

A Formal Risk Assessment Model for Software Evolution*

Juan C. Nogueira

Luqi

Valdis Berzins

Nader Nada

Naval Postgraduate School

2, University Circle.

Monterey, CA. 93943 USA

+1 833 656 2093

jcnoguei@nps.navy.mil

ABSTRACT

This paper introduces a formal method to assess the risk and the duration of software projects automatically. The method has been designed according the characteristics of evolutionary software processes. We introduce a set of metrics to measure productivity, requirement volatility and complexity. We construct a formal method based on these three indicators to estimate the duration and risk of evolutionary software processes. The approach introduces benefits in two fields: a) automation of risk assessment and, b) early estimation method for evolutionary software processes.

Keywords

Risk, software metrics, estimation models

1 INTRODUCTION

Despite progress in formal methods, prototyping, and evolutionary software processes, risk assessment remains as an open issue dependent on human expertise. Software development processes such the hypergraph model for software evolution [15], or the spiral model [3], have a common weakness: risk assessment. In the software evolution domain, risk assessment has not been addressed as part of the model. In the various enhancements and extensions, the graph model did not include risk assessment steps, hence risk management remains as a human-dependent activity that requires expertise. In the evaluation of the spiral model, one of the difficulties mentioned by

Boehm was: *"Relying on risk-assessment expertise, the spiral model places a great deal of reliance on the ability of software developers to identify and manage sources of project risk."* [3].

Many researches [9, 6, 20] have addressed the problem of risk assessment following guidelines, checklists, taxonomies of risk factors, and few metrics. All these methods work fine if a) they are applied by a human educated on risk assessment, and b) he/she has enough experience. The weakness of all current risk assessment practices is human dependency. As a corollary, risk assessment could not be consistent because different experts could arrive at different conclusions from the same scenario.

Our research is focused on transforming the present state of the art about risk assessment into a formal method. This paper introduces an automated and formal software project risk assessment model, based on early metrics and probabilities designed for evolutionary software processes.

2 THE PROBLEM

Studies have shown that early parts of the system development cycle such as requirements and design specifications are especially prone to error [15]. This effect is particularly notorious in projects involving multiple stakeholders with different points of view. Evolutionary software processes offer an iterative approach to requirement engineering to alleviate the problems of uncertainty, ambiguity and inconsistency inherent in software developments. Prototypes are useful to demonstrate system scenarios to the affected parties as a way to: a) collect criticisms and feedback that are sources for new requirements; b) enable early detection of deviations from users' expectations; c) trace the evolution of the requirements; and d) improve the communication and integration of the users and the development personnel.

Despite the unquestionable benefits of evolutionary

* This research was supported by the US Army Research Office under grant #38690-MA and grant #40473-MA.

software processes, we have two concerns. First, the automated risk assessment issue has not been resolved. It is usually viewed as an extra activity layered on the assigned work, or worst, as an outside activity that is not part of the software process [6, 9]. The main line of previous research has addressed the problem in parallel with the development process using informal methods. Basically the proposed methodologies are lists of practices and checklists [20, 6] or scoring techniques [9] that are dependent on human expertise.

The second concern is that prototyping poses a problem to project planning because of the uncertain number of cycles required to construct the product. The industry has been using three classes of tools to estimate effort and time that can be applied at different moments during the life cycle, each category being more precise than the previous one but arriving later: a) very early estimations, b) macro models, and c) micro models. For this last category the estimation is done after the design when it is possible to have a work breakdown structure. The project estimate is the integration of all module estimates based on linear layouts of activities, so they do not fit completely with evolutionary software processes.

A detailed discussion of these techniques is outside the scope of this paper; the details can be read in [1, 2, 4, 6, 14, 16, 17, 18, 19]. None of these techniques consider the following characteristics of software projects: a) requirement volatility, b) personnel volatility, and c) time consumed by communications, exceptions and noise in the process. All the methods use size as an input parameter via some kind of derivation from complexity. In many cases the methods to compute such complexities and sizes are questionable [10, 11, 12].

3 METRICS

In this section we describe a small set of metrics that support our risk identification strategy (requirements, personnel and complexity). We choose metrics presenting the following characteristics: a) robustness, b) repeatability, c) simplicity in terms of the number of parameters, d) easy to calculate, and e) automatically collectable.

