Initial Experiences in Software Process Modeling

Ray Madachy, Ph.D.
Denton Tarbet, Ph.D.
Litton Guidance and Control Systems
5500 Canoga Ave., M/S 50
Woodland Hills, California 91367-6698

818/715-3965
Fax 818/715-2068
{madachyr, tarbetj}@littongcs.com

Abstract

Litton's Guidance and Control Systems (GCS) Division has been using system dynamics to create mostly small-scale models for investigating managerial process issues and supporting personnel training. At the project level, these include models for planning specific projects, studying Brooks' Law and hiring issues, an interactive earned value model, requirements volatility and a detailed peer review model. The perspective of some of the models has been at a multi-project or departmental level including domain learning, product-line reuse processes and resource contention among projects. Insights provided by the models have supported decision-making at different levels and helped galvanize process improvement efforts. The training applications have added spark in classes and improved overall learning.

The models encapsulate collective knowledge of modeling participants, and support organizational learning. By examining the models and simulated behavior, managers share a process vision and can discuss issues against the common models. The models have helped managers understand the key factors in complex scenarios. Knowledge of the interrelated technical and social factors coupled with simulation tools has enabled GCS to improve their planning and management processes.

Modeling is also used to support training of software managers and leads. Topics including earned value techniques, productivity estimation, requirements volatility effects and extrapolation of project tracking indicators have been presented with simulation models. Some of these are "flight training" scenarios that the students interact with to practice project control.

Though we are in the early stages of many of these efforts, they will be followed through and reported on at a later time. This paper describes our introductory experiences and plans for the future.

1. Introduction and Background

GCS achieved CMM Level 4 certification in 1998, and process simulation is being used to support continued improvements. We have started some efforts in the areas of managerial training, project and organizational modeling. As a high maturity organization, existing process performance baselines provide leverage in developing and calibrating meaningful simulation models. Experimenting with these models has identified opportunities for process improvement, including the applicable ranges of conditions for improvement. Longitudinal results of these efforts will be reported in the future.

Software development is a dynamic and complex process as there are many interacting factors throughout the lifecycle that impact the final cost, schedule and quality. Unfortunately these effects are rarely accounted for on software projects. Many organizations and their models gloss over process interactions and feedback effects, but these must be recognized to effect greater improvements. GCS is attempting bring simulation into the everyday fold.

Knowledge gleaned from a global perspective that considers these interactions can be represented in executable simulation models that serve as a common understanding of an organization's processes. Systems thinking, as a way to find and bring to light the structure of the organizational system which influences its dynamic behavior, together with system dynamics as a simulation methodology provide critical skills to manage complex software development.

Models are also an excellent vehicle for learning efforts on both organizational and personal levels. Organizational learning in the context of a software process involves translating the common "mental model" of the process into a working simulation model that serves as a springboard for increased learning and improvement. Mental models must be made explicit to frame concerns
and share knowledge among other people on a team.
Everyone then has the same picture of the process and its
issues. Collective knowledge is put into the models as the
team learns. Elaborated representations in the form of
simulation models become the basis for process
improvement.

For personal learning, we've been using models
to demonstrate basic and advanced project management
concepts through simulation. Student awareness is
heightened when "games" like simulations are used.

Table 1: Characterization of Initial Litton Case Studies

<table>
<thead>
<tr>
<th>Scope / Purpose</th>
<th>Portion of lifecycle</th>
<th>Development project</th>
<th>Multiple, concurrent projects</th>
<th>Long-term product evolution</th>
<th>Long-term organization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strategic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Management</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Planning</td>
<td>stage-based cost/schedule estimation</td>
<td>staffing project cost/schedule/quality estimation</td>
<td>reuse costs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control and operational management</td>
<td>stage tracking</td>
<td>earned value tracking</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process improvement and technology adoption</td>
<td>peer review optimization</td>
<td>peer review effects on project</td>
<td>inter-project reuse processes</td>
<td>product-line reuse processes</td>
<td></td>
</tr>
<tr>
<td>Understanding</td>
<td>requirements volatility</td>
<td>core reuse dynamics</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Training and learning</td>
<td>managerial metrics training</td>
<td>managerial metrics training</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. Implementation

The Software Engineering Process Group (SEPG) is responsible for organizational analysis and training, and develops the models. The SEPG and senior executives first confer on model purpose. Team techniques are then used to elicit process views and to formulate explicit representations of the mental models into executable simulation models. The Software Director helps to "sell" the analyses and results to other affected groups, thus improving intergroup coordination. Hard issues are dealt with using the model results as objective evidence. Previous hand waving is replaced with explicit quantitative models.

