Simulation in Software Engineering with System Dynamics: A Case Study

Minghui Wu
[1]Dept. of Computer Science and Engineering, Zhejiang University City College, Hangzhou, China
[2]College of Computer Science and Technology, Zhejiang University, Hangzhou, China
Email: mhwu@zucc.edu.cn

Hui Yan
Dept. of Computer Science and Engineering, Zhejiang University City College, Hangzhou, China
Email: yanh@zucc.edu.cn

***Abstract*—There are many complexities including dynamic behavior and feedback mechanism as well as various interacting factors in the practical software development. Software Engineering education is facing difficulties because students have limited engineering experience and they can hardly understand typical phenomena occurring in software projects. System Dynamics is a continuous modeling method describing the interaction between project factors. It forces one to consider system behavior in global view. The simulation models encapsulate collective knowledge of software engineering fields. They support the training that the students can interact with to practice project control, which will help students understand the key factors and behaviors in complex scenarios. In this paper, Brooks' Law and the effects of Pair Programming in eXtreme Programming (XP) were used as case study to demonstrate the basic concepts, analysis and modeling processes of system dynamics. The simulation results show the advantages of the approach. Finally, the uses of system dynamics approach in critical areas of software engineering are presented.**

***Index Terms*—System Dynamics; Modeling; Simulation; Software Engineering Education**

I. INTRODUCTION

Software development is a dynamic and complicated process. In the practical development, the software system usually has the large complexity including system uncertainty and random, complex dynamic behavior and feedback mechanism [1]. The behavior of a system is principally caused by its structure. The structure includes not only the physical aspects, but also the policies and processes, both tangible and intangible. There are many interacting factors throughout the lifecycle that impact cost and schedule of the development project, and quality of the developed software product. For example: The budget, the work strength, the schedule, the personnel productivity, the defect ratio, the communication overhead, the number of developers and so on. These variables have mutual influences, form interactions and feedbacks.

University education needs to provide their students majoring in computer not only technology-related skills, but also a basic understanding of typical phenomena occurring in industrial software projects [2]. Software engineering education is difficult because usually students lack of engineering experiences. Therefore, using some appropriate methods and tools to help them is especially important, and the software development process modeling and simulation is a kind of these suitable methods. M.I Kellner et al. [1] had clustered the many reasons for using simulations of software processes into six categories of purpose: 1) strategic management; 2) planning; 3) control and operational management; 4) process improvement and technology adoption; 5) understanding; and 6) training and learning.

Software development is a complex process, which usually cannot be accurately understood by a human being by intuition. Modeling can make people concentrate on what's they have interest and ignore others by abstracting the system. Thus, a simulation is often the effective way to help them. Common purposes of simulation models are to provide a basis for experimentation, predict behavior and answer 'what if' questions. Simulation allows a researcher to estimate the behavior of an existing system under some conditions and can maintain much better control over experimental conditions. Simulation also allows study a system with a long time frame in compressed time, and vice versa.

Software development process simulation models have been used to capture dynamic interactions inherent in software development projects as well as process level issues. Usually, workflow-like discrete event models are used to describe process steps. However, they may not have enough events to represent feedback loops accurately [3]. For instance, the late project progress can increase developers' pressure, driving them to raise the productivity. The project group might choose working overtime to catch up with the schedule. Under high pressure and working overtime may improve the outputs in a month generally, but simultaneously may increase personnel's fatigue, which may lead to increase the error

This paper is based on "System Dynamics Modeling and Simulation in Software Engineering Education" by Minghui Wu, Hui Yan, which appeared in the Proceedings of the 9th International Conference for Young Computer Scientists(ICYCS 2008) Zhangjiajie, China, November 18-21, 2008. © 2008 IEEE.

ratio in high probability. Then the Quality Assurance works and redo works will be increased. And when the degree of fatigue reaches certain high level, the productivity will drop sharply. So the "net effect" of work overtime to project progress is difficult to make a conclusion intuitionally. In the software development these factors which often been neglected have important effects to project success or failure. Systems Dynamics (SD) is a continuous modeling method, can solve these problems well.

The rest of this paper is organized as follows. In the next section, we provide a background of system dynamics. As a case study, section 3 firstly presents the Brooks' Law and models it with system dynamics, followed with the simulation result analysis. In section 4, as another case, a system dynamics model was built to evaluate the effects of Pair Programming in eXtreme Programming. The uses of system dynamics in some critical areas of software engineering domain were presented in section 5. Finally, we conclude in Section 6.

II. System Dynamics Background

The system dynamics introduced by J. W. Forrester applies the engineering principles of feedback and control to social systems [4]. In system dynamics, a system is defined as a collection of elements that continually interact with each other or outside elements over time, to form a unified whole. The two important elements of the system are structure and behavior. The structure is defined as the collection of components of a system, and their relationships. The structure of the system also includes the variables that have important influences on the system. The behavior is defined as the way in which the elements or variables composing a system vary over time. The fundamental philosophy of system dynamics is based on the premise that the behavior of a system is caused principally by its underlying structure [5].

A. Constructs of SD models

System dynamics models describe the system in terms of "flows" that accumulate in various "levels", with "auxiliary" variables and "constant".

