

System Dynamics Modeling of an Inspection-Based Process

Raymond J. Madachy

USC Center for Software Engineering
University of Southern California
Los Angeles, CA 90089-0781
madachy@usc.edu

Software Engineering Process Group
Litton Data Systems
29851 Agoura Rd.
Agoura Hills, CA 91376-6008

Abstract

A dynamic simulation model of an inspection-based software lifecycle process has been developed to support quantitative process evaluation.

The model serves to examine the effects of inspection 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. It extends previous software project dynamics research by examining an inspection-based process with an original model, integrating it with the knowledge-based method for risk assessment and cost estimation, and using an alternative modeling platform.

While specific enough to investigate inspection practices, it is sufficiently general to incorporate changes for other phenomena. It demonstrates the effects of performing inspections or not, the effectiveness of varied inspection policies, and the effects of other managerial policies such as manpower allocation.

The results of testing indicate a valid model that can be used for process evaluation and project planning, and serve as a framework for incorporating other dynamic process factors.

1. Introduction

A large knowledge gap exists for measuring different software development practices and planning large software projects. There are few methods of quantitatively comparing dynamic characteristics of software development processes and assessing project risk. Very little data exists on the tradeoffs of alternative software processes, and expert knowledge is not utilized in tools for cost and schedule estimation or project risk assessment.

This research investigates system dynamics modeling of an inspection-based development process and knowledge-based techniques for cost, schedule and risk

assessment of large software projects. Eventually, a host of alternative software development processes can be modeled. By combining quantitative techniques with expert judgement in a common model, an intelligent simulation capability results to support planning and management functions for software development projects. With an executable simulation model, managers can assess the effects of changed processes and management policies before they commit to development plans.

The process model can also be used as a baseline for benchmarking process improvement when calibrated with metrics for specific environments.

The knowledge-based method aids in project planning by identifying, categorizing, quantifying and prioritizing project risks as an extension to the Constructive Cost Model (COCOMO) [5]. It also detects cost estimate input anomalies and provides risk control advice in addition to conventional cost and schedule calculation. Serving as a front end, it provides an initial project estimate to the simulation model.

The rest of this paper describes the dynamic simulation aspect of this work. Refer to [27] for details of the knowledge-based component and its integration with the simulation model.

2. Background

A major goal of software engineering is to produce higher quality software, while keeping effort expenditure and schedule time to a minimum. One practice used to achieve this is to review intermediate work products for the purpose of eliminating errors when they are less costly to fix. There are many ways to review software artifacts, ranging from very informal to highly rigorous, and involving a variety of participants.

The remainder of this section provides background on inspections and system dynamics simulation.

2.1 Inspections

Inspections were devised by Fagan at IBM as a formal, efficient and economical method of finding errors in design and code [12]. Since their introduction, the use of inspections has been extended to requirements, testing plans and more [13], [17], [32]. Thus, an inspection-based process would perform inspections on artifacts throughout system development such as requirements descriptions, design, code, test plans, user guides or virtually any engineering document.

As a form of peer review, inspections are carried out in a prescribed series of steps including preparation, having an inspection meeting where specific roles are executed, and rework of errors discovered by the inspection. Specific roles include moderator, scribe, inspectors and author.

By detecting and correcting errors in earlier stages of the process such as design, significant cost reductions can be made since the rework is much less costly compared to fixing errors later in the testing and integration phases [13], [32], [18], [25].

A wide range of inspection metrics have been reported. For example, guidelines for preparation and inspection rates often differ geometrically as well as overall effectiveness (defects found per person-hour).

In general, results have shown that extra effort is required during design and coding for inspections and much more is saved during testing and integration resulting in reduced overall schedule. This reference behavioral effect is shown in [13] and represents a goal of this modeling; to be able to demonstrate this effect on effort and schedule as a function of inspection practices.

Inspections can be used to instrument the process by collecting defect data to track patterns. For example, the defect density of artifacts through the lifecycle has been shown to decrease [22], [25].

There have been simplistic static formulas published for net return-on-investment or effort required as a function of pages or code [18], [32], but none integrated together in a dynamic project model. Neither do any existing models provide guidelines for estimating inspection effort as a function of product size, or estimating schedule when incorporating inspections.

2.2 System dynamics

System dynamics refers to the simulation methodology pioneered by Forrester, which was developed to model complex continuous systems for improving management policies and organizational structures [14], [15]. Models are formulated using continuous quantities interconnected in loops of information feedback and circular causality. The

quantities are expressed as levels (stocks or accumulations), rates (flows) and information links representing the feedback loops.

The system dynamics approach involves defining problems dynamically in terms of graphs over time, and striving for an endogenous, behavioral view of the significant dynamics of a system such that the model can reproduce the dynamic problem of concern by itself. Ultimately, understandings and applicable policy insights are derived from the resulting model and corresponding changes are implemented [37].

