Received October 17, 2018, accepted November 7, 2018, date of publication November 12, 2018,
date of current version December 18, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2880998

Change-Oriented Open Source Software Process Simulation

XUAN ZHANG[1,2], XU WANG[3], AND YANNI KANG[1]
[1]School of Software, Yunnan University, Kunming 650091, China
[2]Key Laboratory of Software Engineering of Yunnan, Kunming 650091, China
[3]School of Economics, Yunnan University, Kunming 650091, China

Corresponding author: Xuan Zhang (zhxuan@ynu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61862063, Grant 61502413, Grant 61262025, Grant 61379032, and Grant 61662085, in part by the National Social Science Foundation of China under Grant 18BJL104, in part by the Natural Science Foundation of Yunnan Province under Grant 2016FB106, in part by the Natural Science Foundation of the Yunnan Educational Committee under Grant 2015Z020, in part by the Natural Science Foundation of the Key Laboratory of Software Engineering of Yunnan Province under Grant 2015SE202, in part by the Data Driven Software Engineering Innovative Research Team Funding of Yunnan Province under Grant 2017HC012, in part by the Social Science Foundation of Yunnan Province under Grant YB2016013, and in part by the Software Engineering Innovative Research Team Funding of Yunnan University.

ABSTRACT The goal of our research is to find better ways for assessing the impact of changes to the software projects and the cost-effectiveness of process improvement strategies. To support decision-makers in analyzing changes effects and finding the optimal improvements of software process in a given project, a process simulation model using the system dynamics modeling technique is proposed and used in the context of a case study with open source software. Details of the system dynamics model, its usage scenarios and simulation experiments are provided. With the help of the simulation model, the process quality attributes of the open source software Spring Framework with varying change effects was evaluated. The project effort, delivery time, productivity, schedule, and product quality are impacted as a result of changes. Three different process improvement strategies were evaluated to help decision-makers choose most cost-effective improvement strategies. The simulation models can be used as an effective tool to evaluate the impact of changes, reason about the process improvement, and support consensus building by visualizing dynamic views of the process.

INDEX TERMS Open source software (OSS), change management, software process simulation, system dynamics (SD), software process improvement (SPI).

I. INTRODUCTION

Software development and evolution is a dynamic process and is characterized by change. Software projects often begin with unclear, ambiguous, and incomplete requirements which give rise to intrinsic volatility [1]. Meanwhile, the project team members, software users, environment and technologies are also likely to change throughout the life of the project as different versions of their software are released. When these important factors change and affect the software products, projects, and processes, these changes must be carefully planned [2]. Especially, when additions, deletions and modifications are made to previous generated or in process project artifacts, additional time investment, scrapped effort, and even bugs can result. Therefore, it is important to understand the changing dynamics overtime, the complex interaction effects, and to find the way to control the negative effects of the changes. System dynamics (SD) modeling is one of the best techniques to enable project personnel to software process and model change effects and run the models to better understand the implications of candidate project strategies and decisions [3].

Open source software (OSS) has some distinctive features and characteristics that deserve to be studied and understood [3], [4], so that we can exploit them to find the ways to increase the quality of software and of software process in software organizations. Our research is motivated by the tremendous growth in the open source movement and the success of some OSS projects in their quick response of changes and the capability in high-quality software development. Based on the preceding discussion, change requests are risks of high time pressure, cost, and poor product quality. We are interested in achieving a better understanding of the trade-off effects of change management on the project cost, duration, and software quality. Additionally, much of the OSS process data is recorded and public, such as issue trackers, bug databases, and code repository histories. Therefore, our research subject is the change management in OSS processes and our first goal is to use large-scale, quantitative data of OSS and capture the dynamic nature of the OSS process to understand of how the changes affect the software projects and how to reduce the negative effects on software quality, project duration and cost. Moreover, we would like to improve the project performance as a result of an adequate process improvement. Based on the widely accepted assumption that there is a relationship between software process quality and the quality of the resulting software product [5], past experience with insufficient process quality has motivated the improvement of software processes. Software engineering community has accumulated enormous experiences on software process. But we are here still looking for bullets to improve productivity and quality. One of the important reasons might be the many elements in software processes, such as human, artifact, cost, and schedule *et al.*. Therefore, in this paper, our second goal is to use simulation to gather all relevant factors and parameters in a holistic view for evaluating candidate process improvement strategies. In summary, the simulation from this research will not only serve for process analysis but also for identification of potential process improvement strategies, eventually resulting in modeling the specific software process models and evaluating the candidate process improvements.

By adopting Ferreira *et al.*'s research method [6] and following Münch's simulation models development lifecycle [2], our simulation development procedure was organized in three steps and the rest of the paper is organized by following these steps.

Step1. A rigorous review of the related literatures was performed first. But, in order to show the differences of our work, analyses of their work and the comparison between our work and their work are presented in Section IV.

Step2. Our work is elaborated in section II. GQM (Goal Question Metric) paradigm [7] was used for define simulation goals, questions and the needed measures. A common OSS change management process workflow was analyzed to create a causal model. Based on the cause-effect relationships in the causal model, a SD model was developed by designing subsystems for simulation. Simulation model for each subsystem was also developed and tested.

Step3. In Section III, by inputting empirical data from an OSS – Spring Framework, the simulation model is performed to examine the effects of changes, determine the simulation outcomes, and answer the question about "which process improvement is most cost-effective in reducing negative effects of changes?". The contributions and limitations of our model are also discussed.

Section V summarizes the main results presented in this paper and draws conclusions for future work.

II. CHANGE-ORIENTED SOFTWARE PROCESS SIMULATION

Software process simulation modeling captures the dynamic behavior and uncertainty in the software process [8]. According to Kellner [9], the reasons for using simulations of software processes are clustered into six categories of purpose. They are strategic management, planning, control and operational management, process improvement and technology adoption, understanding, and training and learning. When developing software process simulation models, identifying the purpose and the questions would like to address is central to defining the model scope and data that need to be collected. Therefore, GQM was first used for defining our simulation goals, questions and the needed measures. The purpose and scope of our model are described by the goals (indicated in G*.) below:

G1. Develop a change-oriented open source software process simulation model for analyzing, understanding, planning, and controlling the change effects on project manpower, productivity, schedule, cost and quality.

G2. For continuous software process improvement (SPI), analyze the effects of SPI on the software quality, project duration, and cost. Then, compare, select and customize the best SPI to support the decision of select the most suitable SPI strategies.

According to the purposes of our simulation, the following questions (indicated in Q*.) are formulated that the model should help to answer.

Q1. What are the effects of changes? How an OSS project team manages the changes?

Q2. Is SPI useful in reducing the negative effects of the changes? What does the effectiveness of SPI affect the software quality, project duration, and cost? When possible SPI action may be needed?

Afterward, the parameters of the models are defined as the metrics of GQM. They are the information elements needed to answer the questions that were specified along with the purpose of the model. Depending on the preceding questions, the following basic parameters (indicated in P*.) could be devised for simulation:

P1. Project changes;

P2. Manpower of project and developers' productivity (computed from the changes per week);

P3. Project cost and productivity (computed from the effort, i.e. triage effort, changes resolving effort, and bug fixing effort);

P4. Quality of delivered software (computed from the bug rate);

P5. Duration of project;

Change management behavior and decision made at one point in the process impact others in complex or indirect ways and must be accounted for. For example, in a software process the decision to add new developers or not, inspect change requests or not have multiple impacts. For documenting these complex feedback relationships between process parameters, casual model is developed next.

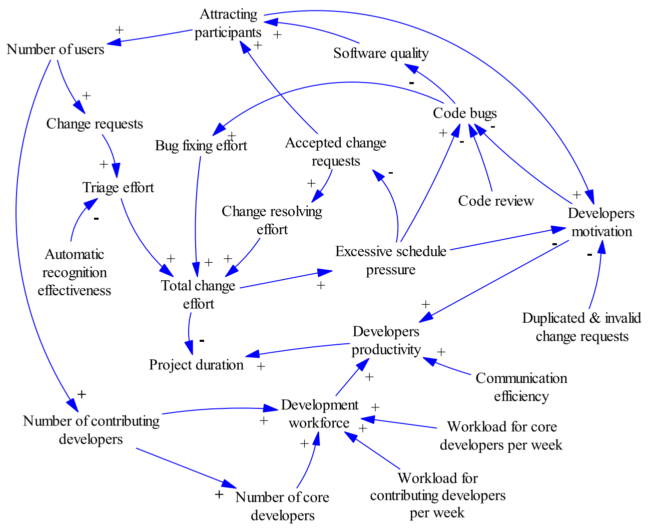


FIGURE 1. **Cause model (cause-effect diagram) for change-oriented process.**

A. CAUSE-EFFECT RELATIONSHIPS ANALYSIS

Based on the analysis of the literature review and discussions with software process experts and experienced software developers from university and company, relevant factors and associated relationships were identified and a causal model was developed, as shown in Figure 1.

This model is used to evaluate which software factors are affected by changes because it illustrates the cause and effect relationships between software factors related to changes. Four structural views were consisted in the model: Process view, effort view, productivity view, and quality view. The relationships between cause and effect factors are represented by arrows and + or −, depending on whether variation of the factors occurs in the same way or in the opposite way. To learn more about the complex interactions of the cause-effect relationships that force a certain simulation output to be generated, a SD model is created in the following. This model provides a tool for software project managers and researchers to perform "what if" analyses, and enables users to examine the effects and impact of various changes and determine project outcomes.

B. SYSTEM DYNAMICS MODELING

In the context of OSS change management, to analyze the performance of their change management process, we developed and applied a simulation model representing the main elements of the process reference model. The simulation model was developed with the help of the SD tool Vensim PLE 6.3[1] (Ventana simulation environment personal learning edition). During the modeling activity, process experts and experienced software developers from university and company were organized to elicit the required knowledge. The modeling procedure was proceeded iteratively and four

[1]Vensim 6.3 was release in 2014 by Ventana Systems, Inc.

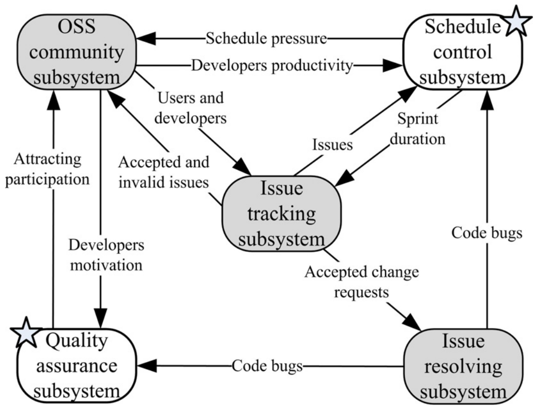


FIGURE 2. **Subsystem design for simulation.**

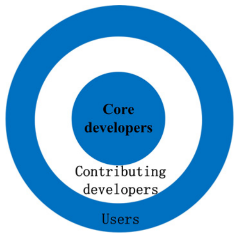


FIGURE 3. **OSS onion model.**

versions of models were developed. The model was first presented to two software project teams in Taili on March 2018. Three meetings were held to discuss the model and many valuable comments and recommendations were collected. During the modification of the model, these discussions were continued until the final model was made. To reduce the complexity, the model is segmented into 5 subsystems, which are organized into logical groupings of related factors. Figure 2 depicts the subsystem design of the model.

In Figure 2, OSS community subsystem, Issue tracking subsystem, and Issue resolving subsystem are three parts of change management subsystems. OSS community subsystem simulates the evolutions of human resources in OSS projects. Issue tracking subsystem simulates managing and addressing issue reports. In OSS projects, by filing issues reports, users helps submit change requests, identify and fix bugs, document software code, and enhance the software via feature requests [10]. Therefore, in the following, we use the issue to indicate the change request. Issue resolving subsystem links issue reports to changes in source code. We use grey shading rounded rectangles to highlight these three subsystems. Due to the additional unplanned changes, severe consequences can potentially occur, including significant schedule and cost overruns, and quality declines. Therefore, the other two subsystems, Quality assurance subsystem and Schedule control subsystem, simulate three key project management indicators (quality, schedule, and cost).

