An Extended Simulation Model for Managing Dynamic Changes in Software Development Projects

Allesha Fogle and Yanzhen Qu

***Abstract*—This research study has utilized an experimental design method to evaluate the impact of integrating process work flows, personnel factors, and rework into a dynamic software process simulation model and its implication for software project managers to respond quickly and correctly to changes that are occurring throughout the software development process. The failure rate of software project has remained high due to the failure to effectively manage the dynamic risks that are present throughout the software process. Current risk management methods, project estimation tools and models in existence have not met the need for continuous re-evaluation of projects as unexpected events occur. Many of the unexpected project events can be quantified and represented in a project plan and thus a simulation of the software process can model the effects of those events on the project outcome. The results of the study showed that extending existing models as input to the software development process and parameterizing the change factors and the project profile within a simulation can provide estimation of the effects of dynamic project changes on the outcome of software development projects in terms of effort and completion time.**

***Index Terms*—Estimation, simulation model, software development process, software development project management.**

I. Introduction

Software projects are known for continuing to have a high occurrence of failure despite the many standard estimation tools, metrics, and risk management techniques that have been developed [1]-[3]. One important factor that leads to software failure is the difficulty in reacting appropriately to unexpected changes in the project resources and schedule. There is a need for a change management tool that can track software project changes and predict the probable impact of those dynamic changes on the schedule and budget. This research study demonstrates how we can apply static software estimation models in a software development simulation model for quantitative software monitoring and prediction of project outcomes based upon response scenarios. This paper will first introduce the existing studies that were used in developing the extended simulation model. Next, the research problem, the methodology and the application of the extended simulation model will be detailed. Finally, the main contributions and future research areas will be discussed.

Manuscript received July 23, 2015; revised October 15, 2015.

Allesha Fogle is with Harris Corporation in San Antonio, Texas 78250 USA (e-mail: alleshafogle@gmail.com).

Yanzhen Qu is with Colorado Technical University, Colorado Springs, CO 80907 USA (e-mail: yqu@coloradotech.edu).

II. Related Work

A risk factor, as it pertains to software engineering projects, is defined by [4] as “A potential problem that, should it become a real problem, will inhibit the ability to deliver acceptable software elements for a software intensive system on schedule and within the constraints of the monetary budget and/or the technology budget.” Many risks have been documented for software projects, and the effective management of those risks is at the center of preventing software failure. Ayad Ali [5] asserts that as a technology advances and more of our software is being developed in web and mobile environments, the complexity of these risks and the challenges in risk management will evolve as well. Management of project risk correlates to managing possible project failure or loss; even when an unexpected risk arises after the project is in implementation phase, effective methods for responding to the problem can determine the success or failure of the project [6].

Existing project estimation tools have been effective in providing some sort of planning guideline at the beginning of the project; however, the accuracy rates of those methods are still quite low by the end of the project cycle. When using modeling techniques that are derived based upon data provided by a few hundred software projects, such as Constructive Cost Model (COCOMO), the problem is not with the estimation technique itself, but the fact that the tool is not designed for real-time adjustments throughout the project. Whenever a risk factor turns into a real problem during the process of developing the software, a combination of risk management and software estimation must take place to counteract the challenges. A good risk management process will ensure the risks with the greatest impact and probability of occurring are identified as project status changes [7], [8]. One way of incorporating risk and estimation into a static method is through simulation of the software development process. Below we will present some of the most relevant and most recent existing theoretical research that focuses on using simulation in software estimation techniques.

A. *Combining Static Models with Dynamic Simulation*

Choi and Bae [9] used the system dynamics model to apply the COCOMO II static probabilities to a theoretical military software project. The authors used the iThink simulation software to create the process model and run the simulation to obtain the total effort results [9]. The study in [9] provides a good baseline for combining static and dynamic estimation methods; however, there are a few weaknesses in the study that provide opportunity for improvement. One limitation is the assumption that the software development methodology is the waterfall method whereas many organizations are shifting toward a more agile methodology to better accommodate requirements changes. Another weakness of the study is that its goal is limited to examining the effects of requirements changes, but does not address other types of dynamic events that commonly occur during the project. The simulation input is mainly based on a percentage of requirements creep in the project; however, it does not address the impact of project changes at different stages of the development process. The next study reviewed involves a discrete event simulation.