The mathematical structure of a system dynamics simulation model is a system of coupled, nonlinear, first-order differential equations,

$$\mathbf{x}'(t) = \mathbf{f}(\mathbf{x}, \mathbf{p}),$$

where \mathbf{x} is a vector of levels, \mathbf{p} a set of parameters and \mathbf{f} is a nonlinear vector-valued function. State variables are represented by the levels.

2.2.3 Software project dynamics: When used for software engineering, system dynamics provides an integrative model which supports analysis of the interactions between activities such as code development, project management, testing, hiring, training, etcetera [3]. Knowledge of the interrelated technical and social factors coupled with simulation tools can provide a means for the organization to improve their processes.

Abdel-Hamid has developed a generic system dynamics model of software development [3]. Others have modified it to support project and process management in particular organizations [24], [39]. Details of the different dynamic models and their uses are provided in [27]. Contrasts between the Abdel-Hamid model and this one are described in section 3.2.2 below.

3. Method

Development of the inspection-based process model has drawn upon extensive literature search, analysis of industrial data, and expert interviews. The purpose of the modeling is to examine the effects of inspection practices on effort, schedule and quality during the development lifecycle. Both dynamic effects and cumulative project effects are investigated per phase. Quality in this context refers to the absence of defects. A defect is defined as a flaw in the specification, design or implementation of a software product.

One goal of the literature search is to help identify reference modes of the system. Reference modes help to define a problem, focus model conceptualization, and are important in later validation phases [36]. Reference behavior modes are identified at the outset as patterns over time, and used during validation to check the

simulation output. After experimental validation of the process model, knowledge of software phenomenology results as well as having a testbed for further experiments.

3.1 Industrial data collection and analysis

Data collection and analysis for Project A using an inspection-based process was performed at Litton Data Systems [25]. As an application for command and control, the software comprises about 500 thousand lines of code (KSLOC). Analysis of additional projects is documented in subsequent internal reports and [28]. Some results with important ramifications for a dynamic model include defect density and defect removal effectiveness lifecycle trends and activity distribution of effort. Calibrations are made to match the model to these defect density trends and activity distributions in selected test cases.

Data for the previous project at Litton in the same product line and having identical environmental factors except for the use of inspections was also collected. It provides several parameters used in calibration of the system dynamics model such as defect density and the average cost to fix a defect found during testing. It also enables a comparison to judge the relative project effects from performing inspections.

Several senior personnel at Litton and international experts were interviewed to obtain data and heuristics for process delays, manpower allocation policies and general project trends relevant to inspection practices.

The model is validated against Litton data such as the proportions of effort for preparation, inspection meeting, rework and other development activities; schedule; and defect profiles. See [25] and [28] for additional analysis of inspection data.

3.2 Model overview

The model covers design through testing activities including inspections of design and code, and is shown in Figure 1. The ITHINK modeling package is used to implement the system dynamics method [38].

It is assumed that inspections and system testing are the only defect removal activities, as practiced in some organizations. The model also assumes a team trained in inspections such that inspections can be inserted into the process without associated training costs.

Project behavior is initiated with the influx of requirements. Rates for the different phase activities are constrained by the manpower allocations and current productivity, and are integrated to determine the state variables, or levels of tasks in design, code, inspection, testing, etc. The tasks are either inspected or not during

design or code activities as a function of inspection practices.

Errors are generated as a co-flow of design and coding activities, and eventually detected by inspections or passed onto the next phase. The detection of errors is a function of inspection practices and inspection efficiency. All errors detected during inspections are then reworked. Note that escaped design errors are amplified into coding errors. Those errors that still escape code inspections are then detected and corrected during integration and test activities. The effort and schedule for the testing phase is adjusted for the remaining error levels.

Other sectors of the model implement management policies such as allocating personnel resources and defining staffing curves for the different activities. Input to the policy decisions are the job size, productivity, schedule constraints and resource leveling constraints. Effort is instrumented in a cumulative fashion for all of the defined activities. Learning takes place as a function of the job completion, and adjusts the productivity. Schedule compression effects are approximated similar to the COCOMO cost driver effects for relative schedule.

The actual completion time is implemented as a cycle time utility whose final value represents the actual schedule. It instruments the maximum in-process time of the tasks that are time-stamped at requirements generation rate and complete testing.

3.2.1 Model calibration: To gauge the impact of inspections, effort expenditure was nominally calibrated to the staffing profiles of COCOMO, a widely accepted empirical model. Correspondence between the dynamic model and COCOMO is illustrated in Figure 2. It is based on the phase effort and schedule of a 32 KSLOC embedded mode reference project.

