Simulation-Based Optimization for Software Dynamic Testing Processes

Mercedes Ruiz
Department of Computer Science and Engineering
University of Cádiz
Cádiz, SPAIN
e-mail: mercedes.ruiz@uca.es

Javier Tuya
Department of Computing
University of Oviedo
Gijón, SPAIN
e-mail: tuya@uniovi.es

Daniel Crespo
Department of Computer Science and Engineering
University of Cádiz
Cádiz, SPAIN
e-mail: dani.crespobernal@alum.uca.es

***Abstract* - Managing software development projects requires the coordination of different processes that may be performed by different teams, e.g., a development team and a separate testing team. This coordination aims at optimizing the trade-off between cost, schedule and delivered quality. Simulation models are a powerful tool to explore what-if scenarios that help managers to achieve this trade-off and to fine-tune different project parameters. This paper presents a simulation model based on a multi-paradigm approach that connects development and testing processes. The testing process model is based on the process model described in the ISO/IEC/IEEE 29119-2:2013 standard. The simulation model is built using two different methods: the discrete-event approach, to simulate the execution of the dynamic testing processes, and the agent-based approach, to in-depth simulate defects life cycle. Results show how the simulation model is used to explore the evolution of a number of process metrics. Then, the simulation model is used to determine the resource distributions in order to optimize two relevant process metrics: the efficiency of the testing process and the average defect life.**

Keywords - software testing; multiparadigm simulation; test management; test process optimization.

I. Introduction

This is a revised and augmented version of our previous work, which appeared in the Proceedings of the Seventh International Conference on Software Engineering Advances (ICSEA 2012) [1]. Software testing is concerned with planning, preparation and evaluation of software products and related work products to: a) determine that they satisfy specified requirements, b) demonstrate that they are fit for purpose and c) detect defects [2]. In general, testing can be viewed as a means of improving the quality of a given product and mitigating risks due to poor quality.

Testing can be carried on using different approaches (e.g., scripted or exploratory), at different levels (e.g., unit, system, integration or acceptance), using different techniques and tools and with different degrees of independency (ranging from testing performed by the producer to third party testing). When testing entails the execution of the system under test, it is often referred to as dynamic testing. Testing exists in an organizational context and is carried on a given project or service. Therefore, the testing activities are tightly interrelated with the development ones, and both shall be planned, monitored and controlled. Problems of quality of the system under test or delays in the development hamper the testing process. Conversely, an inadequate or delayed testing endangers the development process. If not managed properly, both development and testing processes may jeopardize the goals of cost, schedule and quality of a project.

Both development and testing can be described as processes and take advantage of the use of simulation models for helping project and/or test managers in daily tasks of planning, monitoring and control.

Informally, a simulation model can be considered as an abstract view of a complex system comprised of a set of rules that tell how to obtain the next state of the system from the current state. Those rules can be of many different forms: differential equations, state charts, process flowcharts, schedules, etc. The outputs of the model are produced and observed as the model is running.

There is much research on simulation models of the software development process [3]. However, there is lesser research on simulation models for the testing process, usually at the unit level. Furthermore, when testing is considered as part of a simulation model of the development process, it is often over simplified. The goal of this paper is to devise a multi-paradigm simulation model for the testing process to gain insights in how the testing process influences the goals of a given project. The simulation model can be used to simulate the testing process at the system level and to help in decision-making in the test managing processes.

Contributions of this paper extend previous work [1] and include:

1. A multi-paradigm simulation model of the dynamic testing processes, which combines a discrete-event model and agent-based model.
2. The optimization of two key variables of the testing process: process efficiency and average defect life

The main contribution of this work is a multi-paradigm simulation model of the dynamic testing processes that combines a discrete-event model and agent-based model. The model can be used to simulate the testing process at the system level and to help in decision-making in the test managing processes.

The structure of the paper is as follows: Section II shows the works related to our proposal; Section III introduces the multi-layer process model proposed by the International Standards group upon which our simulation model is based; Section IV describes the simulation model; Section V shows two simulation optimization scenarios. Finally, our conclusions and further work are given in Section VI.