A "level" is an accumulation over time of "flows" that come into and go out. The flows increasing and decreasing a level in the speed called "rates".

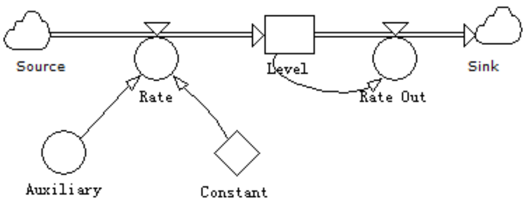

Figure 1. Symbols of system dynamics model

The flows can be dynamic functions of other "auxiliary" variables and "levels". As the simulation advances time in small evenly spaced increments, it computes the changes in levels and flow rates. As figure 1 shown, it's natural to image the system dynamics process as a continuous, fluid-like process of a liquid accumulating in and flowing out a container. In software development, for example, the error generation rate may be treated as a "flow" and the current number of errors could be treated as a "level".

Sometimes, when the origin of a flow is out modeler's interest, the flow's origin is called a "source". Similarly, when the destination of a flow is not of concern, it is called a "sink". For example, where the workforce hired from is a "source" and where they go after them leaving the project is a "sink".

Currently there are many modeling tools available for software process modeling and simulation, such as iThink, PowerSim, and Vensim. In our practices, PowerSim was used in modeling and simulation.

B. Procedure of SD modeling

Based on the recommendations provided in [6], the general idea of system dynamics can be described in the procedure with five main steps, and those steps are expected to be iterated several times.

1) Problem analysis. The first step of SD modeling is to answer following questions: a) what is intended to be modeled? b) what is the scope of the model? c) and what behaviors need to be analyzed in the model? In fact, only when the scope of the problem is reasonably focused, the problem can be analyzed deeply.

2) Eliciting key elements. There many factors are responsible for generating the observed behavior in a system. In this step, important objects and variables, both tangible and intangible, that are believed to be responsible for generating the observed behavior will be identified.

3) Definition of the cause-effect diagram. After elicited the key elements of system, the next step is to identify their cause-effect relationships. The diagram encompasses and links all cause-effect feedback loops and can analyze the system as whole.

4) Building a quantitative SD model. The SD mode is an explicit description including qualitative and quantitative information. The implementation of the initial model requires turning the causal diagram into a set of equations. The model variables must be chosen, the rate equations are precisely defined, and the initial values of the chosen variables are set.

5) Model calibration and simulation. After a model version has passed static verification, dynamic sensitivity analysis is performed in order to test whether all chosen factors are essential to reproduce a given behavior model. By calibrating the simulation model against the data collected from previous projects and literature, it can be used to predict the possible outcomes of different management policies, actions, or decisions through the observed behavior of system.

III. Brooks' Law

Brooks' Law was first publicized in Dr. Fred Brooks' book: *The Mythical Man-Month: Essays on Software Engineering.* Brooks' Law is stated as follows: "*Adding manpower to a late software project makes it later*" [7]. The lack of interchangeability between men and months was recognized by Brooks as being caused by training and intercommunication overheads.

A. Analysis and model for Brooks' Law

In this study, we will study the influences on the total productivity (measured by task/day), the total cost (measured by man-day) and the project duration (measured by day) of adding manpower to a project through the systems dynamics modeling and simulation.

With the manager bringing new personnel into the project, the system dynamic is triggered. Three effects are considered in here: 1) an increase of communication overhead, 2) an increase of training overhead, and 3) an increase of the total manpower available for project development.

When new staffs joining in the project, they require training which will cost experienced personnel's time. At the same time, more group members require higher communication overhead. These communication and training overhead lead to productivity decrease. Another impact is that more people are available to develop. As a result, the productivity will increase. The improvement of the productivity will strengthen the progress, reducing the backlog. Figure 1 is the causal loop diagram. Note that a minus (plus) sign on an arrow means that the two entities connected by the arrow move in the opposite (same) direction.

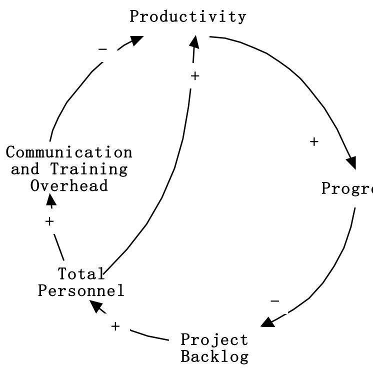

Figure 2. Causal loop diagram of Brooks' Law

Through causes loop analysis by figure 2, we found that adding new personnel will have the positively influence as well as the negative influence on productivity and project progress. What is the net impact on productivity of adding manpower to a late project? It's difficult to draw the conclusion directly only by the qualitative analysis. The detailed influences need further analysis by simulation through the quantitative model.