For the nominal case of no inspections performed, the manpower allocation policy employs the curves to match the rectangular COCOMO effort profiles as a calibration between the models. Thus the area under the design curve matches the aggregated COCOMO preliminary and detailed design, and likewise for the coding and testing curves.

Unlike COCOMO however, the staffing curves overlap between phases. The resulting staffing profiles are scaled accordingly for specific job sizes and schedule constraints.

The integration and test phase also depends on the level of errors encountered, and the curve shown is calibrated to an error rate of 50 errors/KSLOC during testing. The testing effort includes both fixed and variable components, where the variable part depends on error levels.

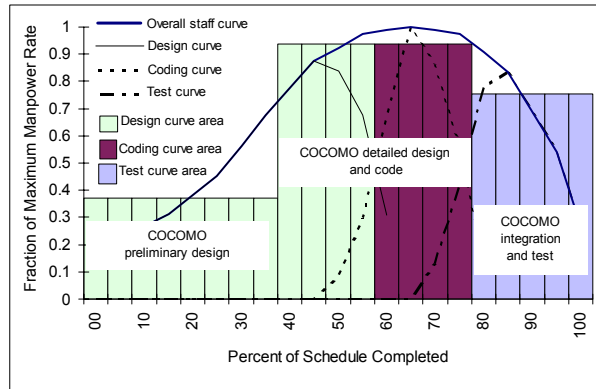


Figure 2: Correspondence Between Dynamic Model and COCOMO (no inspections)

When calibrating the model for other parameters, middle regions of published data trends were chosen as calibration points. Some parameters were set to Litton values when no other data was available.

Per the experience at Litton and accordant with other published data, inspection effort that includes preparation, meeting and rework generally consumes approximately 5-10% of total project effort. Guidelines for inspection rates are always stated as linear with the number of pages (or lines of code), and the inspection effort is set accordingly in the model such that inspection effort does not exhibit diseconomy of scale. The default inspection effort also corresponds to a rate of 250 LOC/hour during preparation and inspection with an average of four inspectors per meeting [18].

The rework effort per error is set differently for design and code. Litton data shows that the average cost to rework an error during code is about 1.3 times that for design. The cost to fix errors from [5] shows a relative cost for code of twice that of design. The model is set at this 2:1 ratio since it is based on a larger set of data. The resulting values are also close to the reported ranges of published data from JPL [22]. The testing fixing effort per error is set at about 5 hours, which is well within the range of reported metrics. The inspection efficiency is nominally set at 60%, representing the fraction of defects found through inspections.

The utility of the model for a particular organization is dependent on calibrating it accordingly per local data. While model parameters are set with reasonable numbers to investigate inspections in general, the results using the defaults will not necessarily reflect all environments.

Parameters to calibrate for specific environments including productivity, error rates, rework effort parameters, test error fixing effort, inspection efficiency and effort, and documentation sizes.

3.2.2 Comparison to previous models: Using the Abdel-Hamid system dynamics model in [3] as a conceptual reference, this model replaces the software production, quality assurance and system testing sectors. The model excludes schedule pressure effects, personnel mix, and other coding productivity determinants so as to not confound the inspection analysis.

Major departures from the Abdel-Hamid model cover manpower allocation, disaggregation of development activities, and error propagation. In [3], only generic quality assurance activities are modeled for error detection on an aggregate of design and code. Instead of the Parkinsonian model in [3] where QA completes its periodic reviews now matter how many tasks are in the queue, resources are allocated to inspections and rework as needed. This organizational behavior is thought to be an indication of a higher maturity level. Instead of suspending or accelerating reviews when under schedule pressure, error detection activities remain under process control.

The other major structural difference is the disaggregation of development phases. In [3], a single rate/level element represents an aggregate of design and coding. In this formulation, phases are delineated corresponding to those in the classic waterfall model so that design and coding are modeled independently.

4. Model demonstration and evaluation

This section shows how the model is tested with various sets of test cases and industrial data, and demonstrates use of the model for several managerial purposes. Results of the simulation runs and risk assessments are evaluated from several perspectives and insights are provided by experimenting with the model.

Model validation in system dynamics extends quantitative validation with an extensive range and depth of qualitative criteria. A battery of tests was used to consider the model's suitability for purpose, consistency with reality, and utility and effectiveness from both structural and behavioral perspectives. Simulation test case results are compared against collected data and other published data, existing theory and other prediction

models. Testing includes examining the ability of the model to generate proper reference behavior, which consists of time histories for all model variables.

Specific modeling objectives to test against include the ability to predict design and code effort sans inspection, preparation and inspection meeting effort, rework effort, testing effort, schedule and error levels as a function of development phase.