Metrics for requirements

We purpose three metrics for requirements: a) birth-rate, b) death-rate, and c) change-rate. We define *birth-rate* (BR) as the percentage of new requirements incorporated in each cycle of the evolution process. This metric shows the introduction of new requirements as a percentage.

We define *death-rate* (DR) as the percentage of requirements that are dropped by the customer in each cycle of the evolution process.

A comparison of birth-rate and death-rate in a bi-dimensional plot that shows four regions: stability region, growing region, volatility region and shrinking region (Fig. 1). Each of these regions has different risk connotations. The arrow shows the normal evolution of a project as time

goes by. During early stages, it is normal for projects to be in the growing region. However, if the project remains in this region after many cycles, or returns to this region after visiting other regions, something wrong happens. After some cycles, the project should be in the volatile region. If the project does not evolve into the stability region, then there is evidence that the requirements engineering activity is not efficient and some corrective action may be needed. If the project evolves to the shrinking region, and the requirements engineering is working right, there is evidence that the customers are cutting down the project. This can be an indicator of a severe cut in the budget. Finally, any return to a previous region should be considered as evidence of threats. In such cases a detailed analysis is required to assess the causes of the anomaly.

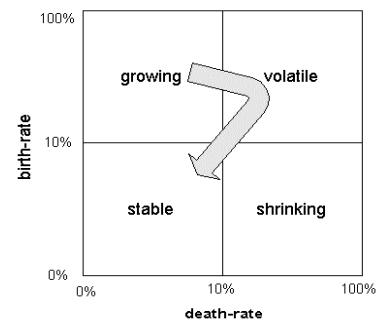


Figure 1: Evolution of requirements

Metrics for fitness

We require measure the fit between people and their roles in the software process. In order to measure personnel both quantitative and qualitative metrics are required. A skill match between person and job is required to estimate the speed in processing information and rate of exceptions. On the quantitative side it is important to measure the number of people and the turnover. This last one provides information about the expected productivity losses due to training, learning curves and communications. This set of metrics is difficult to collect because people are very reluctant to being measured. During the simulations we found that there exists an easier way to measure the productivity fitness observing the ratio between direct working time and idle time as we will discuss in 6.1. Fitness is related to two risk factors: the resources and the process.

Metrics for complexity

Complexity has a direct impact on quality because the likelihood that a component fails is directly related to its complexity. The quality of the product can only be determined at the end of the process. Hence, it is important to measure the complexity as an early predictor to provide a way to assess the duration of the project given some indicators collected during the requirements phase. In such conditions, code is not available, so the only possible

measurements should come from the specification. Complexity is related to one risk factor: the product.

Research on Function Points (FP) [1, 2] showed that there exists a clear relation between complexity and size in terms of lines of code. However, FP are not well suited for real time systems or object-oriented developments [10, 11, 12].

Formal specifications are suitable for being analyzed to compute their complexity. We conducted experiments trying to derive complexity from formal specifications created by CAPS (Computer Aided Prototyping System) [15]. The tool generates specifications in a structured language called Prototyping Specification Design Language (PSDL). We defined two complexity metrics for PSDL: a) *Fine Granularity Complexity* metric (FGC), and b) *Large Granularity Complexity* metric (LGC). The reason to compute different metrics is because we want to detect two classes of threats. First, we need to be aware of excessively complex operators. High complexity of one operator could be caused by poor design and possibly can be solved by further decomposition. Second, we require a metric to compute the total complexity of the system.

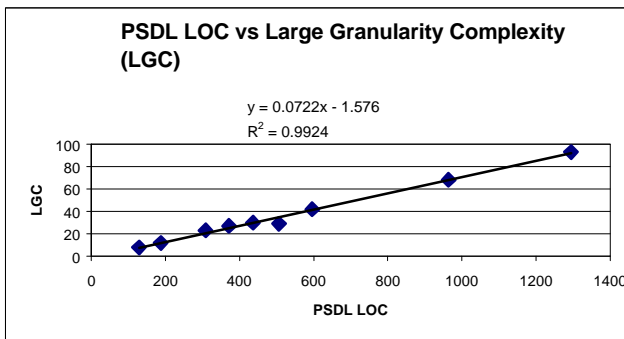


Figure 2: Correlation between PSDL and LGC

FGC expresses the complexity of each operator in the system and is the sum of the fan-in and fan-out data streams related to the operator (FGC = fan-in + fan-out).

