

Evaluating the Effectiveness of Process Improvements on Software Development Cycle Time via System Dynamics Modeling

John D. Tvedt and James S. Collofello

Arizona State University
Department of Computer Science and Engineering
Tempe, AZ 85287-5406 USA
E-mail: {tvedt, collofello}@asu.edu

Abstract

Reducing software development cycle time without sacrificing quality is crucial to the continued success of most software development organizations. Software companies are investing time and money in reengineering processes incorporating improvements aimed at reducing their cycle time. Unfortunately, the impact of process improvements on the cycle time of complex software processes is not well understood. The objective of our research has been to provide decision makers with a model that will enable the prediction of the impact a set of process improvements will have on their software development cycle time. This paper describes our initial results of developing such a model and applying it to assess the impact of software inspections. The model enables decision makers to gain insight and perform controlled experiments to answer "What if?" type questions, such as, "What kind of cycle time reduction can I expect to see if I implement inspections?" or "How much time should I spend on inspections?"

1: Background

1.1: Importance of reducing cycle time

The rush to reengineer processes taking place in many organizations is the result of increasing competitive pressures to shorten development cycle time and increase quality. Reducing the time to produce software benefits not only the customer, but also the software organization. A number of benefits of reduced cycle time are listed by Collier [4].

- Market life of the product is extended.

- Being the first to market along with the high cost of switching products can give a product a great advantage in capturing and obtaining a large market share.
- Higher profit margins. Being first to market gives a company greater freedom in setting profit margins.
- The ability to start later and finish on schedule. Starting later allows the use of technological advances that may not have been available to the competition.

"According to a study released in January 1990 by United Research Co. of Morristown, N.J., six out of 10 CEOs listed shorter product development cycles as vital to their company..." [9].

Most of these cycle time reduction reengineering efforts attempt to take advantage of process improvement technology such as that advocated by the Software Engineering Institute's Capability Maturity Model and ISO 9000. The implication is that a mature development process increases the quality of the work done, while reducing cost and development time [8]. Some software companies that have begun to mature their process have, indeed, reported cycle time reductions via process improvements [5].

1.2: The need for selecting among alternative process improvements

A common problem faced by organizations attempting to shorten their cycle time is selecting among the numerous process improvement technologies to incorporate into their newly reengineered development process. Even though there is promising evidence

trickling into the pages of computer publications, most companies are still skeptical of the investment that must be made before possible improvements will be experienced. They recognize that process improvements do not exist in isolation. The impact an improvement has may be negated by other factors at work in the particular development organization. For example, consider a tool that allows some task to be completed in a shorter period of time. The newly available time may just be absorbed as slack time by the worker, resulting in no change to cycle time. Thus, software organizations need to know, in advance of committing to the process improvement technology, the kind of impact they can expect to see. Without such information, they will have a hard time convincing themselves to take action. In particular, a software organization needs to be able to answer the following types of questions:

- What types of process improvements will have the greatest impact on cycle time?;
- How much effort should be allocated to the improvement?;
- What will the overall impact be on the dynamics of software development?;
- How much will the process improvement cost?;
- How will the process improvement affect cycle time?; and
- How will this improvement be offset by reduced schedule pressure, reduced hiring and increased firing?

Today, the limited mental models often used to answer these questions are insufficient. The mental models often fail to capture the complex interaction inherent in the system. "It is the rare [manager] whose intuition and experience, when he is presented with the structure and parameters of [only] a second order linear feedback system, will permit him to estimate its dynamic characteristics correctly," [2].

1.3: Current approaches for evaluating the impact of process improvements

There does not exist a standard method for determining the impact of specific process improvements on cycle time. An ideal way for performing this assessment is conducting a controlled experiment in

which all factors except the independent variable (the process improvement) are kept constant. Although much can be learned through this approach about the general effectiveness of the process improvement, it is normally not practical to experiment over the entire development life cycle of a significant size project. Thus, it is often impossible to assess the overall impact of the proposed process improvement on development cycle time.