Approximately 80 test cases were designed for model evaluation and experimentation. The reference test case is for a job size of 32 KSLOC, an overall error injection rate of 50 errors per KSLOC split evenly between design and coding, and an error multiplication factor of unity from design to code. Refer to [27] for complete documentation of the simulation output for this test case and substantiation of the reference behavior.

Other test cases were designed to investigate the following factors: use of inspections, job size, productivity, error generation rates, error multiplication, staffing profiles, schedule compression, use of inspections per phase and testing effort.

4.1 Process assessment

This section shows the model being used in various facets of software process planning and analysis. The effects of performing inspections, error generation rates, defect amplification between phases, various staffing policies, schedule compression, and personnel experience are investigated with the system dynamics model. The reader is encouraged to read [27] for complete details on the experimental results.

4.1.1 Effects of performing inspections: Nine test cases were designed to evaluate the model for job size scalability, ability to be calibrated for productivity in specific environments, and to reproduce the effects of inspections.

Test case results are shown in [27] consisting of cost, schedule and error levels per phase. Predictions from COCOMO are provided as a basis of comparison for effort and schedule. One of the salient features of the system dynamics model is the incorporation of an error model, so comparison with the COCOMO estimate provides an indication of the sensitivity of testing effort to error rates (COCOMO does not explicitly account for error rates). It is seen that the simulation results and COCOMO differ considerably for the testing and integration phase as the process incorporates inspections.

It is not meaningful to compare schedule times for the different phases between COCOMO and the dynamic model because the non-overlapping step function profiles of COCOMO will always be shorter than the overlapping staffing curves of the dynamic model as seen in Figure 2.

The remaining errors is zero in each test case since ample resources were provided to fix all the errors in testing. If insufficient testing manpower or decreased productivity is modeled, then some errors remain latent and the resultant product is of lower quality.

Another major result is a dynamic comparison of manpower utilization for test cases with and without inspections. The curves demonstrate the extra effort during design and coding, less test effort and reduced overall schedule much like the reference effect on project effort previously shown by Fagan [13].

Many effects can be gleaned from the results such as the return-on-investment of performing inspections, relative proportions of inspection effort, schedule performance, defect profiles, and others. It is seen that inspections add approximately 10% effort in the design and coding phases for these representative test cases, and reduce testing and integration effort by about 50%. The schedule for testing can also be brought in substantially to reduce the overall project schedule. These results are corroborated by numerous projects in the literature and verified by experts interviewed during this research.

4.1.1.2 Analysis of inspection policies: The results described above correspond to equal inspection practices in design and code, though an alternate policy is to vary the amounts. The model can be used to investigate the relative cost effectiveness of different strategies.

The model provides insight to the interplay of multiple parameters that determine the threshold where inspections are worthwhile (i.e. the cost of fixing errors is more expensive than the inspection effort). The following sections provide some results of the multivariate analysis.

4.1.1.3 Error generation rates: The effects of varying error generation rates are shown in Figure 3 from the results of multiple test cases. The overall defects per KSLOC are split evenly between design and code for the data points in the figure.

It is seen that there are diminishing returns of inspections for low error rates. The breakeven point lies at about 20 defects/KSLOC for the default values of inspection effort, rework and error fixing.

While inspection and preparation effort stay fairly constant, the rework and test error fixing effort vary considerably with the defect rates. If there is a low defect density of inspected artifacts, then inspections take more effort per detected error and there are diminishing returns. For this reason, a project may choose not to perform inspections if error rates are already very low. Likewise it may not always be cost effective to inspect code if the design was inspected well as seen in [27].

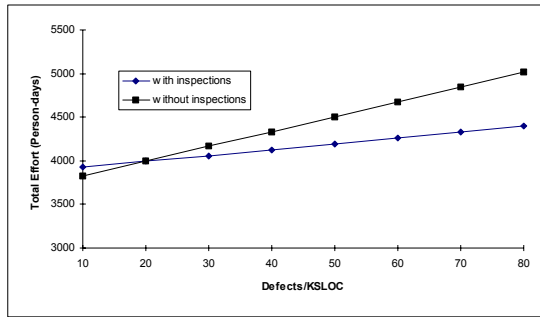


Figure 3: Error Generation Rate Effects

Though not explicitly modeled, there is a feedback effect of inspections that tends to reduce the error generation rate. As work is reviewed by peers and inspection metrics are publicized, an author becomes more careful before subjecting work to inspection.

4.1.1.4 Error multiplication: Defect amplification rates vary tremendously with the development process. For example, processes that leverage off of high-level design tools that generate code would amplify errors more than using a design language that is mapped 1:1 with code.

Figure 4 shows the effect of design error multiplication rate for performing inspections versus not. It is seen that as the multiplication factor increases, the value of inspections increases dramatically.

