Assessing the Risk of Software Development in Agile Methodologies Using Simulation

MARIA ILARIA LUNESU®, ROBERTO TONELLI®, (Member, IEEE), LODOVICA MARCHESI®, (Member, IEEE), AND MICHELE MARCHESI®, (Senior Member, IEEE)

Department of Mathematics and Computer Science, University of Cagliari, 09123 Cagliari, Italy

Corresponding author: Maria Ilaria Lunesu (ilaria.lunesu@unica.it)

This work was supported in part by the Project: “Gestione documentale” - Programmazione Unitaria 2007–2013 Interventi a sostegno della competitività e dell’innovazione, ai sensi della D.G.R. n. 33/41 dell’8.08.2013 - Bando Pacchetti Integrati di Agevolazione “Industria, Artigianato e Servizi” (annualità 2013), in attuazione della D.G.R. n. 46/20 del 31.10.2013 Prot. 283 CUP. E2 6G14000820007 and in part by the Project: ABATA - Applicazioni della Blockchain per l’Autenticità e la tracciabilità Alimentare n. F/200130/02/X45 CUP B21B20000040005 Ministero dello Sviluppo Economico, nell’ambito del Programma Operativo Nazionale “Imprese e Competitività” 2014–2020 - Asse 1 - Innovazione, azione 1.1.3 FESR Fondo per la Crescita Sostenibile - Sportello “Agrifood” D.M. 05/03/2018 Capo III e D.D. 20/11/2018.

ABSTRACT Agile methodologies aim to reduce software development risk using short iterations, feature-driven development, continuous integration, testing automation, and other practices. However, the risk of project failure or time and budget overruns is still a relevant problem. This paper aims to present and discuss a new approach to model some key risk factors in agile development, using software process simulation modeling (SPSM), which can complement other approaches, and whose usage is particularly suited for agile development. We introduce a new approach to modeling some key risk factors – namely project duration, number of implemented issues, and key statistics of issue completion time – using a simulator of agile development, which we developed for this purpose. The approach includes modeling the agile process, gathering data from the tool used for project management, and performing Monte Carlo simulations of the process, to get insights about the expected time and effort to complete the project, and about their distributions. The model’s parameters that can cause risk are errors in effort estimation of the features to develop, variations in developers’ assignment to these features, impediments related to developers’ availability and work completion. To validate the simulator, and to demonstrate how the method can be used, we analyzed three open-source projects, gathering their data from JIRA repositories. We ran Monte Carlo simulations of these projects, showing that the simulator can well approximate the progress of the real project, then varying the identified risk factors and statistically evaluating their effects on the risk parameters. The proposed approach is relevant for project managers, being able to quantitatively evaluate the risks, provided that the process and the project’s data are properly modeled and gathered. As for the future, we are working to improve our risk assessment method, evaluating it on more case studies, scaling the model from a single team to multiple teams involved in one or more projects.

INDEX TERMS Effort estimation error, random issue allocation, risk analysis, software process simulation, Agile, simulation, Monte Carlo methods, software development management.

I. INTRODUCTION

Software technical risk is hard to define and there is no agreement on a common definition. Risk is somehow defined as a measure of the probability and severity of adverse effects [29], inherent in the development of software that does not meet its intended functions and performance requirements [14].

The associate editor coordinating the review of this manuscript and approving it for publication was Jesus Felez.

The Software Engineering Institute (SEI) defines risk as the possibility of suffering loss [5]. In such context, the loss may describe any impact to the project, which could be in the form of diminished quality of the end product, increased costs, delayed completion, loss of market share, failure, and so on.

A broader definition is given by PMBoK (Project Management Body of Knowledge book). Here, risk is defined as an “event or an uncertain condition that, if it occurs, can result in positive (opportunities) or negative impacts (threats) in
one or more project objectives, such as scope, time, cost, and quality” [24].

Wallace et al. [50] identified and studied six dimensions of software project risk, i.e. risks related to: Organization environment, User, Requirements, Project complexity, Planning and control, Team.

Traditional software development methodologies deal with risk by performing a detailed up-front analysis to lower risks related to poor requirements specification and enforcing strict control planning, with frequent risk re-evaluations. However, they tend to integrate the various software modules late in the project, leaving room for integration risks.

A newer approach is adopted by Agile Methodologies, which were introduced in the late 90’s precisely to lower risks due to changing requirements — a characteristic common to most Internet-age software projects — but related also to other issues, such as user involvement, team cooperation, late integration, insufficient feedback [4], [33], [41], [23].

A. PROBLEM STATEMENT

Software risk management is crucial for successful project management, but it is often an aspect not properly taken into account in real projects. One of the main reasons is that project managers often do not have effective tools and practices for software risk management.

In their work [12] Chadli and collaborators presented a research focused on discovering and classifying the various tools mentioned in the literature which support Global software development (GSD) project managers, and on identifying in which way they support group interaction. In this paper, the authors state that “Decision management, risk management, and measurement processes are not adequately supported by tools when compared to the other Software Process Management processes”.

Only recently, a few frameworks and tools have been introduced for better addressing risk management in Agile Software Development (ASD). The results of a survey conducted using a qualitative approach to analyze how risk management is carried out in Scrum software projects were presented in [43]. De Souza Lopes et al. in [17] established a framework called RIsk Management PRoduct Owner (RIMPRO), which intends to support project teams to systematically manage risks related to PO activities that may arise during the project definition proposed by PMBoK.

Risk is an “event or an uncertain condition that, if it occurs, can result in positive (opportunities) or negative impacts (threats) in one or more project objectives, such as scope, time, cost, and quality” (PMI, 2017), in [6] intelligent agents are used to manage risk. It does not simulate the process but only the stochastic impact of the variation [32], [38].

In such a scenario, Software Process Simulation Modeling (SPSM) has emerged as a promising approach to address a variety of issues related to software engineering, including risk management [35], [20]. Clearly, SPSM cannot address risk dimensions such as organization environment stability and management support, lack of user involvement, team motivation, and communication issues. SPSM can be useful to establish the impact of risks on specific topics, such as requirement estimation errors, rework due to software that does not meet its intended functions and performance requirements (software technical risk), project complexity, planning and control practices [49].

However, how and to which extent SPSM can support software risk management is a topic that still needs to be better clarified, especially in the case of ASD.

The goal of this work is to present a viable approach to risk management using SPSM when using an agile development process, that is a process based on implementing requirements or change requests specified as atomic “features” or “issues”. In terms of research questions, this goal can be specified as:

RQ1: To what extent is it possible to automatically import into the simulator data of real projects, from issue management systems?

RQ2: How accurate can the simulator be in predicting project completion times?

RQ3: Can the simulator be useful to estimate project risk (induced by errors in efforts estimation, and random developer issues assignment) with a Monte Carlo approach?

B. CONTRIBUTION

In this paper, we present a system to perform risk assessment of ASD projects using SPSM and a Monte Carlo stochastic approach. Regarding Wallace’s risk dimensions [50], we mainly consider requirement correctness and estimation. Our approach can improve the planning and control of the project, and the management of its complexity. Also, team factors can be taken into account, by explicitly modeling developers’ skills, impediments, and turnover. The dimensions of the organization environment and of user are out of the scope of the proposed approach.

The basic tool of our system is an event-based simulator, able to simulate ASD based on the implementation of User Stories (USs), or features, kept as independent as possible from one another. The agile process may be iterative, as in Scrum, or using a continuous flow approach, as in the Lean-Kanban approach.

The basic inputs to the simulator are the team composition and skills, and the USs themselves. These inputs can be fed to the simulator from the Issue Management tools. We have so far implemented the interface to the popular JIRA tool [1]. Other inputs are parameters driving the random variation of teamwork, and of USs estimation and actual cost, as well as of possible events able to influence the project.

The outputs of the simulation are key risk parameters, such as the distribution of times to complete the project or the iteration, and of the forecast project cost. The simulations are performed hundreds, or thousands, of times, properly varying the inputs, to get the distribution of the required outputs. The resulting distributions can provide hints on the expected outputs and of their variances, including proper risk
percentiles, useful to monitor development and to control risk. In fact, if risk assessment is not done in the initial stage, effort estimation will be strongly affected and the overall project may risk failure [28].