Another approach typically followed is the utilization of some sort of "pilot project" to assess the new technology. Although considerably weaker than controlled experimentation, the pilot studies often reveal the general merit of the proposed technology. It remains difficult to assess the unique impact of the proposed improvement technology on cycle time due to the interaction of other project variables and to generalize the results to other projects.

A third approach to assessing the impact of process improvements is the utilization of traditional cost estimation models such as COCOMO [3] and SLIM [10]. These types of models contain dimensionless parameters used to indicate the productivity and technology level of the organization. The parameter values are determined by calibration of the model to previous projects or by answering a series of questions on a questionnaire. For example, the Basic COCOMO model calculates the number of man-months needed to complete a project as:

$$MM = C (KDSI)^K, \text{ where}$$

MM = number of man-months,
 C = a constant,
 KDSI = thousands of delivered source instructions and
 K = a constant.

The user enters the size of the code along with the two predefined constants to determine the number of man-months. C and K are determined by calibrating the model to previous projects.

An important question for a software organization to answer is how C and K should be modified to reflect the impact of process improvements when no historical data exists. One potential answer is to use the Intermediate or the Advanced COCOMO models. These models contain cost drivers that allow more information about the software project to be input than the Basic model. Cost drivers are basically multipliers to the estimated effort produced by the model. *Modern Programming Practices* is one such cost driver. If a software project is using modern programming practices, the cost driver would be set such that its multiplier effect would reduce the amount of effort estimated. Two questions arise when using this cost driver: what is the definition of modern

programming practices and how should the cost driver's value be altered with the addition of a specific improvement to the existing programming practices. These questions have no simple answers because the cost drivers are at a level of granularity too coarse to reflect the impact of specific process improvements. Some researchers also believe that models such as COCOMO are flawed due to their static nature. Abdel-Hamid states, "The problem, however, is that most such tools are static models, and are therefore not suited for supporting the dynamic and iterative planning and control of [resource allocation]" [2].

2: Proposed approach for evaluating the effectiveness of process improvements

2.1: Overview of system dynamics modeling

Due to the weaknesses of the approaches presented in the previous section, we chose to evaluate the impact of process improvements on software development cycle time through system dynamics modeling. System dynamics modeling was developed in the late 1950's at M.I.T. It has recently been used to model "high-level" process improvements corresponding to SEI levels of maturity [12]. System dynamics models differ from traditional cost estimation models, such as COCOMO, in that they are not based upon statistical correlations, but rather cause-effect relationships that are observable in a real system. An example of a cause-effect relationship would be a project behind schedule (cause) leading to hiring more people (effect). These cause-effect relationships are constantly interacting while the model is being executed, thus the dynamic interactions of the system are being modeled, hence its name. A system dynamics model can contain relationships between people, product and process in a software development organization. The most powerful feature of system dynamics modeling is realized when multiple cause-effect relationships are connected forming a circular relationship, known as a feedback loop. The concept of a feedback loop reveals that any actor in a system will eventually be affected by its own action. A simple example of the ideas of cause-effect relationships and feedback loops affecting people, product and process can be illustrated by the following scenario:

Consider the situation in which developers, perceiving their product is behind schedule (cause), modify their development process by performing fewer quality assurance activities (effect/cause), leading to a product of lower quality (effect/cause), but giving the temporary

perception that the product is back on schedule (effect/close feedback loop).

The developers perceived the product to be behind schedule, took action, and finally perceived the product to be back on schedule, as a result of their actions. Secondary effects due to the developers' actions, however, such as the lower quality of the product, will also eventually impact the perception the developers have as to being on schedule and will necessitate further actions being performed. These cause-effect relationships are explicitly modeled using system dynamics techniques.

Because system dynamics models incorporate the ways in which people, product and process react to various situations, the models must be tuned to the environment that they are modeling. The above scenario of developers reacting to a product that is behind schedule would not be handled the same way in all organizations. In fact, different organizations will have different levels of productivity due to the experience level of the people working for them and the difficulty of the product being developed. Therefore, it is unrealistic for one model to accurately reflect all software development organizations, or even all projects within a single development organization.