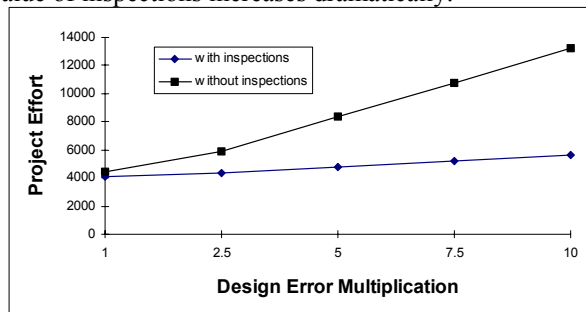


Figure 4: Error Multiplication Rate Effects

The effects of error multiplication must be traded off against error injection rates for different inspected documents. If error multiplication is very high from design to code, then design inspections become relatively more advantageous through multiplication leverage (and code inspections are very important if design inspections are not well-performed). Conversely if error multiplication is low, then inspection of design documents is not as cost effective compared to high multiplication rates.

4.1.1.5 Design vs. code inspections: An organization may opt to inspect a subset of artifacts. Literature suggests that design inspections are more cost effective than code inspections. The model accounts for several factors that impact a quantitative analysis such as the

filtering out of errors in design inspections before they get to code, the defect amplification from design to code, code inspection efficiency, and the cost of fixing errors in test. Experimental results show the policy tradeoffs for performing design inspections, code inspections, or both for different values of error multiplication and test error fixing cost [27].

4.1.2 Staffing policies: Different staffing curves are easily incorporated into the model. Test cases were run for “idealized” and rectangular staffing curves. The idealized staffing curve is a modified Rayleigh-like curve according to [5]. In the example test cases, a fixed staff size entails a longer project schedule by about 30%.

4.1.3 Schedule compression: The effects of schedule compression are demonstrated by varying the relative schedule (desired/nominal) from 1 to .7. The results are shown in Figure 5 with the total manpower rate curve for the four cases. With a reduced schedule, the average personnel level increases and the overall cumulative cost goes up nonlinearly.

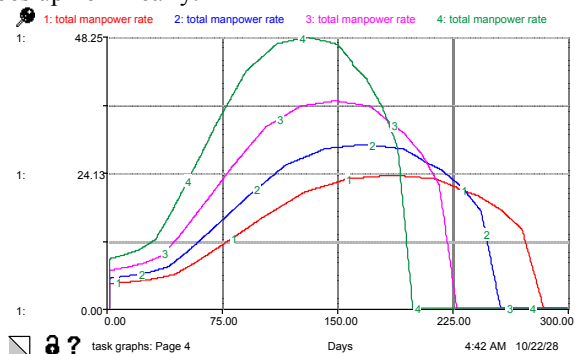


Figure 5: Total Manpower Rate Curves for Relative Schedule

(1: relative schedule=1, 2: relative schedule=.9, 3: relative schedule=.8, 4: relative schedule=.7)

4.1.4 Monte-Carlo simulation: A deterministic point estimate from a simulation exercise represents one of many possibilities. As a form of risk analysis, Monte-Carlo simulation selects random samples from an input probability distribution for multiple simulation runs, and provides a probability distribution as output.

In one example, inspection efficiency was represented using a normal distribution with a mean of .6 and a standard deviation of .2 for 50 runs to gauge the impact on resultant cost and schedule distributions [27].

4.1.5 Experience learning curve: In this example, a learning curve is implemented to simulate increasing productivity as a project progresses. Whereas COCOMO uses discretized experience factors, the dynamic model incorporates a continuous learning curve to better approximate the gradual increase in personnel capability.

The applications experience cost driver is simulated for design activities with a learning curve. The effort multiplier values used during the time span would approximate the effect of experience from a low rating increasing towards nominal over the course of approximately one year. When accounting for learning effects, the design phase takes longer to complete and consumes more personnel resources. A model that uses a realistic learning curve will better support time-phased task planning compared to static models.

4.2 Derivation of a detailed COCOMO cost driver

As an example of dynamic modeling contributing to static modeling, phase sensitive effort multipliers for a proposed cost driver *Use of Inspections* are experimentally derived. The model parameters *design inspection practice* and *code inspection practice* are mapped into nominal, high and very high cost driver ratings according to Table 1. As identified previously, the dynamic model is calibrated to zero inspection practice being equivalent to standard COCOMO effort and schedule.

Table 1: Rating Guidelines for Cost Driver *Use of Inspections*

Simulation Parameters		COCOMO Rating for <i>Use of Inspections</i>
<i>design inspection practice</i>	<i>code inspection practice</i>	
0	0	Nominal
.5	.5	High
1	1	Very High

Figure 6 shows the model derived effort multipliers by COCOMO phase. This depiction is for a composite of test cases, and there is actually a family of curves for different error generation and multiplication rates, inspection efficiencies, and costs of fixing defects in test.