Integrating change management functions and software lifecycle activities, each of the subsystem is developed with a SD stock-level diagram, which elaborated in the following

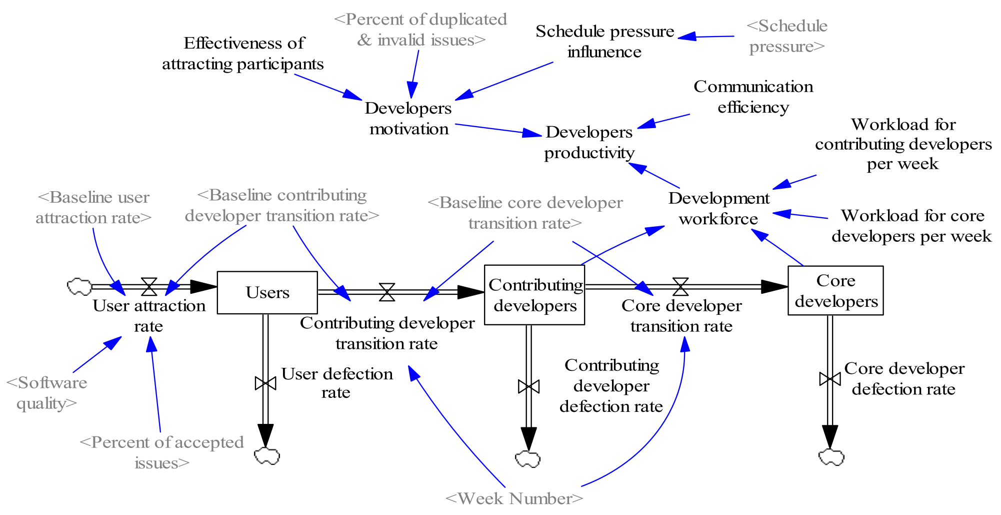

<Percent of duplicated & invalid issues>
Effectiveness of attracting participants
Schedule pressure influence
<Schedule pressure>
Communication efficiency
Developers motivation
Developers productivity
Workload for contributing developers per week
<Baseline user attraction rate>
<Baseline contributing developer transition rate>
<Baseline core developer transition rate>
Development workforce
Workload for core developers per week
Users
Contributing developers
Core developers
User attraction rate
Contributing developer transition rate
Core developer transition rate
<Software quality>
User defection rate
Contributing developer defection rate
Core developer defection rate
<Percent of accepted issues>
<Week Number>

FIGURE 4. **OSS community subsystem (stock-level diagram).**

Section 1) to 5). In these subsystems, two types of parameters were defined: project-specific input and variable parameters. Project-specific inputs are used to represent a specific project. Variable parameters can be changed to analyze the results of the output variables.

1) OSS COMMUNITY SUBSYSTEM

Group dynamics play a big role in OSS development. An investigation of OSS project evolution found that the OSS and the community must coevolve to achieve high quality [11]. In practice, OSS community is commonly organized in onion model, as depicted in Figure 3. "Onion" refers to the successive layers of member types [12]. Individuals increase their involvement through a process of role meritocracy.

The user base performs the bulk of the system testing, sometimes even exclusively, as with Apache [13]. The extensive test generally makes OSS having lower defect. Raymond coined "given enough eyeballs, all bugs are shallow," [3], [14] meaning that if enough people see a software error, at least one of them will probably be able to fix it. Therefore, OSS projects have access to far more human resources and demonstrate its power to develop faster and increase quality [15]. As users move toward the core, they might over time become developers. Developers are often classified according to the core and peripheral contributing roles. A few peripheral contributing developers will eventually join the small team of core developers. Each type of member has certain responsibilities in the system's development and evolution, which relate to the systems overall productivity and quality. Figure 4 depicts the stock-level diagram for OSS community. In this OSS community subsystem, all baseline data in '<>' are project-specific inputs, all the other data in '<>' are derived from the other stock-level diagram. The parameters that without '<>', such as *Developers motivation*, *Developers productivity* are variable parameters.

The OSS community management impacts the developers' productivity directly. Developers' productivity here is the developers' work rate due to the workforce, communication, and motivation of developers. *Developers productivity* and *Developers motivation* in Figure 4 are calculated in the following formula (1) and (2) respectively.

$$p_d = (n_d \times w_d + n_c \times w_c) \times (e + m)/2 \qquad (1)$$

In formula (1), p_d is *Developers productivity*; n_d is the number of *Contributing developers*; w_d is *Workload for contributing developers per week*, which represents the weekly changes that *Contributing developers* can handle. Its unit is changes/(person*week). Similarly, n_c is the number of*Core developers*; w_c is *Workload for core developers per week*. Thus, $n_p \times w_p + n_c \times w_c$ is *Development workforce*. e is *Communication efficiency*, which indicates the impact of communication on the *Developers productivity*. Its value range is from 0 to 1. When it is equal to 1, it is ideal state and means that the communication effectiveness is 100%. If it is 0, the communication effectiveness is 0%. There is no communication between developers. The last variable m is *Developers motivation* which is calculated in the following formula (2).

$$m = (1 - s_f) \times (1 - v) \times (1 + k) \qquad (2)$$

In formula (2), s_f is *Schedule pressure influence*, it is determined by *Schedule pressure*, which will be introduced in the following Section 5). Any change will increase the schedule pressure on the developers, which has been known to cause poor quality designs and code. v is *Percent of duplicated & invalid issue*, which is obtained in the following Issue tracking subsystem (see Section 2)). It is the percentage of duplicated and invalid issues in total issues. k is *Effectiveness of attracting participants*. If it is less than 0, it means that attracting participants is invalid, which declines developers' motivation. On the contrary, if it is greater than

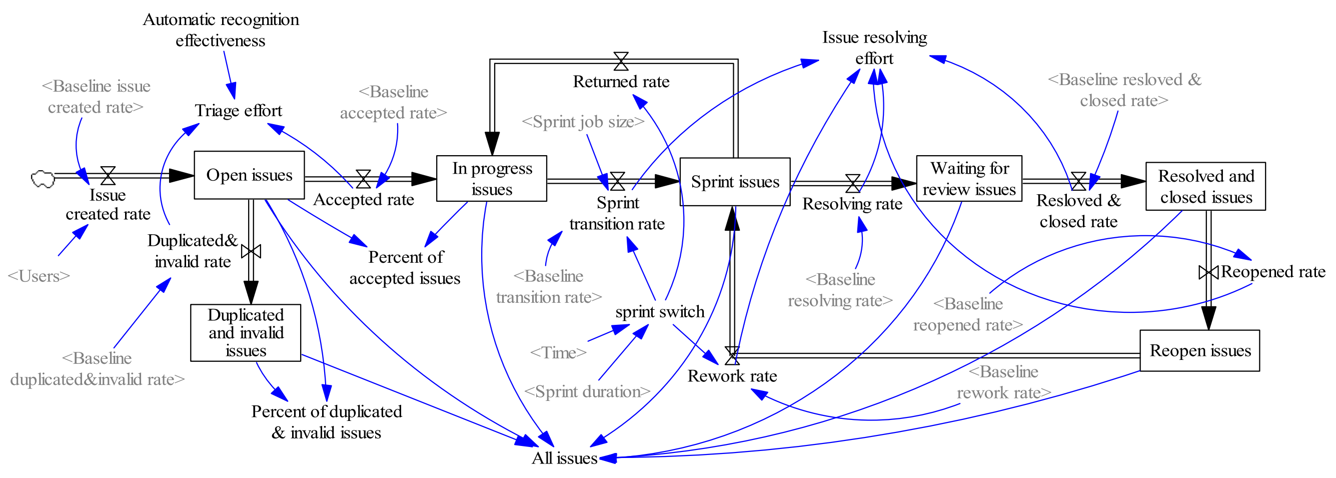


FIGURE 5. Issue tracking subsystem.

0, it is effective and makes developers more motivated. The other parameters are similar and are omitted for the sake of simplicity.

2) ISSUE TRACKING SUBSYSTEM

This subsystem models how OSS project teams handle issues. The dominant and mainstream way to manage the OSS changes from inception to full realization is the adoption of an issue tracking system [16]. Figure 5 illustrates the issue tracking process in OSS projects. The work flow in this process encompasses a normal issue tracking work flow. The normal issue tracking work flow begins at the *Open issues* stock. The *Open issues* stock is initially populated with the *Issue created rate*, which is decided by requirements volatility and user participations. The work then flows through the *In progress issues*, *Sprint issues*, *Waiting for review issues*, and *Resolved and closed issues*. Once the resolution of an issue is failed in test, the issue becomes part of the rework work flow. All issues are generated, investigated, and resolved in this work flow.

There are different types of issues such as questions, bug reports, and feature requests. Triage is the activity of categorizing, updating and adding information to issues so developers know where their efforts are most needed. Oftentimes work related to existing issues already exists. These duplicated issues might be found and closed. Some issues are also often reported incomplete. If the missing information is not provided these issues should be closed and we call them invalid issues in the model (see Figure 5). Therefore, in Issue tracking subsystem, *Triage effort* is used to show how much effort is spent on triage issues. It is calculated in the following formula (3):

$$g = (1 - u) \times (r_v + r_a) \tag{3}$$

In formula (3), g is *Triage effort*; u is *Automatic recognition effectiveness*;r_v is*Duplicated and invalid rate*; r_a is *Accepted rate*. *Automatic recognition effectiveness* is the effectiveness of using tools to help triage automatically. It is a ratio between automatic recognized issues and all reported issues.

Issue resolving effort is the number of issues processed weekly. It is the sum of *Sprint transition rate*, *Rework rate*, *Resolving rate*, *Resolved & closed rate*, and *Reopened rate*.

3) ISSUE RESOLVING SUBSYSTEM

For each accepted issue, corresponding source code might be changed. When a fix is ready for an issue, it is submitted via the commit command in the code version management software. Issue resolving subsystem simulates the code adding, modifying, and deleting. It should be noted that the contents submitted by the commit are not all related issues. Also, issues are divided into bug type or feature type. Therefore, commits are divided into *Issue commits*, *Other commits*, and *Bug commits*, as shown in Figure 6.

Total changed codes is calculated by adding up the codes for all commits, including *Codes for issue commits*, *Codes for other commits*, and *Codes for bug commits*.

4) QUALITY ASSURANCE SUBSYSTEM

According to a 2013 study by the University of Cambridge [17], [18], the global cost of finding and removing bugs from software has risen to \$312 billion annually, and it makes up half of the development time of the average project. Furthermore, constant change makes matters worse. Changes are the main causes of software bugs and major issues faced by the software industry. Studies conducted by Javed *et al.* [1] indicated that there is a significant relationship between change requests and overall bugs. Based on Mozilla quality assurance process [19], the focus of the quality assurance is correcting bugs. Therefore, based on these studies, our Quality Assurance Subsystem is modeled and depicted in Figure 7.