LGC expresses the complexity of the system as a function of the number of operators (O), data streams (D), and types (T) found on the PSDL specification (LGC = O + D + T).

We found a strong correlation between PSDL and LGC (R = 0.996, Fig. 2) and also between Ada code and LGC (R = 0.898, Fig.3). The size of the project in thousands of non-comment lines of code can be estimated as:

$$KLOC = (32 LGC + 150) / 1000 \quad [Eq. 1]$$

This finding provided us with a method to compute the size of the projects given an early measure of their complexity. This conversion is required to compare our approach with Putnam's and Boehm's approaches because they require the size as an input parameter. A caveat of this study is that our sample is small, but it includes all the information we

have at the current time. However, the study suggests the possibility of estimating size in terms of complexity with a useful degree of accuracy.

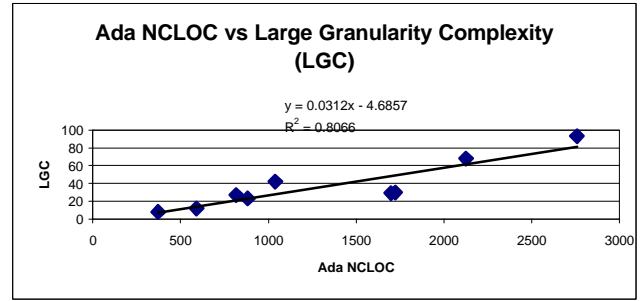


Figure 3: Correlation between Ada code and LGC

4 THE RISK ASSESSMENT MODEL

A probability distribution from the Weibull family can be used to model the development time given the risk factors discussed above. The probability density function and cumulative density function for the model are:

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

[Eq. 2]

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

[Eq. 3]

where:

- x is the random variable under study. In our case, x can be interpreted as development time.
- α is a shape parameter. It determines the width of the peak of the distribution and the expected error. We can associate this behavior with the efficiency of the project, which depends on characteristics of the process and the resources.
- β is a scale parameter that stretches or compresses the graph in the x direction and hence controls the thickness of the tail. This parameter models the extra work introduced by new requirements or changes in requirements.
- Note that the functions start at $x = 0$. We require a third parameter to shift the curves to the right. For that reason we introduce a location parameter γ , which is function of the already discovered system complexity.

5 CALIBRATION OF PARAMETERS

To calibrate productivity (α) and requirement's volatility (β), we conducted simulations with VitéProject [8, 13] using the following scenarios (Fig. 4). Each scenario name consists of three letters describing the value for each of the three variables under study: productivity (α), requirements' volatility (β), and complexity (γ). Each letter could have two values: high (H) or low (L). The tool was configured to

run 100 simulations for each scenario, and the organizational parameters were set to match the characteristics of software development.

Scenario	Productivity	Req. volatility	Complexity
LLL	Low	Low	Low
LLH	Low	Low	High
LHL	Low	High	Low
LHH	Low	High	High
HLL	High	Low	Low
HLH	High	Low	High
HHL	High	High	Low
HHH	High	High	High

Figure 4: Scenario’s characteristics

To analyze the effect of productivity, we compared the results of the simulations of the following scenarios: LLL vs HLL, LLH vs HLH, LHL vs HHL, and LHH vs HHH. We found that for high productivity scenarios (Hxx) the development time improved by 60%.

To analyze the effect of requirement volatility, we compared the results of the simulations of the following scenarios: LLL vs LHL, LLH vs LHH, HLL vs HHL, and HLH vs HHH. We found that high requirement volatility (xHx) degraded the development time by 20%.

To analyze the effect of complexity, we compared the results of the simulations of the following scenarios: LLL vs LLH, LHL vs LHH, HLL vs HLH, and HHL vs HHH. We found that high complexity (xxH) degrade the development time by 30%.

6.1 Productivity (α)

Examining the time distribution between direct work time and idle time, we observed a remarkable pattern that differentiates high productivity scenarios from the low productivity ones. This effect is independent of the other two variables of the simulation. Hence, this suggests that the time distribution can be a good indicator for the parameter α.