This result is unique because it is the first instance of a COCOMO cost driver showing increased effort in some phases and reduced effort in others.

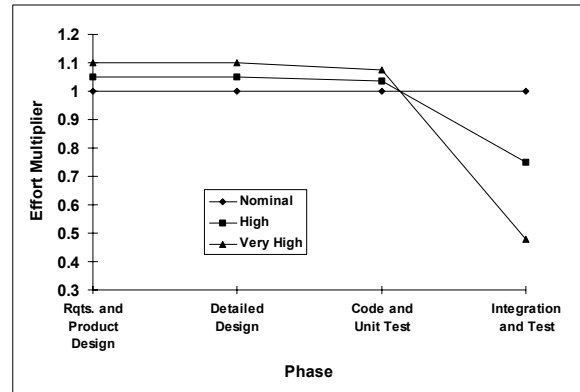


Figure 6: Effort Multipliers by Phase for *Use of Inspections*

4.3 Validation against industrial data

Replication of widely accepted phenomena was demonstrated in previous sections, as well as insights provided by the model regarding inspection effectiveness. In this section, results of comparing model output against specific project data are reported on. Comparison is problematic because no reported data sets contain all the parameters incorporated in the system dynamics model. No project data was found to be complete enough for a global comparison of total effort, schedule and inspection parameters. For instance, published data does not include project size, productivity constants or error multiplication rates and usually does not include phase effort and schedule. The results of performing inspections are normally aggregated for all activities or reported in limited fashion (e.g. “defect rates decreased by 50% during formal test” without providing the specific defect rates). Due to these limitations, the model output can only be validated in certain regards as shown in the following comparisons.

The first comparison is for return from inspections during testing, where the return is the test effort saved. Calculating the return was done for Litton Project A compared to the previous project in the same product line that did not use inspections, and test case 1.1 (full inspections) against test case 1.3 (no inspections) using default model values. The results are shown in Table 2 below. The Litton data is from [28]. This result shows that the model is fairly well-balanced to the experience at Litton, and the difference is acceptable given the wide variability of reported inspection metrics.

Testing effort and schedule performance for the inspection-based project at Litton is also compared against the model. The percentages are already adjusted for a 22% difference in project size. Comparisons are shown in Table 3. Using both default model parameters

and Litton-specific parameters, the model again matches actual project data well.

Table 2: Inspection Return Comparison

Project/Test Case	Return	Inspection Effort	Benefit/Cost
Litton Project A	20206 person-hours saved in test	8716 person-hours	2.32
Test case 1.1	613 person-days saved in test vs. test case 1.3	303 person-days	2.02

Table 3: Testing Effort and Schedule Comparison

Project/Test Case	Test Effort Reduction	Test Schedule Reduction
Litton Project A compared to previous project	50%	25%
Test case 1.1 vs. test case 1.3 with Litton productivity constant and job size for both cases	48%	19%
Test case 1.1 vs. test case 1.3	48%	21%

Another comparison afforded by available data is the ratio of rework effort to preparation and meeting inspection effort. This is done for Litton and JPL data as shown in Table 4, with JPL data from [22]. The Litton data is for 320 inspections, and the JPL data covers 203 inspections. No other published data separates out the rework effort. This demonstrates that the model predicts the relative amount of rework effort to overall inspection effort for these organizations within 9%.

Table 4: Rework Effort Ratio Comparison

Project/Test Case	Rework Effort	Preparation and Meeting Effort	Ratio
Litton Project A	2789 person-hours	5927 person-hours	.47
JPL (several projects)	.5 person-hours per defect	1.1 person-hours per defect	.45
Test case 1.1	100 person-days	203 person-days	.49

4.4 Evaluation summary

A detailed validation summary is provided in [27]. All variables in the reference behavior seem reasonable. Experts that were interviewed have also evaluated the

reasonableness of the time history output, the quantitative effects of inspections, and found no major deficiencies.

The model is shown to be consistent with reality by comparison with project data, idealized inspection trends and expert experience. However, more project data is desired for validation.

The model is suitable for its expressed purpose of demonstrating the effects of inspections from both structural and behavioral perspectives. Policy conclusions are sensitive to reasonable parameter variations of error generation and inspection efficiency. It is useful and effective as a suitable model because it is appropriate for an audience of software practitioners and provides new insights into the development process.

The model has met its goals of demonstrating the behavior of a project using an inspection-based process and is able to predict the relative change in effort, schedule and quality using inspections. When properly calibrated for specific environments, the model is expected to show high fidelity.