Software quality is calculated in the following formula (4):

$$q = b/(l \times 1000) \tag{4}$$

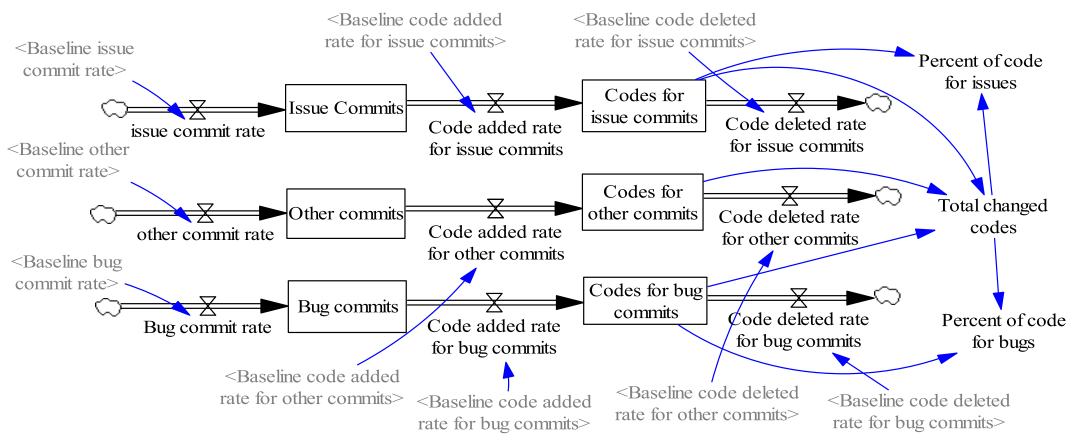

<Baseline issue commit rate>
<Baseline code added rate for issue commits>
<Baseline code deleted rate for issue commits>
Percent of code for issues
Issue Commits
issue commit rate
Code added rate for issue commits
Codes for issue commits
Code deleted rate for issue commits
<Baseline other commit rate>
Other commits
other commit rate
Code added rate for other commits
Codes for other commits
Code deleted rate for other commits
Total changed codes
<Baseline bug commit rate>
Bug commits
Bug commit rate
Code added rate for bug commits
Codes for bug commits
Code deleted rate for bug commits
Percent of code for bugs
<Baseline code added rate for other commits>
<Baseline code added rate for bug commits>
<Baseline code deleted rate for other commits>
<Baseline code deleted rate for bug commits>

FIGURE 6. **Issue resolving subsystem.**

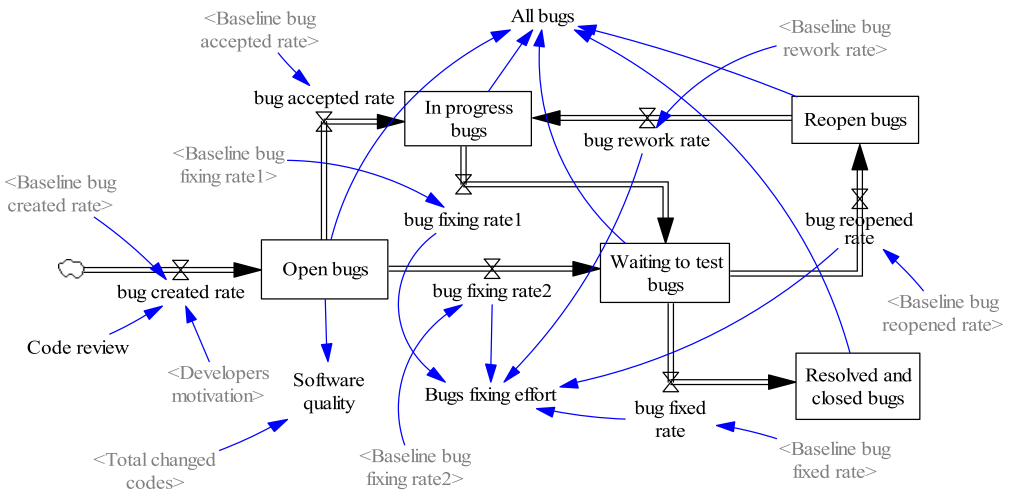


FIGURE 7. **Quality assurance subsystem.**

In formula (4), q is *Software quality*; b is *Open bugs*; l is *Total changed codes*. *Software quality* is expressed by the bug rate, which is equal to the number of *Open bugs* divided by the *Total changed codes*.

In Quality assurance subsystem, *Code review* by numerous reviewers, particularly when carried out in tandem with automatic test, can significantly, positively impact software quality. *Code review* in Figure 7 is a ratio of the effectiveness of the review. If it is equal to 0, it means that the changed code is not reviewed. In the following empirical case study (in Section III), we input a random function to analyze its impact.

5) SCHEDULE CONTROL SUBSYSTEM

In Schedule control subsystem, the workload caused by the requested changes is calculated and the duration of the project is controled according to factors such as schedule pressure of the project and the expected productivity of the project team. Figure 8 depicts the stock-level diagram for Schedule control subsystem.

In Schedule control subsystem, *Expected project duration* is obtained by dividing *Remaining issues & bugs* by *Project actual productivity*, as shown in the following formula (5)

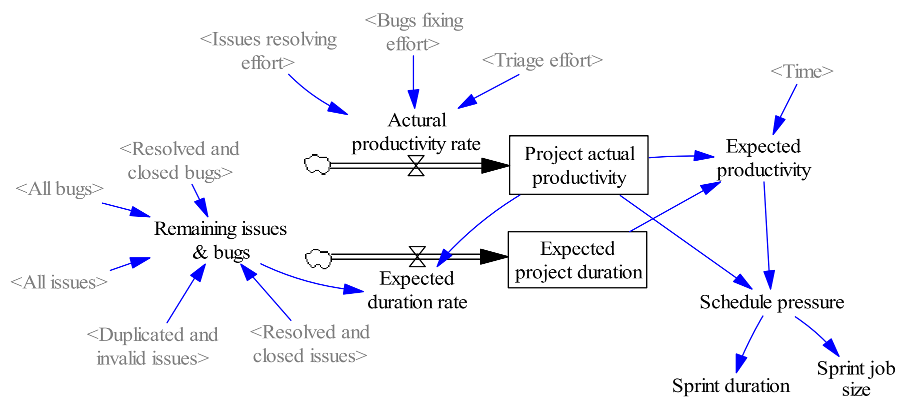
$$x = i/p_p \tag{5}$$

In formula (5), x is *Expected project duration*;i is *Remaining issues & bugs*; p_p is *Project actual productivity*. *Remaining issues & bugs* are all issues and bugs except duplicated, invalid, resolved, and closed issues and bugs. *Project actual productivity* is the sum of *Issues resolving effort*, *Bugs fixing effort*, and *Triage effort*.

Schedule pressure is calculated by the gap between the expected and actual productivity, as shown in the following formula (6):

$$s_p = (p_p \times x)/t - p_p \tag{6}$$

In formula (6), s_p is *Schedule pressure*; p_p is *Project actual productivity*; x is *Expected project duration*; t is*Time*. $(p_p \times x)/t$ is *Expected productivity*, which is the expected work rate due to *Expected project duration* x. Since excessive



<Issues resolving effort>
<Bugs fixing effort>
<Triage effort>
<Time>
Actural productivity rate
Project actual productivity
Expected productivity
<Resolved and closed bugs>
<All bugs>
Remaining issues & bugs
<All issues>
Expected duration rate
Expected project duration
Schedule pressure
<Duplicated and invalid issues>
<Resolved and closed issues>
Sprint duration
Sprint job size

FIGURE 8. Schedule control subsystem.

schedule pressure will increase bugs, decline the quality, and reduce the acceptance of the changes, to reduce its negative impact, schedule pressure is controlled by adjusting the sprint duration and job size.

By integrating all the five subsystems, our SD model represents essential change management in OSS processes. It represents OSS change management as an integrated system rather than in isolation, and incorporates dynamic features and nonlinear cause-effect relationships. Therefore, this model can be used to examine the dynamic nature of the impact of change management on critical project outcomes such as software quality, project schedule, and work effort.

C. MODEL STRUCTURAL TESTS

Once the simulation models are developed, model tests are performed to determine whether the models are suitable for the purpose, address the problems, and reasonable representation of the real process [2]. System dynamics modelers have developed a wide variety of specific tests, including tests of model structure and the ability of the model to reproduce real-life behavior [20], [21]. By adopting the tests from Sterman [21], the test activities were performed by the model developer, and software process and project experts. The identification of the appropriate structure is the first step in SD model test because the structure drives its behavior [22]. Table I summarizes the performed model structural tests and the test results.

The results in table 1 suggest that our model structure closely reflects real OSS change management processes. The model is then ready to include quantitative data. In order to derive the quantitative relationships between the parameters in the simulation models, an empirical study on a real-life OSS project is presented in the following. Also, behavioral tests were assessed in this study to achieve the overall validity of the model.

TABLE 1. Model structural testing lists.

Test	Purpose of test	Testing procedure and results
Boundary Adequacy	Assesses the appropriateness of the model boundary for the purpose of the model.	The model boundary was determined by the subsystem diagram (See Figure 2). After reviewing of the relevant literatures and soliciting expert opinion from interviews, the model was improved and no important feedbacks were omitted from the model.
Dimensional Consistency	Ensures that each equation dimensionally consistent without the use of parameters having no real world meaning.	'Units Check' in Vensim was used for automated dimensional analysis first. Then, the model equations were inspected for suspect parameters. Any dimensional errors or suspect parameters had been corrected.
Structure Assessment	Ensures that the model is consistent with knowledge of the real system relevant to the purpose.	After inspecting the model equations, the model was modified to be conformed to basic physical realities.
Parameter Assessment	Makes sure every parameter is reasonable and has a clear, real-life meaning.	After reviewing the relevant literature and referring to the data collected from OSS projects, all parameters were assessed and modified to have clear and real-life meanings. Then, the values of each parameter were estimated judgmentally using the expert opinion.
Extreme Conditions	Ensures that the model behave appropriately when the inputs take on extreme values.	We tested the output when each input to the equation takes on its maximum and minimum values. Any outputs that violated feasible or reasonable had been corrected.
Integration Error	Ensures that the model is not sensitive to the choice of time step and integration method.	Euler integration was used and the time step was week. There was no change in behavior when the time step was cut in half or different integration methods were used.

III. EMPIRICAL CASE STUDY

The process simulation should be combined with empirical studies to help process understanding and improvement. We demonstrated the applicability of our SD model in a case study of the open source software-Spring Framework Version 3. By parameterizing the elements of the SD model and using public data from Spring Framework to initialize, the baseline performance is evaluated, which includes the changes of project manpower, productivity, and software quality. Then, by simulating three process improvement strategies, their improvement performances were compared. For example, by adding *Code Review* activity in *Quality Assurance Subsystem*, how the software quality and the expected project duration will be changed.

A. CHANGE-ORIENTED OSS PROCESS ANALYSIS

Since Spring Framework Version 3 was released in 2009, all the project-specific inputs were collected in the duration of 440 weeks. After data cleanup and format transformation, these inputs were inputted into the Vensim for simulation. The project performance was then reflected by the simulation output data. Before the simulation analyses, model behavior tests were conducted to improve the model. Table 2 lists these behavior tests.

After the model has been improved and tested to be validated, a more detailed analysis of the simulation results of software project manpower, productivity, and quality is illustrated below.

1) PROJECT MANPOWER

The success of OSS development is highly dependent on the formation and evolution of their supporting communities [3]. Modeling the process factors of community would enable better understanding of open source methodologies. Therefore, using the data from Spring Framework, the number of the *Users*, *Contributing developers*, *Core developers*, *Developers productivity*, and *Expected project duration* outputs were examined and the simulation results are illustrated in Figure 9.

In Figure 9, (a), (b), and (c) are the simulation results of OSS community subsystem, indicating the number of Users, Contributing developers, and Core developers change over time. All of the simulation results show an increasing trend, reflecting the gradual development and expansion of the software project team. Especially after 176th week, Contributing developers increase substantially. Developers' productivity in (d) also shows a corresponding increase. However, in the mid-late stage (after 244th week) of Spring Framework Version 3 development, due to the release of Version 4 and the reduction of change requests, the time spent by developers in Version 3 was gradually reduced. Unresolved change requests were migrated to other version branches. Therefore, after 244th weeks, project developers' productivity fluctuates more slowly.

In Figure 9, (e) is the simulation result of the Schedule control subsystem, indicating *Expected project duration*. The duration required to deal with the software changes decreases with time, and then increases. Before 44th week, the decrease of *Expected project duration* in Fig. 9 (e) is because the fast increasing in the manpower and less change requests in the early stage. But, in the middle to late stages, change requests increase sharply, while the developer's average productivity increase rate is relatively slow, so that the duration of dealing with the software changes is increasing.

TABLE 2. Model behavior testing lists.