B. Discrete Event Simulation in Software Process

Kouskouras and Georgiou [10] focus on modeling the software development processes in detail and exploring how scheduling and defect rates affect the delivery schedule. Although this simulation model is very well designed, there are several limitations in the study from [10]. One limitation is the level of detail and strict procedures represented in the model will not be representative of many other software project processes. The procedures and resource entities used in this model will not be applicable to a broad number of software projects because of its rigidness. Also, this simulation model mainly would apply to a large-scale software project using the waterfall development process. Although the simulation runs were able to show how different project scenarios could affect total effort, its usefulness could be improved by examining the impact of other dynamic events and scenarios such as addition or loss of personnel, non-project related interruptions to the software team, requirements change, and complexity.

C. Software Process Simulation in Risk Management

Application of the system dynamics model to software risk management could be a good method for addressing the need for decision-making tools. Additional research needs to be conducted to examine how simulations can be used in real world risk management tools [11]. Mainly the visual feedback of simulations was found to have a great bearing on how useful the simulation was to the users in providing valuable information to assist in recognizing the project outcomes [12]. One example of such a simulation is ProjScout developed in [13]. The author's dynamic simulation model demonstrates the impact of Brooks's law in the software development process. The results of running the simulation with feedback loops in the model show how adding team members can impact the software development effort in a number of ways. The paper shows that addition of new team members can affect multiple factors such as training overhead, communication, and learning curve [13]. We can make better project decisions when we visualize the risks associated with both the software project itself and with management decisions through simulation [14].

III. Methodology

A. Problem Statement

Existing software estimation tools do not model the impact of dynamic changes that happen during the software development process and thus fail to provide project managers the information to fully assess the impact of these changes as they occur.

Current tools do not represent the realities of the Software Development Life Cycle (SDLC) and are not able to provide a view of the project outcome based on current project status and dynamic changes that occur. Root causes of project failures or delays are not known until after the project has been completed. There may be delayed feedback or no feedback at all once the project has ended to indicate whether the decisions made in response to those dynamic incidents caused performance improvement or degradation. There is little opportunity to improve upon existing decision-making processes without that feedback [14], [15].

B. Hypothesis

If we extend existing software estimation models to overcome weaknesses in analyzing the impact of dynamic changes throughout the development process, we can provide a tool to help software managers manage risks related to unanticipated events that could cause software projects to fail.

C. Research Problems

There were several objectives targeted in development of the extended simulation model. One goal for a new model was to encapsulate all of the small details that typically vary from project to project inside of a larger more general representation of the software process. Even though there are many methodologies in practice, the differences are usually in the amount of time spent in each of the main SDLC phases, the number of artifacts produced, and the delivery schedule [16]. All of these details can be represented numerically in terms of project time and resources required and can therefore be minimized to numerical or functional input parameters for the simulation. If we wrap the varying process details into higher-level processes, such as architecture design or requirements development, the model can be applied to both waterfall and agile types of projects. A weakness noted in the earlier studies, e.g.[9] and [10], was the absence of the consideration for other common dynamic factors in the software project, such as addition of personnel to the team. In order to include this factor in the study, we can use the ProjScout model of Brooks's Law that incorporates the effects of personnel numbers on communication overhead, training overhead and productivity [13]. This model attempts to represent the impact of Brooks's Law, which states in [17], "adding manpower to a late software project makes it later." The model presented by [13] is not detailed enough to account for sequential workflows that are existent in most projects and does not embody the dynamics of requirements creep, defect rates and personnel turnover. However, prior studies addressed these dynamics and can be combined with the ProjScout model to create a more complete model that addresses some of the weaknesses in all three of these research studies.