5. Conclusions and future research

The system dynamics model has successfully demonstrated several phenomena of software projects, including the effects of inspections and management decision policies. It was also shown that the model is scalable for project size and calibratable for productivity, error generation and error detection.

The model endogenously illustrates that under nominal conditions, performing inspections slightly increases up-front development effort and returns more in decreased testing effort and schedule. However, the cost effectiveness of inspections depends on phase error injection rates, error amplification, testing error fixing effort and inspection efficiency.

Complementary features of dynamic and static models for the software process were also shown. Phase effort and schedule from COCOMO was used to nominally calibrate the dynamic model, and the dynamic model furthers COCOMO by accounting for dynamic error generation. Common factors for schedule constraint and experience factors were used, with the dynamic model improving on static assumptions. Integration of the knowledge base and the dynamic model was also demonstrated, as well as derivation of a COCOMO cost driver from the simulation results.

There are many interesting problems and challenging opportunities for extensions of this research. A detailed outline of future directions is provided in [27], and highlights are identified below.

More validation against project data is desired as data becomes available. This may point to aspects of the model that need improvement.

Different error models for error generation and propagation can be incorporated into the model. Accounting for the severity and type of errors would support defect prevention.

Constructs in the model may be modified for other defect finding activities such as walkthroughs, cleanroom reviews, etc., that warrant dynamic examination. Eventually, other lifecycle processes can be incorporated to develop a unified system dynamics model for comparing multiple alternative processes.

Numerous variations on the staffing and scheduling policies can be tested, and the system dynamics model can be refined for finer granularity in the phases.

The model could also be enhanced to “drill down” further into the inspection process for the purpose of optimization, and look into the dynamics of inspection efficiency, rates, etc.

It can be extended for additional cost drivers, and investigation of their dynamic effects can be undertaken. Conversely, one can also derive effort multipliers for different static cost drivers by constructing unique models for investigation.

An experiment has been proposed to compare the discrete-event modeling approach developed at the Software Engineering Institute with system dynamics. Using a common test case to evaluate the effects of inspections, the resultant effort, schedule and quality profiles can be compared to help identify the relative merits of both approaches.

This research is coordinated with other projects at the USC Center for Software Engineering. The risk and anomaly rulebases will be updated for a new set of cost factors and model definitions in COCOMO 2.0 [10]. The new cost model will also incorporate dynamic aspects and risk assessment, and the scheme can be used to assist during negotiations with the Win Win spiral model [9].

6. Acknowledgements

The author would like to thank Barry Boehm, Behrokh Khoshnevis and Eberhardt Rechtin for their guidance, support and inspiration in this dissertation research. Thanks also to Litton Data Systems for their support of this work.

7. Bibliography

[1] Abdel-Hamid T, *Investigating the cost/schedule trade-off in software development*, IEEE Software, January 1990

- [2] Abdel-Hamid T, *Adapting, correcting, and perfecting software estimates: a maintenance metaphor*, IEEE Computer, March 1993
- [3] Abdel-Hamid T, Madnick S, *Software Project Dynamics*. Englewood Cliffs, NJ, Prentice-Hall, 1991
- [4] Ackerman AF, Fowler P, Ebenau R, *Software inspections and the industrial production of software*, in "Software Validation, Inspections-Testing-Verification-Alternatives" (H. Hausen, ed.), New York, NY, Elsevier Science Publishers, 1984, pp. 13-40
- [5] Boehm BW, *Software Engineering Economics*. Englewood Cliffs, NJ, Prentice-Hall, 1981
- [6] Boehm BW, *A spiral model of software development and enhancement*, IEEE Software, May 1988
- [7] Boehm BW, *Software Risk Management*. Washington, D. C., IEEE-CS Press, 1989
- [8] Boehm BW, Royce WE, *Ada COCOMO and the Ada process model*, Proceedings, Fifth COCOMO Users' Group Meeting, SEI, October 1989
- [9] Boehm BW, Bose P, Horowitz E, Scacchi W, and others, *Next generation process models and their environment support*, Proceedings of the USC Center for Software Engineering Convocation, USC, June 1993
- [10] Boehm BW, Clark B, Horowitz E, Westland C, Madachy R, Selby R, *Cost models for future software life cycle processes: COCOMO 2.0*, to appear in Annals of Software Engineering Special Volume on Software Process and Product Measurement, J.D. Arthur and S.M. Henry (Eds.), J.C. Baltzer AG, Science Publishers, Amsterdam, The Netherlands, 1995.
- [11] Day V, *Expert System Cost Model (ESCOMO) Prototype*, Proceedings, Third Annual COCOMO Users' Group Meeting, SEI, November 1987
- [12] Fagan ME, *Design and code inspections to reduce errors in program development*, IBM Systems Journal, V. 15, no. 3, 1976, pp. 182-210
- [13] Fagan ME, *Advances in software inspections*, IEEE Transactions on Software Engineering, V. SE-12, no. 7, July 1986, pp. 744-751
- [14] Forrester JW, *Industrial Dynamics*. Cambridge, MA: MIT Press, 1961
- [15] Forrester JW, *Principles of Systems*. Cambridge, MA: MIT Press, 1968
- [16] Forrester JW, Senge P, *Tests for building confidence in system dynamics models*, in A. Legasto et al. (eds.), TIMS Studies in the Management Sciences (System