Test	Purpose of test	Testing procedure and results
Behavior Reproduction	Checks if the model reproduces the behavior of the interest in the system.	Point-by-point fit was used to compare simulated and actual data in MATLAB 2015b. The coefficient of determination R^2 and root mean square error (RMSE) were used as the measures of fit. Some flaws were discovered and had been revised. Additionally, these revisions were checked to be consistent with all the other tests discussed above.
Behavior Anomaly	Establishes the significance of important relationships by examining whether anomalous behavior arises when the relationship is deleted or modified.	When variables in the casual loop were removed from the model, the model exhibited anomalous behavior. Therefore, all the relationships in casual loop are important and must be included.
Family Member	Ascertain whether the model can generate the behavior of other instances in the same class as the model was built to mimic.	Hadoop Common was used as another instance to test model behavior. The test results will be described in the following forth analysis item (See 4) Simulations in different projects).
Surprise Behavior	Examines unexpected or anomalous behavior.	The behavior of each variable was tested and no unexpected or anomalous behavior was found.
Behavior Mode Sensitivity	Evaluates the impact of changing assumptions on the robustness of the conclusions.	In assessing sensitivity to parametric assumptions, uncertain and influential parameters and relationships were analyzed by defining best and worst case scenarios and comparing these scenarios to the base case. The best and worst cases provide bounds of the model behavior.
System Improvement	Checks whether the modeling process helped change the system for the better.	We proposed three improvements for the change management process. They were code review, attracting participants, and requirements triage. The simulation results showed that the system can be better after taking the improved policies. The details of these improvement tests are described in the following Section B.

2) PROJECT PRODUCTIVITY

In the following, the issues, bugs and project productivity outputs were examined. Their simulation results are presented in Figure 10.

In Figure 10, (a) indicates the number of weekly issue creations, (b) indicates the actual productivity per week. Using curve fitting to analyze the similarity between *Issue created rate* and *Actual productivity rate*, the coefficient of determination $R^2 > 0.6$. That is, the creation rate of the issues affects the actual productivity of project. In Figure 10, (c) represents the percentage of accepted change requests. It can be seen that more than 90% of the change requests were accepted. From the impact of changes on project productivity

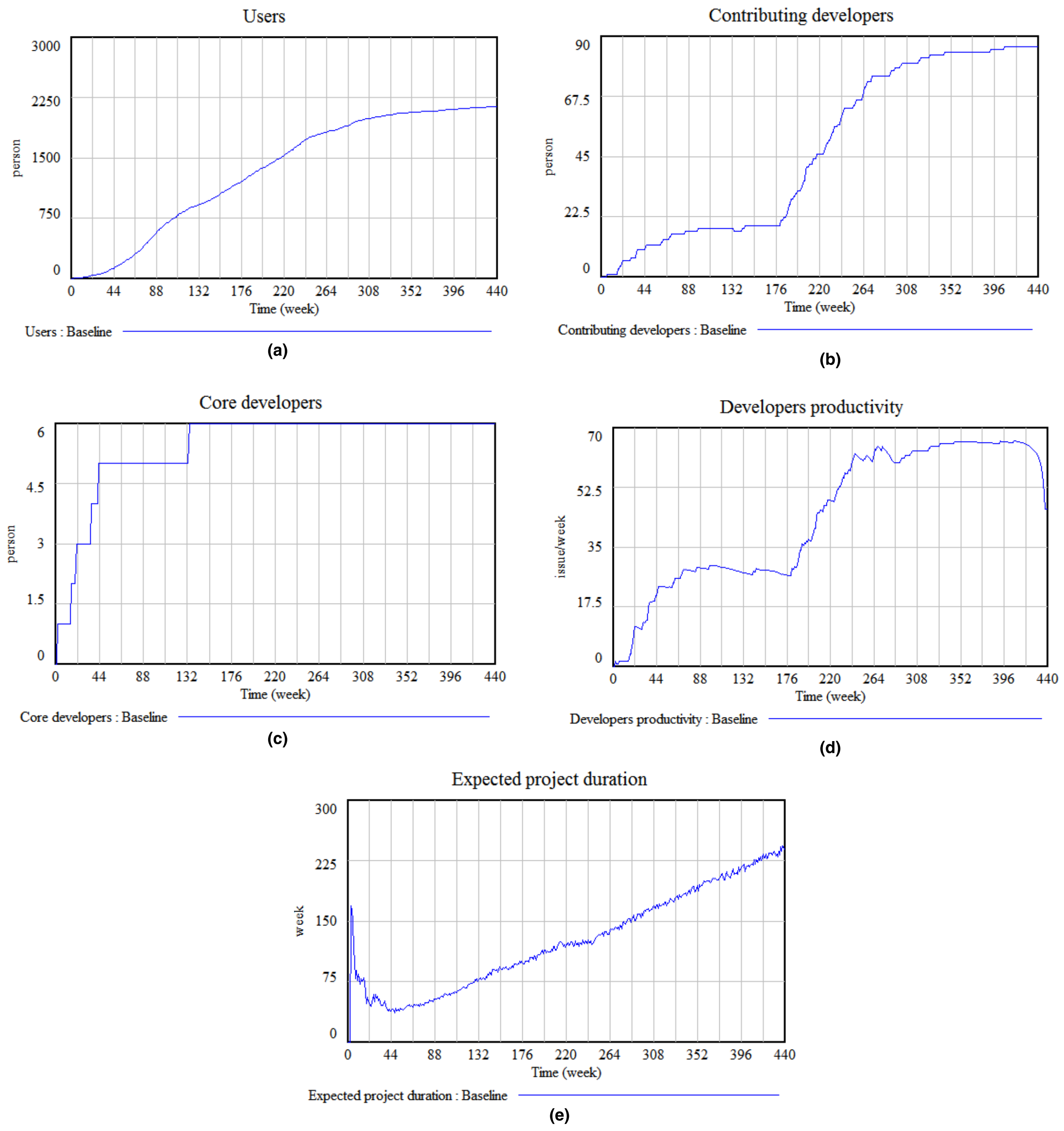


FIGURE 9. Analyses of human resource and developers' productivity. (a) Number of users. (b) Number of contributing developers. (c) Number of core developers. (d) Developers' productivity. (e) Expected project duration.

and the percentage of accepted issues, we can say that in Version 3, the project developers can respond positively and rapidly according to the change requests. After 244th week, due to the release of Version 4, there are the same decrease in both (a) and (b). But, between 220th to 244th week, there is a sharp increase of *Actual productivity rate* in (b), which means the project team resolves and closes most issues in Version 3 and prepares to release Version 4.

Similarly, we use curve fitting to analyze the similarity between *Bug created rate* and *Actual productivity rate* ((d) and (b) in Figure 10), the coefficient of determination $R^2 > 0.3$. There is less impact of bug creation on the project productivity. This is because the bugs are found and reported later than the proposed issues. However, from the simulation result of (e), *Percentage of code for bugs* increases continuously. The project team has a positive response to the fixing of bugs. This is a positive effect on attracting users.

In Figure 10, (f) reflects the percentage changes of duplicated and invalid issues that is rejected by the project team. In the entire life cycle of Version 3, rejected duplicated and invalid issues are within 10% of all issues. In the early stage, this percentage shows an overall rapid upward trend. However, in the later, it can be seen that the project team make more effective control on the rejection percentage and keep it around 6%, showing a steady trend.

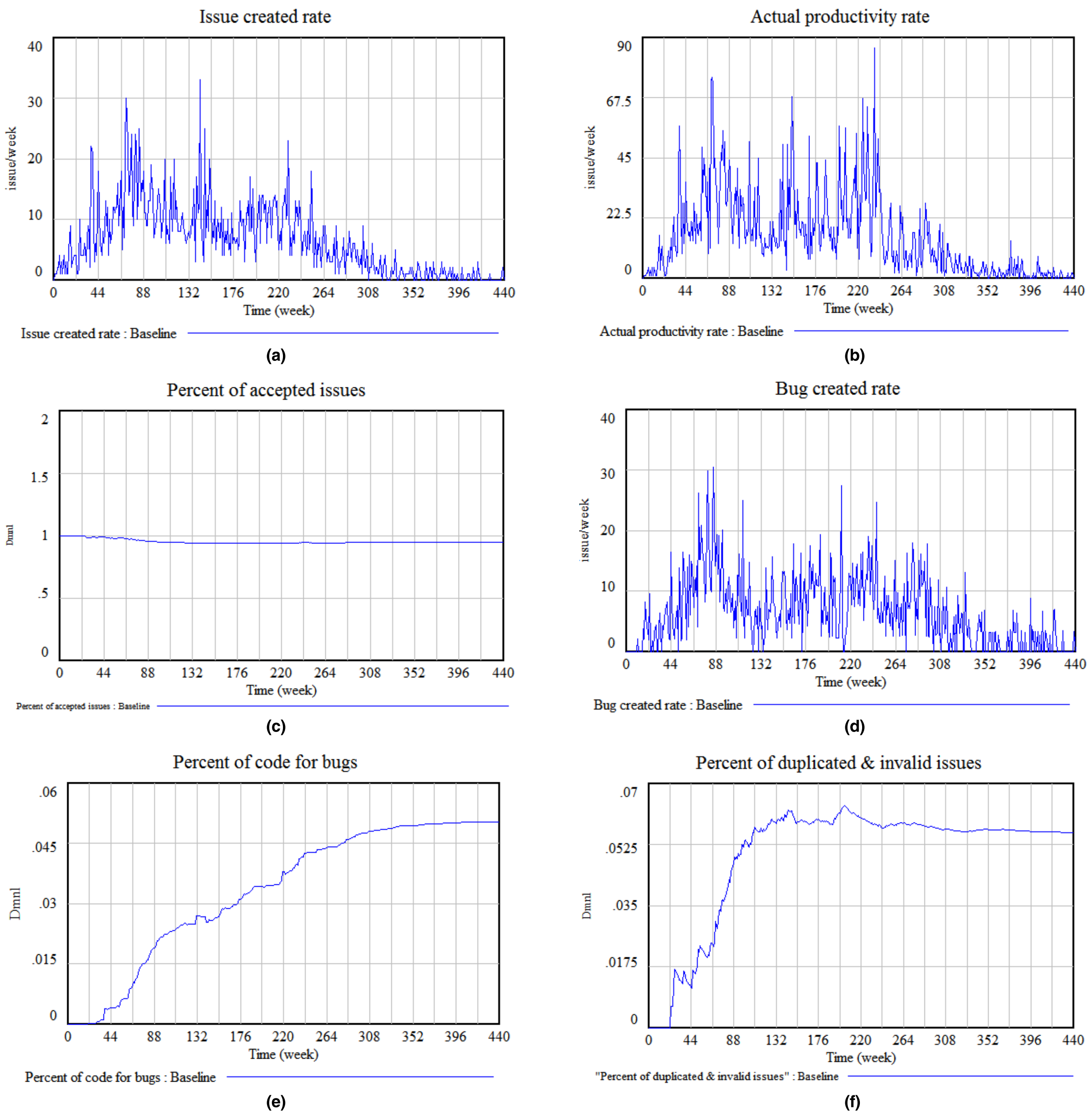


FIGURE 10. Analyses of software changes and project productivity. (a) Issue created rate. (b) Actual productivity rate. (c) percent of accepted issues. (d) Bug created rate. (e) Percent of code for bugs. (f) Percent of duplicated & invalid issues.

3) SOFTWARE QUALITY

Quality dynamics are different in OSS, partly because "Many eyes make all bugs shallow" [3], [14]. With so many open source testers, a high percentage of bugs could be found in every release. In order to analyze the quality dynamics, the simulations of *Resolved and closed issues*, *Total changed codes*, *Developers motivation*, and *Software quality* were examined and the simulation results are presented in Figure 11.

In the Figure 11, (a) is the simulation result of Issue tracking subsystem, indicating the number of the completed issues; (b) is from Issue resolving subsystem, indicating the total changed codes. According to (a) and (b), as the number of completed issues increases, the code lines for software changes is increasing. All the software changes expand the software scale and make the code maintenance more difficult. Therefore, the software changes cause *Developers motivation* and *Software quality* to decline slowly, as shown in Figure 11 (c) and (d). Please note that the software quality is represented by the bugs per KLOC, which means an increase in software bug rate indicates a decline in software quality.