The rest of this section provides a brief overview of some models developed at GCS, where each subsection describes a main application category.

2.1 Software Manager Training

Selected process management principles are demonstrated through the use of live classroom simulations. Software managers and other designated leads receive training from the SEPG in project management, software metrics and other related subjects. Live simulations have been used in the training venues for students to better visualize metrics trends and to improve their control techniques via interactive discussions. Use of the dynamic models has enlivened the training sessions and stirred thought provoking discussions. Using the models in real-time also allows for quick simulation runs to explore issues brought up during discussion. For example, a posed question may be translated into model input and the simulation run for all to see the outcome. This often happens when presenting managerial subjects, as students propose specific

particularly when they participate. Visual dynamic graphs provide faster and easier remembered learning compared to traditional lecture format. Exploration is encouraged through the ability to modify and reply the models.

A process modeling characterization matrix initially developed by Kellner is used to place the various GCS studies in Table 1. See [1] for information regarding the characterization framework.
scenarios that can be quickly evaluated through the simulation model.

The first models used for training purposes are an earned value model, Brooks' Law model and a simple estimated productivity model. The earned value model is also used as a basis for actual project control and evaluating the impact of requirements volatility (see section xx), and the Brooks Law model is also used to determine optimal staffing for some real projects.

2.1.1 Earned Value Model

Earned value is an approach for monitoring performance against budgeted plans. Earned value refers to the dollar value of work accomplished, where the value of a given task is defined in a cost/schedule budget baseline. Value is earned after completion of budgeted milestones as work proceeds. Objective milestones consist of directly observable steps or events in the software process, and earned value is therefore a measure of progress against plan.

The earned value model has been used to 1) demonstrate what earned value is, and 2) show how it can be used to manage a project using feedback. The model implements the basic formulas for Cost Performance Index (CPI) and Schedule Performance Index (SPI) using dynamic cost indicators. CPI represents cost efficiency against plan and SPI represents schedule efficiency against plan. See Figure 1 for the earned value model.

The model is used to educate managers to spot trends early in order to effect control. This is done by monitoring the slope of earned value trends as opposed to current static values only. A classic example involves evaluating progress compared to planned completions. A quick glance at early trends to see whether task completion is behind may be deceiving. Figure 2 shows sample output from the model.

For example, a project that starts out solving easy problems at a fast pace creates an illusion of early overall completion. Whereas the slope of the progress line, also borne out in the dynamic CPI/SPI curves, clearly shows a worsening negative trend. A partial simulation of two comparative projects is shown to students for discussion before completing the simulation. Many people fail to consider the changing slopes of the initial progress trends, and erroneously choose which project will perform best.

Part of the training involves managers interacting with project simulations as they are running. This is often called "flight simulation" analogously. The earned value model is used for hands-on practice in this manner. For example, simulated CPI and SPI trends are monitored by an individual who can control certain parameters during the run. Typically the simulation is slowed down or paused such that individuals can react in time by varying...
sliders. In the earned value simulation, managers can control the staffing levels interactively as the simulation progresses. See Figure 3 for a sample flight simulation interface to the earned value model.

Figure 2: Earned Value Model Output
2.1.2 Brooks' Law Model

In the early software engineering classic *The Mythical Man-Month*, Fred Brooks stated:

> Adding manpower to a late software project makes it later [2].

His explanation for the law was the additional linear overhead needed for training new people and the nonlinear communication overhead (a function of the square of the number of people). These effects have been widely accepted and observed by others. The simple model in Figure 4 describes the situation, and will be used to test the law.

The model is conceived around the following basic assumptions:

- New personnel require training by experienced personnel to come up to speed
- More people on a project entail more communication overhead
- Experienced personnel are more productive than new personnel, on average.