The system dynamics model of Brooks' Law is shown as figure 3 which is established by PowerSim tool. The equations and variables initial values were set as follows:

requirements =5000<<task>>
developed software = 0<<task>>
nominal productivity = 1<<task/(man*day)>>
training overhead % = 25
assimilation delay = 20<<day>>
average daily manpower per staff =1<<day/day>>
new project personnel = 0<<man>>
experienced personnel = 20<<man>>
software development rate = '*nominal productivity*'*(1-'*communication overhead %*'/100)*'*total nominal manpower*'
total nominal manpower = 0.8*'*new project personnel*'+1.2*('*experienced personnel*'-'*experienced personnel needed for training*')
communication overhead % = GRAPH(('*experienced personnel*' + '*new project personnel*'), 0<<man>>, 5<<man>>, {0,1.5,6,13.5,24,37.5,54})
experienced personnel needed for training = '*new project personnel*' * '*training overhead %*' / 100
personnel allocation rate= PULSE(5<<man>>, STARTTIME + 100 <<day>>, 999<<day>>)
assimilation rate = '*new project personnel*' / '*assimilation delay*'
man-day rate = ('*experienced personnel*' + '*new project personnel*')*'*average daily manpower per staff*'

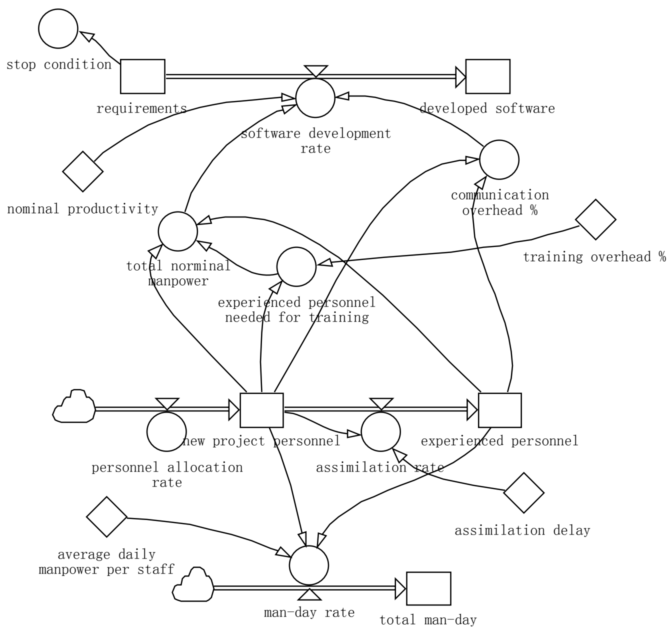

Figure 3. System Dynamics model of Brooks' Law

The '*requirements*' will be developed to the software product gradually. Therefore with the time passing, the '*requirements*' will decrease and '*developed software*' will increase. In the model, the develop productivity is dependent on many factors, including the '*nominal productivity'*, the '*communication overhead'* and the '*total nominal manpower'*. The '*communication overhead'* fits a nonlinear function of total number of personnel, here uses the data (0.6*n*n) in Abdel-Hamid's model [5]. In additional, supposes one experienced personnel can train 4 new personnel, so the value of *'training overhead'* is 25%. '*Assimilation delay'* expressed how many days that a new personnel can be trained to be an experienced one, in our model the value is set to 20 days. Standard productivity '*nominal productivity'* is 1, which expresses that developing one task of requirements will cost one normalized personnel to work one day. In the model, new personnel's productivity is 0.8 time of nominal productivity and the experienced personnel's productivity equals 1.2 times of nominal productivity.

B. Simulation result analysis

In the simulation run, the hypothesis initial condition is 20 experienced staffs to develop 5000 requirements' tasks of the project, the result of project duration and total man-days are shown by the reference curves in figure 4 and figure 5. The total duration is 278 days and costs 5500 man-days. As figure 4 shown, the productivity is a parallel line which value is 18.24 tasks/day.

Suppose that the Project Manager increases 10 new staffs at the 100th day to speed up the project progress. The simulation results are demonstrated in the *'current'* curve of figure 4 and figure 5. Then it will cost 299 days together with 7870 man-days. The productivity curve has a remarkable drop at the 100th day, then it will rise gradually, finally it will stay at 16.56 tasks/day stably. When analyzing the reasons, it is not difficult find that because of sharp rising loses due to communication and training overhead, the productivity dropped sharply. The final result is the total duration of the project has not been shortened, but the development cost actually rises sharply.