In the early stage, *Developers motivation* (see Figure 11(c)) and *Software quality* (see Figure 11(d)) do not decline sharply because the number of issues is small and the productivity of

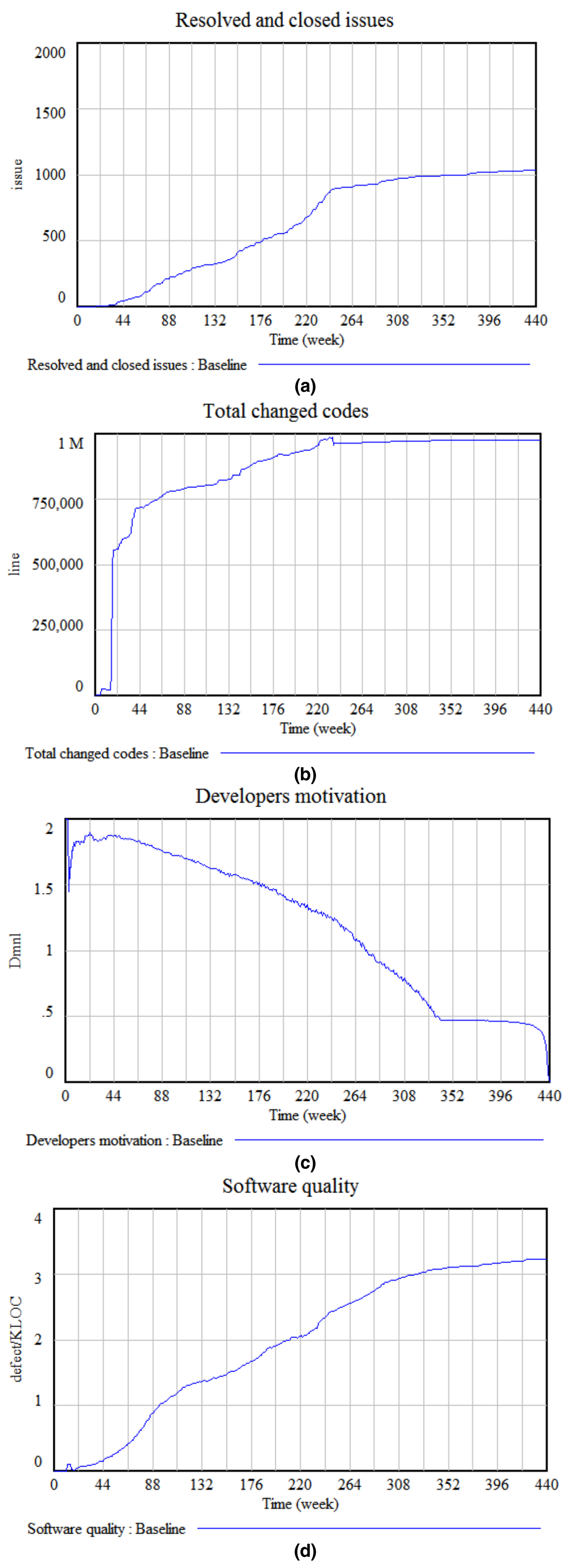


FIGURE 11. Analyses of software quality. (a) Resolved and closed issues. (b) Total changed code. (c) Developers motivation. (d) Software quality.

developers is rising rapidly. But in the middle to late stages, the number of issues increases sharply, the productivity of developers increase relatively slow. This relatively slow speed causes the project schedule pressure to rise, which leads to a gradual decline in *Developers motivation*(see Figure 11(c)), and further leads to the software quality decline slowly (see Figure 11(d)). In addition, at the 310th weeks, Version 3.2.10 was released and the bug reporting rate was reduced. After 310th weeks, *Software quality* and *Developers motivation* in Figure 11 (c) and (d) fluctuate more slowly. Similarly, it can be seen from (a) that after 244th weeks, Version 4 was released and the project team devoted most of their energy to the new version. Therefore, as shown in Figure 11 (a) and (b), after 244th weeks, *Resolved and closed issues* are reduced and the fluctuation of *Total changed codes* are also flat.

4) SIMULATIONS IN DIFFERENT PROJECTS

To ascertain whether the model can generate the behavior of other instances in the same class as the model was built to mimic. Hadoop Common was used as another instance to test the model behaviors. In the following Figure 12, the simulation results of Hadoop Common are presented and further compared with Spring Framework.

In Figure 12, (a) and (b) reflect the expansion of the number of *Contributing developers* in Hadoop Common and Spring Framework respectively. It can be seen that they both have a sharp increase in the middle stage of the project. But, there are more developers in Hadoop Common and they are constantly growing, while in the later stages of Spring Framework, the developers are relatively stable and no longer sustain rapid grow.

The simulation results of (a) and (b) reflect different behavior patterns of developer growth in both projects. In contrast, (c) and (d) in Figure 12 reflect the similar fluctuations in the *Developers motivation*. By further analyzing, the similarity is due to similar fluctuations in *Schedule pressures* in the two projects. As shown in the preceding formula (6), *Schedule pressure* is calculated by the gap between the expected and actual productivity. It is actually calculated by dividing the number of remaining issues and bugs by the actual productivity of the developers. In the early stage of the two projects, the number of issues and bugs was small and the productivity of developers was rising rapidly, reflecting the increased motivation of the developers. In the middle to late stages, the number of issues and bugs increased dramatically, and the average productivity of the developers raised relatively slowly, which led to a gradual decline in *Developers motivation*. However, since the developers in Hadoop Common continue replenish and the number is relatively large, the decline of *Developers motivation* is smoothing.

In Figure 12, (e) and (f) simulate the software quality (represented by the bug rate) changes in two projects. As shown in (e), the quality of Hadoop Common fluctuated greatly in the early stage, and the bug rate is extremely high. However, from 108 weeks to 112 weeks, the defect rate decreases and the late rise gradually flattened. The reason for such large fluctuations was that in 2009, Hadoop Core was renamed to Hadoop Common and a stable version was released in 2011

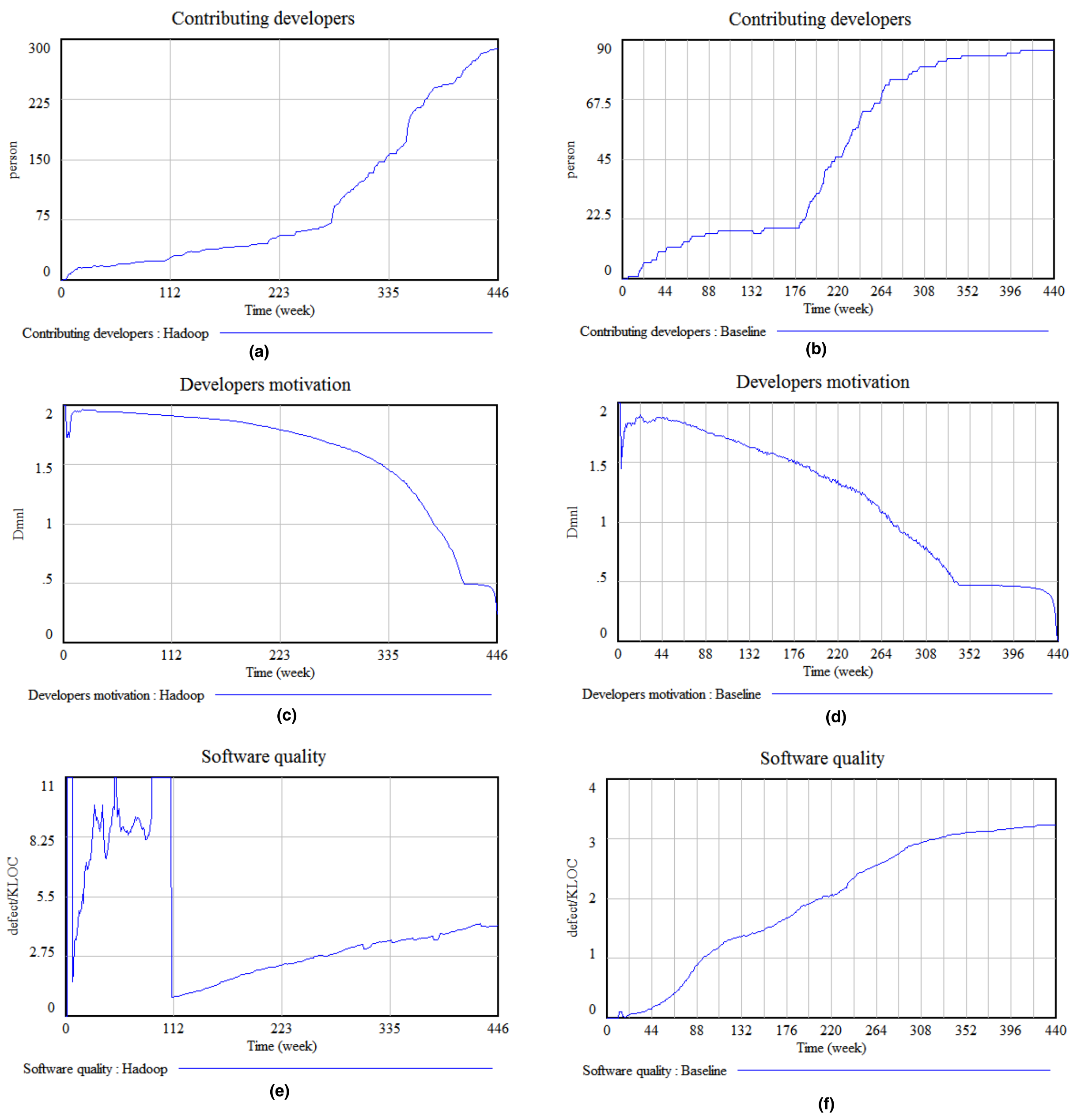


FIGURE 12. **Simulation comparison between Hadoop Common and Spring Framework. (a) Hadoop Common contributing developers. (b) Spring Framework contributing developers. (c) Hadoop Common developers morale. (d) Spring Framework developers motivation. (e) Hadoop Common software quality. (f) Spring Framework software quality.**

(about 108 weeks). Unlike Hadoop Common, Spring Framework has consistently shown a trend of stable increasing.

The preceding understanding of change-oriented software process simulation for OSS would not only help us to understand the phenomenon of open source but would also benefit practitioners in both open source and closed-source software communities. However, software organizations need to improve their software processes continuously to meet the growing demands for products and services. Based on the simulation results, how to improve the software quality? Which software process improvement strategies are effective? The answers of these questions can also be simulated and compared in our model. In the following, we propose and test three improvement strategies.

B. SOFTWARE PROCESS IMPROVEMENT SIMULATION

Humphrey [Humphrey 1989] argued that software quality can be improved by improving its development process. Both academia and industry are striving to find ways for software process improvement (SPI). There are numerous SPI framework and methodologies available today, but they all have one challenge in common: the cost of experimenting with the process change. It is widely claimed that software process simulation modeling can help in predicting the benefits and repercussions of a process change, thus enabling organizations to make more informed decisions and reduce the likelihood of failed SPI initiatives [8].

Currently, mainly two types of SPI approaches are being used in practice: model-based SPI approaches and continuous SPI approaches. Model-based SPI approaches compare organizational processes with a reference model and are used to identify coarse-grained problem areas and potential improvement strategies. Continuous SPI approaches are used to develop process-specific solutions for important problems and assess the effects of improvement actions [2]. Since continuous SPI approaches can be applied, independent of the maturity of an organization, we used continuous approaches in this paper.

Continuous SPI approaches focus on solutions for the most important challenges of a software development organization and usually involve improvement cycles based on an initial baseline that defines the respective starting point of each improvement action [2]. In the following, we describe and discuss three improvement actions that can be used to assess the trade-offs independent from a specific project context. We utilized the simulation to evaluate alternative actions on how to construct the future software processes. For each improvement context, the baseline was the situation when all improvement activities are set to be no effect, while the improvement was to set the random effect for all improvement activities.