The simulation model presented in this study was designed to represent changes during the software project, which will in turn provide project managers with a useful tool for effectively responding to the specific changes modeled in the simulation. The model represents the following project attributes in software projects that differ in size, team dynamics, and methodologies:

1) Rework due to changes during any step of the SDLC
2) Addition or loss of personnel
3) Delays due to reduced productivity and dependencies

The next section reviews the steps designed to address the issues listed above.

D. *Extended Simulation Model*

The chosen methodology for development of this extended simulation model was to combine and extend where necessary the major features of the models that were presented as the basis for creating a model capable of representing dynamic change. The features that were used from previous studies include the components used for modeling Personnel Factors and Software Defects shown in the diagram of the high level overview of the model components in Fig. 1. The new model extended the existing models containing those components by adding an SDLC component, rework feedback loops in every SDLC phase, and dynamic model inputs that serve to address the weaknesses that exist in those models.

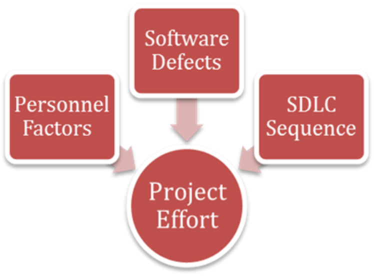

Fig. 1. Factors impacting project outcome.

The starting point was a basic model of the SDLC with the software size in function points as the primary input and the total project effort as the primary simulation output. The major components of the SDLC model were software requirements, designed software, developed software, tested software, and completed software. The major subcomponents were connected in sequence to match the reality of the software development process.

Next, the Personnel components from the model in [13] were used to incorporate the impact of personnel on the project process shown in Fig. 2. This component directly impacts the rate of flow between each of the subcomponents of the SDLC process, defined as productivity rate and, therefore, the total project effort. The Software Defect component from [10] shows the impact of software defects on the flow through the SDLC process. A feedback loop helps to model the way discovered defects would cause the defective component to re-enter the development and testing phases of the SDLC. This rework required due to software defects will ultimately impact the project effort.

Adding feedback loops to the model to represent the possible defects or missed elements in other process areas as well extended the model. These feedbacks depicted in Fig. 3 demonstrate how it is possible that additional work may be required in the process causing a flow of work back through any previous SDLC sub-component.

The components discussed above work together in order to meet the needs of the research. The SDLC component will serve as a base model to show the general sequential flow of work units through the development process. Some of the initial inputs for the model include parameters indicating the number and types of personnel resources, the defect rate, and the size of the project in terms of a number of work units expected for the project.

The personnel component from the ProjScout model is able to model the impact of those personnel changes on productivity [13]. That productivity rate controls how long it takes for each work task to move from one phase in the SDLC component to the next. The defect rate component determines the percentage of implemented software that will have to be reworked after passing through the testing phase. Whenever rework is required, the work units will flow back to a previous phase within the SDLC component instead of moving the project forward toward completion [10]. The feedback loops added to the model simulate how changes causing rework will affect the directional flow of the work through the process. The output of the model consists of both a total effort and a completion time as a result of the initial and dynamic model inputs.

E. *Model Extensions*

1) *Personnel component*

The personnel component was extended from the ProjScout representation of Brooks's Law to represent the differences in productivity and learning time for developer skill levels. This component helps to capture the real differences in productivity between resources of different experiences and skillsets.

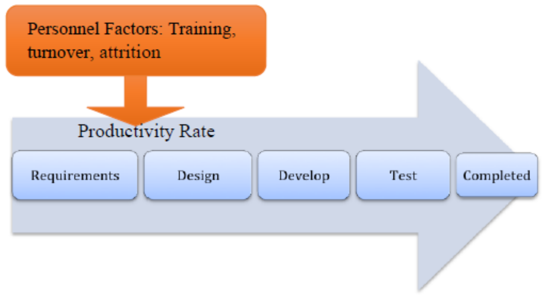

Fig. 2. Personnel factors impacting SDLC process.

2) *SDLC component*

The process component of the model structure encompasses the phases involved in the SDLC in order to produce a completed software project. The importance of using the SDLC model was to capture the sequential nature of software development. Regardless of the number of requirements to be met in the software, the same basic process must occur to produce a finished product.