The main contributions of our work are:

- The development of a flexible and extensible ASD simulator, able to simulate virtually every Agile development process, based on incremental development. This simulator models most aspects of ASD – team members, USs, activities, events – and can be easily extended and customized, due to the full object-oriented approach used for its development.
- The development of a risk management model, considering team factors, requirement correctness, and effort estimation. The input data, parameters, and events, as well as the relevant output distributions, are analyzed and discussed.
- The ability to feed project data from popular issue management tools, namely JIRA, with the ability to extend the simulator taking into account also other tools.
- A risk analysis management performed on 3 medium-sized Open Source projects, comparing the simulation results of different choices of the input parameters. The results highlight both the prediction accuracy of our tool and how it can be used to manage risk.

C. OUTLINE

The remainder of the paper is structured as follows. Section II reports the main related works on software risk management and on the application of SPSM to it. Section III presents the proposed risk-assessment method. Section IV describes the simulation model. Section V presents how the risk-assessment method is applied in this paper. Section VI presents the case studies. Section VII presents the results of the case studies, and how they can be generalized and applied to other projects. Section VIII presents the threats to validity. Section IX eventually concludes the paper and discusses future work.

II. RELATED WORK

1) RISK MANAGEMENT IN SOFTWARE PROJECTS

It is well known that several software projects suffer from various kinds of problems, such as cost overruns, missing delivery deadlines, and poor product quality. One of the factors that cause these problems is the fact that the risks are not handled, as shown by Charette [13].

Risk management in software projects is a key component of the success of a project. Software engineering researchers and professionals have proposed several systematic approaches and techniques for effective risk management as reported by Boehm [11].

A study conducted by the Project Management Institute has shown that risk management is an activity not practiced among all the disciplines of project management in the IT industry [18]. In real software projects, risks are often managed using the insights of the project manager, and the entire process of risk management is rarely followed [34]. One of the main reasons for this is that project managers lack practical techniques and tools to effectively manage the risks.

The paper by Barros et al. [16] presents an approach to develop, retrieve, and reuse management knowledge and experience concerned with software development risks, using scenarios to model risk impact and resolution strategies efficacy.

Shahzad and Al-Mudimigh [40] propose a model that takes care of the most frequently occurring risks and provides a way for handling all of them. The sequence of activities for mitigating all risk factors is presented in a comprehensive model, which is useful for improving management and avoidance of software risk.

Xiaosong et al. [52] use a classical matrix approach, defining 82 software project risk factors, aggregated in 14 kinds of risk, each one with an impact level on the overall project, from Negligible to Critical. Once experts evaluate each factor, the risks are evaluated using the risk matrix. Thereafter, high-level risks should be immediately addressed, whereas medium and low risks will be monitored and kept under control.

Roy et al. identified key risk factors and risk types for each of the development phases of SDLC (Software Development Life Cycle) [37], including services and maintenance of software products.

Thieme et al. in their paper [45] present a study about the process for the analysis of functional software failures, their propagation, and incorporation of the results in traditional risk analysis methods (fault trees and event trees). A functional view on software is taken by allowing integration of software failure modes into risk analysis of the events and effects. The proposed process can be applied during system development and operation to analyze the risk level and identify measures for system improvement.

2) RISK MANAGEMENT IN ASD

Recently, agile software development methods have been introduced, to be able to manage requirement changes throughout project development. ASD might be considered also as a risk management approach because errors and changes in requirements are one of the main risk factors in software development. Among specific studies and approaches to manage project risk in ASD processes we already quoted the RIMPRO framework by Lopes et al. [17]. We also recall the work of Tavares et al. [44] who propose Rm4Am, a tool that identified 5 components and 48 sub-components important in ASD, and conducted an experiment with the supervision and participation of Agile expert.

3) SIMULATION OF SOFTWARE PROCESS

Software Process Simulation Modeling (SPSM) is presented as a promising approach suitable to address various kinds of issues in software engineering [26]. The results of the review conducted by Zhang et al. [53] showed that risk management
is one of the several purposes for SPSM. Liu et al. performed a systematic review on this topic, concluding that the number of SPSM studies on software risk management has been gradually increasing in recent years and that discrete-event simulation and system dynamics are the two most popular simulation paradigms, while hybrid simulation methods are more and more widely used [27].

Practical examples of the use of SPSM are the works by Baum et al. [10], who compares pre-commit reviews and post-commit reviews using process simulation through a parametric discrete event simulation model of a given development process, and by Zhao et al. [54] who present a fine-grained dynamic micro-simulation system based on an agent model at the project level for Open Source Software Development.

Discrete-event simulation of agile development practices was first introduced by Melis et al. [30]. Anderson et al. [7] proposed an event-driven, agent-based simulation model for the Lean-Kanban process, extensible to other Agile software processes, and used it to demonstrate the effectiveness of a WIP-limited approach and to optimize the WIP limits in the various process activities. In a subsequent work, Anderson et al. [8] used an extended version of their simulator to compare Lean-Kanban with traditional and Scrum approaches on the data collected from a Microsoft maintenance project, showing that the Lean-Kanban approach is superior to the others.

Turner et al. worked on modeling and simulation of Kanban processes in Systems Engineering [48], [47]. These two works, though quite preliminary, propose the use of a mixed approach, merging Discrete Event and Agent-based simulation approaches. In particular, Discrete Event simulation is used to simulate the flow of high-level tasks and the accumulation of value, whereas Agent-based simulation is used to model the workflow at a lower level, including working teams, Kanban boards, work items, and activities.

Wang [51] introduces an agent-based simulation tool to compare solo and pair-programming, and other agile practices, in the context of Scrum development. The simulation tool can simulate all types of Scrum context and team composition to test designed strategies under various what-if assumptions in agent-based modeling.

4) AUTOMATED APPROACHES FOR SOFTWARE RISK MANAGEMENT

Another field of interest related to software risk management is the application of Machine Learning, and similar automated techniques.

In this field, Fenton et al. [19] developed a complex causal model based on Bayesian networks for predicting the number of residual defects of a system. Their approach accounts for various phases of software development, among which requirement analysis, design, and programming, testing, and rework. Each phase is modeled by a Bayesian net, whose probability values and functions are set by project managers. The model allows managers to perform various types of what-if analyses and trade-offs, focusing on minimizing the number of defects, which is a key factor in risk management.

Rodríguez et al. [36] present a model based on the A.I. technique called “Rough Set” for selecting and prioritizing, in environments of uncertainty, a set of critical threats in a software development project, to minimize risks. Their model, named “Apollonian”, generated 20 rules that work as a reference for any software development project, to assess the main threats to the project.

Joseph [25] proposes a machine-learning algorithm to generate risk prompts, based on software project characteristics and other factors. His approach uses multiple multilabel artificial neural networks to label software projects according to the risks to which they are most exposed.

Han [22] trained a multi-layer-perceptron to assess the risk level of a software project, using as a learning set the OMRON dataset of 40 projects, each described by 24 risk-related parameters. The results look better than a more traditional logistic regression.

Min et al. [31] applied fuzzy comprehensive evaluation to estimate a project’s risk level. The approach is somewhat similar to, though much simpler than, that of ref. Fenton, but instead of Bayesian nets fuzzy algebra is used to find the probability of risks, given estimated risk factors.

Alencar et al. [6] propose a proactive and automated approach based on agent technology to assist the software project manager in the execution of the Risk Management processes.

Abioye et al. [3] present a lifecycle approach to ontology-based risk management framework for software projects using a dataset gathered from literature, domain experts, and practitioners. The risks are conceptualized, modeled, and developed using Protégé. The framework was adopted in real-life software projects.

Asif et al. [9] identify the relationship between risk factors and mitigation actions automatically by using an intelligent Decision Support System. The DSS is rule-based and identifies the rules using the Equivalence Class Clustering and bottom-up Lattice Traversal (ECLAT) algorithm, starting from expert knowledge and literature. 26 highly cited risk factors and 57 risk mitigations were identified from the literature and associated through the rules of the DSS, to help project managers to mitigate the risks.