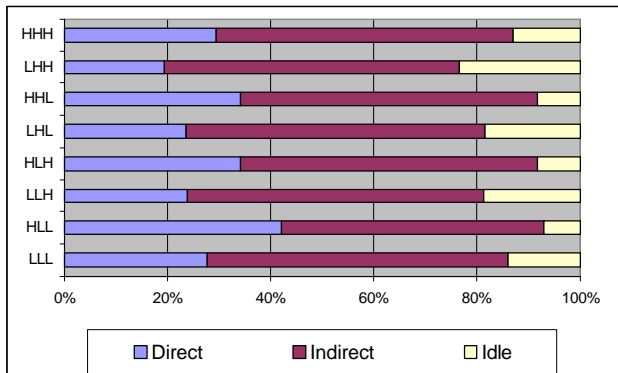


Figure 5: Time distribution from each scenario

Figure 5 presents the distribution times for the eight scenarios simulated. A pattern of time distributions can be

clearly observed. Scenarios with low productivity have a percentage of idle time greater than 13% of the total development time.

We can recognize low productivity scenarios also by the ratio of the percentage of direct time over percentage of idle time, which we call productive ratio (PR):

$$PR = \alpha = \text{Direct\%} / \text{Idle\%} \quad [\text{Eq. 5}]$$

For high productivity scenarios $2.0 < PR < 6.0$, and for low productivity scenarios $0.8 < PR < 2.0$.

We observed that using PR as the value of α, the model behaves as the simulations. That is on high productivity scenarios the total development is 60% shorter than in low productivity ones. The reasons why the ratio PR is related to productivity require further study. However, we conjecture the reason could be related to:

- Fit of job and people skills.
- People turnover, generating noise and productivity losses derived from training and learning curves.
- Number of people, influencing the productivity by excess or default of working force.

In the model the use of α ranging from 0.8 (low productivity) to 6 (highest productivity), corresponds to the results observed in the simulations.

6.2 Requirement’s volatility (β)

β, the extra delay factor caused by requirements’ volatility (late requirements and changes in previous requirements), is obtained by the following formula:

$$\beta = \text{INT}((BR + DR) / 10) \quad [\text{Eq. 6}]$$

Our simulations showed a 20% increase on the development time when the requirement’s volatility is high.

6.3 Complexity (γ)

Having found a complexity metric suited for our purpose, the next step was to find for the existence of some sort of relationship between LGC and development time. We conducted a simple experiment using the conversion ratio [Eq. 1] to obtain the size inputs for the sample. We used sample points from 1000 LGC to 30000 LGC, which means sample projects from 32 KLOC to almost 1MLOC. We compute the average estimation for the development time using COCOMO and Putnam. The sample points are plotted with a smoothing thick line. The logarithmic trendline is plotted as a thin red line. We found a strong logarithmic correlation ($R^2 = 0.9699$) with the following function (Fig. 6).

$$\text{Time (months)} = \gamma = 13 \text{Ln(LGC)} - 82 \quad [\text{Eq. 7}]$$

This equation gives a conservative estimation for projects between 4000 and 20000 LGC (128 and 640 KLOC of Ada). The estimation seems to be too optimistic for projects smaller than 2000 LGC or greater than 25000 LGC. Figure 9 shows the effects of complexity over different scenarios.

The development time increases by 20% when the complexity is high.

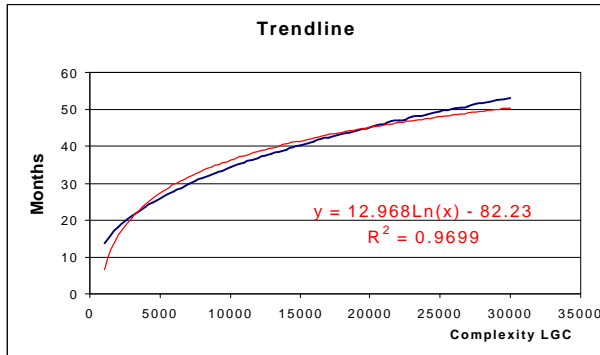


Figure 6: Complexity-time correlation

6.4 The complete model

Our model requires three parameters (α , β , γ) that can be derived from metrics automatically collected from the development environment (Eq. 5, 6 and 7). If the development environment does not have the functionality to collect metrics, then a manual procedure could provide the data. Using these values in Eq.3 we obtain the probability of finishing the project at any given time (x in months) (Fig. 7). The model enables to refine the estimation from the knowledge captured at each evolutionary cycle. As the development progress γ increases (known complexity) and β decreases (less tail).