3) *Defect component*

The design and implementation defects have the effect of rerouting the functions back to a previous process step. The model was designed to route all defects to either the Requirements level, which indicates a design defect, or the design Software level, which would indicate an implementation defect.

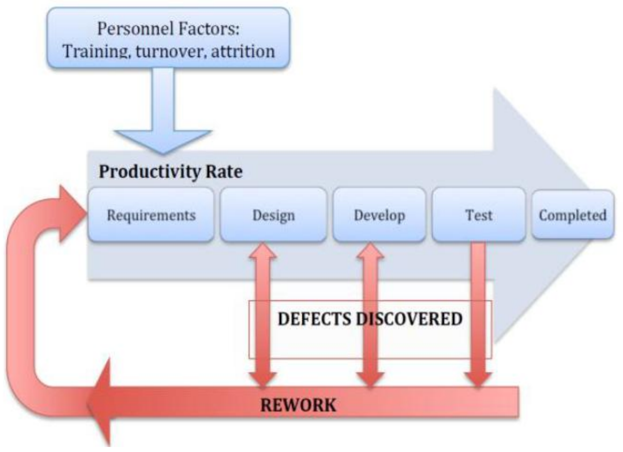

Fig. 3. Introduction of defects and rework in SDL.

IV. Implementation

A. Personnel Component

The Personnel extension added the ability to estimate process flow based on team experience and skill levels. There were several factors to consider when deriving the productivity rate for each of the skill levels represented:

1) Skill level

Each resource was categorized into one of three skill levels: junior, mid-level, and senior. For the purpose of the model, junior level developers are considered to have the least amount of experience and education in software development and senior developers are considered to have the highest amount of experience and education in software development. Mid-level developers were considered to have a combination of education and experience that is higher than a junior developer and less than a senior developer.