5) SPSM FOR RISK MANAGEMENT

The literature about SPSM applied to risk management is still quite limited. Ghane [21] observes that ASD delivers workable software in short cycles, and this helps with collecting more heuristic data as compared to traditional waterfall methodologies. Such data can be used as quantitative metrics for time and effort estimation. He then introduces a risk management model that uses project simulation to produce risk metrics used to help with risk avoidance and mitigation. These metrics are used to adjust project factors such as time, cost, and scope during the lifespan of a project.
Singh et al. [42] represent the PERT graph of a software project, where nodes are the states, and arcs represent activities. Each arc bears the mean and standard deviation of its cost. A Monte Carlo simulator stimulates the model with thousands of executions, to find the cost distribution and the critical paths.

III. RISK ASSESSMENT THROUGH SIMULATION

A. THE RISK-ASSESSMENT METHODOLOGY

Our starting points in Risk assessment are the Six dimensions of risk, as defined by Wallace et al. [50]. They are:

1) Organizational Environment Risk, including change in organizational management during the project, corporate politics with a negative effect on the project, unstable organizational environment, organization undergoing restructuring during the project.

2) User Risk, including users resistant to change, the conflict between users, users with negative attitudes toward the project, users not committed to the project, lack of cooperation from users.

3) Requirements Risk, that is continually changing system requirements, system requirements not adequately identified or incorrect, system requirements not properly defined or understood.

4) Project Complexity Risk, encompassing a high level of technical complexity, the use of new or immature technology.

5) Planning & Control Risk, including the setting of unrealistic schedules and budget, lack of an effective project management methodology, project progress not monitored closely enough, inadequate estimation of required resources, project milestones not clearly defined, inexperienced project manager.

6) Team Risk, including inadequately trained and/or inexperienced team members, team member turnover, ineffective team communication.

We recall that risk management involves the following three steps:

1) Risk identification: where possible risks related to a project are enumerated and discussed, typically preempting what might go wrong in a proactive way.

2) Risk analysis: where identified risks are quantitatively and qualitatively evaluated, to ascertain the probability and critical level of their impact.

3) Risk mitigation: actions to both lower the probability that the adverse event occurs, and reduce its impact on the project before it happens.

SPSM can be applied to risk analysis, greatly helping the quantitative assessment of some risk dimensions. Our approach is intended to work together with other risk management frameworks, and not as a stand-alone method.

The use of the SPSM approach addresses mainly dimensions 3, 4, and 5, and specifically inadequate estimations of requirements, project complexity in terms of number and complexity of requirements (including sequence constraints), and poor quality of software artifacts, due again to requirements not properly understood, or to issues in project management and planning. Team risk can also be modeled by setting proper developers’ skills, by representing developers’ turnover, task switching, and absences due to various causes.

The Risk-assessment methodology we propose is performed in subsequent steps:

1) The development (or maintenance) process is modeled (activities, team, process, issues, constraints), and the simulator is configured to simulate it.

2) Key quantitative risk factors are identified; in our case, they are estimation errors in efforts to complete features or resolve issues, percentage of rework, variations in the skills of team members, probability of events that stall the development of single features, or block one or more developers, and so on.

3) Probability distributions are given to these risk factors, for instance, the probability distribution of actual effort needed to fix an issue, or the probability that a developer is blocked, together with the probability distribution of the time length of this block.

4) Key process outputs are identified, such as project total time, throughput, average and 95% percentile of lead and cycle times to resolve issues.

5) Hundreds or thousands of Monte Carlo simulations of the project are made varying the risk factors accordingly to their probability distributions, and recording the process outputs.

6) The resulting distributions are analyzed, assessing for instance the most likely duration and cost of the project, the average time – or the distribution of times – to implement an issue or to fix a bug, the probability that a given percentage of issues is implemented within a given time.

This Monte Carlo assessment can be performed also on an ongoing project, by simulating the continuous flow of new requirements or maintenance requests, or just the remaining features to be implemented.

In creating the proposed approach, we were inspired by the ever increasing use of Issue Tracking Systems (ITS) that allow developers to easily gather all data related to change requests and defect correction, as well as to schedule and track the project flow. We started by creating a connection with the JIRA system [1], one of the most popular ITS worldwide, throughout REST calls, using JIRA APIs. Once the connection is established, it is possible to download project data and all related information in a file in JSON or CSV formats. In particular, the simulator can collect detailed data related to developers, issues, process activities, and others, as shown in Fig. 1.

FIGURE 1. Interaction between JIRA issue tracking system and the simulator.
IV. THE SIMULATION MODEL
Our SPSM model uses an approach that is both event-driven and agent-based. The operations of the system are represented by a sequence of events in chronological order. Each event occurs at an instant of the simulation time and causes a change in the state of the system. The simulator is also based on agents (the team members), who exhibit autonomous behavior. The agent’s actions are not fixed but depend on the simulated environment.

A. BASIC COMPONENTS
The basic components of the SPSM model are the issues, the activities, the team members, and the events:

- **Issues** are the atomic work items in a development or maintenance project. They correspond to the features described in the Lean-Kanban approach and are similar to Scrum USs. Each unit of work is characterized by a unique identifier, a collection of effort values (in man-days) expressing the amount of work needed in each activity to complete the issue, a priority (an integer in a given range, expressing the importance of the issue), and information of the actual effort spent in each activity. When the last activity is completed, the issue becomes closed.

- **Activities** represent the kinds of work that has to be done on the issues to complete them. Typically, they are Planning, Analysis, Coding, and Testing, but they can be configured on the specific development process chosen by the team.

- **Team members** (Developers) hold the information on the actual developers, which includes the skills in the various activities. If the skill is equal to one, it means that the team member will perform work in that activity according to the declared effort. For instance, if the effort is one man-day, the member will complete that effort in one man-day. If the skill is lower than one, the actual developer, which includes the skills in the activity to complete the issue, a priority (an integer in a given range, expressing the importance of the issue), and information of the actual effort spent in each activity. When the last activity is completed, the issue becomes closed.

- **Events** represent what happens at a given time, which is relevant to the development. We manage three kinds of events: i) the creation of an issue; ii) the start of work on an issue in a given activity; iii) the end of the work on the issue in a given activity. Each event has a time and is executed by the simulator when its time arrives.

B. THE SIMULATION PROCESS
The modeled development process proceeds through a sequence of steps. Preliminarily, one has to define the ASD process, that is what are the activities of the process, their sequence, and their average relative weight on the total effort to complete an issue. Secondly, the development team members must be created, each having different skills in the various activities.

Then, the simulator is started, executing steps with the following characteristics:

1) The simulation starts at time zero: \( t = 0 \). Time \( t \) is expressed in working days. Each day involves 8 hours of work. At the beginning, the system may already hold an initial backlog of issues to be worked out.

2) The simulation proceeds by time steps of one day, until a predefined end, or until the event queue is empty.

3) Issues are entered at given days, drawn from a random distribution, or given as input to the system. Each issue is endowed with a collection of effort values for each activity (in days of work), to be performed to complete the issue. These values can be drawn from a distribution, or obtained from real data.

4) Each issue passes through the activities. Each activity takes a percentage of the whole effort to process the issue. The sum of the percentages of all the activities must be 100%. When an issue enters an activity, the actual effort (in man-days) needed to complete the activity is equal to the total effort of the issue multiplied by the proper percentage.

5) At the beginning of each day, the developers choose the issue (and the activity) they will work on in the day. This choice is driven by the developer’s skills, by the issue priority, and by the preference to keep working on the same issue of the preceding day, if possible. Whenever an issue is processed for the first time, or in the case of developer switching from another issue, a multiplicative penalty factor \( p \), with \( p \geq 1 \), is applied to compute the time effort, to model time waste due to task switching (extra time needed to study the issue, which is proportional to the size of the task). The task switching problem is well known in software engineering, as reported also recently by Abad et al. [2].

6) When the work on an issue in a given activity ends, the issue is pulled to the next activity, where it can be chosen by a developer, or is closed in the case of the last activity. The developer who ended the work will choose another issue to work on, which might be the same issue, in the subsequent activity, or not.

The presented simulation model can represent development, or maintenance processes, and can be customized to cater to the specific process of an organization. From this generic model, we can derive various specific models. For instance, one might introduce WIP limits, as suggested by the Lean-Kanban approach. For a WIP-limited process, the model has to be complemented by adding limits to the maximum number of issues that can be processed at any given time inside an activity.