Once a system dynamics model has been created and tailored to the specific development environment, it can be used to find ways to better manage the process to eliminate bottlenecks and reduce cycle time. Currently, the state-of-the-practice of software development is immature. Even immature software development environments, however, can benefit from this technique. The development of the model forces organizations to define their process and aids in identifying metrics to be collected. Furthermore, the metrics and the models that use them do not have to be exact in order to be useful in decision-making [13]. The model and its use result in a better understanding of the cause-effect relationships that underlie the development of software. The power of modeling software development using system dynamics techniques is its ability to take into account a number of factors that affect cycle time to determine the global impact of their interactions, which would be quite difficult to ascertain without a simulation. The model may be converted to a management flight simulator to allow decision-makers the ability to perform controlled experiments of their development environment.

2.2: Modular system dynamics modeling

Due to the large number of process improvements available for evaluation and the necessity of conducting each evaluation in the context of a system dynamics

model customized to the organization for which the evaluation is being performed, a modular system dynamics modeling approach was adopted. A modular system dynamics model consists of two parts, the base software process model and the process improvement model. Once a base model of the software development process exists, any number of process improvement models may be plugged in to determine the effect they will have on software development cycle time. This modularity gives the process improvement models the advantage of being used in more than one base model of a software development environment. For example, the process improvement model could be integrated with a base model of development environment "A" and also a base model of development environment "B". The only change to the process improvement model would be the values of its parameters in order to calibrate it to the specific development environment. The models of the development environments may have to be modified structurally, such that they include all necessary model elements, e.g., system dynamics modeling rates and levels, for integration.

A conceptual image of the modular model is shown in Figure 1. On the left is a box representing an existing system dynamics model of a software development process. This model would be tailored to a particular organization. On the right is a box representing the model of the process improvement.

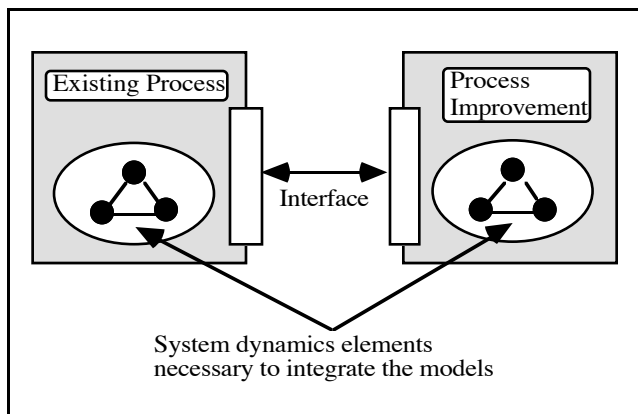


Figure 1. Graphic of existing process with process improvement.

The two models are integrated through the interface pictured. Information will flow in both directions between the models. This interface is dependent on certain model elements, such as levels and rates, existing in both models. The ovals in Figure 1 represent a cut-away view showing the underlying structure of the system dynamics model and the elements needed for the interface. An existing model of an organization may need to be

modified to take advantage of the proposed model, such that it contains the necessary elements for the interface.

2.3: Approach for constructing modular system dynamics models

An organization wishing to utilize system dynamics modeling for assessing the impact of new technologies on cycle time must perform the following steps:

1. Construct a base model of the software development process.

Construction of a base model requires detailed modeling of the organization's software development process as well as identification of cause-effect relationships and feedback loops. Sources of information for model construction include: observation of the actual software process, interviews of personnel, analysis of metrics and literature reviews of relevant publications. An organization wishing to adapt an existing generic model for rough assessments might consider the model developed by Abdel-Hamid [1].

2. Construct the process improvement models.

For each process improvement being considered, a modular process improvement model must be developed utilizing the same approach as that of the base model.

3. Validate the base and process improvement models.

Both the base and process improvement models must be validated to their accuracy requirements in order to build confidence in the models ability to replicate the real system. The validation process must consider the suitability of the model for its intended purpose, the consistency of the model with the real system, and the utility and effectiveness of the model [6], [7], [11].