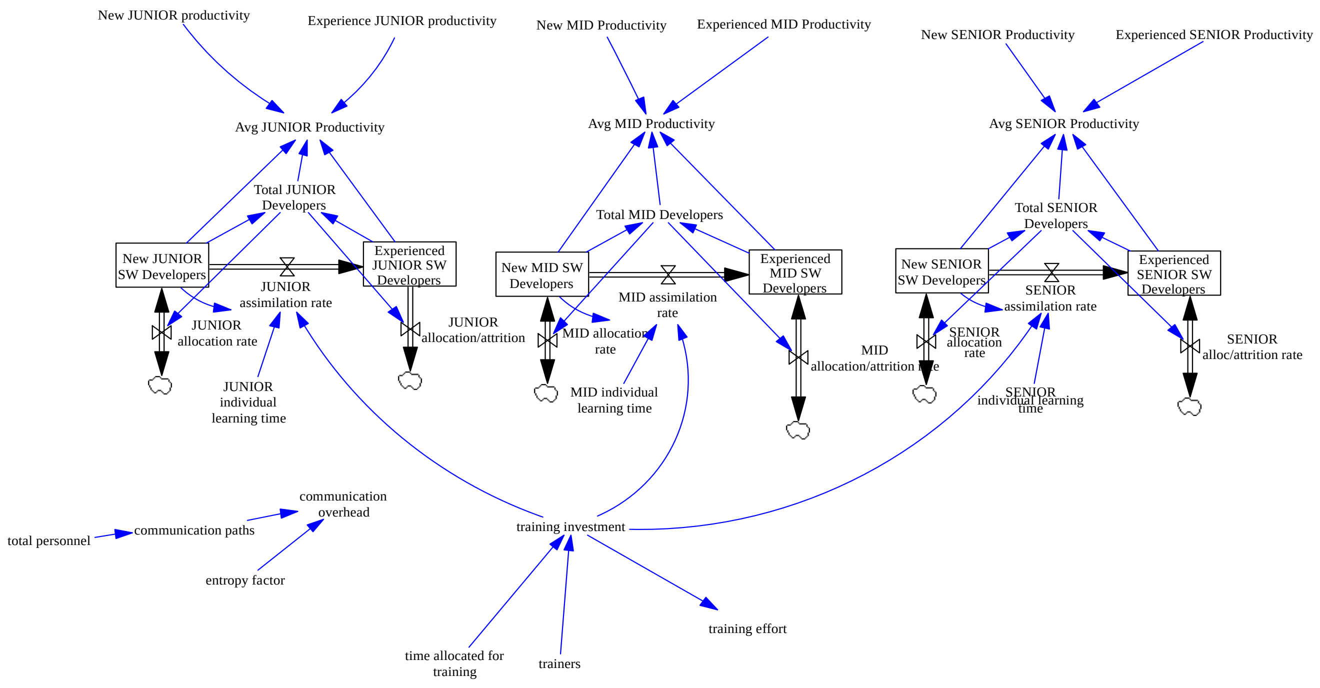

Fig. 4. Personnel component model structure.

2) Project/domain experience

A determination was made of whether the resource was new to the project with no prior domain knowledge or experienced in the domain and familiar with the project. Each development resource was classified as either "New" or "Experienced" to differentiate between those developers with project experience and those without experience.

3) Relative productivity

The relative productivity was the percentage of increase or decrease in what was considered nominal productivity based on both the skill level and the project and domain experience level. Nominal productivity was considered to be the number of functions developed, designed, or tested per hour for mid-level developers. Junior and senior productivity multipliers were used to adjust productivity to match the skill level and the project experience level of the developer.

The model structure for the personnel component is displayed in Fig. 4 and includes a standard communication overhead, which will reduce the overall productivity rates.

An average productivity equation was derived for each skill level based on the fraction of "New" versus "Experienced" resources available. The equations used to calculate average mid-level productivity are shown in (1), (2), and (3) Each of the average productivity rates was calculated using the same formula and were dynamic based upon the number of developers working on a set of tasks for each phase.

$$\text{NominalProductivity} = 0.0125 \tag{1}$$

$$\text{NewMIDProductivity} = 0.8 * \text{NominalProductivity} \tag{2}$$

$$\text{ExperiencedMIDProductivity} = 1.2 * \text{NominalProductivity} \tag{3}$$

The initial requirements for the project begin as a set of functions in the Requirements level and the design, implementation, and testing rates determine how quickly the functions flow from the Requirements level to the Completed

Software level. For the purpose of this research, the model structure uses levels as the backlog of functions waiting to be processed either in the design, implementation or testing phase and rates to define the amount of time for each function to flow from one phase to the next. The output productivities from the personnel component are used to drive the rates in the SDLC process component. In cases where there are multiple teams working on separate project components, an additional process is utilized for integration testing as well as the resolution of defects discovered during integration testing.

The primary consideration for calculating the rates of flow was the distribution of resources among each phase. Since the productivity is already known, the percentage of each skill and experience level available for work on each phase was needed for the rate calculations. For the design rate, only senior developers would perform work, however, for the implementation and testing rate, all available developers would be available to perform work. Since the goal of this research was not to determine optimal work distribution, a simple method of matching resource distribution to workload was used. For example, if 50% of the functions were in the Designed Software level waiting to be implemented, then 50% of the resources available for implementation were assigned to complete the work. For senior developers, the workload percentage was calculated among all three phases, but for mid-level and junior developers, the percentage was calculated among only the implementation and testing phases since they would not be performing any design work. Also, since senior developers must spend a fraction of their time in training if there are any new developers, the training effort must be subtracted from the senior developers' availability. Equations (4) and (5) demonstrate how the design and implementation rates were calculated within the modeling tool. In those equations, the function ZIDZ (*numerator*, *denominator*) returns *numerator*/*denominator* OR returns zero if the denominator is zero.

```
design rate =
IF THEN ELSE (Design Completed, 0, (1-communication
overhead)
*ZIDZ (Requirements, Total Tasks Remaining)
*(Total SENIOR Developers−training effort) *Avg SENIOR
Productivity)
```
(4)

```
development rate=
IF THEN ELSE (Implementation Completed, 0,
(1-communication overhead)*(ZIDZ (Designed Software, Total
Tasks Remaining) *(Total SENIOR Developers−training
effort)*Avg SENIOR Productivity +ZIDZ (Designed Software,
Designed Software+Implemented Software)*(Total MID
Developers *Avg MID Productivity+Total JUNIOR
Developers*Avg JUNIOR Productivity)))
```
(5)

Note: The function ZIDZ (numerator, denominator) returns numerator/denominator OR returns zero if the denominator is zero.