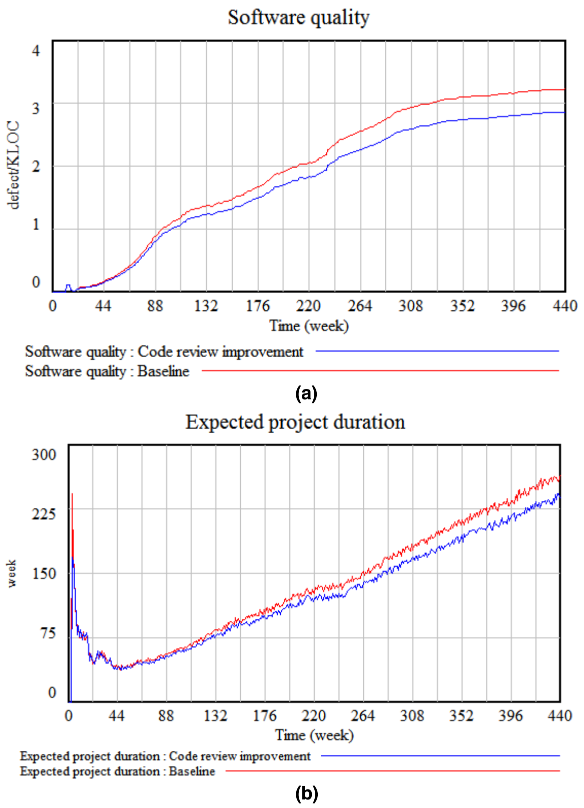


FIGURE 13. Improvement of using code review. (a) Software quality. (b) Expected project duration.

1) CODE REVIEW

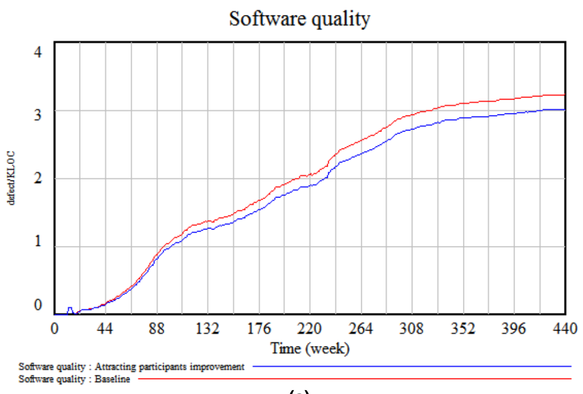
Code review is a continuous process in OSS development [23]. Overall, the research agreed that code review by numerous reviewers can significantly, positively impact software quality. To test the effectiveness of code review, the parameter of *Code review* in Quality assurance subsystem was given in formulas RANDOM NORMAL(0, 1, 0.05, 0.1, 0) and all the other input parameters were kept constant. The simulator was executed and the values of the quality and duration were examined according to the variation in the extent of code review.

In Figure 13, all the red lines in (a) and (b) provide the original software project performance of the process without code review activity. They are the initial baselines. The blue lines show the performance of the improved process, which we add the code review improvement. In Figure 13(a), it can be observed from the two lines that the improved process yields an improved product quality of fewer bugs per KLOC. In addition, as shown in Figure 13(b), at the beginning of the software project lifecycle, the simulation results indicate a time overrun of about 5% compared the baseline. This delay, however, is changed in the later project lifecycle and has an apparently decrease.

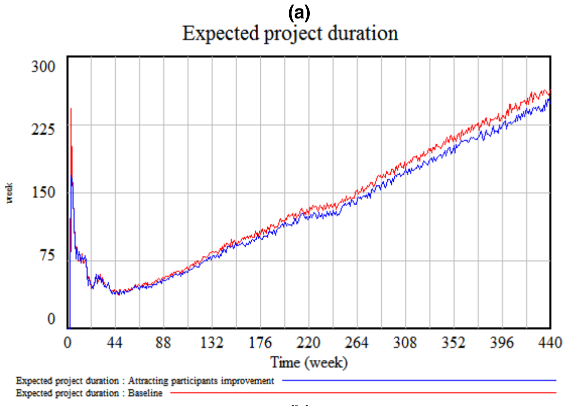
2) ATTRACTING PARTICIPANTS

People management plays a vital role in developing high-quality software [15]. Many studies concluded that creating a sustainable community should be an OSS project's key objective, and a sustainable community relied on attracting participants. Specifically, research has found that reasons to participate include enjoyment in helping others improve software, enjoyment in tackling complex programming problems, improving programming skills, gaining financial benefits, signaling competence to potential employers, improving future job prospects, gaining recognition from peers, enhancing reputation in the field, and identifying with the project teams [24]. Many strategies for attracting participants are proposed. These include establishing an effective environment and culture. For instance, modular design would make project tasks less time demanding and more achievable, which would better satisfy participants' needs for competence [24]. Furthermore, social networks should be developed to facilitate communication, coordination, and collaboration among participants [24]. Therefore, our second SPI simulation is to examine the effectiveness of attracting participants. To test the effectiveness of attracting participants, the parameter of *Effectiveness of attracting participants* in OSS community subsystem was given in formulas RANDOM NORMAL(0, 1, 0.05, 0.1, 0) and all the other input parameters were kept constant.

Very similar to Figure 13, (a) and (b) In Figure 14 show the simulation comparisons between baseline data and improvement data. It can be concluded that attracting participants can



(a)



(b)

FIGURE 14. **Improvement of using strategies for attracting participants. (a) Software quality. (b) Expected project duration.**

reduce the software bug rate (the reduction of the bug rate means the increase of the software quality) and reduce the duration required for software projects. Therefore, for higher quality, open source development requires a critical mass of participants for testing the software and providing suggested changes. People dynamics and their motivation are essential areas of study for OSS processes.

However, compare (a) in Figure 13 and (a) in Figure 14. When using the same random function, the decline of blue line in Figure 13 is more than the blue line in Figure 14. Using code review may be more effective than using attracting participants. The comparison of *Expected project duration* will be elaborated in the following.

3) ISSUES TRIAGE

In the software change management, invalid and duplicated issues consume the cost and time of the software project team and degrade developer motivation and software quality. In order to reduce this impact, it is necessary to categorize and filter issues. In Figure 5, issue triage has been designed in Issue tracking subsystem. To illustrate the necessity of issue triage, a new stock-level diagram is designed by deleting issue triage, as shown in Figure 15 (a). Due to the lack of issue categorization and filtering, all issues will be accepted. Only when the issues are executed, duplicated and invalid issues will be found. The result is a waste of labor and time.

TABLE 3. **Comparing improvement strategies.**

Improvement Strategies	Labor cost	Time cost
Code review	High	high
Attracting participants	Normal	Normal
Issues triage	Low	Low

(b), (c), (d), and (e) in Figure 15 are the simulation comparison results.

In Figure 15 (b), the blue line indicates the simulation result of Figure 15 (a), that is, the issue triage is not taken. The red line indicates the simulation result of Figure 5, that is, the issue triage is taken. By comparing and analyzing Figure 15(b), it can be concluded that the issue triage activity makes the *Project actual productivity* higher. Thereby, reduce the *Expected project duration*, as shown in (c). In Figure 15(d), the uselessness of implementing invalid and duplicated issues makes *Developers motivation* lower, which in turn leads to an increase in the number of bug reports. Ultimately, it leads to a decline in *Software quality*, as shown in (e). Therefore, under the same conditions, issue triage is useful in reducing the loss caused by invalid and duplicated issues to the software.

When we further compare these simulation results with the results of the previous two improvements, the impact of the issue triage activity on *Software quality* and *Expected project duration* is more significant.

From the comparison of simulation results, we can see that different process improvement strategies can be adopted for different software organizations or different software projects. But, how do we choose the most appropriate process improvement strategies? We try to answering this question in the following.

4) PRIORITIZATION OF IMPROVEMENTS

Due to the fact that software processes are usually human-based and depend on the development context, improvements to these processes also cause significant costs and should be considered carefully [2]. In the following Table 3, the preceding three improvement strategies are evaluated with respect to their implementation labor cost and time cost in the case of improving the same software quality (reducing the same bug rate).

In general, software process improvements require different labor costs and time costs. When dealing with change requests, code review requires one or several humans to check a program mainly by reading and reviewing its source code. If the number of change requests is large, it may take more labor and time costs. Therefore, OSS project teams usually recommend that the modified codes should be as small as possible. When attracting participants to an OSS project team, the preceding methods that we described in Section 2) are typically used. Comparing to the other two improvement options, the labor and time costs are less than

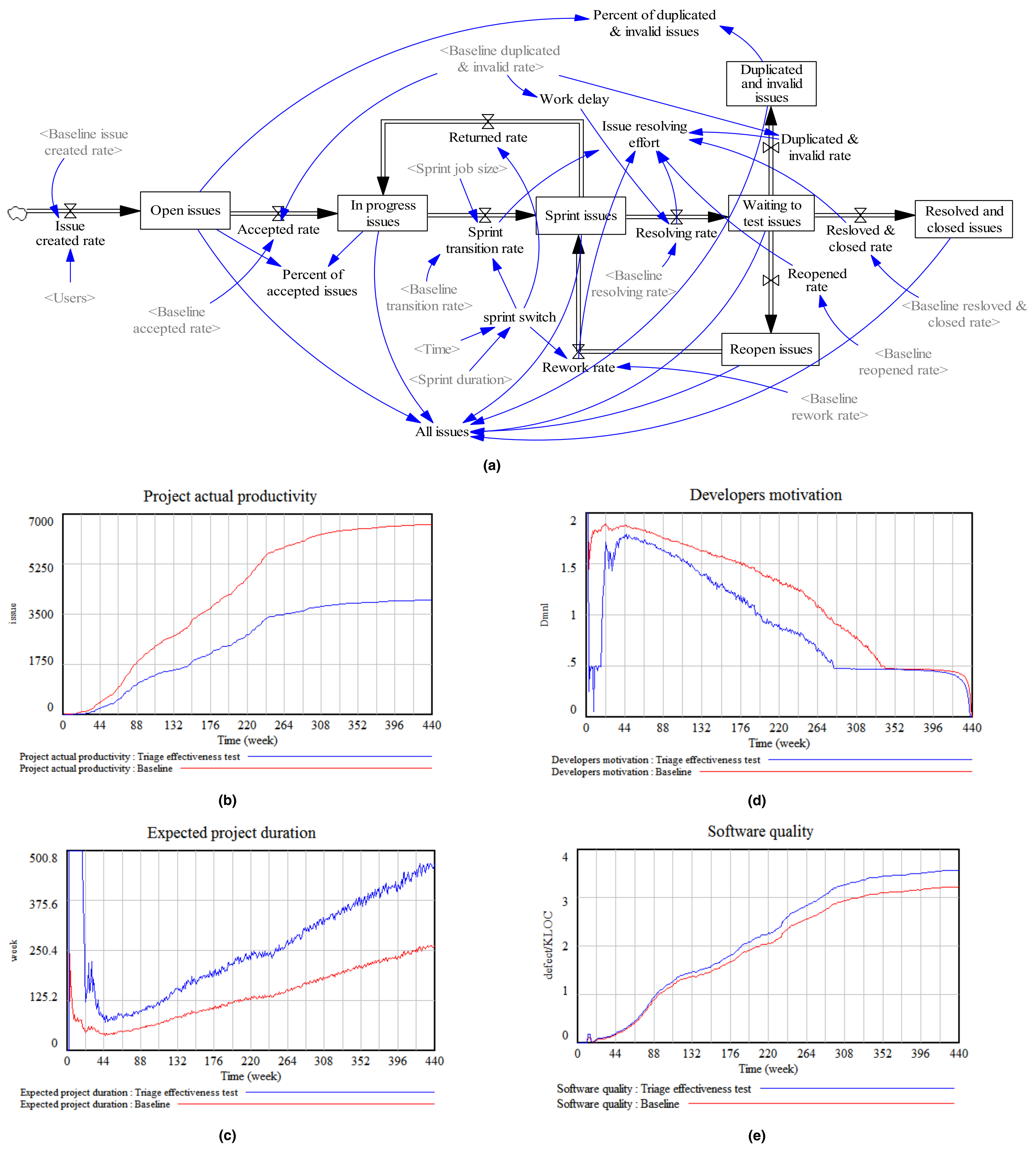


FIGURE 15. **Triage effectiveness test stock-level diagram and simulation comparisons. (a) Triage effectiveness test stock-level diagram. (b) Project actual productivity. (c) Expected project duration. (d) Developers motivation. (e) Software quality.**

code review but more than issues triage. However, it must be mentioned that the methods may be invalid, delayed or even counterproductive. These negative impacts should be fully considered that may reduce the motivation of developers. For the last improvement strategy, issues triage, since most of the current large and medium-sized OSS projects use the issue tracking system to manage the change requests [10], the tool itself can provide issues classification and filtering capabilities. In comparison with the other two improvement options, the labor and time costs required may be less. That is why this option is used by most large and medium-sized OSS projects.

