# A Risk Assessment Model for Evolutionary Software Projects[1]

Luqi, J. Nogueira
Naval Postgraduate School
Monterey CA 93943 USA

**Abstract**

Current early risk assessment techniques rely on subjective human judgments and unrealistic assumptions such as fixed requirements and work breakdown structures. This is a weak approach because different people could arrive at different conclusions from the same scenario even for projects with a stable and well-defined scope, and such projects are rare. This paper introduces a formal model to assess the risk and the duration of software projects automatically, based on objective indicators that can be measured early in the process. The model has been designed to account for significant characteristics of evolutionary software processes, such as requirement complexity, requirement volatility and organizational efficiency. The formal model based on these three indicators estimates the duration and risk of evolutionary software processes. The approach supports (a) automation of risk assessment and, (b) early estimation methods for evolutionary software processes.

## 1. Introduction

Software applications have grown in size and complexity covering many human activities of importance to society. The report of the President's Information Advisory Committee calls software the "new physical infrastructure of the information age". Unfortunately, the ability to build software has not increased proportionately to demand [Hall, 1997. pp xv], and shortfalls in this regard are a growing concern. According to the Standish group, in 1995 84% of software projects finished over time or budget, and $80 billion - $100 billion is spent annually on cancelled projects in the US. Developing software is still a high-risk activity.

There have been many approaches to improving this situation, mostly focused on increasing productivity via improvements in technology or management. Although better productivity is certainly welcome, closer examination shows that these efforts address only half of the problem. A project gets over time or over budget if actual performance does not match estimates. Current estimation techniques are far from reliable, and tend to systematically produce overly optimistic estimates. More accurate early estimates could help reduce wasted resources associated with overruns and cancelled projects in two ways: if costs are known to be too high at the outset, the scope of the project could be reduced to enable completion within time and budget, or it could be cancelled before it starts, and instead the resources could be used to successfully complete other feasible projects.

This paper therefore focuses on improved risk assessment for software projects. We address project risks related to schedule and budget, and focus mostly on completion time of the project. Current risk assessment standards are weak because they rely on subjective human expertise, assume frozen requirements, or depend on metrics difficult to measure until it is too late. This paper describes a formal risk assessment model based on metrics and sensitive to requirements volatility. Further details can be found in [Nogueira 2000]. The model is specially suited for evolutionary prototyping and incremental software development.

Section 2 defines the problem we are addressing. Section 3 analyzes relevant previous work. Section 4 presents and evaluates our project risk model. Section 5 outlines how systematic risk assessment fits into iterative prototyping. Section 6 concludes.

[1] This research was supported in part by the U. S. Army Research Office under contract/grant number 35037-MA and 40473-MA, and in part by DARPA under contract #99-F759.

## 2. The Problem

As the range and complexity of computer applications have grown, the cost of software development has become the major expense of computer-based systems [Boehm 1981], [Karolak 1996]. Research shows that in private industry as well as in government environments, schedule and cost overruns are tragically common [Luqi 1989, Jones 1994, Boehm 1981]. Despite improvements in tools and methodologies, there is little evidence of success in improving the process of moving from the concept to the product, and little progress has been made in managing software development projects [Hall, 1997]. Research shows that 45 percent of all the causes for delayed software deliveries are related to organizational issues [vanGenuchten 1991]. A study published by the Standish Group reveals that the number of software projects that fail has dropped from 40% in 1997 to 26% in 1999. However, the percentage of projects with cost and schedule overruns rose from 33% in 1997 to 46% in 1999 [Reel 1999].

Despite the recent improvements introduced in software processes and automated tools, risk assessment for software projects remains an unstructured problem dependent on human expertise [Boehm 1988, Hall 1997]. The acquisition and development communities, both governmental and industrial, lack systematic ways of identifying, communicating and resolving technical uncertainty [SEI 1996].

This paper explores ways to transform risk assessment into a structured problem with systematic solutions. Constructing a model to assess risk based on objectively measurable parameters that can be automatically collected and analyzed is necessary. Solving the risk assessment problem with indicators measured in the early phases would constitute a great benefit to software engineering. In these early phases, changes can be made with the least impact on the budget and schedule. The requirements phase is the crucial stage to assess risk because: a) it involves a huge amount of human interaction and communication that can be misunderstood and can be a source of errors; b) errors introduced at this phase are very expensive to correct if they are discovered late; c) the existence of software generation tools can diminish the errors in the development process if the requirements are correct; and d) requirements evolve introducing changes and maintenance along the whole life cycle.