4. Execute the models.

The base model and any of the process improvement models can be combined to assess the impact of various process improvements on cycle time. Analysis of outputs may also lead to new ideas and understanding of the system triggering additional process reengineering as well as consideration of other process improvements.

3: Demonstration of approach

3.1: Model development

In order to illustrate the feasibility and usefulness of system dynamics modeling for process improvement assessment, we applied our approach to the software inspection process. Our model has the ability to provide answers to the types of questions, concerning process improvements, posed in Section 1.2. For the purpose of our demonstration, we focus mainly on the question of cycle time reduction. We initially developed a base model corresponding to a typical organization's waterfall software development process. We then constructed a model of the software inspection process which we integrated with the base model. The software inspection model enables manipulation of a number of variables connected to the inspection process in order to understand their impact on software development cycle time. Direct manipulation of the following variables are allowed:

- The time spent on each inspection task per unit of work to be inspected (e.g., the preparation rate and the inspection rate);
- The size of the inspection team;
- The percent of errors found during inspection;
- The percent of tasks that undergo reinspection; and
- The defect prevention attributable to the use of inspections.

Our software inspection model is based on the following assumptions:

- Time allocated to software inspections takes time away from software development;
- A larger inspection team will consume more man-hours per inspection than a smaller team;
- Software inspections find a high percentage of errors early in the development life cycle; and
- The use of inspections can lead to defect prevention, because developers get early feedback as to the types of mistakes they are making.

Our software inspection model does not incorporate the following:

- Software developers achieve higher productivity due to an increase in product knowledge acquired through the software inspection process;
- Software developers achieve improved design estimates due to attention paid to size estimates during inspection; and
- Software inspections lead to increased visibility of the amount of work completed.

The model also excludes the interaction that the inspection team size and the time spent performing inspection tasks have on the percent of errors found during inspection. The inspection team size, the time spent on inspection tasks and the percent of errors found during inspection can be set at the beginning of a model simulation according to historical metrics or altered during the actual simulation.

In order to judge the impact that software inspections have on software development cycle time, the software inspection model must be integrated into a model of the software development process. Once integrated, the software inspection model will impact a number of elements in the software development process model. Figures 2 and 3 are an incomplete, but representative view of the integrated model. Figure 2 represents the process steps and effort involved in inspecting work products. Figure 2, however, does not reveal how time and manpower are allocated to perform each step in the inspection process, in order to keep the diagram and ideas presented simple. Each rate in Figure 2 requires that manpower be consumed in order to move work products from one step to the next. Figure 3 shows an incomplete, but representative implementation of the interface between the base model of the software development process and the process improvement model, that is shown abstractly in Figure 1. Figure 3 represents the modeling of errors in the base process model of software development and illustrates the impact inspections have on error generation and error detection in the base process model. The impacts that software inspections have on software development are: software inspections consume development man-hours, errors are less expensive to find during software inspection than system testing, software inspections promote defect prevention and software inspections reduce the amount of error regeneration. Before discussing Figure 2 and Figure

3, a brief discussion of flow diagrams, a notation used to represent system dynamics models, is in order.

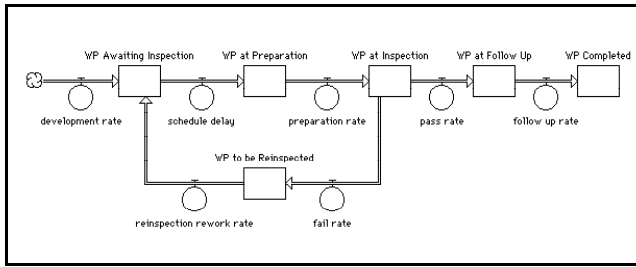


Figure 2. Flow diagram of simplified inspection process steps.

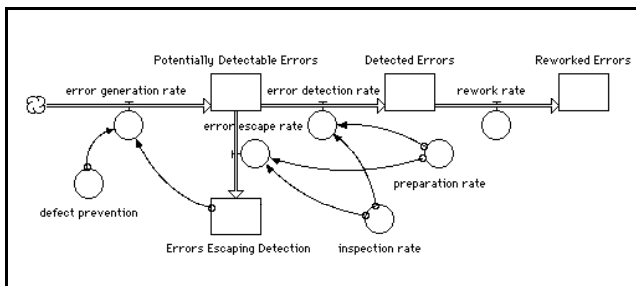


Figure 3. Flow diagram of inspection's impact on errors.