Representing projects with multiple teams working in parallel involved the addition of integration testing process components. An input parameter to indicate the number of functions that need to be integration tested as well as an additional process flow specifically for integration testing was designed. The integration flow process is similar to the SDLC process presented previously, however, in this study the resources available for integration testing and debugging only include senior developers from each team. Unlike the normal SDLC process, the integration testing process does not begin until all team project components have been fully designed, developed and tested independently of each other.

The integration test rate flow controls when the integration testing begins and during integration testing, the defects flow back through the normal SDLC process for resolution.

B. *Defect Component*

The probability of defects occurring is controlled by the Defect Fraction parameter. The model was designed to route all defects to either the Requirements level, which indicates a model is either a design defect or an implementation defect, an increase in the fraction of design defects corresponds to a decrease in the fraction of implementation defects.

C. *Project Changes*

Once the baseline estimate was established, the changes in resources and productivity were simulated in the model using parameters for the change to be applied and the actual time at which to apply the change. For this study, resource and productivity changes were applied at three distinct times in the simulation in order to show how the effort and completion time would vary based upon whether the change was made early, in the middle, or late in the project. For early, middle, and late changes, the change was applied when 25%, 50%, and 75% of the original estimated time for the project elapsed respectively. For project scenarios that involve multiple teams, the changes were applied to Team One workflows only.

The changes in defect rate were applied by modifying the overall fraction of functions that were found to be defective. Additionally, the defects were categorized as either implementation defects or design defects. Changes to the ratio of design defects to implementation defects were simulated as well.

D. *Model Outputs*

The specific project outputs that were captured during the simulation were project effort and total project completion time as:

Project Completion Time: Total number of hours for the design, implementation and successful testing of software project.

Project Effort: The total person-hours used for design, implementation and testing to deliver a fully implemented software project.

The status of the tasks remaining within the SDLC and Integration components enable calculation of the total project completion time. A model variable was set up to capture the number tasks in the Requirements, Designed Software, and Implemented Software level for the SDLC component for each team as well as those same levels in addition to the Integration Requirements level within the Integration component. This data was used to determine when the total tasks remaining to be executed are equal to zero at which time the project can be deemed completed.

The model was designed to accumulate project effort as long as the design rate, development rate, testing rate, or integration testing rate was non-zero since a non-zero rate indicated that the work was being completed by resources for the project. Effort for all personnel resources was considered to be equivalent regardless of skill level. The effort equation for project effort at any given time was calculated as the summation of the rates of flow between each of the phases. The total effort was determined by observing the maximum effort, which occurred at the project completion time.

V. Applications

To demonstrate how this model can be used to help manage a software project, we present one of the scenarios used in testing the model. We will use P1 to represent a project scenario of creating a single deliverable software component consisting of three major functional requirements by a single small team. The resources initially available for the project and their skill levels are documented in Table I. The total effort and completion time for this project are used as the baseline estimates that would be produced prior to starting project development.

TABLE I: Project P1 Profile

Input Parameter	Baseline value
Initial Requirements (functions)	3
Experienced Mid-level (person)	2
Experienced Senior (person)	1
Nominal Productivity (function/(person*Hour)	0.0125
Design Defect Fraction	0.15
Overall Defect Fraction	0.15
Total Effort (PERSON-HOUR)	**673**
Total Time to Complete (HOURS)	**225**

If we assume that the software project manager has just received a letter of resignation from one of his senior developers with about 50% of the project requirements completed. This manager needs to know what the impact will be on the project timeline. Based on the model presented here, such an estimated impact can be made through simulating the scenario within the model.