Part of the problem is misinterpreting the importance of risk management. It is usually and incorrectly viewed as an additional activity layered on the assigned work, or worse, as an outside activity that is not part of the software process [Hall 1997, Karolak 1996]. One of the goals of our research is to integrate a risk assessment model with previous research on CAPS[2] at NPS [Harn 99]. This integration is required in order to capture metrics automatically in the context of a modern evolutionary prototyping and software development process. This should provide project managers with a more complete tool that can enable improved risk assessment without interfering with the work of a projectÕs software engineers.

A second source of problems in risk management is the lack of tools [Karolak 1996]. The main reason for this lack of tools is that risk assessment is apparently an unstructured problem. To systematize unstructured problems it is necessary to define structured processes. Structured processes involve routine and repetitive problems for which a standard solution exists. Unstructured processes require decision-making based on a three-phase method (intelligence, design, choice) [Turban et al 1998]. An unstructured problem is one in which none of the three phases is structured. Current approaches to risk management are highly sensitive to managersÕ perceptions and preferences, which are difficult to represent by an algorithm. Depending on the decision-maker's attitude towards risk, he or she can decide early with little information, or can postpone the decision, gaining time to obtain more information, but losing some control.

A third source of risk management problems is the confusion created by the informal use of terms. Often, the software engineering community (and most parts of the project management

[2] CAPS stands for Computer Aided Prototyping System [Luqi 1988].

community [Wideman 1992]) uses the term "risk" casually. This term is often used to describe different concepts. It is erroneously used as a synonym of "uncertainty" and "threat" [SEI 1996, Hall 1997, Karolak, 1996]. Generally, software risk is viewed as a measure of the likelihood of an unsatisfactory outcome and a loss affecting the software from different points of view: project, process, and product [Hall 1997, SEI 1996]. However, this definition of risk is misleading because it confounds the concepts of risk and uncertainty. In general, most parts of decision-making in software processes are under uncertainty rather than under risk. Uncertainty is a situation in which the probability distribution for the possible outcomes is not known.

In this paper the term "risk" is reserved to indicate the probabilistic outcome of a succession of states of nature, and the term "threat" is used to identify the dangers that can occur. We define risk to be the product of the value of an outcome times its probability of occurrence. This outcome could be either positive (gain) or negative (loss). This abstraction permits one to address not only the classical risk management issue, but also to discover opportunities leading to competitive advantage.

We address the issue of risk assessment by estimating the probability distribution for the possible outcomes of a project, based on observed values of metrics that can be measured early in the process. The metrics were chosen based on a causal analysis to identify the most important threats and a statistical analysis to choose the shape of the probability distribution and relate its parameters to readily measurable metrics.

## 3. Related Work

There are three main groups of research related to risk:

- **Assessing Software Risk by Measuring Reliability.** This group follows a probabilistic approach and has successfully assessed the reliability of the product [Lyu 1995, Schneidewind 1975, Musa 1998]. However, this approach addresses the reliability of the product, not the risk of failing to complete the project within budget and schedule constraints. These approaches could be used to assess risks related to failures of software projects, which are outside the scope of the current paper. A concern with these approaches is that the resulting assessments arrive too late to economically correct possible faults, because the software product is mostly complete and development resources are mostly gone at the time when reliability of the product can be assessed by testing.

- **Heuristic approaches:** Other researchers assess the risk from the beginning, in parallel with the development process. However, these approaches are less rigorous, typically subjective and weakly structured. Basically these approaches use lists of practices and checklists [SEI, 1996, Hall 1997, Charette 1997, Jones 1994] or scoring techniques [Karolak 1996]. Paradoxically, SEI defines software technical risk as a measure of the probability and severity of adverse effects in developing software that does not meet its intended functions and performance requirements [SEI, 1996]. However, the term "probability" is misleading in this case because the probability distribution is unknown.

- **Macro Model Approaches:** A third group of researchers uses well known estimation models to assess how risky a project could be. The widely used methods COCOMO [Boehm 1981], and SLIM [Putnam, 1980] both assume that the requirements will remain unchanged, and require an estimation of the size of the final product as input for the models [Londeix 1987]. This size cannot be actually measured until late in the project.

The standard tools used to control all types of projects, including PERT, CPM, and Gantt, do not consider coordination and communication overhead. Such models represent sequential interdependencies through explicit representation of precedence relationships between activities. This simplified vision of a project cannot address the dynamics created by reciprocal requirements of information in concurrent activities, exception management, and the impact of

actor interactions. Since the missing factors increase time requirements, the estimates resulting from these generic project estimation models are overly optimistic.