Flow diagrams are composed of levels, material flows, rates, auxiliaries and information links. Levels, depicted as rectangles, represent the accumulation of a material in a system. For example, work products and errors are materials that reside in levels in Figure 2 and Figure 3. Material flows, depicted as hollow arrows, indicated the ability for material to flow from one level to another. Material flows connected to clouds indicate a flow to, or from, a portion of the system not pictured. Rates, depicted as circles attached to material flows, represent the rate of material flow into and out of a level. For example, error detection rate in Figure 3 controls the rate at which potentially detectable errors are detected. Auxiliaries, depicted as circles, aid in the formulation of rate equations. In Figure 3, defect prevention is an auxiliary that affects the error generation rate. Information links, depicted as arrows, indicate the flow of information in a system. Information about levels, rates and auxiliaries are transferred using information links. Information links, unlike material flows, do not affect the contents of a level.

The first impact that software inspections have on software development is in the allocation of man-hours for development. Software inspections require that time be set aside for them to be done properly. The time consumed by software inspections is time that cannot be

used for other activities, such as writing software. The amount of time that an inspection consumes is based on the size of the inspection team and the time spent on inspection tasks, e.g., the preparation rate and the inspection rate. Figure 2 is a simplified view of the steps that must be taken to perform inspections. It implicitly models the effort expended to move work products through all of the process steps associated with inspection; effort that takes time away from development activities.

The second impact that software inspections have on software development is in the detection of errors. Errors are found early in the life cycle. Errors are less expensive to find and correct during development than during testing. This impact is not explicitly shown in Figure 3, except that a higher error detection rate will increase the number of errors found early in the life cycle, rather than later during testing.

The third impact that software inspections have on software development is in the prevention of defects. Successful inspections attack the generation of future defects. Developers that are involved with inspections learn about the types of errors that they have been making and are likely to make during the project. This feedback about the errors they are making leads to fewer errors being made during the project. Figure 3 shows defect prevention, due to inspections, impacting the rate of error generation.

The fourth impact that software inspections have on software development is a reduction in the regeneration of errors. Errors that remain undetected often lead to new errors being generated. For example, undetected design errors will lead to coding errors, because they are coding to a flawed design. Software inspections detect errors early in the life cycle, thus reducing the amount of error regeneration. Figure 3 indicates that the errors escaping detection impact the error generation rate. The number of errors escaping detection is dependent on the preparation and inspection rates, as shown in Figure 3.

The four impacts that software inspections have on software development, mentioned above, are the foundation for the theory upon which the software inspection model is based. Many details of the implementation of the theory, using system dynamics techniques, are absent from Figure 2 and Figure 3, but are shown in detail elsewhere [14].

3.2: Example model output

The integrated model has been turned into a simple management flight simulator, allowing for simple experimentation. This section describes the user interface of the simulator and the output generated by its use.

Figure 4 shows the simple interface to the simulator. Input to the simulator is in the form of sliders. The simple interface has just five slider inputs: an on/off switch for inspections, the size of the inspection team, the percent of errors found during inspection, the percent of work products that fail inspection and the percent of incorrect error fixes.

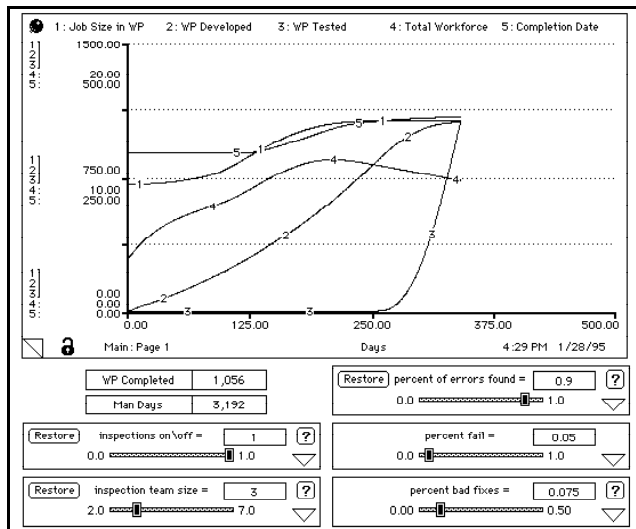


Figure 4. User interface of the inspection simulator.