If a Scrum-like development has to be modeled, this can be accomplished by defining the length of the iteration (Sprint) and managing the number and total effort of new issues entering at the beginning of each Sprint.
C. SIMULATOR DESIGN

The simulator is implemented using Smalltalk, a language very suited to event-driven simulation and very flexible to accommodate any kind of changes and upgrades to the model. The simulator design is fully object-oriented, allowing to easily add new features if needed.

The simulation model records all the significant events related to issues, developers, and activities during the simulation, to be able to compute any sort of statistics and to draw various diagrams.

In the simulator, software requirements are decomposed into issues, that can be independently implemented. The implementation is accomplished through a continuous flow across different activities, from Open to Closed. The work is performed by a team of developers, able to work on one or more activities, depending on their background.

The different entities of the simulator are represented in the class diagram shown in Fig. 2. Here you can find the four basic classes outlined in section IV-A, plus other key classes. The Simulator is a singleton (a class with just one instance) managing the simulation through the event queue. A Project defines its activities and holds the pertaining issues. The present version of the simulator allows simulating a single project, with one team working on the issues.

The team is composed of a given number of developers. Each developer $i$ is characterized by a skill array (one skill for each Activity), and a productivity factor at time $t$, $q_i(t)$, obtained as the ratio between the number of closed issues of $i$-th developer at time $t$, $C_i(t)$ and the number of project days elapsed:

$$q_i(t) = \frac{C_i(t)}{t}. \quad (1)$$

Each developer works on an available issue until the end of the day, or until the issue has been completed. When the state of the system changes, for instance, because a new issue has been introduced, an issue is pulled to an activity, the work on an issue ends and, in any case, at the beginning of a new day, the system looks for idle developers and tries to assign them to the issues available to be worked on in the activities they belong to.

As reported in section IV-B, the developer’s productivity may be affected by the penalty factor $p$ used to compute issues time effort in case of developer switching. The penalty factor $p$ is equal to one (no penalty) if the same developer, at the beginning of a day, works on the same issue s/he worked the day before. If the developer starts a new issue or changes issue at the beginning of the day, it is assumed that s/he will have to devote extra time to understand how to work on the issue. In this case, the value of $p$ is greater than one and the required effort to complete the issue is the original effort, multiplied by $p$.

1) ISSUES
The issues that make up a project are categorized into various types: features, tasks, epics, bugs, stories, which are given
The first and the last activities are Backlog and Live. The first and the last activities are Backlog and Live. They are special activities because no work is performed on them – so their effort percentage is zero. The former is a placeholder, containing all issues put into processing, for instance at the beginning of each Sprint in Scrum. The latter is where the completed issues are kept.

When work starts on one issue within a given activity, the actual effort of development of the issue in the activity is computed. This is accomplished by randomly increasing or decreasing it of a percentage drawn from a given distribution. Of course, the average effort of all issues must be equal to the average of their initial estimates. A way to obtain this behavior is to multiply the estimated effort by a random number r drawn from a log-normal distribution with a mean equal to 1 and standard deviation depending on the wished perturbation.

3) EVENTS
The simulator is event-driven, meaning that the simulation proceeds by executing events, according to their time ordering and priority. When an event is executed, the time of the simulation is set to the time of the event. The simulator holds an event queue, where events to be executed are stored sorted by time and priority. When an event is executed, it changes the state of the system, and it can generate new events, with times equal to, or greater than, the current time, inserting them into the event queue. The simulation ends when the event queue is empty, or possibly if a maximum time is reached, marked by an “End of simulation” event. Fig. 4 shows an activity diagram explaining the simulator’s workflow. After an initial configuration, which includes inserting into the event queue the issue creation events, and possibly the end of the simulation event, the simulator main cycle starts and is shown in the diagram.

The time is recorded in nominal working days, from the start of the simulation. A day can be considered to have 8 nominal working hours, but developers can work more or less than 8 hours. If we want to consider calendar days, it is always possible to convert from nominal working days to them. The supported events are described in Table 1.

The mentioned events are enough to manage the whole simulation. The simulation is started by creating the initial issues, putting them in the Backlog activity, and generating a number of IssueToPull events for the first activity equal to the number of issues ready to be pulled in the next activity, and then generating as many IssueCreation events for future times as required. The simulator is then asked to run, using the event queue created in this way. When the work on all issues has been completed in all activities, no more issues can be pulled and no more work can start, so the event queue becomes empty, and the simulation ends.

4) COMPONENT VALIDATION
The proposed SPSM model reflects standard concepts of ASD: features (here called issues), development activities, team members, events (which are instrumental to simulate an iterative, or continuous-flow development). Most of its parameters are directly taken from real project data through

FIGURE 3. The possible states of an issue.
A few parameters, however, must be properly estimated and validated. The penalty factor $p$ described above was set to $p = 1.15$, meaning a 15% increment in the work to be done when a developer changes the issue s/he is working on. This figure is consistent with the data by Tregubov et al. [46] who estimate that “developers who work on 2 or 3 projects spend on average 17% of their effort on context switching between tasks from different projects”. In our case, the switching can be also between different issues of the same project, and the penalty is applied only to the time to work on that issue for the present day.

Another key parameter to be estimated, which may differ from project to project, is the percentage of work to assign to the various activities of a project. Following the same approach reported in [15], we assigned most of the effort (70%) to the “in progress” activity – representing the actual development made on an issue – and a residual effort to testing (15%) and deploying (15%) activities.

This choice was reviewed and approved by six expert project managers (more than 10 years of experience in managing medium-to-large size Java and Python projects) of three firms we work with, and by one expert consultant in Python and Javascript development. Their experience is mainly in Web applications (including apps for mobile devices), ERP platforms, business intelligence applications. Using a Delphi online panel, we got seven answers (with percentages obtained by rounding to five percent), made a refinement round, and adopted the median result.

D. JIRA INTERFACE

The simulator reads data directly from JIRA through its APIs. JIRA REST APIs allow users to use several kinds of REST calls to get information about projects, issues, developers, and more. Our system asks all data about a project with the specific query:

```
--data '{"jql":"project = GROOV", "startAt":0, "maxResults":100}, "http://localhost:8099/rest/api/2/search"
```

JIRA responds by providing a JSON file that includes the following fields: project name, project starting date, project workflows, developers, a backlog of issues, issues arrival dates, issues types, issues estimate, times, issues original times, issues times spent. Then the simulator parses the file and organizes the data to be processed. In particular, it builds the initial queue (backlog) of issues according to their priority. For each issue, the simulator sets all attributes (key, effort spent, original effort estimate, issue type, developer, issue state, issue description, ...), then creates the collection of developers, the collection of activities, and finally sets up the project’s starting date.

V. RESEARCH DESIGN

Our overall research objective is to better understand how to use SPSM to perform risk analysis on ASD projects. The inputs to our model, taken from an ITS such as JIRA and related to an ASD project, are information on team members involved, and information on all issues managed at a given time. The latter kind of information includes the date an issue entered the system, its original effort estimate, the actual time spent on it, and an estimate of the remaining effort. Information on the ASD process used must also be collected from the team. It includes:

- the sequence of activities performed by the team; at least they include: Backlog, In progress, Done;
TABLE 2. The simulator inputs, highlighting the inputs imported from JIRA. "PART." means "partially".