These issues are addressed by Vit Project [Levitt 1999, Thomsen et al. 1999]. Vit Project is applicable to projects in which a) all activities in the project can be predefined; b) the organization is static, and all activities are pre-assigned to actors in the static organization; c) the exceptions to activities result in extra work volume for the predefined activities and are carried out by the pre-assigned actors; and d) actors are assumed to have congruent goals. The model is well suited for simulating organizations that deal with great amounts of information processing and coordination. Such characteristics are extremely relevant in software processes [Boehm, 1981]. However, this approach requires a fixed work breakdown structure, and therefore does not apply at the early stages when requirements are changing and the set of tasks comprising the project are still uncertain.

By using informal risk assessment models, using estimation models based on optimistic assumptions that require parameters difficult to provide until late, and using optimistic project control tools, project managers condemn themselves to overrun schedules and cost.

## 4. The Proposed Project Risk Model

Our approach is based on metrics automatically collectable from the engineering database from near the beginning of the development. The indicators used are Requirements Volatility (RV), Complexity (CX), and Efficiency (EF).

*Requirement Volatility (RV):* RV is a measure of three characteristics of the requirements: a) the Birth-Rate (BR), that is the percentage of new requirements incorporated in each cycle of the evolution process; b) the Death-Rate (DR), that is the percentage of requirements dropped in each cycle; and c) the Change-Rate (CR) defined as the percentage of requirements changed from the previous version. A change in one requirement is modeled as a birth of a new requirement and the death of another, so that CR is included in the measured values of BR and DR. RV is calculated as follows: RV = BR + DR.

*Complexity (CX):* Complexity of the requirements is measured from a formal specification. A requirements representation that supports computer-aided prototyping, such as PSDL [Luqi 1996], is useful in the context of evolutionary prototyping. We define a complexity metric called Large Granularity Complexity (LGC) that is calculated as follows: LGC = O + D + T, where for PSDL O is the number of atomic operators (functions or state machines), D is the number of atomic data streams (data connections between operators), and T is the number of abstract data types required for the system. Operators and data streams are the components of a dataflow graph. This is a measure of the complexity of the prototype architecture, similar in spirit to function points but more suitable for modeling embedded and real-time systems. The measure can also be applied to other modeling notations that represent modules, data connections, and abstract data types or classes. We found a strong correlation between the complexity measured in LGC and the size of PSDL specifications (correlation coefficient R = 0.996). Most important, we also found a strong correlation (R = 0.898) between the complexity measured in LGC and the size of the final product expressed in non-comment lines of Ada code, including both the code automatically created by the generator and the code manually introduced by the programmers.

*Efficiency (EF):* The efficiency of the organization is measured using a direct observation of the use of time. EF is calculated as a ratio between the time dedicated to direct labor and the idle time: EF = Direct Labor Time / Idle Time. We found that this easily measurable quantity was a good discriminator between high team productivity and low team productivity in a set of simulated software projects [Nogueira 2000].

We validated and calibrated our model with a series of simulated software projects using Vit Project. This tool was chosen because of the inclusion of communications and exceptions in its project dynamics model, and because it has been extensively validated for many types of engineering projects, including software engineering projects. The input parameters for the simulated scenarios were RV, EF and CX, and the observed output was the development time. Given that the proposed model uses parameters collected during the early phases and given that Vit Project requires a complete breakdown structure of the project, which can be done only in the late phases, there was a considerable time gap between the two measurements. This time gap is less than for a post-mortem analysis, but it is sufficient for model calibration and validation purposes.

The simulation results were analyzed statistically, with the finding that the Weibull probability distribution was the best fit for all the samples. A random variable x is said to have a Weibull distribution with parameters $\alpha$, $\beta$ and $\gamma$ (with $\alpha > 0$, $\beta > 0$) if the probability distribution function (pdf) and cumulative distribution function (cdf) of x are respectively:

$$\text{pdf: } f(x;\alpha,\beta,\gamma) = \begin{cases} 0, & x < \gamma \\ (\alpha/\beta^{\alpha})\,(x-\gamma)^{\alpha-1}\exp(-((x-\gamma)/\beta)^{\alpha}), & x \geq \gamma \end{cases}$$

$$\text{cdf: } F(x;\alpha,\beta,\gamma) = \begin{cases} 0, & x < \gamma \\ 1 - \exp(-((x-\gamma)/\beta)^{\alpha}) & x \geq \gamma. \end{cases}$$

The random variable under study, x, can be interpreted as development time in our context. The shape parameter $\alpha$ controls the skew of the pdf, which is not symmetric. We found that this is mostly related to the efficiency of the organization (EF). The scale parameter $\beta$ stretches or compresses the graph in the *x* direction. We found that this parameter is related to the efficiency (EF), requirements volatility (RV), and complexity (CX) measured in LGC. The shifting parameter $\gamma$ is shifts the origin of the curves to the right. We found that it is mostly related to the complexity measured in LGC.

