2003

Decision Support for Software Projects: The Role of SPC and Simulation Metamodeling

Omar F. El-Gayar
Dakota State University

Decision Support for Software Projects: The Role of SPC and Simulation Metamodeling

Conference Paper · January 2003

CITATIONS
0

READS
7

1 author:

Omar F. El-Gayar
Dakota State University
156 PUBLICATIONS **1,500** CITATIONS

SEE PROFILE

DECISION SUPPORT FOR SOFTWARE PROJECTS: THE ROLE OF SPC AND SIMULATION METAMODELING[1]

Omar F. El-Gayar
College of Business and Information Systems, Dakota State University,
Madison, South Dakota 57042, U.S.A. email: omar.el-gayar@dsu.edu

ABSTRACT

While many researchers have attempted to directly apply statistical process control (SPC) to the software domain, several difficulties in applying SPC to software development, in particular, the inability to compute meaningful control limits for the process.

In this research, we propose a framework for applying SPC to software projects. The framework integrates SPC concepts and simulation metamodeling to create meaningful control limits on process and project inputs. The framework is demonstrated using a case study.

Keywords: Software process simulation modeling; Statistical process control; Simulation metamodeling; Visualization

INTRODUCTION

The demand for software continues to outpace our production capabilities in terms of quality and quantity [1]. In 1995, U.S expenditure for software development and maintenance reached $ 250 billion and encompasses an estimated 175,000 projects [2].

In spite of the significant increase in expenditure, the software industry continues to experience what is commonly referred to as the " software crisis", namely, budget overruns, schedule delays, poor quality and users dissatisfaction [3]. In 1995, U.S companies spend an estimated $59 billion in cost overruns on IS projects and another $81 billion on canceled projects [2].

In this vein, and over the past decade, concepts and methods associated with process improvement have gained wide acceptance in the software community [4]. Of particular importance is the application of statistical process control (SPC) to the software process. The strengths of SPC are that they provide easy to use graphical tools that enable the manager to visualize the performance of the Manufacturing process (system) along multiple dimensions of performance.

While the application of SPC in manufacturing dates back to the 1920's and was pioneered by the work of [5], and SPC has now been accepted world wide as the best method for tracking quality of manufacturing processes, it is only during the mid-eighties that attempts have been made to apply SPC to software [1] [6] [7] [8]. Since then, a number of applications have been reported in the literature [9] [10] [11]. Moreover, the Quantitative Process Management Key Process Area of the Capability Maturity Model requires that projects manage their processes quantitatively and recommend that SPC be applied at this level.

However, inherent in existing applications of SPC to the software process is the view of software as a manufactured product. Accordingly, techniques of SPC can be applied to software processes just as they have been applied to manufacturing processes [12]. Nevertheless, as noted by [1], there are fundamental differences between

[1] This material is based upon work supported by the National Science Foundation/EPSCoR Grant #EPS-0091948 and by the State of South Dakota. The author also thanks Peter Lakey of Cognitive Systems for graciously providing the model used in this paper. The usual disclaimer applies.

software process and manufacturing process. Software development process are very different from manufacturing processes in terms of many key process dimensions such as product volume, level of automation, standardization of processes, among others. Most notably, is that in software processes is the transformation of user requirement into software is dominated by cognitive activities, this contrasts with manufacturing activities where cognition is minimized. Moreover, in software processes, the input and output for the software process are different for each instance of the process [1]. In other words, each software product generated by a particular process is different (as determined by the user requirements). The interaction between cognitive dominated activities and different inputs and outputs create challenges to the application of SPC for software. In particular,

1. The data regarding input and outputs experience great variability, which make it particularly difficult to develop meaningful control limits.
2. It is difficult to assign cause of variability in the data.
3. It is difficult to visualize the performance of the process using the traditional control limits as they typically contain too much variability to provide any useful guidance in managing the process

These weaknesses call into question the applicability and the usefulness of SPC to the software development domain because these weaknesses are common to many software development projects. Against this background, the general aim of this research is to improve the applicability of SPC to software development processes. The specific objectives and tasks are as follows:

- Utilize management control maps to create meaningful limits on the overall software development project performance measures. We accomplish this by creating visual control maps based upon management's goals for the project and utility for project outcomes.
- Investigate the relationship between the variability of individual process and product metrics to overall project performance using simulation to show whether the process is capable of achieving the desired project goals.