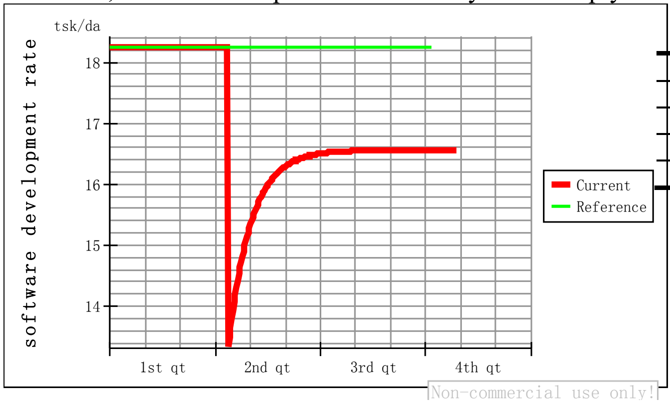

Figure 4. Software productivity diagram

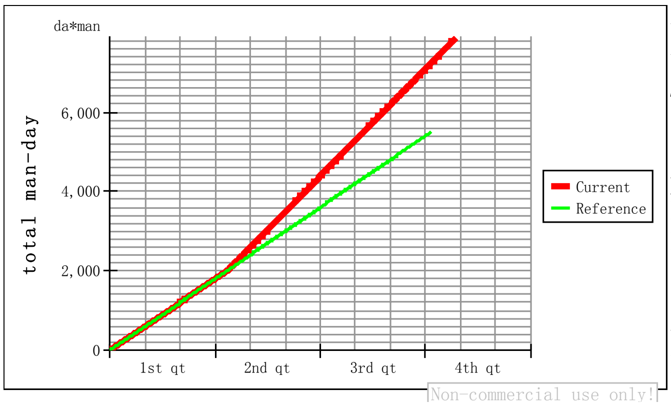

Figure 5. Total man-days diagram

Figure 4 and figure 5 demonstrate this model's simulation run in some scenes, and the Brooks' Law is accuracy in some extent. In fact, we may adjust new staff's quantity or change the entraining time of new personnel's joining the project to predict the project result under other conditions. For instance, if we increase 5 new staffs instead of 10 at the 100th day, the project duration can be slightly shortened, which needs 275 days, but total costs 6295 man-days which is still higher than original 5500 man-days. Table 1 lists different scenarios simulation results.

The above described a simplified Brooks' Law model, but it serves our original purposes. When it comes to practical software development, we need to refine it and consider more factors. For example, the product defect ratio of new staffs is usually higher than experienced staffs and this will increase the workload of QA and redoing it. Certainly, the productivity can be affected by schedule pressure, the process maturity of organization and other factors. Some researchers have done deep study about using SD to simulate Brooks' Law and get some interesting conclusions, the details can be found in [5].

Our experimental results provide insight into Brooks' Law. Adding more people to a late project always causes it to become more costly, but it does not always cause it to be completed later.

TABLE I. SIMULATION LISTS OF DIFFERENT SCENARIOS

Scenario No.	New Staff Joining Time	New Staff Number	Project Duration (Day)	Total Man-Day
1	-	-	278	5500
2	50	5	274	6520
3	50	10	305	8520
4	100	5	275	6295
5	100	10	299	7870

IV. EFFECTS OF PAIR PROGRAMMING IN XP

Lightweight development is continuously gaining its popularity in recent years. Now, it seems to be a common view that current methodologies of lightweight development are especially fit to a medium sized project and team [8]. Extreme programming (XP) is the most famous one among them. XP comes from a real project (a project at Daimler Chrysler called C3) led by Kent. It indicates that XP succeeded first in practice rather than theoretical analysis. It's so close to the real application that Kent described XP as a "*humanistic discipline of software development*" [9]. After some refinement and abstraction, Kent lays out a set of 12 core practices that serve as a starting point for an XP team [1]. The 12 practices are: The Planning Game, Short Releases, Metaphor, Simple Design, Testing, Refactoring, Pair Programming, Collective Ownership, Continuous Integration, 40-Hour Week, On-Site Customer and Coding Standards. The 12 practices are along with the 4 values of XP (named Communication, Simplicity, Feedback, and Courage) and the 5 basic principals (named Rapid feedback, Assume simplicity, Incremental change, Embracing change and Quality work) [8] have made XP more acceptable and applicable.

Now more and project managers are convinced that XP will be at least a substantial help to their project. The 12 practices are the most important guidelines to implement XP. Some items of the 12 practices are obvious positive to a successful project, for example: Metaphor and Coding Standards. Some items are "*double-edged swords*", for example: Collective Ownership and Continuous Integration, but if properly applied, they can be positive factors in all. These two groups of practices are the practices accepted by most of the people and applied if XP is adopted. They take up 11 out of the total 12 practices. The only one that is under suspicion greatly is Pair Programming. It's natural to think that letting two programmers to do the work that's formerly done by only one will waste time nearly by 100%. As a result, lots of people are reluctant to accept the practice of Pair Programming, though the rest 11 practices are more or less applied. More over, some objectors have done researches on the topic and published papers to illustrate that solo programming outperforms pair programming [10].

As the most controvertible item of the 12 practices, pair programming also has some advocators. They believe that pair programming is an essential part of XP and helps to improve the overall performance of the development. Researchers have done a lot of exploration on pair programming. Some of the researches are based on theoretical analysis [11][12], some are based on statistic results [13]. But all these kinds of research are not very suitable for evaluate pair programming. Because software development, even lightweight, is a complicated process or system, just from theoretical analysis on some aspects or statistic results on some crucial data is far to enough. Besides the above methods, controlled experiment is an effective and credible way [14][15]. But the cost confines the experiment and we cannot change some of the variables in the experiment and redo it to see how it will affect the result.

So we built a SD model to evaluate the effects of pair programming in XP.

A. SD Model for Pair Programming in XP

XP process is a complicated system, whose factors affect each other in different causal loops. Simulation is well suit for dealing with causal loops. Some researchers have been working on the field: S. Kuppuswami et al. [16] use simulation model to explore the XP issues, but not for pair programming. P. Wernick and T. Hall [17] use system dynamics model to investigate the impact of pair programming on evolution of software systems, but it's not about pair programming in the whole development process.