II. RELATED WORK

During the recent years a lot of research has been done in the field of software testing. These studies are mainly oriented to enhance and optimize the software testing process improving the results obtained after the development of software projects. Several techniques and methods have been used to reach this goal. Knowledge Management has been a recurrent tool due to its usefulness for revising software testing processes [4], learning from the errors committed in the past [5], collecting, analyzing and managing lessons learned [6] and improving the quality of software testing [7].

Sometimes, it is interesting to study the behavior of processes in order to detect weaknesses so that improvement can be effectively performed. Process modeling techniques have been widely used to address these issues. Models can be used to estimate process outcomes such us the number of defects remaining and the time required to detect defects either for a subsequent optimization [8], to estimate effort, cost and schedule [9] or to perform cost control management [10]. Modeling also plays an important role in decision-making support. Reference [11] presents a goal-driven measurement model for software testing process so that software organizations can deduce the appropriate measurement process according to the process goals they determine. On the other hand, reference [12] provides a quantitative defect management model that can be improved to be practically useful for determining which activities need to be addressed to improve the degree of early and cost-effective software fault detection with assured confidence. Finally, reference [13] proposes a competence model that could be applied to train staff in software testing activities and to recruit the appropriate profiles improving their performance.

Some authors have developed their own frameworks to study software test processes. Reference [14] describes a conceptual framework to specify and explicitly evaluate test process quality aspects and [15] proposes a software testing improvement framework based on the Plan–Do–Check–Act (PDCA) method.

Some other techniques employed for software testing process improvement include Bayesian networks for process evaluation [16], Markov decision models to optimize software testing by minimizing the expected cost with given software parameters of concern [17], multi-objective feature prioritization for testing planning and controlling [18], system dynamics to formulate and quantify the software testing processes [19] and even the usage of software engineering standards to improve the testing process [20].

Although the above mentioned techniques are extremely useful to improve the software testing processes, sometimes it is necessary to look into the processes with more detail. In order to effectively optimize a process the current behavior must be examined. Furthermore, all the possible variations suffered by the process due to the different scenarios that may occur, should be taken into account in order to analyze the results derived from one situation or another. Software process simulation provides the means to accomplish this goal in a cost effective way.

The search string “simulation” AND “software testing process” AND “management” and others alike used in several digital libraries and citation databases of peer-reviewed literature retrieves only a few number of papers. In many of the papers retrieved, the term “simulation” is frequently used to describe experiences in which simulation is used as a tool for the testing process. In other works, the term “simulation” makes reference to a set of formulas that are solved by analytical means.

As an example of the first usage, in their collection of works, Lazić, Mastorakis and Velasěvić [21] to [25] aim at raising awareness about the usefulness and importance of computer-based simulation in support of software testing. In their works, simulation is used to ease the design and execution of the testing processes of real military and defense systems.

Some analytical models of the software testing process can also be found. Zhang, Zhou, and Luo [26] propose a reward-Markov-chain-based quantitative model for sequential iterative processes and show how to use it to estimate the time for the software testing process. Similarly to this, Lizhi, Weiqin, Zhou and Zhang [27] propose an approach to model the testing process based on hierarchical time colored Petri Nets (HTCPN). However, while Petri-nets are good at modeling resources and parallel processing, simulation modeling models system components and their interactions, making it possible to conduct arbitrary time-related performance analysis, something which is not easy using Petri-nets.

Consequently, to overcome the problems of analytical methods, simulation modeling can be applied in the context of testing processes mainly because: a) it enables to find solutions when analytical methods fail; b) it is a more straightforward process than analytical modeling since the structure of the simulation model naturally mimics the structure of the real system, and c) it is scalable, flexible, and easy to communicate since the modeling tools use visual languages.

However, despite these advantages there is a small number of contributions of simulation modeling in the field of software testing processes. Saurabh [28] presents a System Dynamics (SD) model of software development with a special focus on the unit test phase. This work is partially based on Collofello’s et al. work about modeling the software testing process under the SD approach [29].