Based on best fit to our simulation results, the model parameters can be derived from the project metrics using the following algorithm:

```
If (EF > 2.0)    then  α = 1.95;
                       γ = 22 * 0.32*(13*ln(LGC)–82);
                       β = γ /(5.71+(RV-20)*0.046);
                 else  α = 2.5;
                       γ = 22 * 0.85*(13*ln(LGC)–82);
                       β = γ /(5.47-(RV-20)*0.114);
                 end if;
```

The model estimates the following cumulative probability distribution for project completion on or before time x:

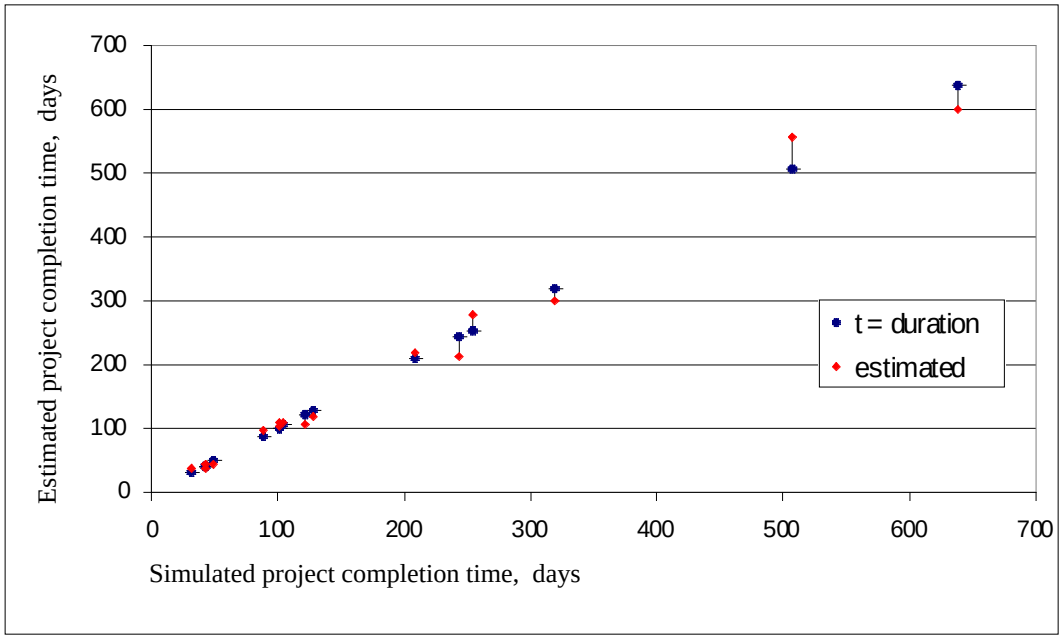
$$P(x) = 1 - \exp(-(((x-\gamma)/\beta)^{\alpha})) \quad \text{// where x is time in days}$$

This equation can be inverted to obtain the schedule length needed to have a probability P of completing within schedule, with the following result.