In order to investigate the impact of pair programming on XP development, we create an XP model. The thought of modeling comes from Abdel-Hamid's book [5]. Considering the trait of XP, we change some elements and make some simplification of Abdel-Hamid's description. In Abdel-Hamid's model, the workload is measured as "task". In our XP model, the workload is measured as 'story', because in XP process, the developing unit is story. In Abdel-Hamid's model, there are 8 sub-models. In our XP model, there is only one integrated model, because Abdel-Hamid describes a heavyweight software development process, but XP process is a lightweight one, some sub-models in Abdel-Hamid's model can be combined, others can be omitted to fit the characteristic of XP.

The high level of our model is shown in figure 6.

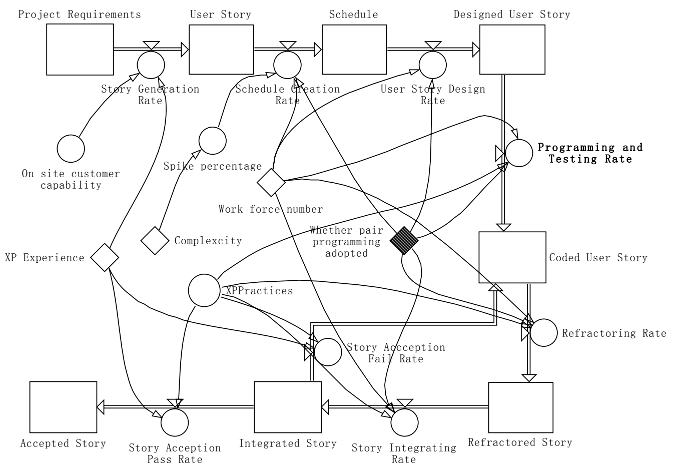

Figure 6. High level of SD model for XP

Here, for brevity, we explain only "Programming and Testing Rate" to give an idea of our model. 'Programming and Testing Rate' is one of the key rates in XP process, it represents the velocity of coding and testing. This rate is affected by several factors:

WF – Work force number;

XP Practices – some of them: Short Release Cycles, Pair Programming, 40 Hour Week, Code Standards, Simple Design.

Every XP Practice that is related to "Programming and Testing Rate" has an impact scale on the rate, and each has its own performance boost. The impact scale is how much this practice can be applied during the XP process, the more difficult it is to be applied, the lower impact scale it will be. The performance boost means how helpful the practice is to the XP process. For example, Short Release Cycles has an application difficulty 4.4%, and Pair Programming has an application difficulty of 15.6%, the relative application scale of Short Release Cycle is 1/4.4%=22.7, and that of Pair Programming is 1/15.6%=6.4. That means using the same effort, implementing Short Release Cycles is 3.5 times (22.7/6.4=3.5) easier than implementing Pair Programming. The calculation of performance boost is the same as application scale. Each practice has a rate of helpfulness in percentage so we can get its relative performance boost. All the data we need here comes from B. Rumpe's paper [18]. Getting the required data, we can make our "Programming and Testing Rate".

Programming and Testing Rate = ('Scale1''Boost1' + 'Scale2'*'Boost2' + 'Scale3'*'Boost3' + 'Scale4'*'Boost4' + 'Scale5'*'Boost5')*'NPATR'*(1-'CO')*'RNWL'*'RAL'*

Here:

Scale1 to *Scale5* means the application scale of the 5 practices related to Programming and Testing Rate mentioned above.

Boost1 to *Boost5* means the performance boost of the 5 practices.

NPATR means Nominal Programming and Testing Rate.

CO means Communication Overhead.

RNWL means Reduced Non-working Loss. Two people working as a pair will have less slack time than one working along.

RAL means Reduced Assimilation Loss. Pair Programming, especially with rotation helps to reduce assimilation time by one learning the other while pairing.

Our simulation model does not differentiate the complexities of different stories.

B. *Simulation and Results*

The simulation investigates how pair programming will affect the XP process. Concretely, it helps to determine whether pair programming outperform solo programming when other XP practices are applied.

To achieve this goal, we make controlled simulations. We set a switch variable in our model to charge the on and off of pair programming. Some parameters are set and changed to see the different result: project story number and work force number. First, we display the situation of a 100-story project. We set work force number from 2 to 20 to see the results. No matter how many programmers we assign to the project, pair programming will outperform solo programming, with a little advantage. Figure 7 illustrates that.

Then, we change the story number to 150 and 200 to see the results for larger projects. In order to make the results clearer, we use "Man day reduction" to represent the improvement of pair programming. Figure 3 shows the simulation results.

From figure 8, we can see that in all the situations, pair programming is better, and especially for the 150-story project. Different story number and work force number have different impact on the improvements of pair programming.