It is built on two connected flow chains representing software development and personnel. The software development chain assumes a level of requirements that need to be implemented. The requirements are transformed into developed software at the software development rate. The level of developed software represents progress made on implementing the requirements. Project completion is when developed software equals the initial requirements. Software size is measured in function points, and the development rate is in function points/person-day.
The software development rate is determined by the levels of personnel in the system: new project personnel who come onto the project at the personnel allocation rate, and experienced personnel who have been assimilated (trained) into the project at the assimilation rate.

The model is a high-level depiction of a level-of-effort project, and it will be exercised by adding more people via the personnel allocation rate. This will allow for tracking the software development rate over time and assessing the final completion time to develop the requirements under different hiring conditions.

As time progresses, the number of requirements decreases since it represents requirements still left to implement. These requirements are processed over time at the software development rate and become developed software, so requirements decline as developed software rises. The software development rate is constrained by several factors: the nominal productivity of a person, the communication overhead %, and the number of personnel. The effective number of personnel equals the new project personnel plus the experienced personnel minus the amount of experienced personnel needed for training the new people. The communication overhead % is expressed as a nonlinear function of the total number of personnel that need to communicate. The experienced personnel needed for training is the training overhead percentage as a fraction of a full-time equivalent experienced personnel. The default of .25 indicates one quarter of an experienced person is needed to train a new person until they are fully assimilated.

The bottom structure for the personnel chain models the assimilation of new project personnel at an average rate of 20 days. In essence, a new person is trained by one fourth of an experienced person for an average of 20 days until they become experienced in the project.

The nominal productivity is set to 1, with the productivities of new and experienced personnel set to .8*nominal productivity and 1.2*nominal productivity respectively as a first-order approximation.

2.1.2.1 Sample Results

The default steady state behavior of the model shows a final completion time of 274 days to develop 500 function points with a constant staff of 20 experienced personnel. Experimental runs are then performed to optimize the schedule finish. The first perturbation run injects an instantaneous pulse of 5 new people at the 100th day, and on the next run 10 people are added at the 100th day. Figure 5 is a sensitivity plot of the corresponding software development rates for the default condition and the two perturbation runs.
With an extra staff of five people (curve #2), the development rate nosedives, then recovers after a few weeks to slightly overtake the default rate and actually finishes sooner at 271 days. However, when 10 people are added the overall project suffers (curve #3). The initial productivity loss takes longer to stabilize, the final productivity is lower with the larger staff and the schedule time elongates to 296 days. The plunge and smooth recovery seen on both are the training effect. The extra staff gains in the first case are greater than the communication losses, but going from 20 to 30 people in the second case entails a larger communication overhead compared to the potential productivity gain of having more people.

The model shows how the law holds only under certain conditions, since there is a tradeoff between the number of added people and the time in the lifecycle. There is a threshold of new people that can be added until the schedule suffers. A specific addition of people may be tolerable if injected early enough, but not later. Conversely the project time determines how many can be effectively added.

Based on the insight provided, we may now clarify Brooks' Law. Adding manpower to a late software project makes it later if too much is added too late. That is when the additional overhead is greater than productivity gains due to the extra staff. See [3] for more details.

2.1.3 Estimated Productivity

A simple model of estimated productivity has been developed, which represents the ongoing estimate of the productivity to be achieved at project completion. It combines an estimate-to-complete (EAC) with a size stability indicator (estimate of the final size). Tracking this indicator includes periodic updates to the EAC accounting for actuals-to-date and monitoring of size stability. The model demonstrates to managers how to combine these dynamic trends in order to track estimated productivity and stay on top of the project. Figure 6 shows sample output from the model.
2.2 Project Planning and Control

A new project is being used as a pilot for dynamic planning and control to raise the visibility of certain planning issues and to monitor the project. The critical project is a large integrated system development with incremental releases. Major elements of the system dynamics model include incremental development, Brooks' Law effects, hiring delays, earned value and requirements volatility.

A first-cut baseline of the project staffing dynamics and schedule milestones was derived with the COCOMO cost model. The Costar cost estimation tool was used to develop a baseline staffing profile for the planned increments using a sophisticated incremental version of COCOMO. The profile was then translated into manpower addition and transfer rates to achieve the desired staffing, and used as a basis for exploring requirements volatility and hiring issues. The COCOMO model assumes breakage from requirements changes is constant throughout the project, but different behavior is expected on this particular project. Due to software dependency on unprecedented hardware development and other factors, most breakage is expected during integration. This indicates a model clash between static COCOMO and assumed actual breakage patterns.