| Class   | Parameter          | Type    | Description                                      | From JIRA | Notes                                      |
|---------|--------------------|---------|--------------------------------------------------|-----------|--------------------------------------------|
| Project | ProjectName        | String  | Name of the project                              | YES       |                                            |
| Project | ProjectStartDate   | Date    | Start date of the project                        | YES       | Corrected to the date of the first completed issue |
| Project | NoOfActivities     | Integer | Number of activities                             | YES       |                                            |
| Activity| ActivityID         | String  | Unique short identifier                         | NO        | Generated by the system                    |
| Activity| ActivityName       | String  | Name of the activity                             | YES       |                                            |
| Activity| ActivityEffort     | Percentage | Effort perc. of the activity                | NO        | Chosen through expert consultation         |
| Team    | NoOfDev            | Integer | No. of developers                               | YES       |                                            |
| Developer| DevID              | String  | Unique short identifier                         | NO        | Generated by the system                    |
| Developer|_devName            | String  | Name of the developer                           | YES       |                                            |
| Developer|Skills              | Float   | Array of the dev. skill for each activity       | PART      | Estimated from JIRA data                  |
| Issue   | IssueID            | String  | Issue description                               | YES       |                                            |
| Issue   | IssueDescr         | String  | Issue description                               | NO        | Not used                                   |
| Issue   | IssueStartDate     | Date    | Start date of issue development                | YES       |                                            |
| Issue   | IssueCreationDate  | Date    | Creation date of the issue                      | YES       |                                            |
| Issue   | IssueType          | String  | Type of the issue                               | YES       |                                            |
| Issue   | IssueState         | String  | State of the issue                              | NO        | Dynamic value                             |
| Issue   | IssuePriority      | String  | Priority of the issue                           | YES       |                                            |
| Issue   | IssueOrigEstimate  | Float   | Original estimated effort                       | YES       |                                            |
| Issue   | IssueEffortSpent   | Float   | Original remaining effort                       | YES       |                                            |
| Issue   | IssueWorkRatio     | Percentage | Actual percentage of work performed             | NO        | Dynamic value                             |
| Issue   | Assignee           | String  | Name of developer in charge                     | YES       | Converted to DevID of the developer        |
| Simulator| PenaltyFactor     | Percentage | Penalty factor p for issue switching         | NO        | See Subsection IV-C                        |
| Simulator|StatDevPentErr      | Float   | Standard dev. of the perturbation on issue estimate | NO       | Typically set to 20%                      |

- the duration of a Sprint, and the way Story Points are computed to decide how many USs to include in each Sprint if Scrum is used;
- the maximum number of USs allowed in each activity if the Lean-Kanban approach is used.

Starting from this information, it is possible to compute the outputs of SPSM, which typically are:
- The project duration if no more issues are entered;
- The number and effort of issued forecast to be closed on a given date;
- An estimate of average, standard deviation, and median of the issue cycle time, that is the time needed from the start of item processing to its completion.

Fig. 5 shows the basic inputs and outputs of our model, based on JIRA issue data. Table 2 shows the parameters used in the simulation, including the class they belong to, their type, and whether they were read from JIRA, estimated from JIRA data, set after expert consultation, or dynamically computed during the simulation.

To perform risk analysis using SPSM, first of all, we need to assess the suitability of the simulator to effectively model the software development process. This can be accomplished by running the simulator on real data and verifying its ability to mimic the real development.

Subsequently, we need to define what are the causes of risk, which can affect the desired outcomes of the project – in our case duration and cost. The main risk factors we identified are erroneous effort estimates of the issues, suboptimal allocation of the issues to developers, and blocks and impediments to the work of developers. Once the statistics of these factors are precisely defined, a Monte Carlo simulation can be used to assess the risks quantitatively.

VI. EXPERIMENTAL DATA
We analyzed three open-source projects which comply with the following preconditions:
1. they are tracked on the JIRA System,
2. they have CreationDate, OriginalTimeEstimate, TimeSpent and Assignee fields filled out for each issue; the last two are reported in man hours;
3. they are medium size (meaning a number of issues above a few hundreds, and below one thousand).

The medium-sized requirement allows us to perform a deeper analysis of the project and the correspondent outcomes when two-month intervals are examined, which would be too time consuming for large size projects. The size is also kept limited to be able to perform several Monte Carlo runs (≥ 100).
The selected projects have a duration of around 15-20 months and an average period of 330 days. The number of team members varies between 15 and 60, but in the latter case team members do not work simultaneously, so on average, there are 15-20 developers active. These projects, all tracked in JIRA, are built for different domains and purposes, so they present differences in topic, duration, team size and composition, workflow. The simulator can quite faithfully reproduce each of them.

A. PROJECT: TEST ENGINEERING

The first development project is TE (Test Engineering), carried on by edX (www.edx.org), a consortium of more than 160 universities offering courses and programs through an e-learning platform completely open-source. TE is an internal project to perform testing and continuous integration of the software developed for edX and by edX partners.

TE is an ongoing project, which we analyzed for a total of 570 working days, including 675 issues classified as bug, epic, story, and task. The team is composed of 13 developers with different skills and productivity, inferred by analyzing the number and effort of issues completed by each developer in the considered time interval. Issues effort estimation follows a very skewed, fat-tail distribution, which is fairly approximated by a Pareto distribution:

\[ p(X) = \frac{b}{X^{b+1}} \]

where \( X \) is expressed in man-days, and the shape value is \( b \approx 1.35 \).

The basic statistics for the TE project are shown in Table 3.

### Table 3. Projects’ statistics. Effort statistics in man days.

| Project name      | Days | Issues | Developers | Mean | Std Dev | Median |
|-------------------|------|--------|------------|------|---------|--------|
| Test Engineering (TE) | 570  | 675    | 15-20      | 2.9  | 4.2     | 1.33   |
| Platform          | 622  | 853    | 14-15      | 4.4  | 6.4     | 1.1    |
| CORD              | 192  | 523    | 13         | 4.3  | 6.4     | 1.0    |

B. PROJECT: PLATFORM

The second project is also carried on by edX and regards the e-learning platform of the edX consortium. In fact, it is called “Platform”. Platform is an ongoing project, which we analyzed for a total of 622 working days, including 853 issues classified as: subtask, bug, story, and epic.

The team size is 65 people. At a first glance, it might seem too big, but after a careful check, we found that there was a high turnover. Most team members worked only for a limited amount of time so that the average team size is about 14-15 people, similar to the other projects.

Issues effort estimation approximately follows a Pareto distribution with shape value \( b = 1.38 \). The basic statistics for the Platform project are shown in Table 3.

C. PROJECT: CORD

The last project is called CORD (Central Office Re-architected as a Datacenter). It is a project of the Open Networking Foundation (ONF), a non-profit operator-led consortium driving transformation of network infrastructure and carrier business models. CORD is an open-source project aiming to develop a platform to build agile datacenters for the network edge.

We analyzed the CORD project for a total of 192 working days, including 523 issues classified as: subtask, feature, bug, story, and epic. Issues effort estimation approximately also follows a Pareto distribution with shape value \( b = 1.51 \). The basic statistics for the CORD project are shown in Table 3.

D. SIMULATOR ASSESSMENT

The first step to assess the validity of our simulator was to check whether it can produce time duration for completion of all issues similar to those of the real projects.

We performed simulations giving as input all issues which resulted in Closed state, or with a remaining effort equal to zero, among all issues read from JIRA repositories. The work on each issue was performed by the same team member of JIRA data, using the developer’s productivity as estimated from the same data.

For performing the simulations, we shortened their duration to an interval of about 70% of the total, starting from the date of the first completed issue. This is because almost all issues created afterward were not yet completed.

Two of the selected projects have a simulation duration of about 14 months (408 and 445 days). CORD was tested for a shorter period of 138 days, or 4.5 months. The number of team members is around fifteen people in all projects, remembering that this is an average team size, obtained by putting to work only the developers who are active in the proper time interval, as inferred by JIRA data.

These projects are built for different domains and purposes, so they present differences in topic, duration, team size and composition, workflow. The simulator can faithfully reproduce each of them. All projects are decomposed into several issues. Each issue has a total effort estimation given in man-days. Different kinds of issues belonging to a project may have a different workflow. This is modeled according to the real schema including all main activities (states).

The results for the three projects are shown in Table 4, which shows the actual duration with the average duration of the simulations. Standard deviations are shown within round brackets after the means.

### Table 4. Total project duration. Mean (st. dev.) over 100 simulations.

| Project name      | N. Issues | N. days to close project |
|-------------------|-----------|--------------------------|
|                  | Real case | Simulated case          |
| Test Engineering (TE) | 285       | 408                      | 433 (31) |
| Platform          | 321       | 445                      | 415 (26) |
| CORD              | 205       | 138                      | 154 (12) |

Besides running the simulator for the whole development time, we also refined the effectiveness tests by considering shorter time intervals. We considered time intervals...
of 60 days, being two months the shortest time suitable to get significant results. Trying shorter duration did not work, both because some issues require at least 60 days to be completed, and because the number of issues to test in each interval becomes too small.

We divided the project duration into intervals of 60 days, considering for each interval the issues actually created, and the team members actually working in the real project. The issues not yet completed at the end of the interval are left to the next one for the remaining part and are considered to be completed only when their status becomes Closed. In these tests, the key parameter is not the duration of the project, but the number of issues completed in each interval, and the total number of issues completed at the end of the last interval.

