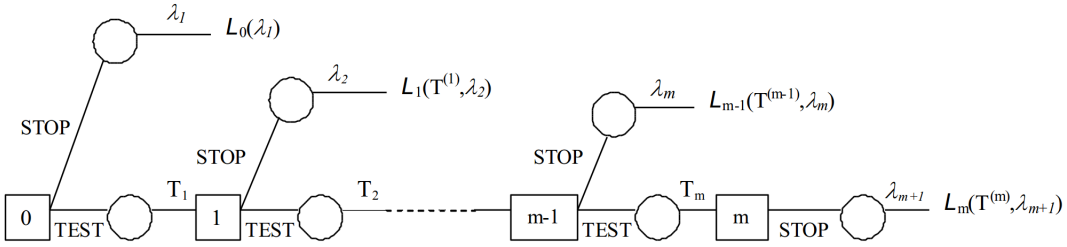

Figure 1: The m-stage decision tree for the optimal release problem

3 Games in Software Testing

Previously we have discussed approaches based either on optimal stopping rules minimizing the total cost in testing, possibly combined with specific reliability constraints, or on optimal allocation of testing efforts. In literature, minor emphasis has been placed on the aspect of competition between rival producers of software. The adversarial nature of the problem problem has been considered by Zeephongsekul and Chiera (1995) and Dohi, Teraoka and Osaki (2000) who used a game theoretic approach. The former authors consider (for simplicity) the case of only two competitors, labeled i, $i = 1, 2,$, which are producing software performing the same set of tasks and with life cycle length non exceeding T.

The player i, $i = 1, 2$, can decide to release the software at any time t in $[0, T]$ and he succeeds in selling the product with probability $A_i(t)$. The functions $A_i(t)$, $i = 1, 2$, are supposed to be

continuously differentiable, concave and such that $A_i(0) = A_i(T) = 0$ with a unique maximum at time η_i. The choice of such functions is made not only for mathematical convenience but it is also justified by the actual behavior. More specifically, the success probability is expected to be close to 0 both at the beginning and the end of the life cycle $[0, T]$, because of initial poor reliability and final obsolescence, respectively.

Zeephongsekul and Chiera (1995) make an assumption typical of game theory papers, i.e. that both players know the functions $A_i(t)$, $i = 1, 2$. Such assumption is sometimes unrealistic and it could be addressed within an Adversarial Risk Analysis (ARA) framework where each player has just guesses about the other's probabilities. In this case the problem would be seen from the viewpoint of one player, say 1, and she would have her opinion on the probability $A_1(t)$, either as a unique function or a distribution on it, whereas she should elicit a distribution on the space of the possible functions $A_2(t)$. More details on the ARA approach can be found in Banks, Rios and Rios Insua (2015).

Zeephongsekul and Chiera (1995) consider a cost function quite similar to the one proposed by Okumoto and Goel (1980), without assuming an infinite cycle length. They consider the expected cost $c_i(t)$ incurred by player i in releasing the software at time t as

$$c_i(t) = c_{1i}t + c_{2i}m(t) + c_{3i}\left(m(T) - m(t)\right), \tag{4}$$

where c_{1i} is the cost of testing per unit time, c_{2i} the cost of removing a fault during testing, c_{3i} the cost of removing a fault during operation and $m(t)$ the expected number of faults detected up to time t. Since fixing an error is more expensive after release than before it, then $c_{3i} > c_{2i}$ is assumed. Such assumption, combined with the choice of an increasing, concave and differentiable $m(t)$, with $m(0) = 0$, implies that the function $c_i(t)$ is convex with a minimum at

γ_i such that

$$m_i^{'}(\gamma_i) = \frac{c_{i1}}{(c_{3i} - c_{2i})}.$$

The authors assume that T is sufficiently large so that $\gamma_i < T$.

The novelty of the work by Zeephongsekul and Chiera is that they introduce the notion of competition where a player has to consider not only her cost but also the action of a competitor, and his costs. If player 1 releases software at time x and player 2 at time y, then $M_i(x, y)$ is the expected unit profit to player i. Such profit is the consequence of the difference between the unit price $p_i > 0$ of the software produced by player i and the cost incurred c_i given by (4). In general, it holds $M_1(x, y) \neq M_2(x, y)$, where

$$M_1(x,y) = \begin{cases} p_1 A_1(x) - c_1(x) & 0 \leq x < y \leq T \\ p_1(1 - A_2(y))A_1(x) - c_1(x) & 0 \leq y < x \leq T \end{cases}$$

and $M_2(x, y)$ can be described similarly. It is obvious that this is a non-zero sum game.