The motivation of these works is closely related to ours, but the models are built under a different simulation approach. System Dynamics approach operates at high abstraction level and is mostly used for strategic modeling. Hence, since a simulation model can only be used at the abstraction level in which it has been created, such a highly abstract model is not adequate for the operational and tactical levels in which decision-making regarding the testing processes takes place. In our case, since our main interest is to simulate the testing processes the discrete event (DE) modeling, with the underlying process-centric approach, has been selected. Furthermore, we have also selected the agent-based (AB) approach to be used together with the discrete-event one resulting in a multi-paradigm simulation model.

Generally, each simulation approach (SD, DE, AB) provides a set of different abstractions. If the system being modeled is complex enough, and software development is, then it is preferable to integrate different simulation methods than using one single approach, since the final model will represent the real system more realistically.

When we used the search string ("multi-paradigm" OR "multi-method") AND "simulation" AND "software testing process" and others alike in the digital libraries and citation databases, no single work was retrieved. Therefore, given the results of the systematic literature review performed, not fully documented here for space reasons, to the best of our knowledge our proposal is the first one that aims at using multi-paradigm simulation modeling to improve decision making in software testing management.

III. MULTI-LAYER TEST PROCESS MODEL

Testing processes include a variety of management and technical activities that are organized in a process model in part 2 of the ISO standard for software testing: ISO/IEC/IEEE 29119-2:2013 [30]. The purpose of this international standard is to define a generic process model for software testing that can be used by any organization when performing any form of software testing. Testing is structured in a multi-layer process model that defines the software testing processes at (1) the organizational level, (2) test management level and (3) dynamic test level. More specifically, the dynamic test level describes how dynamic testing is carried out within a particular phase of testing (e.g., unit, integration, system and acceptance) or type of testing (e.g., performance testing, security testing and usability testing). It is composed of four processes that are depicted in Figure 1.

- Test Design & Implementation Process: Describes how test cases and test procedures are derived; these are normally documented in a test specification, but may be immediately executed.
- Test Environment Set-Up & Maintenance Process: Describes how the environment in which tests are executed is established and maintained.
- Test Execution Process: Describes how the test procedures generated as a result of the Test Design & Implementation Process are run on the test environment established by the Test Environment Set-Up & Maintenance Process.
- Test Incident Reporting Process describes how the reporting of test incidents is managed.

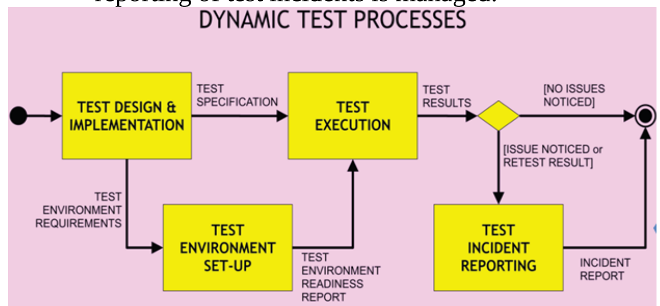

Figure 1. Dynamic Test Processes in ISO/IEC/IEEE 29119-2:2013.

The Test Execution Process is run after the tests have been specified and the environment has been established, which leads to a strong dependency on the previous processes. This process may need to be performed a number of times as all the available test procedures may not be executed in a single iteration. Additionally, this process must be reentered as a consequence of detected failures after the underlying defects have been corrected (retesting).

Besides, the Test Design & Implementation Process, and the Test Environment Set-Up & Maintenance Process may be reentered whether additional tests are needed after execution or some problems are detected in the testing environment. The Test Incident Reporting Process may be also reentered as a result of: a) the identification of test failures, b) something unusual or unexpected occurred during test execution, or c) retest activities.

IV. MODEL DESCRIPTION

The simulation model developed is described below in terms of its scope, result variables, process abstraction and input parameters. The description is organized following Kellner's proposal for describing simulation models [31].

A. Model Proposal and Scope

To determine a model proposal, the key questions that the model needs to address need to be identified. Then, model scope is set so that it is large enough to fully solve the key questions posed. In the context of this work, model proposal is to help in decision-making in software testing process management. Accordingly, the scope for this model will be a portion of the life cycle, with a short time span (i.e., the months in which the testing activities take place), one software product and two teams (i.e., development and testing teams) organizational breadth.