Several other dynamic effects are also being explored on this project. A Brooks' Law model is used to determine optimal staffing-up rates and team sizes (see Section 2.1.2 for more detail on the Brooks' Law formulation). An earned value model is connected with the actual and planned task completions in order to predict and monitor earned value trends like CPI and SPI.

The earned value model was modified to incorporate requirements volatility. The delayed effect on CPI and SPI of unbudgeted requirements changes has been demonstrated during senior management reviews of selected projects including this one. The public visualization has been used to improve communication between software engineering and project control personnel.
2.3 Brooks' Law and Hiring

GCS is also using the Brooks' Law model to support hiring and project transfer decisions. The model shows when under what circumstances hiring people on a late project will help, and those when it won't. Effort is increased in every case, and schedule time can be recovered only under certain stringent conditions. In particular, there are losses due to training of new staff and communication overhead. If the teams are relatively small, and hiring doesn't continue very late, then schedule performance can be slightly improved. This is a multi-attribute problem that system dynamics provides insight for.

Hiring policies have also been analyzed. Despite project plans and aggressive recruiting, bringing on new hires entailed months of delays. The average hiring delay is slightly greater than two months for an individual, though business management virtually never accounted for these effects. After the simulation results were presented, management started producing more realistic and conservative hiring plans.

2.4 Project Contention Model

A model of resource switching was motivated by reactive behavior on the part of executive management whereby senior people were juggled between projects to fix short-term problems per customer whims. The software engineering department wanted to demonstrate that the practice was counter-productive in terms of cost/schedule performance. Both projects suffered learning curve and communication overhead drains that overwhelmed any gains on the initial "troubled" project. Additionally, the juggled individuals themselves experienced losses due to the multiple context switching. Their net output is much less when attempting to work on several tasks at once compared to a single task.

The model is shown in Figure 8. One key to the model is expressing the learning curve as a function of volume (i.e., tasks produced) rather than time per se. See [4] for a detailed discussion of software learning curves. When there are interruptions and context switching between projects, then an individual going back to a project requires longer than the interruption time to reach previous productivity levels.

A Delphi-poll and other queries were used to gauge the learning curve losses. It was estimated that 2-3 weeks are needed after a one-week break to get back into the productive mode of the original project. Added together, the respective losses and gains for the involved projects produced a net loss compared to not hot-switching personnel. Even more important, the anticipated schedule slips were directly correlated to customer dissatisfaction. These results were shown to the executives who could not disagree with the conclusions, and they have changed their corresponding policies. Figure 9 shows sample output.

The effects of working on multiple tasks has also been quantified by Weinberg [5]. He developed a table of relative productivity vs. the number of tasks that validates the results found in our model, and is a good lesson for software organizations to keep in mind.
Figure 8: Project Contention Model Diagram

Figure 9: Project Contention Model Results
2.5 Peer Review Model

An inspection model was initially developed at Litton’s Data System Division (DSD) in conjunction with USC to answer the research question: What are the dynamic project effects of performing inspections? It has now been modified at GCS to model other types of peer reviews, particularly walkthroughs. The model portion for the product and error chains is shown in Figure 10.

The dynamic model serves to examine the effects of peer review practices on cost, schedule and quality throughout the lifecycle. It uses system dynamics to model the interrelated flows of tasks, errors and personnel throughout different development phases and is calibrated to industrial data. Details on the original model are in [6].

The original inspection model was calibrated to data at DSD. GCS walkthrough data was used to parameterize the model and scenarios were run to quantify the expected cost and schedule impacts of peer reviews. The experimentation has helped to solidify plans for peer reviews on specific projects. On others, it was shown that performing additional formal peer reviews may not be worthwhile. The model has also helped to educate executives and developers alike on the importance of peer reviews, and provided motivation to optimize the peer review processes.