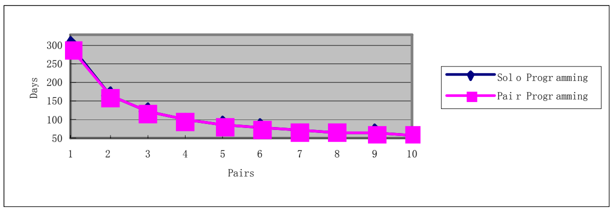

Figure 7. Simulation of 100-story project with work force number form 2 to 20

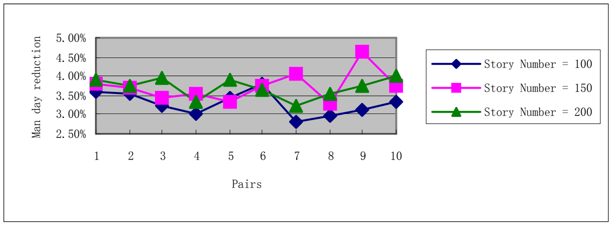

Figure 8. Man day reduction of 100, 150 and 200-story projects with force number from 2 to 20

Any way, in all the situations we simulated, pair programming always outperforms solo programming. But the improvement will not be very significant, at about 3.5% reduction of development time in average. This result is quite close to K.M. Lui and Keith C.C. Chan's result in their paper [19]. They investigated by two methods. Method 1 indicated a reduction of time by 4.2% and method 2 a reduction of 5.3%.

The reasons for the time reduction are manifold. Here are the reasons we summarized: reduced communication overhead, less assimilation time, higher morale, less slack time, better code quality, better design and so on. To illustrate how these reasons perform on the model, we use our formerly introduced "Programming and Testing Rate" to give out an explanation. In that rate, reduced communication overhead, less assimilation time and less slack time are considered. All of these are converted to factors:

Reduced communication overhead – 1.05 ;

Less assimilation time – 1.51 ;

Less slack time – 1.48.

All these data come from [5][11][18].

With the positive simulation results we get, we can say with certainty that pair programming will perform a little better than solo programming in XP development process. It's worthy point out that pair programming will help to escalate the overall developing ability of the team by letting programmers learning from each other by pairing and rotation, this advantage is not considered in our simulation. More details about the SD model for XP please refer our previous work in [20].

V. SYSTEM DYNAMICS USING IN CRITICAL AREAS OF SOFTWARE ENGINEERING

Simulation can be applied in many critical areas of software engineering. It enables one to address issues before they become problems. Simulation is more than just a technology, as it forces one to consider system behavior in global terms [21]. So it is easy to make student understood in software engineering education.

A. *Project management*

The Abdel-Hamid and Madnick's software project system dynamics model represents one of the first efforts in this area [5]. The model is divided into four major parts: human resource management, software production, controlling, and planning. The human resources subsystem focuses on the view of the personnel who participates in the software development. It includes hiring, training, dismissal, transferring personnel during different projects. Software production subsystem focuses on different development activities, for instance design, code, test, redo and quality assurance. This subsystem also processes the team motivation; the development personnel fatigue degree, as well as communication and so on. The software control subsystem describes in view of management measure. This subsystem control overtime, progress pressure etc. The software plan subsystem provides the software project the initialization parameter values, for instance project scale, initial team scale, anticipated closure time etc.

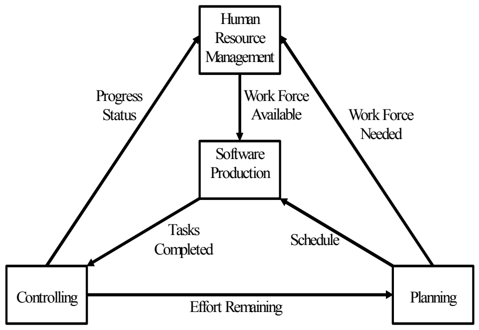

Figure 9. Abdel-Hamid's Subsystem Model of Software Project Management

The model was used as an experimentation vehicle to study an array of managerial policies and procedures. Three areas were studied: *software cost and schedule estimation*, *the economics of quality assurance* and *staffing.*

B. *Software cost estimation*

Traditional cost estimation models such as COCOMO / COCOMO Ⅱ [22] contain dimensionless parameters used to indicate the productivity and technology level of a software organization. Such kind of static models are designed to provide point estimates. However, the project resources, project scope, and schedule may go through many changes during the development life cycle. They fail to capture management decision-making dynamics and their impact on project behavior [23]. One of disadvantages of those models is that they attempt to map the correlation among the various project variables based upon their statistical patterns without looking into the internal structural correlation among the project variables, which inevitably will generate unrealistic results [24].

Madachy introduced the Dynamic COCOMO in [22], which is an extension based on the cost parameters varying over time versus traditional static assumptions. It is necessary to re-calculate the effort estimation equation of COCOMO II because of the external volatility and feedback from user-driven change requests. He applied the concept of Dynamic COCOMO to a spiral life-cycle model to estimate the cost and schedule for multiple increments [25].

K. Choi proposed a simulation method for dynamic project performance in terms of effort, schedule, and defect density changes in a dynamic project environment by combining COCOMO II with system dynamics [26]. The approach can be an alternative measure for organizations having not enough project data. They combine COCOMO II with system dynamics as follows: First, derive a small-time incremented development rate of each phase to bring dynamics to COCOMO II by using the effort and schedule distribution data. Second, integrate the effort, schedule, and defect density estimation models to analyze the trade-offs among them. Finally, incorporate additional project factors to represent the effects of the dynamically changing project environment.

C. *Concurrent software development*

Concurrent software engineering exploits the potential for simultaneous performance of development activities between projects, product deliveries, development phases, and individual tasks. The problems of incremental development arise from the complex interdependencies between development activities and their effect on process behavior. These dependencies exist from their reliance on common resource or shared work-products in time [27].