Output from the simulator comes in two forms: numeric displays and graphs. Numeric displays show the current value of a simulation variable. Man-Days and Work Products Completed are two examples of numeric displays. Graphs, on the other hand, display the value of simulation variables versus time. Each output curve is labeled with a number for ease of reading. There may be multiple units of measure on the vertical axis, each matched to the number of the curve it is representing. The unit of measure on the horizontal axis is days. The five output curves represent: 1) currently perceived job size in terms of work products, 2) cumulative work products developed, 3) cumulative work products tested, 4) total size of workforce and 5) scheduled completion date.

A demonstration of the use of the system dynamic models for predicting the cycle time reduction due to a process improvement is in order. Using the integrated model of the baseline waterfall development life cycle and the software inspection process improvement, it will be shown how this modeling technique can be used for evaluating the impact that a proposed process improvement would have on development cycle time.

The following demonstration is a simulation of a hypothetical software team employing the simple inspection model presented in this paper. The project

being developed is estimated to be 64,000 lines of code requiring a total workforce of eight developers at the height of development. Two scenarios of the project development are simulated holding all variables fixed, except for the size of the inspection team and the percent of errors found during inspection.

Figure 5 is the output generated by executing the model with an inspection team size of six developers discovering 40 percent of the errors during inspection. When interpreting the graphical output, the story of the project is revealed. From Figure 5, the following story emerges. Curve 1, the currently perceived job size in work products, reveals that the project size was initially underestimated. As development progressed, the true size of the project was revealed. Curve 5, the scheduled completion date, was not adjusted even as it became apparent that the project had grown in size. Instead, curve 4, the total size of workforce, indicates that the workforce was increased in size. In addition, though not shown on this graph, the workforce worked longer hours to bring the project back on schedule. Curve 2, cumulative work products developed, reveals that the project appeared to be back on schedule, because there were no visible delays in development of work products. It was not until system testing that problems in development were discovered. Curve 3, cumulative work products tested, reveals that system testing did not go as smoothly as expected. The poor performance of the inspection team pushed the discovery of errors back to system testing. During system testing it was revealed that there was a good amount of rework to be done and as a result, the scheduled completion date, curve 5, was once again pushed back.

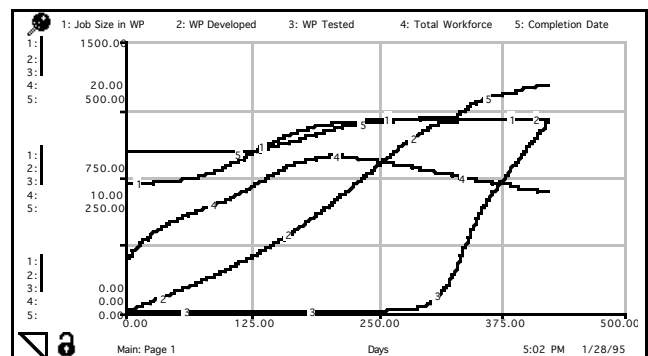


Figure 5. Software inspection scenario 1.

Figure 6 is the output generated by executing the model with an inspection team size of three developers discovering 90 percent of the errors during inspection. The story is much the same as that shown in Figure 5. The big difference between Figures 5 and 6 is shown by curve 3, cumulative work products tested. Using more effective software inspections, this project was able to

discover errors early in the life cycle and correct them for much less cost than if they had been found in system test. In addition, there were no major surprises in system testing as to the quality of the product developed. Therefore, with no major amount of rework to be performed in system test, the project was able to finish close to its revised schedule.