$$x = \gamma + \beta(-\ln(1-P))^{1/\alpha}$$

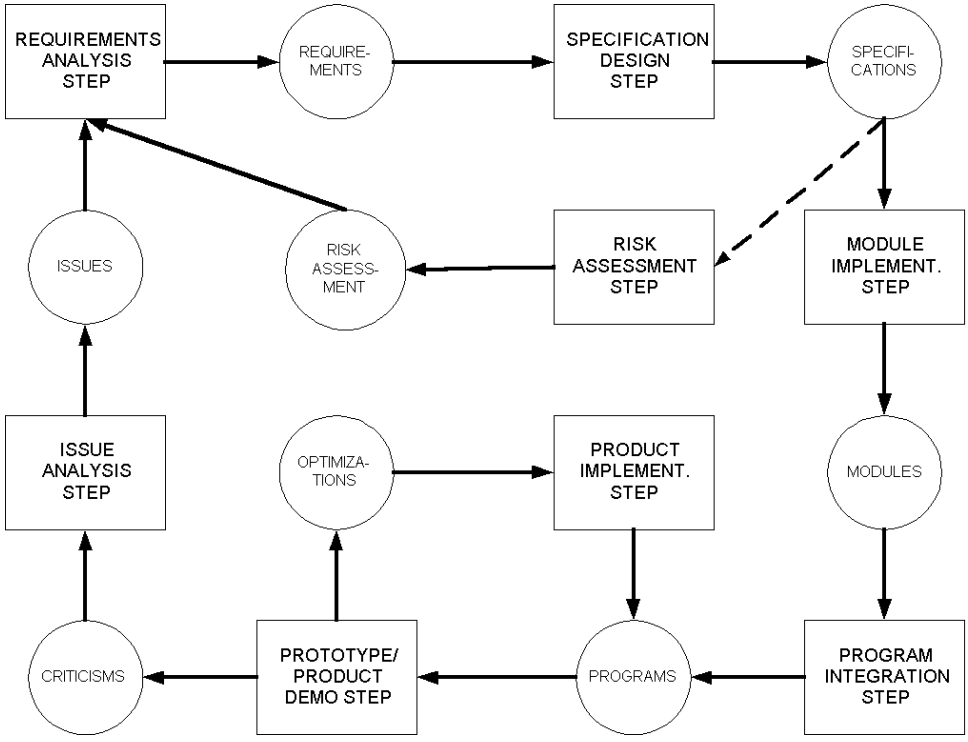
The probability P can be interpreted as a degree of confidence in the ability of the project to successfully complete within a schedule of length x. Applying the above equation to estimate the development time needed for a 95% chance of completion within schedule for 16 different

scenarios simulated using Vit Project, we observed a standard error of 22 days. The worst case was an error of 60 days for a project of 520 days (12%). The comparison of estimated time and simulated time is shown below.




## 5. Integrating Risk Assessment into Prototyping

The model presented in the previous section is designed to support an iterative prototyping and software development process. In this process, an initial problem statement, a prototype demo or problem reports from a deployed software product trigger an issue analysis, followed by formulation of proposed requirements changes, and specification of a proposed adjustment to the software requirements, which can be initially empty. At this point in each cycle, the project manager should perform a risk assessment step. The results of the risk assessment step guide the degree of detail to which requirements enhancements are demonstrated, and the set of requirements issues to be considered in the next prototyping cycle, if any.



The first measurement-based risk assessment step can be performed after specification of the first version of the prototype architecture, based on the requirements volatility, LGC and efficiency measurements from the steps just performed.

In cases where risk assessments are required even earlier, before any prototyping has been done, estimates of team efficiency and requirements volatility can be based on measurements of similar past projects, and initial complexity estimates can be based on subjective guesswork of the kind currently used in the macro model approaches. This kind of estimate may be less reliable than those based solely on measurements, but it can provide a principled and reasonably accurate basis for deciding whether or not to start a prototyping process to determine the requirements for a proposed development project. Thus parts of our approach can be used truly at the very beginning of the process.

If a prototyping effort is approved, early measurements of the process could be used to refine the initial estimates of the model parameters using Bayesian methods, thus providing a balanced and systematic transition from subjective guesswork, coded as an *a priori* distribution, to assessments increasingly based on systematic measurement. Such an approach also supports incorporation and systematic refinement of measurements from previous cycles of the iterative prototyping process.

The results of risk assessment can provide guidance on the degree to which the project can afford to explore requirements enhancements requested by the customers. It can also help customers or marketing departments to decide how much they really want possible improvements, in the context of the resulting time and cost estimates. Systematic cost/benefit analysis becomes possible only with the availability of reasonably accurate estimates.

The risk assessment step can thus provide a balancing force to stabilize the requirements formulation process. In the absence of information on how much potential enhancements will cost, stakeholders are prone to unrealistic requirements amplification — of course they would always like to have a better system, no matter how good the existing one is, if you do not ask them to pay for the improvements. The proposed risk assessment steps can provide a realistic basis for incorporating time and cost constraints and cost/benefit tradeoffs early in the process, when the situation is fluid and many options are open.