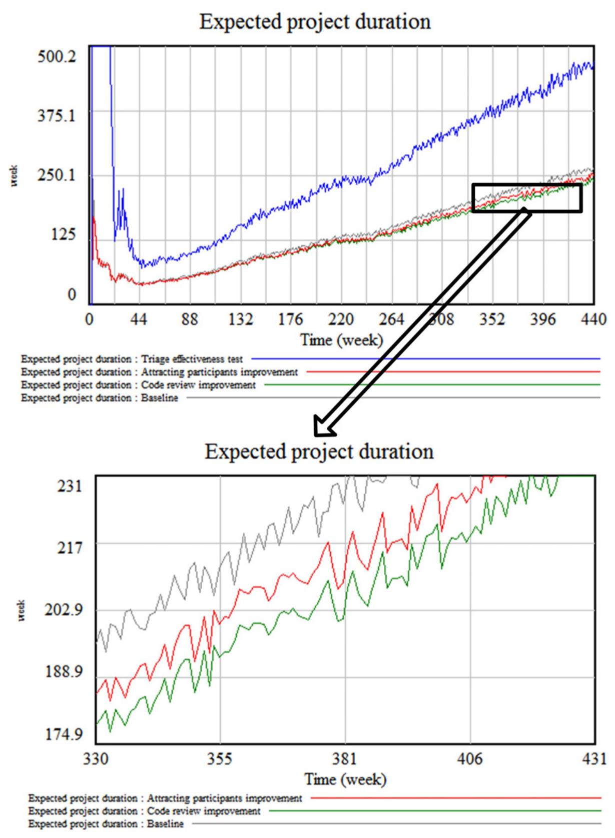


FIGURE 16. **Comparing improvement options.**

Next, we compare the simulation results of these improvement stratigies on *Expected project duration*. The comparison simulation results are presented in Figure 16.

In Figure 16, the second figure is an enlarged partial image of the first figure. The grey, red, and green lines in the second figure indicate *Expected project duration* that has taken the improvements of issues triage, attracting participants, and code review respectively. When the gray line is used as the baseline, both improvement options are effective, and the effect of using code review is better.

According to the above simulation results and the cost comparison analysis, issues triage should be used first. For the remaining two options, if attracting participant methods can be use appropriately, it can be given priority because the costs is relatively low and the project duration and software quality improvement will also have a good effect. In the case that the labor cost of the project team is relatively sufficient, although code review consumes a certain cost, it has the greatest improvement effect on the project duration and software quality.

In summary, the simulation model assisted in reasoning about why certain alternatives were better than others. The comparing outcomes can help select and customize the best process for a specific context.

C. DISCUSSION

OSS favors distributed development and the use of tools to support their change management tasks [25]. This attracts the company because they involve geographically distributed developers and are trying to rely more on tools for their change management. Therefore, we used the SD modeling to simulate and analyze the performance of Spring Framework change management process. The model and results were presented to the project leaderships in Taili Technologies. Overall, they were pleased with the results of the modeling effort. From the simulation results, many valuable lessons had been learnt.

The benefit of using SD modeling is that this modeling approach provides the means to gather all relevant factors and parameters in a holistic view. Moreover, while the variables included in the model can be diverse, that is, representing different process improvement strategies. The application and improvement analyses presented in Section A and Section B are meant to illustrate how the SD model can be applied in a software project in order to assess the effectiveness of change management and SPI strategies. Therefore, our SD model can be used to visualize the critical project behavior and to discuss the assumptions about the cause-effect relationships that are supposed to be responsible for the generated behavior. The design of our models is such that it has the following positive properties:

(1) Multi-causality: the model offers the possibility to assess the impact of various parameters not only individually but also concurrently. In other words, it is a holistic model that does not limit itself to mono-causal dependencies but facilitates the consideration of complex inter-dependencies among impact factors.

(2) Comprehensiveness: our SD model offers the possibility to assess the impact of various model parameters not only on one performance parameter but also several performance parameters simultaneously, thus allowing for trade-off analyses and holistic assessment of the cost-effectiveness of SPI strategies.

However, two limitations were also found and discussed.

(1) The model structure and the simulation results are not reviewed by experts of Spring Framework or Hadoop Common. Our modeling goal is a common change management process for all OSS projects. By studying the change management processes of some OSS projects, we found that the processes of different OSS projects have more or less differences. Therefore, Our SD models were defined at a level of abstraction such that it can be considered experimental change bed containing all essential elements representing the typical steps in change management processes. However, without the review by the open source project team, it is still a threat to the effectiveness of our empirical research.

(2) When simulating the software process improvements, we used the random values. In the real project, to effectively detect the impact of the improvement strategies real data that according to the actual context should be used.

Regarding modeling cost, the setup of our first model structure was consumed three persons in three months of effort. More difficult is the simulation study and the calibration of the SD models. After the first model structure was determined, team members cost about two months for modification and calibration. In addition, calibration effort and success depends on the availability of valid data in the software project to which the model shall be calibrated, and the ability to estimate those parameters for which measurement data are available.

Simulation models like the one presented in this paper have the preceding limitations, but once in place they allow for easy evaluation of many different configurations of all software processes in a software project. Therefore, the model will be continuously adjusted for the project specific demand and simulation data will be collected and inputted into the model to provide more valuable simulation results.

IV. RELATED WORK

System Dynamics (SD) was first proposed by Forrester for industry simulation with its origins in the 1950s [22], [26]. In the late 1980s, software process simulation was started from the work by Kellner and Hansen [27], Lin and Levary [28], and Abdel-Hamid and Madnick [29], and has become an active and growing area of research [18]. In 1998, the first international workshop on software process simulation modeling (ProSim'98) held at Silver Falls. Then, many companies have experimented with using process simulation models as a management and decision-support tool [18].

A software process simulation model focuses on some particular software development/maintenance/evolution process [9]. Although there are some significant applications of SD to model and simulate the traditional software process, our model focuses on the change-oriented aspect of OSS processes. In this area, only a small number of the published simulation models focus on the analysis and improvement of change-oriented software process. To the best of our knowledge, our work is the first to apply SD simulation to model and analyze change-oriented OSS processes.

A. SIMULATION OF CHANGE-ORIENTED SOFTWARE PROCESSES

Software requirements and software processes in practice are not static but changing. In this environment of perpetual change, a means of effectively predicting the impact of such changes on software development would be valuable.

In 1989, Lin and Levary [28] presented a dynamic model for the software process to assist in assessing the impact of both changed requirements and new technology on software development schedules and costs. They found a change in the requirements phase leading to changes in all succeeding phases, and also leading to a delay in schedule and staff level. In the year 1994 and 1995, Pfahl and Lebsanft developed a prototype PSIM (project simulator) model to explore the potentials of the SD approach [26]. Using a fictitious example, the effects of unexpected change requests were simulated in PSIM. The simulation results showed the SD model can be used to find an optimal policy to avoid time overrun and/or quality loss. In 2000, they used a simulation model in an industrial application to demonstrate that requirements volatility is extremely effort consuming [30]. Similarly, Houston *et al.* [31] used stochastic simulation and survey to show that requirements creep is the most significant risk factor. Donzelli and Iazeolla [32] proposed a hybrid simulation modeling to study the effects of requirements instability on various process quality attributes. Martin and Raffo [33] presented a hybrid model to evaluate simultaneous changes to both the process and the project environment. All their work was applied to a traditional process, but hybrid models can examine questions that cannot be answered by either system dynamics models or discrete event models alone. We will try to use hybrid models in the future. In 2008, Madachy presented software process dynamics [3] to explore the dynamics of the software process. In his book, the model for requirements volatility was introduced by using the work from Ferreira ea al.. Ferreira *et al*. developed a simulator for better understanding the requirements engineering process and the impact of requirements volatility [6]. After the research of traditional processes [6], Ferreira *et al*. suggested that agile processes are another valuable area to study because they welcome requirements changes. For the objective of researching on the agile development practices, Cao *et al.* [20] first applied SD simulation to model essential aspects of agile software development and studied the dynamic implications of two agile practices (refactoring and pair programming). Although several agile methods emphasize co-location and do not focus on using tools to support its development tasks [25], both OSS and agile development embrace change by using short feedback loops with frequent releases [34]. Therefore, our SD models have same goal in change management function, but we elaborated more details in OSS change management about the different people factors, group dynamics, quality attributes, different evolution dynamics, and other phenomena compared to closed source development. In addition, they focused on agile practices, whereas we emphasized SPI in the change-oriented context.

The first law of software evolution [35] is continuing change. In attempt to simulate the factors that affect the software evolution behavior and better manage software evolution, Lehman and Ramil [35], Wernick and Lehman [36], Wernick *et al.* [37], Kahen *et al.* [38], and Ali *et al.* [39] developed several simulation models and continuously improved their models. However, except the model in [38], the other models abstracted the modeling of changes. Although the changes were defined explicitly in [38], the corresponding model variables did not refer to any particular measure.

The target of these articles is not open source type projects but more traditional closed source development type projects. In addition, little empirical research has been carried out on the topic of software change that considered the factors involved and the integrated quantitative effects of change on factors related to key project indicators, such as schedule, quality, and cost.

B. SIMULATION OF SOFTWARE PROCESS IMPROVEMENT

Although software process improvement is an intensively researched topic in software engineering, not much support is available to decide what improvement strategies is worthwhile adopting, and what are not. Full-fledged empirical investigations and many experiments are required to find the best management and improvement strategies for a specific context. Because the costs for such studies are high, the idea of building simulation models representing the essential process activities may be a good compromise.

An early study of process improvement dynamics in a high-maturity organization was commissioned by the SEI and conducted by Steve Burke [40]. His model was used for determining SPI suggestions and reaching higher CMM level. Almost at the same time, Madachy and Khoshnevis [41]. Christie [42], Pfahl and Lebsanft [26] reviewed how software process simulation can be used to support software process improvement. Madachy and Khoshnevis [41] developed a dynamic simulation model of an inspection-based software process to support quantitative process evaluation and aid in process improvement. Christie [42] proposed how to apply process simulation at all levels of the CMM (capability maturity model). Pfahl and Lebsanft [26] used a fictitious example to show that SD model PSIM can be used to evaluate candidate process improvement strategies. In addition, they proposed to combine SD modeling with measurement-based quantitative modeling. This integration was good at complementing each other and was adopted in our research. Powell *et al.* [43] used SD modeling to evaluate strategies for lifecycle concurrency and iteration. The technique of SD was proven to be useful in understanding incremental process dynamics, especially when the effectiveness of alternative strategies is dependent on the process context [43]. Ruiz *et al.* [44] used SD to model and simulate COTS (commercial-off-the-shelf)-based software development process and process improvement. Their work only included the first results and without obtaining real data to validate. By means of SimSE environment [45], the simulation in Hsueh *et al.*'s work [46] provided an opportunity for organization to improve process and reduce many efforts before pilot project. Ali *et al.* [47] proposed a framework to perform simulation-assisted value stream mapping and illustrated which wastes and process improvements could be identified. The results showed that the use of simulation was particularly helpful in having more insightful discussions and led to realistic improvements with a high likelihood of implementation.

These existing simulation models discussed in the literature did not include change management processes considered in concert with the rest of the lifecycle or other critical project factors. Also, most of them used model-based SPI approaches which focused on the maturity of an organization. We used continuous SPI approaches to involve improvement cycles based on an initial baseline.