B. Result Variables

The result variables are the information elements needed to answer the key questions regarding the purpose of the model. In our model, several process metrics have been identified to help us understand the simulated process capability. According to this, process metrics have been classified into effectiveness and efficiency process metrics.

Effectiveness process metrics measure the extent to which a process produces a desired result [32].

The following result variables fall into this category:

- Defect Closure Period (DCP). The longer a reported defect takes to go from discovery to resolution, the higher the project risk associated with the underlying defect. Unresolved defects may: a) delay testing, b) make development less efficient or c) prevent the delivery of the software to the final customers. DCP measures the difference between the time required to repair a defect and the time required to confirm the defect is repaired.
- Defect Open Count. This measure tracks the number of times a defect report is opened. When the report is first submitted this count is set to one. This count is incremented each time the same defect report is reopened due to a failure in the confirmation test (retest).
- FixBacklog: Shows the percentage of defects closed per all the defects opened in a given time.
- Average Defect Life: Shows the average elapsed time from the moment a defect is found until it is successfully closed.
- Total Planned Test: The metric shows the evolution of the number of planned test cases along the testing project.
- Total Executed Tests: Shows the evolution of actual test cases that are executed along the project.
- Total Passed Tests: Shows the actual test cases that are executed and successfully passed (e.g., did not find any defects).
- Total Failed Tests: Shows the actual test cases that are executed and failed (e.g., did find defects).

Efficiency process metrics measure the extent to which a process produces its desired results in a not wasteful way and, ideally, minimizing the resources used [32].

Result variables in this category follow:

- Actual Test Time: Shows the total length of the testing process.
- Total Team Size and number of people per activity: Shows the total size of the testing team and the number of resources allocated to each activity of the process, respectively.
- Average Defect Cost: Shows the ratio between the total number of defects closed and the number of working hours invested.
- Process Efficiency: Shows the ratio of the number of defects closed per the number of defects found.

C. Process Abstraction

When developing a simulation model, the key elements of the process, their inter-relationships, and behavior need to be identified. The focus should be on those aspects of the process that are especially relevant to the purpose of the model, and believed to affect the result variables [31].

One of the decisions that need to be made in this phase is the simulation paradigm that it is going to be used to build the model. A simulation paradigm is a general framework for mapping a real world system to its model. The choice of paradigm should be based on the system being modeled and the purpose of the modeling. When modeling complex systems, it is frequent that different parts of the system are most naturally modeled using different paradigms. In this case, a multi-paradigm model is built.

In order to build our model, the multi-paradigm approach has been selected. First, to model and simulate the dynamic testing processes, the paradigm selected has been the discrete-event or process centric approach. Under this approach, the system being modeled is considered as a process, i.e., a sequence of operations being performed across entities, and this makes this paradigm the most natural and adequate to build process simulation models. The model is specified graphically as a process flowchart, where blocks represent the operations to be done along the process.

Although a simulation model following this approach allows us to analyze the evolution of the testing activities, the resource consumption and the number of defects detected, it would be interesting to add an extra functionality to the simulation model allowing the user to track the life of every defect since it is found until it is closed. It is important to notice that to achieve this aim the level of abstraction used needs to be changed from process-centric to individual-centric. Agent-based modeling is a simulation approach that allows the modeler to build a model under a bottom-up perspective, that is, describing the behavior of individuals (e.g., agents) and, if needed, their interactions. Frequently, the behavior of an agent is formalized by means of a state chart-like diagram. Therefore, this approach seems to be most natural and adequate to describe the lifecycle of defects found during the testing phase. As a consequence, a multi-paradigm simulation model was our choice for our modeling problem.

In summary, the model consists of two connected models. A description of each of these models follows:

1) Discrete event model (DE).

The discrete event model represents the Dynamic Test Processes in ISO/IEC/IEEE 29119-2:2013 [30], previously described in Section III.

The development process produces two main artifacts that are the input for the testing processes:

1. The test basis, usually the software specification, which is modeled as a set of features.
2. The executable code that is to be exercised by the tests.

The availability of the test basis enables the execution of the Test Design & Implementation Process, which leads to a number of test cases. However, test cases are not ready to be executed until the test environment has been established (Test Design & Implementation Process) and the executable code released. Once the code is installed in the testing environment, the Test Execution Process can begin. Failed test cases are the input for the Test Incident Reporting Process and the results communicated to the development processes through the Agent-based model. Test execution reenters when previously detected defects have been fixed by development.