The simplifying assumption behind this model is that the success of a player in selling her product implies the impossibility for the other player to sell his own. The model could be acceptable when there is just one customer, interested in a unique purchase. More complex models could be possible in other scenarios, e.g. when the existence of many customers can provide opportunities for both players. A possibility could be offered by lowering the price of the last marketed software. Therefore p_i should be in this case a function of $A_{3-i}(x)$, $i = 12$.

The expected utility for each player depends on four factors:

- testing cost;
- fault removal cost (with an higher one for faults after release);
- reliability of the released product;
- release of software by the other player.

In general, late releases imply higher reliability and higher costs combined with the risk of an earlier successful release by the competitor.

The final goal of Zeephongsekul and Chiera (1995) consists of finding the optimal release policies among Nash equilibrium points. Dohi, Teraoka and Osaki (2000) found their solution restricted to just a particular case and computationally quite intractable and proposed a different approach addressing those issues.

Alternative approaches have been proposed in the literature. It is worth mentioning the work by Feijs (2001) who considered a game where the two players have very specific, distinct roles. One is an implementer (I) who is rewarded if she delivers an (almost) error-free piece of software, where the other is a tester (T) who is rewarded only if he performs a thorough testing job. Fejis considers an Idealized Testing Game (ITG) which is a two-player strategic game where each player has two choices about performing their jobs: *bad* (B) or *good* (G) quality. Intermediate quality levels are also possible, as discussed in Fejis (2001). A pair (x, y) of payoffs is associated with each combination of job quality, where x is the payoff for I and y for T. The payoff matrix is given by

	B	G
B	$(2,2)$	$(0,3)$
G	$(3,0)$	$(1,1)$

The payoffs $(0,3)$ and $(0,3)$ have a clear interpretation: this is the case where one player (I in the first case) is unable to detect faults whereas the other succeeds where fails. This explains why one player get the lowest payoff in the matrix and the other gets the highest. The payoffs $(2,2)$ and $(1,1)$ have a less evident explanation. The former pair corresponds to the case of bad quality job by both the player: they are of course penalized because unable to discover many faults but they are rewarded since they did not make significant efforts. The latter pair corresponds to the opposite situation: the players are rewarded because they were able to

discover many causes of faults but that occurred because of a large effort (and related cost).

The idea behind this model is that implementer I and tester T choose their performance level simultaneously. Once the problem has been structured and the software specification is available, I starts implementing the software whereas T starts looking for test cases and describing them thoroughly. In this way, the overall project duration is reduced as much as possible, leaving just the actual testing phase after the implementation of the software.

An alternative model, briefly described in Feijs (2001), corresponds to the case in which I chooses first the quality of her job and then T decides what to do after observing what I has done. This is another common situation where the tester can decide to make extra efforts if he believes the implementer did a poor job (or vice versa). Considering the previous payoff matrix, then I has two possible choices: B or G. In the former case T is left with just the first row of the matrix and he can choose between payoffs $(2, 2)$ and $(0, 3)$. Of course he would choose the second one, leaving I a payoff of 0. Should I choose good quality, then the choice of T would be between the payoffs $(3, 0)$ and $(1, 1)$, with an obvious preference for the second one. From the viewpoint of the implementer, she gets a payoff of 0 if she performs a bad quality job and 1 if the quality is good. Therefore, rational behavior leads to a pair of payoff $(1, 1)$ corresponding to good quality jobs by both players.

Those games, resembling the famous Prisoner's Dilemma, could be rethought in an ARA framework, as described earlier.

As pointed out in Feijs (2001), the actual software development is more complex and requires balancing between different aspects like development time, code-size, reusability, etc. Another critical aspect is the large number of possible test cases and the search for a restricted number of them which could lead to a significant improvement of the software quality and, at the same time, can be performed in a reasonable amount of time: good quality and quick delivery are often clashing goals!