To optimize the performance of the process as a whole it is necessary to effectively balance the levels of concurrency and iteration. In paper [27], they developed a model to highlight the trade-off within concurrent and iterative lifecycles. They use four basic elements of structure common to system dynamics models of production process structure, namely resource, time, effort, and work. The proposed model of incremental development focuses on the predicted deviation of a process from the measured process capability. This evaluation is used as a basis for the formulation of robust plans, definition of acceptable limits on control, and identification and evaluation of improvements.

Focused on concurrent software development, C.T. Hsu have classified different types of Concurrent Software Engineering (CSE) practices and identified the specific benefits, potential risks, and the dynamic cause-effect implications of different types of them. Based on analysis, he developed a SD model named CSE-SD [28]. The model is an economic and effective management policy exploration tool for pre-assessing the benefits and potential risks of future projects.

D. *Software process improvement*

When software development organizations attempt to shorten their cost and cycle time without decreasing quality, they will adopt some process improvement technologies to incorporate into their newly reengineered development process. But the 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 [29]. Thus, software development organizations need to know the impact they can expect to see before committing to the process improvement technology.

In the conventional way, how to change the software process is mainly depended on manager experience, which combined with high risk and expensive. Modeling and simulation is possible to provide a certain extent foresight of a process before its true realization. This kind of insight can help the process designer to appraise the candidate plan. M. Ruiz et al. according to CMM's different ranks proposed a correspondence level dynamic integration framework named DIFSPI in [30, 31] to support a qualitative and quantitative assessment for software process improvement and decision making, so helps the management team to define, to appraise and realizes the different rank process improvement. There are some other important works in this area, for example, Madachy [32], Tvedt [33] and Burke [34].

E. *Risk management*

One purpose of system dynamics modeling and simulation is the management of software development risks. An approach to modeling risk factors and simulating their effects as a means of supporting certain software development risk management activities is proposed by Houston et al. [35]. In the study, qualitative and quantitative surveys were used to study the factors and their potential effects. Six common and significant software development risk factors out of 150 risk factors were studied. A base model was then produced for stochastically simulating the effects of the selected risk factors. The model has following sectors: Planned staffing, Actual staffing, Effort allocation, Project planning, Project control, Adjustment of job effort, Productivity, Work flow, and Quality management. The model is designed specifically for the risk management activities of assessment, mitigation, contingency planning, and intervention.

VI. Conclusions

In the practical software development, because of the complex dynamic and feedback in development process, there are many interacting factors throughout the lifecycle which led to some phenomenon violating the intuition. The Brooks' law is a classical instance of this phenomenon. The traditional education method is difficult to discover the underground reasons, while simulation can be used to impart information in a more meaningful and dynamic way compared to traditional methods. System dynamic modeling and simulation can describe the system characteristics and represent some dynamic behaviors. It is an inexpensive way to gain deep insights when the conditions of manipulating the real system are prohibitive. With system dynamics, students can easily build the model, change parameters, and repeat the simulation. So it's possible to analyze the result under different conditions, which enforce and enhance them to get understanding of typical phenomena occurring in software projects.

Acknowledgment

This work is partially supported by ZUCC Education Foundation (No. PP0601, No. JG0801) and Zhejiang Province Education Science Planning Project (No.2009-SCG196)

References

[1] M.I Kellner, R.J. Madachy, D.M. Raffo, "Software process simulation modeling: why? what? how?". Journal of System and Software, 1999; 46: 91-105

[2] D. Pfahl, M. Klemm, G. Ruhe, "A CBT module with integrated simulation component for software project management education and training". Journal of System and Software, 2001; 59: 283-298

[3] R.H. Martin, D.Raffo, "A Model of the software development process using both continuous and discrete models". Software Process Improvement and Practice, 2000; 5: 147-157

[4] J.W. Forrester, Industrial dynamics, Cambridge, MA: MIT Press, 1961

[5] T. Abdel-Hamid, S.E. Madnick, Software project dynamics: an integrated approach. Englewood Cliffs, NJ, Prentice Hall. 1991.

[6] D. Pfahl, K. Lebsanft, "Knowledge acquisition and process guidance for building system dynamics simulation models: an experience report from software industry" . International Journal of Software Engineering and Knowledge Engineering, 2000; 10(4): 487-510

[7] F.P. Brooks, The mythical man-month. MA: Addision-Wesley Publishing, 1978

[8] K. Beck, Extreme programming explained, Addison-Wesley, 1999

[9] K. Beck, "Extreme Programming: A uumanistic discipline of software development", in: Proceedings of International Conference Fundamental Approaches to Software Engineering (FASE 98), 1998

[10] M.M. Muller, "Are reviews an alternative to Pair Programming?", Empirical Software Engineering, 2004 , 9, 335-351

[11] W. C. Wake, Extreme programming explored, Addison-Wesley, 2001

[12] L. Williams, "Integrating Pair Programming into a software development process", in: Proceedings of International Conference Software Engineering Education and Training ,2001, 27-36

[13] A. Cockburn, L. Williams, The costs and benefits of pair programming, Extreme programming examined, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 2001

[14] G. Canfora, A. Cimitile, C. A. Visaggio, "Empirical study on the productivity of the pair programming", Extreme Programming and Agile Processes in Software Eng.—Proc 6th Int'l Conf. XP 2005, LNCS 3556, Springer, 2005, 92–99.