2) Agent-based model (AB).

During the software development process, each defect has a lifecycle in which it reaches different states. In order to simulate the different states that a defect reaches the agent-based paradigm has been used. Under this approach, we formalize the defects found as agents and their behavior as a state chart that reflects the different states and transitions of defect lifecycle. A description of each state in which the agents can be follows:

- *New*: An agent reaches this state when a defect is reported by the tester for the first time and is yet to be approved.
- *Analyzed*: Once a defect is reported, the manager has to analyze it in order to approve it as a genuine defect, reject or defer it. The agent remains in this state during the time in which this activity takes place. When the activity is done, the information for deciding what to do with the defect is available, and so, the agent moves to the next state, which can be one of the following: a) *Rejected*: If a defect is found to be invalid, b) *Deferred*: If a defect is decided to be fixed in upcoming releases, and c) *Assigned*: If a defect is found to be valid and assigned to a member of the development team to fix it.
- *Fixed*: An agent moves to this state once the developer communicates the defect is fixed. The defect goes to the testing team for validation by injecting a task in the DE model to indicate that the test case that found this defect has to be executed again (retest). The result of this execution will determine the next state of the agent.
- *Closed*. If the tester finds that the defect is indeed fixed and is no more a cause of concern, the agent moves to the state Closed. Otherwise, if the defect is not fixed or partially fixed, the agent will go again to the state Assigned in which the work of a developer working on its fixing will be simulated again.

D. Input Parameters

The input parameters to include in the model largely depend upon the result variables desired and the process abstractions identified. Input parameters allow setting up different scenarios for simulation. The input parameters of the simulation model are the following:

- Software size: Size of the software product under development.
- FPA per Feature: Adjusted Functional Points per feature.
- Number of Test Cases per Feature: Number of test cases that need to be designed and executed per feature.
- Initial number of tasks in Environment Setup. Initial number of tasks that need to be done for the common and global environment setup.
- Estimated Time for Environment Setup. Time estimated to develop each environment setup task.
- Environment Setup Resources. Number of people allocated to the Environment Setup processes.
- Estimated Time for Test Design and Implementation. Time estimated to develop each task of the Test Design and Implementation processes.
- Test Design and Implementation Resources. Number of people allocated to the Test Design and Implementation process.
- Estimated Time for Test Execution. Time estimated to develop each task of the Test Execution processes.
- Test Execution Resources. Number of people allocated to the Test Execution processes.
- Estimated Time for Test Incident Reporting. Time estimated to develop each task of the Test Incident Reporting processes.
- Test Incident Reporting Resources. Number of people allocated to the Test Incident Reporting Processes.
- Estimated time to fix a defect. Time estimated to a fix a defect by a developer.
- Code released for Test Execution. Indicates when the code is released for testing. This value is provided as a percentage of delay measured regarding the initial estimated time for the testing project.
- Probability of finding a defect per Test Case Execution. Probability that a Test Case finds a defect when the test case is executed the first time.
- Probability of finding a defect per Test Case in Retest Execution. Probability that a Test Case finds a defect when the defect has been reported as fixed.

In order to achieve more realistic results, the model accepts a triangular distribution for most of the above input parameters.

V. SIMULATION OPTIMIZATION

Even though simulation runs are useful to visualize the effect of different values of the input parameters in the process performance, that is, to execute what-if scenarios in managerial decision-making, a key benefit can be obtained when we use together simulation and metaheuristic optimization algorithms in a process called simulation optimization. In this case, it is possible to obtain which values need to take the input parameters in order to maximize or minimize an output variable.

This section presents two optimization scenarios regarding the following exploratory questions:

- RQ1: Is it possible to maximize the efficiency of the test process by controlling the moment in which the executable code is available for testing? The optimization will determine the distribution of the human resources that maximizes the Process Efficiency.
- RQ2: Is it possible to minimize the time life span of a defect? (time from detection to closing). The optimization will determine the distribution of the human resources that minimizes the Average Defect Life.