This process refinement provides some additional insight into the dynamics of iterative prototyping: the iterative process should stop when the customers have determined what requirements they can afford to realize, and which of many possible improvements they will be willing to pay for, if any. It is not necessarily the case that the set of criticisms elicited by the final round of prototype demonstrations is empty — that is true only in an idealized world with adequate budgets and patient customers.

## 6. Conclusion

This paper introduces a formal risk assessment model for software projects based on probabilities and metrics automatically collectable from the project baseline. The approach enables a project manager to evaluate the probability of success of the project very early in the life cycle, during an iterative requirements formulation process, based on well-defined measurements rather than just guesswork or subjective judgments.

For more than twenty years, estimation standards have been characterized by a common limitation: the requirements should be frozen in order to make estimates. This model presented in this paper removes this important limitation, facing the reality that requirements are inherently variable.

The model is perfectly suited for any evolutionary software process because it follows the same philosophy. The risk assessment and estimation steps are conducted at each evolutionary cycle with increasing knowledge and decreasing variance. The research formalizes an

improvement in the evolutionary software process, introducing a risk assessment step that can be automated, and that can help shape the planning of the project in the early stages when there is still substantial freedom to allocate available time and budget.

**References**


[Boehm 1981] B. Boehm, *Software Engineering Economics*, Prentice Hall, 1981.

[Boehm 1988] B. Boehm, A Spiral Model of Software Development and Enhancement, *Computer*, May 1988.

[Charette 1997] R. Charette, K. Adams, & M. White, Managing Risk in Software Maintenance, *IEEE Software*, May-June, 1997.

[Gilb 1977] T. Gilb, *Software Metrics*, Winthrop Publishers, Inc., 1977.

[Hall 1997] E. Hall, Managing Risk, *Methods for Software Systems Development*, Addison Wesley, 1997.

[Harn 1999] M. Harn, V. Berzins, Luqi, Computer-Aided Software Evolution Based on a Formal Model, *Proceedings of the Thirteenth International Conference on Systems Engineering*, Las Vegas, Nevada, August 9-12, 1999, pp. CS: 55-60.

[Jones 1994] C. Jones, *Assessment and Control of Software Risks*, Yourdon Press Prentice Hall, 1994.

[Karolak 1996] D. Karolak, Software *Engineering Management*, IEEE Computer Society Press, 1996.

[Levitt 1999] R. Levitt, *The ViteProject Handbook: A User's Guide to Modeling and Analyzing Project Work Processes and Organizations*, Vit ' 1999.

[Londeix 1987] B. Londeix, *Cost Estimation for Software Development*, Addison-Wesley, 1987.

[Luqi 1988] Luqi, M. Ketabchi, A Computer Aided Prototyping System, *IEEE Software*, Vol. 5, No. 2, p. 66-72, March 1988.

[Luqi 1989] Luqi, Software Evolution Through Rapid Prototyping, *IEEE Computer*, May 1989.

[Luqi 1996] Luqi, Special Issue: Computer-Aided Prototyping, *Journal of Systems Integration*, Vol. 6, Nos. 1-2, March 1996.

[Lyu 1995] M. Lyu, *Software Reliability Engineering*, IEEE Computer Society Press. 1995.

[Musa 1998] J. Musa, Software *Reliability Engineering: More Reliable Software, Faster Development and Testing*, McGraw-Hill, 1998.

[Nogueira 2000] J. Nogueira, *A Formal Risk Assessment Model for Software Projects*, Ph.D. Dissertation, Naval Postgraduate School, 2000.

[Putnam 1980] L. Putnam, *Software Cost Estimating and Life-cycle Control: Getting the Software Numbers*, IEEE Computer Society Press, 1980.

[Reel 1999] J. Reel, *Critical Success Factors in Software Projects*, IEEE Software, May - June 1999.

[SEI 1996] Software Engineering Institute, Software Risk Management, Technical Report CMU/SEI-96-TR-012, June 1996.

[Schneidewind 1975] N. Schneidewind, Analysis of Error Processes in Computer Software, *Proceedings of the International Conference on Reliable Software*, IEEE Computer Society, 21-23 April 1975, p 337-346.

[Turban et al 1998] E. Turban and J. Aronson, *Decision Support Systems and Intelligent Systems*, Prentice Hall, 1998.

[vanGenuchten 1991] M. van Genuchten, Why is Software Late? An Empirical Study of the Reasons for Delay in Software Development, *IEEE Transactions on Software Engineering*, June, 1991.

[Wideman 1992] R. Wideman, *Risk Management: A Guide to Managing Project Risk Opportunities*, Project Management Institute, 1992.