From a practical standpoint, this research creates a useful graphical method for managers to visualize project performance and to manage their software development project. From a theoretical standpoint, this work explains the inherent difficulties of directly applying SPC to software development in a manner that opens the possibility of a variety of alternative approaches for making SPC a more useful tool in the software development domain.

THE FRAMEWORK

The proposed framework for process and project control centers on the establishment of a feedback links between process and project inputs (such as productivity, detected defects, module complexity and others) and the desired model outputs of cost, quality, and schedule. Once such links are established, we can then utilize such links to map the level and variability (expressed as control limits) into corresponding visual control maps for process inputs.

In effect, establishing a link between process and/or project inputs and outputs calls for establishing a model between inputs and outputs. While several software process modeling approaches have been proposed in the literature [13] [14], in this framework, we are particularly interested in modeling approaches that quantitatively link inputs to outputs, e.g., regression models as used in COCOMO as well as software process simulation models (SPSM). Figure 1 illustrates a general framework linking process inputs to outputs through the use of a model.

The model depicted can also be expressed mathematically as:

$$Y = f(X) \qquad (1)$$

Where:

Y, Y', Y'': is a vector of real, simulation, and simulation meta model output variables, respectively.

X : is a vector of input variables.

The mathematical representation is also valid even if the model is not originally depicted as described in equation 1. For instance, a simulation model can be approximated and described as in equation 1 through the construction of a simulation meta-model.

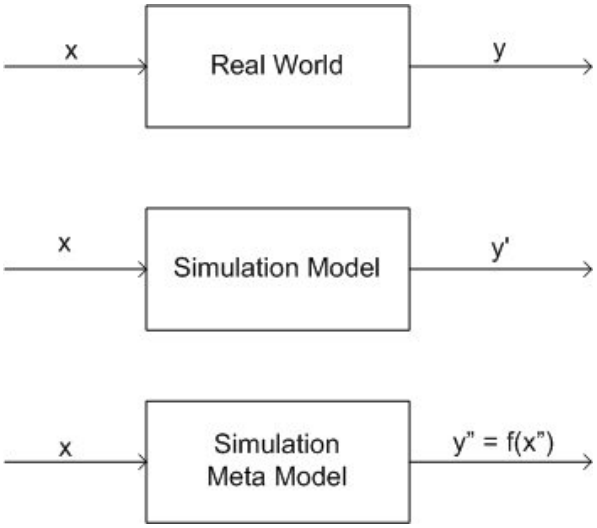

Figure 1: general framework linking process inputs to outputs through the use of a model

Irrespective of the quantitative formulation of the relationship between process inputs and outputs, the resultant models are always constructed with the output variables as the dependent variables and the input variables as the independent variables. Such models are normally used in estimating and forecasting process outputs, e.g., schedule and cost as a function of process inputs such as input quality, and, programmers' productivity. Other application include 'what if' analysis, mostly for planning purposes. Accordingly, such models are of limited value when the purpose is to prescribe the level and variability for the input variables that result in acceptable (to managers) levels and variability for the output variables, i.e., 'goal seeking'. In effect, from a project management and control perspective, such models are descriptive as opposed to prescriptive.

As such, the next step, demands the formulation of a reverse model in which the process outputs, such as schedule and cost, are the independent variables and process inputs, such as productivity and input complexity are the dependent variables as shown in Figure 2. Since software process simulation offers a versatile approach to modeling software development process and as evidenced by the proliferation of SPSM techniques and applications reported in the literature [15], it is reasonable to assume that the model in Figure 1 is a simulation model. In this case, the second step in our framework reduces to the development of a reverse simulation meta-model as described in Figure 2.