The model is general enough that it required no structural changes to handle other peer review types besides inspections. The following calibration parameters were determined for each project based on existing GCS baseline data as refined for specific projects:
- nominal productivity
- review efficiency
- design defect density
- code defect density
- average design defect amplification

Project specific parameters include:
- job size
- COCOMO effort parameter
- schedule constraint.

The parameters below represent management decisions regarding schedule processes for each project:
2.6 Core Software Reuse Model

Product line reuse is a major risk item. GCS has several product lines, one of which has a team developing reusable components for multiple projects. The core software is shared among projects within the product line. Unfortunately the planned reuse levels are rarely met on a project. It is not known whether current reuse practices involving the core software library are economical or not. Changes to the core library by one project often adversely affect other projects. These side effects create new problems that often lead to cost and schedule overruns. Experience indicates a conflict with business development policies, which presume significant cost savings.

The observed effects are due to increasing software entropy. A software system that undergoes continuous change will grow in complexity and will become more disorganized over time. This phenomenon is often attributed to conventional software architectures, whereby the entropy increases when interfaces are changed for tactical reasons. Generally the software lifetimes are much shorter than commonly expected. At Litton, this problem was exacerbated by the fact that over a dozen projects use the core reuse library simultaneously. Changes from a single project ripple into problems on other projects.

A product-line reuse model is being developed to analyze the dynamics of core software reuse. The effort is currently in the conception and data collection phases. Some past trend data serves as reference behavior for the model in terms of the core dynamics. In order to parameterize the model, reuse process data is currently being collected and analyzed, and the collection is also being automated.

Based on the eventual findings, an architecture team will investigate ways to improve reusability across projects over longer time horizons. Conversely, reuse criteria and reuse planning processes will be revamped. Figure 11 shows a high-level model of the situation.

![High-Level Model of Product Line Software Reuse](image)

3. Summary and Conclusions

System dynamics simulation is well suited to exploring process issues. Even small models are highly valuable for providing insight into dynamic trends. In fact, smaller is better when presenting results to people not familiar with process modeling or simulation. The
models are also well received as training vehicles for a fun and interactive learning experience. Simulation has proven to be a real "utility" player for the organization in that it can play different roles and attack varied types of problems. It supports learning at the organizational and individual levels.

An important thrust of the modeling work has been multi-project in nature, where studies look at specific well-defined problems. Rather than producing a complex model of the entire organization at this time, we have been using system dynamics for partitioned focused studies on common issues and problems in our environment. This has served to provide individual lessons which collectively have improved our management vision and actions.

There are important advantages to using simulation in the classroom. Simulation can be used to impart information in a more meaningful and dynamic way compared to traditional methods. Some principles are better visualized through time-based simulations. Live demonstrations also keep up the interest of the student audience as opposed to strict lecture, and interactivity serves to drill in the learning experience. Not all lessons can necessarily be enhanced by demonstrating management principles through live simulations. When to use live simulation should be considered on a per-case basis.

References


Biographies

Dr. Raymond Madachy is the Manager of the Software Engineering Process Group at Litton Guidance and Control Systems, which achieved CMM Level 4 in December 1998. He is also an Adjunct Assistant Professor in the Computer Science department at the University of Southern California teaching graduate-level Software Engineering. He has published over 35 articles and is currently writing the book Software Process Dynamics with Dr. Barry Boehm. He received his Ph.D. in Industrial and Systems Engineering at USC, has an M.S. in Systems Science from the University of California, San Diego and a B.S. in Mechanical Engineering from the University of Dayton. He is a member of IEEE, ACM, INCOSE, Tau Beta Pi, Pi Tau Sigma, serves as program chair for the International Forum on COCOMO and Software Cost Modeling, on the program committee for the California Software Symposium and the steering committee for the LA Software Process Improvement Network.

Dr. Denton Tarbet is the Director of Software Engineering at Litton Guidance and Control Systems. Previously he managed Systems Engineering at Edwards Air Force Base, was Director of the Computer Science Center at Tektronix, Vice-President of Engineering at Hydro-Aire, and has had other management and consulting positions for embedded real-time software development. He teaches seminars in Configuration Management, Project Management and Quality Assurance. He received a Ph.D. in Systems Engineering from the University of Houston, an M.S. in Operations Research from the University of Houston and a B.S. from Truman University.