We can simulate a resource loss change at 50% requirements completion and observe the estimated impact on the total project completion time. Based on whether that projected impact is acceptable to the stakeholders, the project manager can make an informed decision to make no resource additions to the team or to add additional resources to maintain the original project timeline. If additional resources should be added, the project manager can use the model to simulate the addition of resources so that the impact on the project completion time can be estimated. The simulation results can help determine whether resource additions are likely to decrease the project completion time.

The completion time results showed that when positive resource and productivity changes are applied earlier in the project, the completion time is shorter than when those same changes are made late in the project. For this small project scenario, the addition of resources reduced the completion time from a baseline of 225 hours to 206 hours. Conversely, the loss of resources led to a longer completion time of 279 hours.

VI. Conclusions

In this research study we have made two contributions to the field. First we have provided an extended simulation model used for change tolerant software project management related parameters estimation. Secondly we have also developed a simulation model implementation of the extended simulation model capable of estimating the impact of unforeseen changes and decisions for projects of different sizes, team dynamics, and methodologies.

The model from [13] was a basis for the personnel component by adding dynamic addition/loss of personnel as well as adding representation of different skill levels and average productivity. The component provided a productivity rate to the model as well as staff availability. Both productivity and availability have a direct impact on the project progress rate. The SDLC component combined with the personnel components controls the speed and direction of flow of work through the project completion. The SDLC component enabled application of the basic rules of sequential workflow to the model including any dependencies between SDLC phases. The combination of components allowed the application of work distribution and resource availability to the model for each team involved in the simulation. The model from [10] provided basis for simulating the effect of defects on the flow of work through the process. The feedback loops enabled representation of the effects of design and implementation defects on the project outcome within the SDLC and Integration testing process. The integration component accounted for the necessity for integration testing when some requirements rely on components from multiple teams to be completed. Team dependencies could be represented and demonstrated by using the primary SDLC output from each team and the number of integration testing requirements as inputs to the integration component.

The results presented in this paper show that in comparison to existing estimation techniques and simulation models, the extended simulation model presents a unique advantage of allowing the impact the aforementioned dynamic changes to be estimated in both waterfall and agile processes. The previous works were not designed to handle all of these types of changes within the context of a general SDLC model. The focus of this paper was extending the existing modeling techniques and demonstrating a technique that could capture and accept dynamic project change input for software estimation. The next section will outline a few of the improvements and extensions that can further enhance the extended simulation model presented in this paper.

VII. Future Works

Many project changes were not specifically addressed in the model. Some of these include the addition of brand new requirements, workload distribution for the project, and dedication of individual resources to multiple projects. Although these types of changes can be built into some element of the model parameters, there was no specific examination of how each of these elements impacts the project outcome. A future study could examine how changes in these areas will impact the project effort and completion time.

Another important area for development would be to test the validity of this model in terms of how closely it would be able to model real project scenarios. Estimations are inaccurate by nature, but running the same tests using real project data and comparing the estimate with the actual project outcome could help to discover ways to calibrate the model for future application.

The extension and calibration of the model to accommodate additional changes and more closely estimate the outcome of real projects clearly creates the potential for development of a usable software estimation tool to be used in practice. A programmatic implementation of the extended simulation model would be much more flexible in terms of the number of changes that can be accommodated and would eliminate the limitations of using simulation modeling tools alone. There are a number of project management tools available on the market that could provide the project data that would drive an estimation tool based on the change-tolerant extended simulation model. Creating a product that could employ the extended simulation model combined with existing management tools would be a significant contribution to the software engineering and software development project management field.

REFERENCES

[1] P. G. Armour, "Twenty percent," *Communications of the ACM*, vol. 50, pp. 21-23, June 2007.

[2] K. E. Emam and A. G. Koru, "A replicated survey of IT software project failures," *IEEE Software*, vol. 25, pp. 84-90, September/October 2008.

[3] R. L. Glass, "IT failure rates — 70 percent or 10-15 percent?" *IEEE Software*, vol. 22, pp.112, 110-111, May/June 2005.

[4] R. E. Fairley and I. C. Society, *Managing and Leading Software Projects*, Hoboken, NJ: Wiley, 2009.