Table 5 shows statistics and results of the simulations for the three examined projects. Note that overall time intervals considered are longer than those of the preceding simulations, and run as long as possible, compatibly with the constraint to be multiple of 60 days. The reported mean and standard deviations are made on real issues, over the 60 days time intervals. The simulated issues are averaged over 100 simulations for each project.

TABLE 5. Results of simulations with 60 days intervals.

| Project name       | Days | Issues | Mean | Std Dev | Simul. Issues | Diff |
|--------------------|------|--------|------|---------|---------------|------|
| Test Engineering   | 540  | 369    | 41   | 24.5    | 370           | 1    |
| Platform           | 600  | 431    | 43   | 25.9    | 426           | -5   |
| CORD               | 180  | 265    | 83.3 | 62.9    | 267           | 2    |

To give further insight on these simulations, in Table 6 we report the number of issues closed in every 60-day interval for project Test Engineering. The number referring to simulated issues is again the average of over 100 simulations. The last columns report the real values. Similar behavior can be found in the issue management of the other projects. As you can see, the number of closed issues varies a great deal from two months to two months. The simulation can reproduce quite well the number of closed issues in every 60-day interval.

TABLE 6. Issue management in 60-day intervals.

| Test (Test Engineering) Project | Days | Closed Issues (Mean) | Real Closed Issues |
|---------------------------------|------|----------------------|-------------------|
| 60                              | 56   | 54                   |
| 120                             | 10   | 9                    |
| 180                             | 39   | 36                   |
| 240                             | 58   | 60                   |
| 300                             | 44   | 46                   |
| 360                             | 84   | 85                   |
| 420                             | 25   | 26                   |
| 480                             | 10   | 9                    |
| 540                             | 44   | 44                   |

We deemed these results good enough to consider the simulator suitable for its use in risk analysis.

VII. RISK ASSESSMENT THROUGH SIMULATION

We applied the Risk assessment methodology outlined in Section III-A on the reported cases, to test its effectiveness. The key risk factors identified are variations in efforts estimated by developers to complete USs, and random developer issues assignment – to be compared with assignment according to real data. In this preliminary study, we did not consider variations in the skills of team members, and events stalling the development of single features or blocking one or more developers, though the simulator could account also for these factors.

The effort estimation error of each issue is modeled using random variations. The Percentage differences between estimated and actual times to close an issue in the three projects are very close to zero, and show a standard deviation between 0.17 and 0.26. Averaging on all closed issues of the three projects, the standard deviation is 0.22, which we approximate to 0.2.

So, the effort estimation error is obtained by multiplying the original issue effort value by a number that follows a log-normal distribution with an average equal to one, and a standard deviation equal to 0.2. In this way, the efforts averaged on all issues remain equal to the average of original issues, and the standard deviation of errors is very similar.

The key process outputs whose variations are checked are the project total time and statistics on cycle times to implement a feature. The time from the project start to the time when the last feature is released is inversely proportional to the throughput and is an excellent indicator of the good performance of the project.

The statistics on cycle times, measuring time to give value to the customer, are:

- average time to implement a feature;
- standard deviation of the time;
- median time, measuring the most likely time to complete a feature;
- 95% percentile of times, measuring the limit of the worst 5% times;
- 5% percentile of times, measuring the limit of the best 5% times;
- maximum time, measuring the worst case.

We perform a given number of Monte Carlo simulations – typically one hundred, but they might be more – for each choice of the tested risk factors, varying random perturbation, or developers’ assignments to issues, recording the key outputs.

From these outputs, it is then possible to compute the desired statistics. On each of these values, it is possible to set risk thresholds that, if reached or overcome, trigger proper mitigation actions.

A. REAL CASES RISK ANALYSIS

We used the three open-source projects described in Section VI, considering the time interval from the first issue project creation date until the end of the projects. We start by analyzing the time estimation of project duration when only
time estimation variations are considered as risk factors, but issue resolutions are made by the same developers of the real case. For a better interpretation of the results, we divided the simulations into time intervals of 60 days, as made before.

Figures 6, 7 and 8 show the results for projects TE, Platform and CORD, respectively. Here we show, for each time interval, the values of the real number of issues completed in that interval, of the average on 100 simulations varying the issue estimates as described in Section VII, and of the 5% and 95% percentiles of the simulated completed issues. We do not show the medians of completed issues because they are very similar to the averages, and would not add significant information to the figures.

In all projects, the number of completed issues every 60 days greatly varies. This variability is due to different reasons, such as the different commitments of the developers in different months, the tendency of issues to be completed in “batches” rather than in a continuous flow, and the low number of active developers, which makes the project output more susceptible to random variations.

Regarding the TE project, the real number of completed issues and the average of simulated numbers are quite close to each other, being the maximum percentage error equal to 36%. The real number of completed issues is almost always contained between 5% and 95% percentiles, but for the result at 300 days, where the real number is lower than the 5% percentile.

The results of the CORD project are shown in Fig. 8. Here we have only three simulated intervals of 60 days, with very few completed issues after the first one. For this reason, the differences between real and average numbers over 100 simulated cases of the completed issues are higher, being 25% after the second time interval.

The average of total closed issues after 180 days is 291, versus the real value of 265 (the difference is 10%). The real number of completed issues is always contained between 5% and 95% percentiles of the simulated cases.

B. RISK ANALYSIS WITH RANDOM DEVELOPER ALLOCATION

In this section, we consider the effect of the application of both risk factors in the simulations, namely random estimation errors and random allocation of issues among developers. The estimation errors are introduced using the same approach of the previous section, multiplying the original estimates by a random number following a log-normal distribution with an average equal to one, and a standard deviation of 0.2.

The allocation of developers to issues is not equal to the real case, but developers simply “decide” what issue to work on depending on a random choice of available issues with the highest priority. This mimics better the real situation, where of course it is not known in advance which developer will work on which issue, and a risk analysis must consider an issue allocation not known in advance.

As in the previous Section, we divided the simulations into time intervals of 60 days, and report the number of issues completed in each of them. We report the number of issues

number of completed issues (423) and the average number of simulated ones over 100 simulations (416) is about 2%.

Considering the cumulative number of completed issues over time, the error is always under 6%, but after the first 120 days, where it is 25% after 60 days, and 13% after 120 days.

With respect to the TE project, 5% and 95% percentiles over 100 simulations include the real number of completed issues, but for the result at 300 days, where the real number is lower than the 5% percentile.

The results of the Platform project are shown in Fig. 7. Here we have only three simulated intervals of 60 days, with very few completed issues after the first one. For this reason, the differences between real and average numbers over 100 simulated cases of the completed issues are higher, being 25% after the second time interval.

The average of total closed issues after 180 days is 291, versus the real value of 265 (the difference is 10%). The real number of completed issues is always contained between 5% and 95% percentiles of the simulated cases.

FIGURE 6. TE project: N. of completed issues vs. time. Averages and percentiles over 100 simulations.

FIGURE 7. Platform project: N. of completed issues vs. time. Averages and percentiles over 100 simulations.
of the real case, the average and the 5% and 95% percentiles computed over 100 simulations.

Fig. 9 shows the results for the TE project. As you can see, the four curves are closer than in the case without random developer allocation, as shown in Fig. 6.

These results are consistent with those found without risk parameters. The overall percentage difference between the total number of completed issues (369) and the average number of simulated ones over 100 simulations (362) is about 2%.

Considering the cumulative number of completed issues over time, the error is always under 5%. The 5% and 95% percentiles over 100 simulations always include the real number of completed issues.

The reproducibility of real data is well provided by the model. Having in place both risk parameters results in outputs that, on average, are closer to real data with respect to using only effort estimation perturbations.

The data on the Platform project are reported in Fig. 10. Also here the four curves are closer than in the case without random developer allocation, as shown in Fig. 7.

The overall percentage difference between the total number of completed issues (426) and the average number of simulated ones over 100 simulations (431) is about 1%.

Considering the cumulative number of completed issues over time, the error is always under 11%, but for the first time interval where it is 100%, due to the very low of completed issues (4 in the real case, 8 in the average of simulated cases). The 5% and 95% percentiles over 100 simulations include the real number of completed issues, but again at the end of the first time interval.

Eventually, Fig. 11 shows the results for the CORD project. The results do not differ much from the case with variations only in issue effort estimation and are slightly better.