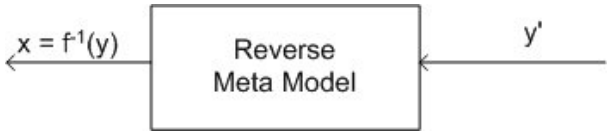

Figure 2: Reverse simulation metamodel

Once a reverse meta-model is constructed, the third and final step is the creation of visual control maps (levels and control limits) using the management control maps (representing the voice of the customer) and the simulation meta-model developed in the preceding step.

CASE STUDY

The model

In this study, we utilize a model developed by Peter Lakey of Cognitive Concepts [16]. The model is a discrete event simulation model, originally developed as a project management tool for estimating schedule, effort and quality during the detailed design for a computer software configuration item (CSCI).

The underlying development process is an iterative and incremental process in which the CSCI is developed incrementally over a number of iterations. Each iteration includes a development phase, a review phase, and a rework phase. For each of the phases, the model tracks the effort involved as well as a number of other metrics such as the number of defects generated, found, and escaped during the process.

An important feature of this model is the incorporation of System Dynamics modeling concept of feed back loops. Specifically, when evaluating these metrics the model takes into account a variety of process and product factors. Examples of product factors include the size quality and complexity of the input artifacts,

while examples of process factors include schedule pressure, communication overhead and tools support.

The reverse simulation meta-model

Several simulation meta-modeling and reverse meta-modeling are presented in the literature. See [17] [18] [19] for a state of the art review. In this case study we illustrate our framework by constructing an artificial neural network (ANN) reverse meta-model for the aforementioned simulation model. The steps we used are summarized as follows:

1. Select the inputs and outputs.
2. Generate the training and testing data sets.
3. Select and design an ANN
4. Train the ANN using the training data set.
5. Test the ANN using the testing data set.

Step 1: Select input and outputs

The inputs for the reverse simulation meta-model are the outputs of the simulation model. These are the variables that the project manager ultimately cares about. In Lakey's model these variables are: Effort and Number of defects. Schedule is another variable that we have omitted as it exhibits the same statistical properties as Effort. Since we are interested in visualizing the variation in project output, we use the coefficient of variation of the aforementioned variables. The list of inputs to the simulation-meta-model then becomes:

Effort coefficient of variation ($\bar{x}/\sigma$)

Number of Defects coefficient of variation $\bar{x}/\sigma$

On the other hand, the outputs of the reverse simulation model are the inputs of the simulation model. In Lakey's model there are a number of process and product inputs. Since the purpose of this case study is to demonstrate the applicability of the proposed framework, the study, at this stage, focused on the input variability (denoting the extent of variation in the size and quality of input artifacts) as the controllable and thus the output variable of the simulation meta-model.

Step2: Generate the training and testing data sets

Once the inputs and outputs are selected, the next step (step 2) is to generate the data sets needed for training and testing the ANN. Each of the training and testing data sets are comprised of 83 data points (experiments). For each data set, the simulation model is run for 83 experiments. Each experiment is comprised of 15 runs using different streams of random numbers for each of the runs resulting in a total of 1245 simulation runs per data set.

Moreover, for each experiment the mean ($\bar{x}$), standard deviation (σ), and coefficient of variation ($\bar{x}/\sigma$) are calculated for both output variables, namely, Effort and Number of defects. The result from each simulation experiment is an observation in our master data set.

Step 3: Select and design an ANN

In step 3 we design and implement the ANN. Of the several ANN paradigms that are presented in the literature [20], the feed-forward networks using back propagation learning rule are known for their robustness and learning ability and are thereby utilized in this study. Since there is very little experimental evidence as to the size of the network (in terms of the number of hidden layers and the size of each layer), we evaluate various designs.