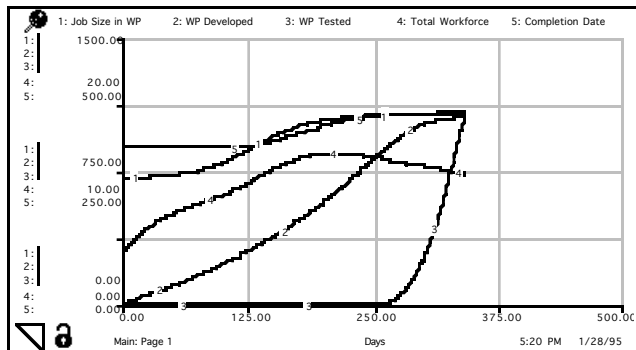


Figure 6. Software inspection scenario 2.

4: Conclusions and future work

Our research grew out of the questions posed in Section 1.2 concerning the impact of process improvements on software development cycle time. Our approach in answering those questions has been to use system dynamics modeling to model the software development process, allowing experimentation with the system. We have tried to demonstrate how this technique may be used to evaluate the effectiveness of process improvements. At this point in our work we have developed a base model of the waterfall development life cycle and a process improvement model of software inspections. We plan to continue this effort by developing a base model of the incremental development process and creating a library of process improvement models. Some examples of process improvements that we plan to add to our library are meetingless inspections, software reuse and risk management. We then plan to validate our base and process improvement models in several software development organizations. Finally, another area of future research is to generalize the interface between the base process models of software development and the models of process improvements. The interface is analogous to tool interfaces. A well defined, generalized interface would facilitate integration of base process models with process improvement models.

References

[1] Tarek Abdel-Hamid and Stuart E. Madnick, SOFTWARE PROJECT DYNAMICS An Integrated

Approach, Prentice-Hall, Englewood Cliffs, New Jersey, 1991.

[2] Tarek K. Abdel-Hamid, "THINKING IN CIRCLES," American Programmer, May 1993, pp. 3-9.

[3] Barry W. Boehm, SOFTWARE ENGINEERING ECONOMICS, Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

[4] Ken W. Collier and James S. Collofello, "Issues in Software Cycle Time Reduction," International Phoenix Conference on Computers and Communications, 1995.

[5] Raymond Dion, "Process Improvement and the Corporate Balance Sheet," IEEE Software, July 1993, pp. 28-35.

[6] Jay W. Forrester, Industrial Dynamics, The M.I.T. Press, Cambridge, MA, 1961.

[7] Jay W. Forrester and Peter M. Senge, "Tests for Building Confidence in System Dynamics Models," System Dynamics. TIMS Studies in Management Sciences, 14 (1980), pp. 209-228.

[8] Mark C. Paulk, Bill Curtis, Mary Beth Chrissis and Charles V. Weber, "Capability Maturity Model, Version 1.1," IEEE Software, July 1993, pp. 18-27.

[9] T. S. Perry, "Teamwork plus Technology Cuts Development Time," IEEE Spectrum, October 1990, pp. 61-67.

[10] Lawrence H. Putnam, "General empirical solution to the macro software sizing and estimating problem," IEEE Transactions on Software Engineering, Vol. SE-4, No. 4, July 1978, pp. 345-361.

[11] George P. Richardson and Alexander L. Pugh III, Introduction to System Dynamics Modeling with DYNAMO, The M.I.T. Press, Cambridge, MA, 1981.

[12] Howard A. Rubin, Margaret Johnson and Ed Yourdon, "With the SEI as My Copilot Using Software Process 'Flight Simulation' to Predict the Impact of Improvements in Process Maturity," American Programmer, September 1994, pp. 50-57.

[13] George Stark, Robert C. Durst and C. W. Vowell, "Using Metrics in Management Decision Making," IEEE Computer, September 1994, pp. 42-48.

[14] John D. Tvedt, "A System Dynamics Model of the Software Inspection Process," Technical Report TR-95-007, Computer Science and Engineering Department, Arizona State University, Tempe, Arizona, 1995.