V. CONCLUSION

Software companies are facing with constant changes and increased demands for improvements to cycle time, development cost, and quality. Software process simulation is emerging as an effective tool to help evaluate and manages changes made to software projects and development organizations. Due to the fact that process models glue together all activities, products, and resources, the relevance of process models for the success of software projects is enormous [2]. System dynamics simulation is useful in assessing the impact of these elements in software processes. In this paper, our simulation model can represent the complexity of relationships between large quantities of interrelated factors and effectively illustrates the effects of changes. The model audience is software project managers and researchers in software project teams who are seeking to understand the effects and management of changes. The simulation model can be use to study the effects of changes on various process quality attributes, such as effort, delivery time, productivity, schedule, and product quality. Moreover, by adding process improvement strategies, the simulation results show that the use of the model can provide qualitative and quantitative suggestions on how to improve the software process or on how to satisfy specific organizational needs.

ACKNOWLEDGMENT

"Xuan Zhang thanks the researchers and developers in Taili Co., Ltd for their contribution in providing their valuable industrial advices and suggestions during the modeling and empirical case study."

REFERENCES

[1] T. Javed, M. Maqsood, and Q. S. Durrani, "A study to investigate the impact of requirements instability on software defects," *ACM SIGSOFT Softw. Eng. Notes*, vol. 29, no. 3, pp. 1–7, 2004.
[2] J. Münch, O. Armbrust, M. Kowalczyk, and M. Soto, *Software Process Definition and Management*. Berlin, Germany: Springer-Verlag, 2012.
[3] R. J. Madachy, *Software Process Dynamics*. Hoboken, NJ, USA: Wiley, 2008.
[4] A. Fuggetta, "Open source software—An evaluation," *J. Syst. Softw.*, vol. 66, no. 1, pp. 77–90, 2003.
[5] W. S. Humphrey, *Managing the Software Process*. White Plains, NY, USA: Longman, 1989.
[6] S. Ferreira, J. Collofello, D. Shunk, and G. Mackulak, "Understanding the effects of requirements volatility in software engineering by using analytical modeling and software process simulation," *J. Syst. Softw.*, vol. 82, no. 10, pp. 1568–1577, 2009.
[7] V. R. Basili, G. Caldiera, and H. D. Rombach, *Goal Question Metric Paradigm*. New York, NY, USA: Wiley, 1994.
[8] N. B. Ail, K. Petersen, and C. Wohlin, "A systematic literature review on the industrial use of software process simulation," *J. Syst. Softw.*, vol. 97, pp. 65–85, Nov. 2014.
[9] M. I. Kellner, R. J. Madachy, and D. M. Raffo, "Software process simulation modeling: Why? What? How?" *J. Syst. Softw.*, vol. 46, nos. 2–3, pp. 91–105, 1999.
[10] T. F. Bissyandé, D. Lo, L. Jiang, L. Réveillère, J. Klein, and Y. Le Traon, "Got issues? Who cares about it? A large scale investigation of issue trackers from GitHub," in *Proc. IEEE 24th Int. Symp. Softw. Rel. Eng. (ISSRE)*, Pasadena, CA, USA, Nov. 2013, pp. 188–197.
[11] K. Nakakoji, Y. Yamamoto, Y. Nishinaka, K. Kishida, and Y. Ye, "Evolution patterns of open-source software systems and communities," in *Proc. Int. Workshop Princ. Softw. Evol.*, vol. 5, 2002, pp. 76–85.
[12] M. Aberdour, "Achieving quality in open-source software," *IEEE Softw.*, vol. 24, no. 1, pp. 58–64, Jan./Feb. 2007.

[13] A. Mockus, R. T. Fielding, and J. Herbsleb, "A case study of open source software development: The Apache server," in *Proc. Int. Conf. Softw. Eng. (ICSE)*, Limerick, Ireland, vol. 6, 2000, pp. 263–272.

[14] E. S. Raymond, *The Cathedral & The Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. Newton, MA, USA: O'Reilly Media, 1999.

[15] M. Aberdour, "Achieving quality in open-source software," *IEEE Softw.*, vol. 24, no. 1, pp. 58–64, Jan./Feb. 2007.

[16] T. Bhowmik, N. Niu, P. Singhania, and W. Wang, "On the role of structural holes in requirements identification: An exploratory study on open-source software development," *ACM Trans. Manage. Inf. Syst.*, vol. 6, no. 3, 2015. Art. no. 10.

[17] T. Britton, L. Jeng, G. Carver, P. Cheak, and T. Katzenellenbogen, "Reversible debugging software-quantify the time and cost saved using reversible debuggers," Ph.D. dissertation, Judge Business School, Univ. Cambridge, Cambridge, U.K., 2013.

[18] V. Garousi and D. Pfahl, "When to automate software testing? A decision-support approach based on process simulation," *J. Softw., Evol. Process*, vol. 28, no. 4, pp. 272–285, 2016.

[19] C. Jensen and W. Scacchi, "Process modeling across the Web information infrastructure," *Softw. Process Improvement Pract.*, vol. 10, no. 3, pp. 255–272, 2005.

[20] L. Cao, B. Ramesh, and T. Abdel-Hamid, "Modeling dynamics in agile software development," *ACM Trans. Manage., Inf. Syst.*, vol. 1, no. 1, Dec. 2010, Art. no. 5.

[21] J. Sterman, *Business Dynamics: Systems Thinking and Modeling for a Complex World*. New York, NY, USA: McGraw-Hill, 2000.

[22] J. W. Forrester, *Industrial Dynamics*. Cambridge, MA, USA: Productivity Press, 1961.

[23] J. Dinkelacker, "Progressive open source," in *Proc. 24th Int. Conf. Softw. Eng. (ICSE)*, vol. 5, 2002, pp. 177–184.

[24] W. Ke and P. Zhang, "The effects of extrinsic motivations and satisfaction in open source software development," *J. Assoc. Inf. Syst.*, vol. 11, no. 12, pp. 784–808, 2010.

[25] A. M. Magdaleno, C. M. L. Werner, and R. M. De Araujo, "Reconciling software development models: A quasi-systematic review," *J. Syst. Softw.*, vol. 85, no. 2, pp. 351–369, 2012.

[26] D. Pfahl and K. Lebsanft, "Integration of system dynamics modelling with descriptive process modelling and goal-oriented measurement," *J. Syst. Softw.*, vol. 46, nos. 2–3, pp. 135–150, 1999.

[27] M. I. Kellner and G. A. Hansen, "Software process modeling: A case study," in *Proc. 22nd Annu. Hawaii Int. Conf. Syst. Sci.*, vol. 2, 1989, pp. 175–188.

[28] C. Y. Lin and R. R. Levary, "Computer-aided software development process design," *IEEE Trans. Softw. Eng.*, vol. 15, no. 9, pp. 1025–1037, Sep. 1989.

[29] T. Abdel-Hamid and S. Madnick, *Software Project Dynamics: An integrated Approach*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1991.

[30] D. Pfahl and K. Lebsanft, "Using simulation to analyse the impact of software requirement volatility on project performance," *Inf. Softw. Technol.*, vol. 42, no. 14, pp. 1001–1008, 2000.

[31] D. X. Houston, G. T. Mackulak, and J. S. Collofello, "Stochastic simulation of risk factor potential effect for software development risk management," *J. Syst. Softw.*, vol. 59, no. 3, pp. 247–257, 2001.

[32] P. Donzelli and G. Iazeolla, "Hybrid simulation modelling of the software process," *J. Syst. Softw.*, vol. 59, no. 3, pp. 227–235, 2001.

[33] R. Martin and D. Raffo, "Application of a hybrid process simulation model to a software development project," *J. Syst. Softw.*, vol. 59, no. 3, pp. 237–246, 2001.

[34] S. Koch, "Agile principles and open source software development: A theoretical and empirical discussion," in *Proc. Int. Conf. Extreme Program. Agile Processes Softw. Eng.*, Calgary, AB, Canada, in Lecture Notes in Computer Science, vol. 3092. Berlin, Germany: Springer-Verlag, 2004, pp. 85–93.

[35] M. M. Lehman and J. F. Ramil, "The impact of feedback in the global software process," *J. Syst. Softw.*, vol. 46, nos. 2–3, pp. 123–134, 1999.

[36] P. Wernick and M. M. Lehman, "Software process white box modelling for FEAST/1," *J. Syst. Softw.*, vol. 46, nos. 2–3, pp. 193–201, 1999.

[37] P. Wernick, T. Hall, and C. L. Nehaniv, "Software evolutionary dynamics modeled as the activity of an actor-network," *IET Softw.*, vol. 2, no. 4, pp. 321–336, 2008.

[38] G. Kahen, M. M. Lehman, J. F. Ramil, and P. Wernick, "System dynamics modelling of software evolution processes for policy investigation: Approach and example," *J. Syst. Softw.*, vol. 59, no. 3, pp. 271–281, 2001.

[39] S. M. Ali, M. Doolan, P. Wernick, and E. Wakelam, "Developing an agent-based simulation model of software evolution," *Inf. Softw. Technol.*, vol. 96, pp. 126–140, Apr. 2018.

[40] S. Burke, "Radical improvements require radical actions: Simulating a high-maturity software organization," Softw. Eng. Inst., Pittsburgh, PA, USA, Tech. Rep. CMU/SEI-96-TR-024, 1996.

[41] R. Madachy and B. Khoshnevis, "Dynamic simulation modeling of an inspection-based software lifecycle process," *Simulation*, vol. 69, no. 7, pp. 35–47, 1997.

[42] A. M. Christie, "Simulation in support of CMM-based process improvement," *J. Syst. Softw.*, vol. 46, nos. 2–3, pp. 107–112, 1999.

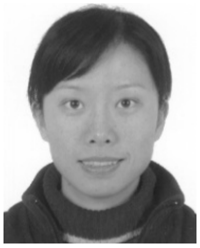
[43] A. Powell, K. Mander, and D. Brown, "Strategies for lifecycle concurrency and iteration—A system dynamics approach," *J. Syst. Softw.*, vol. 46, nos. 2–3, pp. 151–161, 1999.

[44] M. Ruiz, I. Ramos, and M. Toro, "Using dynamic modeling and simulation to improve the COTS software process," in *Proc. 5th Int. Conf. Product Focused Softw. Process Improvement (PROFES)* in Lecture Notes in Computer Science, vol. 3009, F. Bomarius and H. Iida, Eds. Berlin, Germany: Kausai Science, 2004, pp. 568–581.

[45] E. O. Navarro and A. van der Hoek, "Software process modeling for an educational software engineering simulation game," *Softw. Process, Improvement Pract.*, vol. 10, no. 3, pp. 311–325, 2005.

[46] N. L. Hsueh, W.-H. Shen, Z.-W. Yang, and D.-L. Yang, "Applying UML and software simulation for process definition, verification, and validation," *Inf. Softw. Technol.*, vol. 50, nos. 9–10, pp. 897–911, 2008.

[47] N. B. Ali, K. Petersen, and B. B. N. de França, "Evaluation of simulation-assisted value stream mapping for software product development: Two industrial cases," *Inf. Softw. Technol.*, vol. 68, pp. 45–61, Dec. 2015.



XUAN ZHANG was born in Kunming, Yunnan, China, in 1978. She received the B.S. and M.S. degrees in computer science and the Ph.D. degree in system analysis and integration from Yunnan University, Yunnan, in 2000, 2003, and 2014, respectively.

From 2003 to 2012, she was an Assistant Professor with the School of Software, Yunnan University. Since 2012, she has been an Associate Professor with the School of Software, Yunnan University. She has authored two books and more than 60 articles. She has been a Principal Investigator for more than 10 national, provincial, and private grants and contracts. Her research interests include software process, trustworthy software, and requirement engineering.

XU WANG was born in Kunming, Yunnan, China, in 1976. He received the B.S., M.S., and Ph.D. degrees in economics from Yunnan University, Yunnan, in 1998, 2006, and 2010, respectively.

From 2010 to 2016, he was an Assistant Professor with the School of Economics, Yunnan University, Yunnan, China. Since 2016, he has been an Associate Professor with the School of Economics, Yunnan University, Yunnan, China. He has authored three books and more than 50 articles. His research interests include business process management, monetary economics and banking, and financial security.

YANNI KANG was born in Kunming, Yunnan, China, in 1992. She received the B.S and M.S. degrees in information security from Yunnan University, Yunnan, in 2014 and 2018, respectively.

She has authored three articles. Her research interests include software process and requirement engineering.

• • •