[5] A. A. Keshlaf, "Risk management for web and distributed software development projects," in *Proc. the Fifth International Conference on Internet Monitoring and Protection*, 2010, pp. 22-28, Barcelona, Spain.

[6] H. Stefan, M. Aust, M. Schermann, and H. Krcmar, "Comparing risks in individual software development and standard software implementation projects: A delphi study," in *Proc. the Hawaii International Conference on System Sciences*, 2012, pp. 4884-4893, Hawaii.

[7] D. W. Hubbard, *How to Measure Anything: Finding the Value of Intangibles in Business*, Hoboken, NJ: John Wiley & Sons, 2010.

[8] R. E. Fairley, *Risk Management for Software Projects*, 2008.

[9] K. Choi and D. H. Bae, "Dynamic project performance estimation by combining static estimation models with system dynamics," *Information and Software Technology*, vol. 51, pp. 162-172, 2009.

[10] K. G. Kouskouras and A. C. Georgiou, "A discrete event simulation model in the case of managing a software project," *European Journal of Operational Research*, vol. 181, pp. 374-389, 2007.

[11] D. Liu, Q. Wang, and J. Xiao, "The role of software process simulation modeling in software risk management: A systematic review," in *Proc. the International Symposium on Empirical Software Engineering and Measurement*, Lake Buena Vista, Florida, 2009.

[12] R. Agarwal, "Software development process animation," *Association of Computing and Machinery Southeast Regional Conference*, Kennesaw, Georgia, 2011.

[13] D. G. Chernoguz, "The system dynamics of Brooks' Law in team production," *Simulation*, vol. 87, pp. 947-975, January 2011.

[14] C. Mizell and L. Malone, "A project management approach to using simulation for cost estimation on large, complex software development projects," *Engineering Management Journal*, vol. 19, pp. 28-34, December 2007.

[15] G. Lee and W. Xia, "Toward agile: An integrated analysis of quantaitative and qualitative field data on software development agility," *MIS Quarterly*, vol. 34, pp. 87-114, March 2010.

[16] A. Cockburn, *Agile Software Development: The Cooperative Game*, Boston, MA: Addison-Wesley, 2007.

[17] F. P. Brooks, *The Mythical Man-Month: Essays on Software Engineering*, Crawfordsville, Indiana: Addison-Wesley Pub, Co., 1995.

Allesha Fogle currently is a software engineer working as a consultant for Harris Corporation in San Antonio, Texas, USA. She has over eleven years of experience in software engineering and has involvement in planning, architecting, designing and implementing software solutions in a variety of application domains including asset tracking, financial marketing and the medical field. Her primary research interest is in software development process improvement using process modeling and simulation tools. She received her Bachelor of Science degree in Mathematics from Fort Valley State University, USA in 2001; Bachelor of Science in Computer Engineering from Georgia Institute of Technology, USA in 2004; Master of Science in Software Engineering from Mercer University in Macon, USA in 2009, and Doctor of Computer Science from Colorado Technical University, USA in 2014. Dr. Fogle is a member of IEEE and ACM.

Yanzhen Qu currently is the university dean of College of Computer Science and Technology, and professor in computer science and information technology at Colorado Technical University, USA. He received his B.Eng. in electronic engineering from Anhui University, China, M. Eng. in electrical engineering from the Science Academy of China, and Ph.D. in computer science from Concordia University, Canada. Over his industrial career characterized by many the world first innovations, he has served at various senior or executive level Product R&D and IT management positions at several multinational corporations. He was also the chief system architect and the development director of several world first very large real-time commercial software systems. At Colorado Technical University, Dr. Qu is the dissertation supervisor of many computer science doctoral students, and his recent research interests include cloud computing, cyber security, data engineering, software engineering process and methods, data mining over non-structured data, affective computing, artificial intelligence, scalable enterprise information management system, big data analytics as well as embedded and mobile computing. He served as general/program/session chair or keynote speaker in various professional conferences or workshops. He is also a visiting professor of over twenty universities. He has published many research papers in the peer reviewed conferences and professional journals, and is currently serving as a member of editorial board of several professional journals. He is a senior member of IEEE and IACSIT.