The model implementation and the simulation runs have been performed using Anylogic™ software [33] with the Enterprise Library. The model logic is written in Java. Optimizations have been carried on using the optimizer OptQuest® [34] built-in Anylogic™.

TABLE I. BASE SCENARIO CONFIGURATION

Input parameter	Value
Software size	800 FPA
FPA per Feature	5
Number of Test Case per Feature	(0.5, 2, 4)
Initial number of tasks in Environment Setup	5 tasks
Estimated Time for Environment Setup	(10, 14.4, 20) hours
Environment Setup Resources	1 person
Estimated Time for Test Design and Implementation	(3, 4.5, 6) hours
Test Design and Implementation Resources	4 people
Estimated Time for Test Execution	(1.5, 3.2, 4.5) hours
Test Execution Resources	4 people
Estimated Time for Test Incident Reporting	(1.5, 3, 4.5) hours
Test Incident Reporting Resources	1 person
Estimated time to Fix a defect	(3, 4.5, 6) hours
Code released for Test Execution	15%
Probability of finding a defect per Test Case Execution	(5%, 15%, 25%)
Probability of finding a defect per Test Case in ReTest Execution	(10%, 20%, 30%)

The first step will be to configure a base scenario. Then optimizations will be determined starting from this scenario.

A. Base Scenario Setup

In this scenario, the base simulation is run to determine the values of the result variables and analyze the results of the process. In order to obtain a set of reasonable parameters, we have estimated the costs of the different activities using a set of ratios observed in average risk profiles [35]. We consider functional testing for a system test phase in a project with waterfall development, experienced builders and a structured test approach driven by risk:

- Development process ratios: Ratios of functional design, realization and functional test are 1:2:1.
- Test process ratios: the ratios of test design & implementation, execution, reporting and environment set-up are 50:40:5:5, respectively.

The values of the input parameters in this scenario are displayed in Table I.

B. Base Scenario Run

Once the input parameters of the model have been set to the values shown in Table I, the model is ready to simulate the base scenario. The number of test cases in each state is depicted in Figure 2, which shows that initially 160 test cases were planned for the initial features. At the end of the simulation the total of fulfilled tests is 511 with 416 passed tests (81.41%) and 95 failed tests (18.59%).

Figure 3 depicts the number of defects in each state. At the end of the simulation, 4 of them were rejected and 3 of them deferred; 75 defects were closed and 9 reopened. The Process Efficiency reached with this setting is 91.0%, which is reasonable in practice, showing the consistency of the model when using the above parameters.

Figure 4 and Figure 6 display the time evolution of the number of test cases in each state and the number of defects in each state until the end of simulation, respectively. These figures are included later in the article to facilitate the comparison against the optimization runs. The increasing of the number of test cases in each states (Figure 4) is fairly linear from the moment in which testing begins. The number of defects in each state (Figure 6) follows a different trend, as there is a significant delay from detection of failures to their closing. This is related to the Average Defect Life, which will be optimized later.

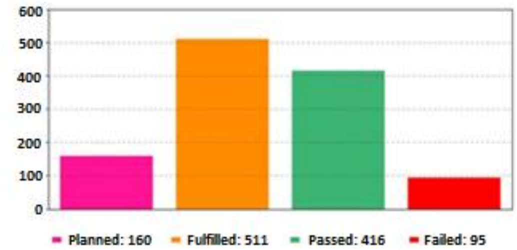

Figure 2. Number of test cases in each state at the end of the simulation.

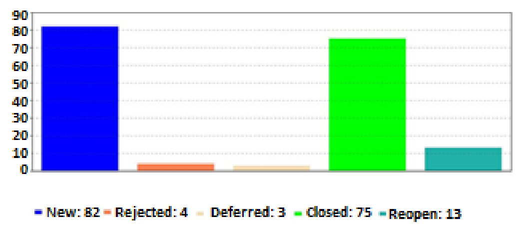

Figure 3. Number of defects in each state at the end of the simulation.