6 CONCLUSION

We introduced a formal method for risk assessment that solves the issue of human dependency, characteristic of the current risk assessment methodologies. This method is supported by a small set of metrics that can be automatically collected from the development environment.

One of the metrics introduced, productivity ratio, constitutes an objective method to assess the productivity level of an organization without subjective judgement of experts.

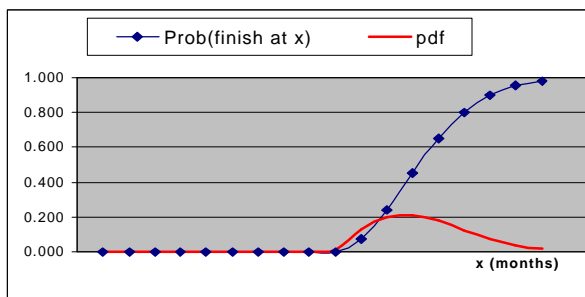


Figure 7: Distribution functions

Although our model is based on a complexity metric developed specifically for PSDL, the method can be generalized for other methodologies using Object Points or number of classes instead of LGC.

The model provides an easy way to estimate, very early in the life cycle, the duration of a project, and indirectly, its cost. This method enables an earlier assessment of the duration of the project and solves the problems of: a) human dependency on risk assessment, and b) difficulties in estimating time on evolutionary prototyping software processes. Further research is required to generalize the method for larger systems and for different domains.

REFERENCES

1. Albrecht, A. Measuring Application Development Productivity. Proceedings IBM. October 1979.
2. Albrecht, A. and Gaffney, J. Software Function Source Lines of Code and Development Effort Prediction. IEEE Transactions Software Engineering, SE-9, 1983.
3. Boehm, B. A Spiral Model of Software Development and Enhancement. Computer. May, 1988.
4. Boehm, B. Software Engineering Economics. Prentice Hall, 1981.
5. Boehm, B., Madachy R., Selby, R. Cost Models for Future Software Life Cycle Processes: COCOMO 2.0. <<http://sunset.usc.edu/COCOMOII/cocomo.html>>
6. Hall, E. Managing Risk. Methods for Software Systems Development. Addison Wesley, 1997.
7. Humphrey, W. Managing the Software Process. Addison-Wesley, 1989.
8. Jin, Y. and Levitt, R. The Virtual Design Team. Paper to appear in Computational and Mathematical Organization Theory. 1996.
9. Karolak, D. Software Engineering Management. IEEE Computer Society Press, 1996.
10. Kitchenham, B., Kansala, K. Inter-item Correlations among Function Points. Software Metrics Symposium. IEEE Computer Society Press. 1993.
11. Kitchenham, B., Linkman, S. Estimates, Uncertainty, and Risk. IEEE Software. May-June, 1997.
12. Kemerer, C. Reliability of Function Points Measurements. Communications ACM, V. 36 No 2. 1993.
13. The ViteProject Handbook. Vité ©. 1999.
14. Londeix, B. Cost Estimation for Software Development. Addison-Wesley, 1987.
15. Luqi and Ketabchi, M. A Computer-Aided Prototyping System. IEEE Software. March, 1988.
16. Putnam, L. Software Cost Estimating and Life-cycle Control. IEEE Computer Society Press. 1980.
17. Putnam, L. and Myers, W. Measures for Excellence. Yourdon Press, 1992.
18. Putnam, L. and Myers, W. Executive Briefing. Controlling Software Development. IEEE Computer Society Press. 1996.
19. Putnam, L. and Myers, W. Industrial Strength Software. Effective Management Using Measurement. IEEE Computer Society Press, 1997.
20. Software Engineering Institute. Software Risk Management. CMU/SEI-96-TR-012. June, 1996.