Step 4: Train the ANN using the training data set

Overall, our objective is to find the 'optimal' network design. By optimal, we refer to network designs with the minimum number of unknown and the lowest mean square error (MSE). To minimize the number of unknowns, we seek networks with the minimum number of hidden layers and minimum number of neurons in each layer. Given the size of the data set, we used MATLAB to evaluate network designs of up to 3 layers (including the output layer) and 6 hidden neurons in each layer. Figure 3 depict the resultant network with two layers, in which layer 1 has 2 neurons with logsig functions and layer 2 (the output layer) has one neuron with purlin function.

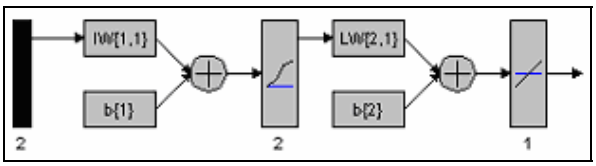

Figure 3: ANN architecture

Step 5: Test the ANN using the testing data set

Once we train the ANN, the next step is to test if the ANN exhibit good generalization, i.e., good performance with data not seen before. In this case we used a data set of 83 data points as described in step 2. The resultant MSE is comparable to the MSE corresponding to the training data set, thereby indicating good generalization.

The visual control charts

With the reverse simulation model in place, a management control map such as in Figure 4a&b denoting the preference of the project manager with respect to project variables such as Effort and Quality (Number of defects) can then be mapped into corresponding control maps for project inputs as shown in Figure 5. In effect, Figure 5 depict the range of variation denoted with upper and lower control limits around the average size of input scenarios that is acceptable to keep the variability in key project variables within acceptable limits (denoted on Figure 4)

CONCLUSIONS

This paper presents a framework for project and process control that centers on the establishment of feedback links between process and project inputs (such as productivity and module complexity) and the desired model outputs such as effort and quality. The proposed framework is demonstrated using a case study where we developed a neural network reverse simulation model to map acceptable levels of variability in project output to a corresponding acceptable level of variability on project inputs. As such, the proposed framework provides software project managers with the means to a-priori evaluate and control their project at a point in time. By comparing the current measurements to the control limits set forth by the reverse simulation model, the project manager can take action if any of such measurements fall outside these limits.

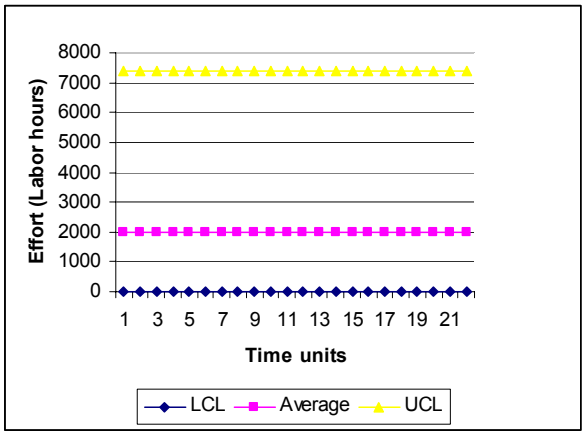

Figure 4: Management control map for Effort

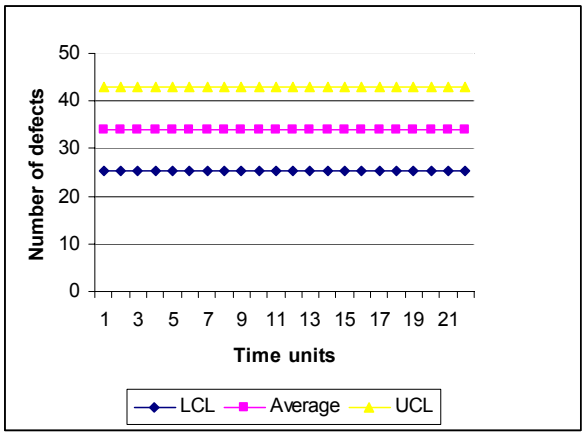

Figure 4b: Management control map for Quality (Number of Defects)