[15] S. L. Syed-Abdullah, J. Karn, M. Holcombe, T. Cowling, and M. Gheorge, "The positive affect of the XP methodology", Extreme Programming and Agile Processes in Software Eng.—Proc 6th Int'l Conf. XP 2005, LNCS 3556, Springer, 2005, 218-221

[16] S. Kuppuswami, K. Vivekanandan, P. Rodrigues, "A system dynamics simulation model to find the effects of XP on cost of change curve", in: Extreme Programming and Agile Processes in Software Engineering, XP 2003, LNCS, vol. 2675, Springer-Verlag, 2003, 54-62

[17] P. Wernick and T. Hall, "The impact of using pair programming on system evolution a simulation-based study", Proceedings of the 20th IEEE International Conference on Software Maintenance, ICSM'04, 2004: 422-426

[18] B. Rumpe and A. Schroeder, "Quantitative survey on extreme programming projects", in: Proceedings of the 3rd International Conference on eXtreme Programming and Agile Processes in Software Engineering, XP2002, 2002. 95-100

[19] K.M. Lui, K.C.C. Chan, "When Does a Pair Outperform Two Individuals?", in: Extreme Programming and Agile Processes in Software Engineering, XP 2003, LNCS, vol. 2675, Springer-Verlag, 2003, 225-233

[20] Minghui Wu. Xuejian Luan, HKN Leung, etl, "Using system dynamics model to simulate the effect of pair programming in XP". WEASE Transactions on Computer, 2007; 10: 1111-1115

[21] A.M. Christie. "Simulation - an enabling technology in software engineering". http://www.sei.cmu.edu/publications/articles/christie-apr1999/christie-apr1999.html.

[22] B.W. Boehm, C. Abts, A.W. Brown, et al., Software cost estimation with COCOMO II, Prentice-Hall, 2000

[23] C.Y. Lin, T. Abdel-Hamid, J.S. Sherif, "Software-engineering process simulation model (SEPS)", Journal of Systems and Software, 38(3), 1997, 263-277

[24] Z.Y. Ma, "Intelligent system dynamics modeling and decision analysis for software schedule slippage recover". Ph.D. dissertation, Arizona State University, 2000

[25] R. Madachy, B. Boehm, J. Lane, "Spiral lifecycle increment modeling for new hybrid processes". in: Proceedings of the International Software Process Workshop and International Workshop on Software Process Simulation and Modeling (SPW/ProSim 2006), China, LNCS, vol. 3966, Springer-Verlag, 2006, 167-177

[26] K. Choi, D.-H Bae. "Dynamic project performance estimation by combining static estimation models with system dynamics". Information and Software Technology 51 (2009): 162–172

[27] A. Powell, K. Mander, D. Brown. "Strategies for lifecycle concurrency and iteration – A system dynamics approach". Journal of System and Software, 1999; 46: 151-161

[28] C.T. Hsu. "A system dynamics model for concurrent software engineering". Phd dissertation, The University of Texas at Arlington, 1999

[29] J.D. Tvedt and J.S. Collofello. "Evaluating the effectiveness of process improvements on software development cycle time via system dynamics modeling". in: Proceedings of the Computer Software and Applications Conference (COMPSAC 95), 1995; 318-325

[30] M. Ruiz, I. Ramos, M. Toro. "A dynamic integrated framework for software process improvement". Software Quality Journal, 2002, 10, 181–194

[31] M. Ruiz, I. Ramos, M. Toro. "An integrated framework for simulation-based software process improvement". Software Process Improvement and Practice, 2004; 9: 81-93

[32] R.J. Madachy. "A software project dynamics model for process cost, schedule and risk assessment". Phd dissertation, University of Southern California, 1994

[33] J.D. Tvedt. "An extensible model for evaluating the impact of process improvements on software development cycle time", Phd dissertation, Arizona State University , 1996

[34] S. Burke. "Radical improvements require radical actions: simulating a high-maturity software organization". 1997, Software Engineering Institute Report CMU/SEI-96-TR-024.

[35] D.X. Houston, G.T. Mackulak, J.S. Collofello. "Stochastic simulation of risk factor potential effects for software development risk management". Journal of System and Software, 2001; 59: 247-257

Minghui Wu received the BS degree in Computer Science and Engineering from Nanchang University in July 1997 and MS degrees in Computer Science and Engineering from Zhejiang University in March 2000. Now he is the Ph.D. candidate in Computer Science of Zhejiang University. Since Dec 2006, he serves as an associate professor of Computer Science at Zhejiang University City College. His major interests include Software Engineering, System Dynamics Modeling, Model-Driven Development, Semantics Web, and Model Checking.

Hui Yan received the BS and MS degrees both in Computer Science and Engineering from Zhejiang University in July 1986 and March 1991, respectively. Since Dec 2005, she serves as a professor of Computer Science at Zhejiang University City College. Her major interests include Software Engineering, and Embedded System.