Kukreja, Halfond and Tambe (2013) consider software testing as a security game where a Defender uses her limited resources for protecting public infrastructure (e.g. airports) from an Attacker (e.g. terrorist). In particular, they define a testing game in which the tester T is playing the role of Defender and the implementer I is the Attacker.

The tester T is willing to ensure high software quality and therefore, and is interested in developing a testing strategy which will execute the most efficient test cases, given constraints on resources and/or time constraints. Therefore, T is the Defender of the software quality, whereas I is treated as an Attacker who might produce a software full of bugs which could have a negative impact on its quality. The tester should detect such bugs before the software release. Implementers are Attackers not because they get a reward from bad quality software but because they might get credit for the quick development of the software rather than for a delayed one, even if due to careful testing. Kukreja, Halfond and Tambe (2013) associate utilities, for both players, to the test cases and compute a distribution that maximizes the tester payoff. Like in security games, the defender T may employ non-deterministic strategies, i.e., selecting a particular action with some probability. This use of a distribution decreases the predictability of T, making the task of the Attacker I harder.

4 Multi-objective Optimization in Software Testing

In Section 2.1 we formulated the optimization problem to determine the optimal release time under three different scenarios: an unconstrained problem where the objective function was depending on both (testing and failure) costs and software reliability and two constrained problems where either costs were minimized subject to the achievement of a minimal reliability level or, reliability was maximized subject to fixed cost level. Although simple to formulate, those approaches can hardly describe the management's attitude, especially the unconstrained problem where there is the issue of how much reliability should be weighted with respect to cost.

A more natural approach, although computationally more complex, consists of minimizing costs and maximizing reliability simultaneously. Therefore, the problem becomes a multi-objective optimization one where the goal is to find the optimal release time t^* solution of

$$\max_{t>0} R(x|t) \ \& \min_{t>0} E[C(t)], \tag{5}$$

where x is the useful life or warranty time of the software once released.

Different approaches have been presented in literature as discussed in Li, Xie and Ng (2012). A first approach is based on trade-off analysis which has the goal of identifying nondominated actions which are solutions to (5). In the current context, we say that an action a (in this case a release time) is nondominated if there is no other action b such that $R_b(x|t) \geq R_a(x|t)$ and $E[C_b(t)] \leq E[C_a(t)]$, with strict inequality for at least one of them. In this case subscripts describe which action we are referring to. The nondominated solutions, called also Pareto optimal solutions, are not inferior to any other solution. The approach simplifies the management's decision process since it reduces the search from all feasible solutions to a subset where a rational compromise can be made among the different solutions.

Multi-attribute utility theory (MAUT) addresses the problem of having different objectives in different scales and units. MAUT solves the problem considering weights and a single utility function. For each attribute $d_i, i = 1, \ldots, n$, (reliability and costs in our context) an utility function $u(d_i)$ is specified and then the multiattribute function

$$U(d_1, \ldots, d_n)0 \sum_{i=1}^{n} w_i u(d_i)$$

is considered, where $w_i, i = 1, \ldots, n$, are the importance weights assigned to each utility functions. In this way each attribute is converted to a value in $[0, 1]$ through its utility function where the weights, adding up to 1, determine the relative importance of each attribute. Methods for

elicitation of each utility function and the corresponding weight have been proposed, although it should be remarked that they are subject to the same sensitivity concerns discussed earlier. In practice, the utility is often chosen as a linear function for each attribute and management has just to provide upper and lower value on the corresponding attribute. On the other hand, the choice of the weights could be obtained comparing a certain scenario and a lottery. A thorough illustration of the approach can be found in Keeney and Raiffa (1976).

5 Conclusions

In the chapter we have presented several game and decision theoretic formulation of problems in software testing. We have not taken a firm position about the choice between a frequentist or a Bayesian approach, since we recognize merits of both of them. Nonetheless, we believe that the Bayesian approach can make better use of available information and preferences, providing also a more coherent theoretical approach. As mentioned in the chapter, Adversarial Risk Analysis (ARA) is an emerging field, thoroughly described in Bank, Rios and Rios Insua (2015), which could be successfully applied in this context.

6 Bibliography

Banks, D., Rios J. and Rios Insua D. (2015). *Adversarial Risk Analysis.* Chapman and Hall/CRC, Boca Raton,, FL, USA.