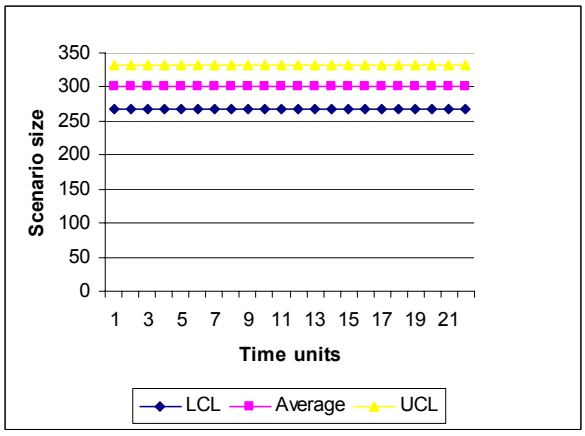

Figure 5: Management control map for Input Variability for input scenarios

REFERENCES

[1] Lantzy, M. Application of Statistical Process Control to the Software Process. *ACM* 1992 113-123.

[2] Keil, M., P. Cule, K. Lyytinen, and R. Schmidt. A Framework for identifying software project risks. *Communications of the ACM*, November 1998, vol. 41, No. 11, pp. 76-83.

[3] Abdel-Hamid, T. and S. Madnick. *Software Project Dynamics, An Integrated Approach*. New Jersey: Prentice Hall, 1991.

[4] Florac, W. and A. Carleton. *Measuring the Software Process: Statistical Process Control for Software Process Improvement*, Addison Wesley, 1999.

[5] Shewhart, W. *The Economic Control of Quality of Manufactured Product*. New York: D. Van Nostrand Company, 1931, reprinted by ASQC Quality Press, Milwaukee, Wisconsin, 1980.

[6] Cho, C. *Quality Programming: Developing and Testing Software with Statistical Quality Control*. New York: John Wiley and Sons, Inc., 1987

[7] Lanphar, R. Quantitative process management in software engineering, reconciliation between process and product views. *The Journal of Systems and Software*, July 1990, v12, n3, pp 243-249.

[8] Putnam, L. Trends in measurement, estimation, and control. (making software engineering and management more quantitative). *IEEE Software*, March 1991, v8, n2, pp 105-108 .

[9] Weller, E. Practical applications of statistical process control. *IEEE Software*, May-June 2000, v17, i4, pp 97-107.

[10] Florac, W., Carleton, A., Barnard, J. Statistical process control: analyzing a Space Shuttle Onboard Software process. *IEEE Software*, July-August 2000, v17, i4, pp 97-107.

[11] Lewis, N. Assessing the evidence from the use of SPC in monitoring, predicting & improving software quality. *Computers and Industrial Engineering*, 1999, 37(1), 157-160.

[12] Florac, W., Park, R., and Carleton, "A. Practical Software Measurement: Measuring for Process Management and Improvement," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1997.

[13] Curtis, B., Kellner, M.I., and Over, J. "Process Modeling," *Communications of the ACM* (35:9), September 1992, pp 73-90.

[14] Fuggetta, A. "Software Process: A Roadmap," *Proceedings of the conference on the future of software engineering*, Limerick, Ireland, 2000.

[15] Kellner, M., Madachy, R.J., and Raffo, D.M. Software Process Simulation Modeling: Why? What? How?, *Journal of Systems and Software*(46:2/3), April 1999, pp 1-18.

[16] Lakey, P.B. Discrete Event Software Project Management Tool. Cognitive Concepts, 2000.

[17] Barton, R.R. Metamodels for simulation input-output relations, *Proceedings of the 24th conference on Winter simulation*, p.289-299, December 1992, Arlington, Virginia, United States.

[18] Barton, R.R. Simulation Metamodels: a state of the art review, *Proceedings of the 26th conference on Winter simulation*, p.237-244, December 1994, Orlando, Florida, United States.

[19] Barton, R.R. Simulation Metamodels, *Proceedings of the 30th conference on Winter simulation*, p. 167 - 176, December 1998, Washington, D.C., United States.

[20] Haykin, S. *Neural Networks: A Comprehensive Foundation*. Englewood Cliffs, New Jersey: Prentice Hall, 1999.