The average of total closed issues after 180 days is 271, versus the real value of 265 (the difference is 2%). The real number of completed issues is well contained between 5% and 95% percentiles of the simulated cases, but after the first time interval, for the usual reason of a very low number of issues.

C. DISCUSSION

Starting from the analysis of three real projects tracked on JIRA, we made four different types of analysis with four scenarios.
The first scenario had the aim to test the simulator reliability by comparing the number of days needed to close the project in the real and simulated cases for the three projects, as shown in Table 5. The simulator reproduces the project duration in days with an error margin between 6% and 11%, considering the average of 100 simulations, with a standard deviation between 6% and 8% of the related average.

The second scenario is aimed at improving forecasts by reducing the time interval of predictions to time intervals of sixty days each. In this way, the model simulates the number of closed issues for a limited period and resets itself at the end of the interval to perform the next interval forecast, according to the supplied real data. The results show that in this way the model performs better than using the entire project lasting time.

Here the key output is not the duration of the project, but the number of issues that are actually completed during each time interval. In the case of issues only partially completed, they are not taken into account but are moved to be completed to the next time interval. The results of these simulations, obtained again by performing 100 simulations for each interval and averaging the number of issues completed after each simulation, show that the percentage error between the average and the real data is typically less than 10% in all intervals, except sometimes the first one, characterized by a low number of completed issues in the real case.

Summing up all completed issues on all intervals yields a total number of completed issues (averaged over 100 simulations) which differ less than 1.5% with respect to the real number of completed issues.

We deemed that these scenarios prove that the simulator can reproduce with enough precision the real ASD process. Referring to the Research Questions asked in the Introduction, we are able to answer to RQ1 and RQ2:

**RQ1**: To which extent it is possible to automatically import into the simulator data of real projects, from issue management systems?

Importing issue data directly from the popular system JIRA is quite straightforward. However, importing the sequence of activities actually used for a specific project, as well as estimating the skills and time commitment of developers need manual intervention. This must be done before the simulation starts, by analyzing project data. During project execution, new issues can be periodically read to update the simulation state, without further intervention.

**RQ2**: How accurate can the simulator be in predicting project completion times?

The case studies analyzed show that the simulator can be quite accurate in reproducing a real project progress – that is, the project completion time – if it is fed with accurate data about issue estimates and team composition and skills. Simulating whole projects, which lasted between six and twenty months, the error margin is of the order of 10%. Shortening the simulation intervals to two months, and then resynchronizing the simulator, yields an even lower error of the order of 1-2% averaged over 100 simulations. So, the answer to this question is that the simulator accuracy to predict project completion times is high.

The third and fourth scenarios introduce the use of the simulator for risk analysis. Basically, in both scenarios, random perturbations are applied to issue estimations, chosen from a log-normal distribution, derived from statistical analysis of the real data set. The fourth scenario adds a random choice of the developers who subscribe for resolving an issue.

We ran 100 simulations for each considered project, for both scenarios, to check the extent to which the simulated outputs differ from the real ones, in the case of random estimation errors. So, we computed not only the mean value of the forecasted number of closed issues in the time intervals but also proper percentiles, helpful to check if the real values stay within these limits.

We found that just applying random perturbations to issue estimation – which in a real project would mean that the original estimation was wrong, whereas the perturbated one is the “right” value – has limited impact on the goodness of the approximation of the simulation results with respect to the real case. This is shown in Figures 6, 7, and 8, and in the discussion thereof.

When random issue assignment to developers is added, the results are even better, meaning that the simulation outputs tend to be even closer to the real ones, as shown in Figures 9, 10, and 11, and in the discussion thereof. This result can be attributed to the fact that in the real case simulated work on the issues is performed by the developers who registered it in JIRA. So, the simulator applies a rigid assignment, which mimics the delays and unavailability of real development. When issues are randomly assigned, however, when work on an issue is needed in a given activity, all available developers are polled, and one is chosen at random. If there are available developers, work on the issue begins with no delay, and this justifies the better results in terms of completion time.

Practically, the simulator can be applied to actual risk management by applying the proper random variations to the parameters that could be the major causes of risk, performing a sufficiently high number of simulations, and evaluating the distributions of key output parameters. Moreover, in a real case, future issues can be randomly generated, using their expected effort distribution, to get a more realistic simulation of the work to be performed.

Being still an ongoing research project, the simulator has not been yet used in real development. To get feedback, we presented our results to the same experts involved in helping us to estimate the penalty factor \( p \), as reported in Section IV-C4 “Component Validation”. The majority of experts encouraged us to continue the work, believing that
assessing project risks using SPSM is a valid and promising approach, provided that other kinds of risks are considered, besides error in issue estimation. A couple of the project managers stressed that this tool might be useful, but only if provided with a user interface able to ease the tuning of the simulation parameters and to immediately highlight the risks to exceed time and costs.

This leads to the answer to the last research question:

**RQ3: Can the simulator be useful to estimate project risk (induced by errors in efforts estimation, and random developer issues assignment) with a Monte Carlo approach?**

By varying parameters – such as the variance of issue estimation errors, and developers’ availability – and performing Monte Carlo simulations, a project manager can compute statistics on forecast project completion times and average time to complete issues, and take proper action to control this kind of risk. Consequently, also the answer to this question is positive. Clearly, much more causes of risk might be accounted for, as described below, and we are actively working to include them in the simulator.

In this paper, for the sake of simplicity, we limited to use just issue effort perturbations, and random developers’ assignment. However, other causes of risk can be easily simulated, such as:

- Sequentiality constraints between issues, so that delaying or stalling the development of an issue directly influences other issues.
- Problems related to specific issues, whose development is delayed or stalled for external reasons.
- Change or deletion of existing issues.
- Different skills of developers in the process activities, and consequent preferential choices of issues to work on.

Other important causes of risk, which can also be simulated, and which embody random components and add “uncertainty” to the process are shown in the followings. The distributions of the value of these factors must be studied and defined in advance, typically by analyzing the past history of the project, or of similar projects:

- Arrival of new issues, real or simulated to account for forecasts of future work to be done; here the distribution of issue effort estimates, issue priorities, and issue arrival time must be defined. Also, change requests of existing issues not yet implemented might be considered.
- Issues which do not pass quality controls, and have to be reworked, whose probability and extent of rework must, in turn, be specified.
- Changes in the availability of team members, due to various causes (vacation, illness, resignation, more important commitments to carry out). This is an important risk factor, able to substantially change the project schedule. Again, the probability and duration of developers’ unavailability must be defined in advance.

Clearly, in real projects, this approach should be embedded in other risk management approaches, which include risk identification and risk mitigation. Our SPSM-based approach can substantially help in quantitative risk analysis, but cannot cover different kinds of risk that cannot be modeled and simulated, such as Organizational Environment Risk, User Risk, and Team Risk [50]. In other words, we do not claim that the simulation technique is better than other known techniques to manage risk, but we claim that it should be used as another tool, able to complement other approaches.

For instance, in Rm4Am risk management tool for ASD [44], our tool might be used for assessing the risk of increments (i.e. user stories/features), in the risk analysis of Product and Sprint backlogs. This should be done during the Risk weekly meeting, a subcomponent specifically added in Rm4Am to manage project risk.

**VIII. THREATS TO VALIDITY**

The goal of this section is to discuss the threats to validity that we need to consider regarding the study of JIRA open source projects performed to evaluate the presented risk assessment method. The typical threats to validity taken into consideration in a software process development are: construct validity, internal validity, and external validity [39].

- **Internal Validity** is the approximate truth about inferences regarding cause-effect or causal relationships. Thus, it is only relevant in studies that try to establish a causal relationship and it is not relevant in most observational or descriptive studies. In our case, the analysis is focused to demonstrate the abilities to manage the risk using statistical analysis, and internal validity is not relevant to find this kind of relationship.

- **Construct validity** is focused on the relation between the theory behind the experiment and the observations. Even when it has been established that there is a causal relationship between the execution of an experiment and the observed outcome, the treatment might not correspond to the cause one thinks to have controlled and altered, or the observed outcome might not correspond to the effect people think they are measuring. In our case, the main threat to construct validity is whether our models are accurate enough to be realistic. In other words, are issues, activities, and developers, as modeled by us, enough reliable to get plausible results? Other studies on empirical data seem to answer favorably to this question [8], [15], but more research is clearly needed. Another issue related to constructing validity is the characteristics of feature effort variations and the random issues to developers’ allocation. Though these characteristics are common to many projects, the exact distribution of effort variations and the random assignment performed might be improved. Moreover, there are other possible risk factors, such as inaccurate requirements and team-related issues, such as the resignation of developers, the introduction of new developers,
intra-team dependencies, and so on. What we proposed is a model able to represent some key aspects of software development, without trying to model every possible aspect.