- Dynamics), North-Holland, The Netherlands, 1980, pp. 209-228
- [17] Gilb T, *Principles of Software Engineering Management*. Addison-Wesley, Wokingham, England, 1988
- [18] Grady R, Caswell D, *Practical Software Metrics for Project Management and Process Improvement* Prentice-Hall, Englewood Cliffs, NJ, 1992
- [19] Green C, Luckham D, Balzer R, Cheatham T, Rich C, *Report on a Knowledge-Based Software Assistant*, Kestrel Institute, RADC#TR83-195, Rome Air Development Center, NY, 1983
- [20] Humphrey W, *Managing the Software Process*. Addison-Wesley, 1989
- [21] Kellner M, *Software process modeling and support for management planning and control*. Proceedings of the First International Conference on the Software Process, IEEE Computer Society, Washington D.C., 1991, pp. 8-28
- [22] Kelly J, Sherif J, *An analysis of defect densities found during software inspections*, Proceedings of the Fifteenth Annual Software Engineering Workshop, Goddard Space Flight Center, 1990
- [23] Khoshnevis B, *Systems Simulation - Implementations in EZSIM*. McGraw-Hill, New York, NY, 1992
- [24] Lin C, Abdel-Hamid T, Sherif J: *Software-engineering process simulation model*. TDA Progress Report 42-108, Jet Propulsion Laboratories, February 1992
- [25] Madachy R, Little L, Fan S, *Analysis of a successful inspection program*, Proceedings of the Eighteenth Annual Software Engineering Workshop, NASA/SEL, Goddard Space Flight Center, Greenbelt, MD, 1993
- [26] Madachy R, *Knowledge-based risk assessment and cost estimation*, Automated Software Engineering, Kluwer Academic Publishers, September 1995
- [27] Madachy R, *A software project dynamics model for process cost, schedule and risk assessment*, Ph.D. Dissertation, Department of Industrial and Systems Engineering, USC, December 1994
- [28] Madachy R, *Process improvement analysis of a corporate inspection program*, Proceedings of the Seventh Software Engineering Process Group Conference, May 1995
- [29] Mills H, Dyer M, Linger R, *Cleanroom engineering*, IEEE Software, September 1987
- [30] Paulk M, Curtis B, Chrissis M, and others, *Capability Maturity Model for Software*. CMU/SEI-91-TR-24, Software Engineering Institute, Pittsburgh, PA, 1991
- [31] Putnam L (ed.), *Software Cost Estimating and Lifecycle Control: Getting the Software Numbers*, IEEE Computer Society, New York, NY, 1980
- [32] Radice RA, Phillips RW, *Software Engineering - An Industrial Approach*, Englewood Cliffs, NJ, Prentice-Hall, 1988, pp. 242-261
- [33] Raffo D, *Evaluating the impact of process improvements quantitatively using process modeling*, working paper, Dept. of Industrial Administration, Carnegie Mellon University, 1993
- [34] Rehtin E, *Systems Architecting*, Englewood Cliffs, NJ, Prentice-Hall, 1991
- [35] Reddy Y, Fox M, Husain N, McRoberts M, *The knowledge-based simulation system*, IEEE Software, March 1986
- [36] Richardson GP, Pugh A, *Introduction to System Dynamics Modeling with DYNAMO*, MIT Press, Cambridge, MA, 1981
- [37] Richardson GP, *System dynamics: simulation for policy analysis from a feedback perspective*, Qualitative Simulation Modeling and Analysis (Ch. 7), Fishwick and Luker, eds., Springer-Verlag, 1991
- [38] Richmond B, and others, *IThINK User's Guide and Technical Documentation*, High Performance Systems Inc., Hanover, NH, 1990
- [39] Smith B, Nguyen N, Vidale R, *Death of a software manager: How to avoid career suicide through dynamic process modeling*, American Programmer, May 1993
- [40] Toth G, *Software technology risk advisor*, Automated Software Engineering, Kluwer Academic Publishers, September 1995
- [41] Weller E, *Three years worth of inspection data*, IEEE Software, September 1993, pp. 38 - 45