Barron, E.N. (2013). *Game Theory: An Introduction, 2nd Edition.* Wiley, Chichester, United Kingdom.

Boland, P.J. and Singh, H. (2002). Determining the optimal release time for software in the geometric Poisson reliability model. *International Journal of Reliability Quality and Safety Engineering*, 9, 201-213.

Dalal, S.R. and Mallows, C.L. (1986). When should one stop testing software? *Journal of the American Statistical Association*, 83, 872-879.

Dohi, T., Teraoka, Y. and Osaki, S. (2000). *Journal of Optimization Theory and Applications*, 105, 325-346.

Feijs, L. (2001). Prisoner's dilemma in software testing. *Proceedings of 7e Nederlandse Testdag*, 65-80. TU/e, Eindhoven, The Netherlands.

Forman, E.H. and Singpurwalla N.D. (1977). An empirical stopping rule for debugging and testing computer software. *Journal of the American Statistical Association*, 72, 750-757.

Forman, E.H. and Singpurwalla N.D. (1979). Optimal time intervals for testing hypotheses on computer software errors. *IEEE Transactions on Reliability*, R-28, 250-253.

French, S. and Rios Insua, D. (2000). *Statistical Decision Theory: Kendall's Library of Statistics 9*. Arnold, London, United Kingdom.

Kukreja, N., Halfond, W.G.J. and Tambe, M. (2013). Randomizing Regression Tests Using Game Theory. *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering (ASE) – New Ideas track*.

Li, X., Xie, M. and Ng, S.H. (2010). Sensitivity analysis of release time of software reliability models incorporating testing effort with multiple change-points. *Applied Mathematical Modelling*, 34, 3560-3570.

Li, X., Xie, M. and Ng, S.H. (2012). Multi-objective optimization approaches to software release time determination. *Asia-Pacific Journal of Operational Research*, 29, 1240019-1 - 1240019-19.

McDaid, K. and Wilson, S.P. (2001). Deciding how long to test software. *Statistician*, 50, 117-134.

Morali, N. and Soyer, R. (2003). Optimal Stopping in Software Testing. *Naval Research Logistics*, 50, 88-104.

Musa, J. D. (1993). Operational profiles in software reliability engineering. *IEEE Software*, 10, 14-32.

Okamura, H., Dohi, T. and Osaki, S. (2011). Bayesian Inference for Credible Intervals of Optimal Software Release Time. *Software Engineering, Business Continuity, and Education* (Kim T. et al. Eds). Springer, Berlin, Germany.

Okumoto, K. and Goel, A.L. (1980). Optimum release time for software systems, based on reliability and cost criteria. *Journal of Systems and Software*, 1, 315-318.

Ozekici, S. and Catkan N.A. (1993). A dynamic software release model. *Computational Economics*, 6, 77-94.

Ozekici, S. and Soyer, R. (2001). Bayesian testing strategies for software with an operational profile. *Naval Research Logistics*, 48, 747-763.

Ross, S.M. (1985). Software reliability: the stopping rule problem. *IEEE Transactions on Software Engineering*, SE-11, 1472-1476.

Singpurwalla, N.D. (1989). Preposterior analysis in software testing. *Statistical Data Analysis and Inference (Dodge Y. Ed.)*. Elsevier, New York, NY, USA.

Singpurwalla, N.D. (1991). Determining an optimal time interval for testing and debugging software. *IEEE Transactions on Software Engineering*, SE-17, 313-319.

Singpurwalla N.D. and Wilson S.P. (1999). *Statistical Methods in Software Engineering*. Springer-Verlag, New York, NY, USA.

van Dorp, J.R, Mazzuchi, T.A. and Soyer, R. (1997). Sequential inference and decision making during product development. *Journal of Statistical Planning and Inference*, 62, 207-218.

Yamada, S., Narihisa, H. and Osaki, S. (1984). Optimum release policies for a software system with a scheduled software delivery time. *International Journal of Systems Science*, 15, 905-914.

Zeephongsekul, P. and Chiera, C. (1995). Optimal Software Release Policy Based on a Two-Person Game of Timing. *Journal of Applied Probability*, 32, 470-481.