- **External validity** is concerned with whether one can generalize the results outside the scope of the study and hold them across different experimental settings, procedures, and participants. If a study possesses external validity, its results will generalize to a larger population not considered in the experiment [8].

In this specific study, we performed hundreds of simulation runs. An important aspect to be taken into consideration is that data analyzed are real but the analysis refers to just three simplified test cases. Although the scope of this paper would not have allowed simulating more complex cases, this has to be taken into account when considering external validity. Also, the fact that all features were available at the beginning of the simulated project, and that no more feature was added, limits the generalization of our results.

**IX. CONCLUSION AND FUTURE WORK**

Risk management is essential to software development. Agile methodologies were introduced precisely to minimize the risk inherent in traditional waterfall processes, with very high risks related to requirement changes and final integration. However, only a few studies have been devoted to explicit risk management of agile processes.

In this paper we presented a risk assessment procedure based on process modeling and simulation and tested it on three open-source projects developed using an agile approach, with requirements collected as user stories and managed as an atomic unit of work, using JIRA popular issue management tool. The process can be organized into a sequence of basic activities, and can thus be modeled using an event-based simulation. The developers are in turn modeled as agents, who decide the units of work they develop and complete.

To be able to work with real data, we linked the simulator with JIRA to collect the needed information (used process, team composition, project size, number of issues, estimated effort) and show the reliability of the tool.

We have shown how it is possible to run the simulator in a Monte Carlo fashion, varying the identified risk factors and statistically evaluating their effects on the critical outputs. In this way, a risk manager will be able to analyze the expected parameters of the examined project, including the expected optimal closing time of the project, and of the various issues and features, and evaluate the percentiles of their distributions, to assess the probability of adverse and favorable variations.

We validated the simulator using three open-source medium-sized projects, whose data are available on open JIRA repositories, and considered four kinds of scenarios. The first two were used to test the reliability of the simulator; the third and fourth scenarios were used to make a risk analysis introducing the variation in the estimated effort to complete features, and when there are changes in their allocation to developers.

The proposed approach is clearly relevant for project managers, who get a tool able to quantitatively evaluate the risks, provided that the process and the project’s data are properly modeled. This is made possible by the relative simplicity of Agile processes, that typically operate on atomic requirements — the features, or issues — through a sequence of activities.

Clearly, in real projects the approach should be complemented with other risk management approaches, able to cover different kinds of risk that cannot be modeled and simulated, such as Organizational Environment Risk, User Risk, and Team Risk [50].

**A. LIMITATIONS**

In this work, we limited our study considering only the effort variations and the random assignment of developers to issues. Even though the obtained results are limited to the analyzed projects, our model could be customized and adapted for other projects.

**B. FUTURE WORK**

In the future, we will improve our risk assessment method, evaluating it on many other case studies, and also exploring the optimal parameter settings that can minimize the overall risk. An improvement we are considering is to scale the model from a single team to multiple teams, involved in one project, or even in several projects. This would greatly improve the utility of the tool for risk management in large organizations.

**REFERENCES**

[1] *Jira*. [Online]. Available: https://www.atlassian.com/software/jira

[2] Z. S. H. Abad, M. Noaeen, D. Zowghi, B. H. Far, and K. Barker, “Two sides of the same coin: Software Developers’ perceptions of task switching and task interruption,” in Proc. 22nd Int. Conf. Eval. Assessment Softw. Eng., Jun. 2018, pp. 175–180.

[3] T. E. Abioye, O. T. Arogundade, S. Misra, A. T. Akinwale, and O. J. Adeniran, “Toward ontology-based risk management framework for software projects: An empirical study.” J. Softw., Evol. Process, vol. 32, no. 12, Dec. 2020, Art. no. e2269.

[4] A. Albadarneh, I. Albadarneh, and A. Qasef, “Risk management in agile software development: A comparative study,” in Proc. IEEE Jordan Conf. Appl. Electr. Eng. Comput. Technol. (AEECT), Nov. 2015, pp. 1–6.

[5] C. J. Alberts and A. J. Dorofee, “Risk management framework,” Carnegie-Mellon Univ. Pittsburgh PA Softw. Eng. Inst., Pittsburgh, PA, USA, Tech. Rep., 2010.

[6] T. Alencar, M. Cortés, N. Veras, and L. Magno, “A proactive approach to support risk management in software projects using multi-agent systems,” in Proc. 20th Int. Conf. Enterprise Inf. Syst., 2018, pp. 415–424.

[7] D. Anderson, G. Concasa, M. I. Lunesu, and M. Marchesi, “Studying Lean–Kanban approach using software process simulation,” in Proc. Int. Conf. Agile Softw. Develop. Berlin, Germany: Springer, 2011, pp. 12–26.

[8] D. J. Anderson, G. Concasa, M. I. Lunesu, M. Marchesi, and H. Zhang, “A comparative study of scrum and Kanban approaches on a real case study using simulation,” in Proc. Int. Conf. Agile Softw. Develop. Berlin, Germany: Springer, 2012, pp. 123–137.

[9] M. Asif and J. Ahmed, “A novel case base reasoning and frequent pattern based decision support system for mitigating software risk factors,” IEEE Access, vol. 8, pp. 102278–102291, 2020.
MARIA ILARIA LUNESU received the degree in electronic engineering from the University of Cagliari and the Ph.D. degree in electronic and computer engineering from the University of Cagliari, in 2013. She is currently a Research Fellow at the Department of Mathematics and Computer Science, University of Cagliari. Her research interests include the study and applications of Agile and Lean methodologies evaluating their potential for risk management in software development processes, the study of startup dynamics and ecosystem, and the study of software engineering practices for blockchain development and distributed applications and ICOs main success factors analysis. She received the title of Doctor Europaeus.

ROBERTO TONELLI (Member, IEEE) received the Ph.D. degree in physics, in 2000, and the Ph.D. degree in computer engineering, in 2012. He is currently a temporary Researcher and a Professor with the University of Cagliari, Italy. The main topic of his research has been the study of power laws in software systems within the perspective of describing software quality. Since 2014, he has been extended his research interest to blockchain technology. His research interests include widespread and multidisciplinary. He received the Prize for the top-50 most influential papers on blockchain, in 2018 and January 2019, from Blockchain Connect Conference, San Francisco, CA, USA.

LODOVICA MARCHESI (Member, IEEE) graduated in computer science at the University of Cagliari, in February 2018, with a final grade of 110/110. She is currently pursuing the Ph.D. degree with the Department of Mathematics and Computer Science, University of Cagliari. She worked as a Researcher for the grant on the study of blockchain technology applied to systems for managing complex documents at the Department of Electrical and Electronic Engineering, University of Cagliari. Her research interests include the application of blockchain technology in different sectors, cybersecurity for blockchain applications, the study of software engineering practices for blockchain development and distributed applications, and the study of machine learning algorithms and economic models and financials related to the cryptocurrency market.

MICHELE MARCHESI (Senior Member, IEEE) received the degree in electronic engineering from the University of Genova, in 1975. He has been a Full Professor with the Faculty of Engineering, University of Cagliari, since 1994. Since 2016, he has been a Full Professor with the Department of Mathematics and Computer Science, University of Cagliari, where he teaches software engineering courses. He has authored over 200 international publications, including over 70 in the magazine. He has been one of the first in Italy to deal with OOP, since 1986. He was a Founding Member of TABOO, the Italian association on object-oriented techniques. He has also worked on object analysis and design, UML language, and metrics for object-oriented systems since the introduction of these research themes. In 1998, he was the first in Italy to deal with extreme programming (XP) and Agile methodologies for software production. He organized the first and most important world conferences on Extreme Programming and Agile Processes in Software Engineering, Sardinia, from 2000 to 2002. Since 2014, being among the first in Italy, he has been extending his research interest to blockchain technologies, and obtaining significant results in the scientific community.