Figure 12 displays the time evolution of the Process Efficiency, which has been defined before as the ratio between the number of closed defects and the number of defects found. At the beginning of the simulation the number of defects found is zero (because test cases are still in preparation), so that the simulator returns 100%. Just after the first test case is available, the efficiency goes to zero as there are not closed defects. After the first defect has been closed, efficiency increases up and grows towards its final value (91.0%).

C. Optimization of the Process Efficiency

To answer RQ1, we ran an optimization experiment to determine whether it is possible to improve the efficiency of the test process by controlling the moment in which the executable code is available for testing. The optimization will determine the distribution of the human resources that maximizes test efficiency when the code is released for testing in a range that varies from 5% to 50% from the moment the testing process begin [1]. Table II displays the input values for the control parameters of the experiment, the constraints imposed and the results obtained in the optimized process compared with the base case.

The results of the optimization experiment show that, under the constraints imposed, it is possible to achieve 97% of efficiency in the process allocating 7 people to the process and having a maximum delay of the code released for testing of 27% of the initial estimated time. This will result into a process that is 97% efficient in closing defects but finishes one month later than the base scenario.

TABLE II. OPTIMIZATION OF THE PROCESS EFFICIENCY COMPARED WITH BASE SCENARIO

Input parameter	**Control Input**	**Result (Base scenario)**	**Result (Optim. scenario)**
Initial number of tasks in Environment Setup	3-5 tasks	5	5
Environment Setup Resources	1-4 people	1	1
Test Design and Implementation Resources	1-4 people	4	2
Test Execution Resources	1-4 people	4	3
Test Incident Reporting Resources	1-4 people	1	1
Code released for Test Execution	5% - 50%	15%	27%
Constraints	**Value**		
Testing Team Size	<= **7** people		
Maximum Testing Time Overrun	<= 1 month		
Process Efficiency obtained (percent)		90%	97%

TABLE III. OPTIMIZATION OF THE AVERAGE DEFECT LIFE COMPARED WITH BASE SCENARIO

Input parameter	**Control Input**	**Result (Base scenario)**	**Result (Optim. scenario)**
Initial number of tasks in Environment Setup	3-5 tasks	5	5
Environment Setup Resources	1-4 people	1	1
Test Design and Implementation Resources	1-4 people	4	2
Test Execution Resources	1-4 people	4	4
Test Incident Reporting Resources	1-4 people	1	3
Code released for Test Execution	5% - 50%	15%	20%
Constraints	**Value**		
Testing Team Size	<= 10 people		
Maximum Testing Time Overrun	<= 1 month		
Average Defect Life (working hours)		28.31	13.91

The conclusion drawn from this particular experiment with regard to the base scenario is that if the project is adequately scheduled, it is possible to reduce the total number of test resources as well as increase the process efficiency.

D. *Optimization of the Average Defect Life*

To answer RQ2 we run an optimization experiment to determine whether it is possible to improve the time span between fault detection and closing by controlling the moment in which the executable code is available for testing (in an range that varies from 5% to 50% as in previous subsection). In this case, the optimization will minimize the Average Defect Life. Table III displays the input values for the control parameters of the experiment, the constraints imposed and the results obtained in the optimized process compared with the base case.

The results of the optimization experiment show that under the constraints imposed, it is possible to reduce by more than a half (down to 13.91%) the Average Defect Life by allocating the same amount of people in a different way to the process and having a maximum delay of the code released for testing of 20% of the initial estimated time.

As in the previous optimization, this case also requires a team size of 10 people allocated to the testing tasks. However, the optimization brings new light regarding the allocation of people to the tasks resulting in a considerable advantage regarding the average defect life.

Now, a comparison on trends of the main variables of the process will be provided. Figure 4 and Figure 5 display the time evolution of the number of test cases in each state for the base and optimized scenarios, respectively. Figure 6 and Figure 7 display the time evolution of the number of defects in each state until the end of simulation for the base and optimized scenarios, respectively. In Figure 7, it can be seen that there is a shorter delay between the moment in which failures are detected and their closing, at the expenses of a larger test time.

Figure 8 and Figure 9 display the values of the Average Defect Life (base and optimized scenario, respectively). In the optimized scenario, the variable starts growing earlier than in the base scenario, but with a lower maximum value. Just after reaching the maximum begins a continuous decrease until its optimum value (13.9 working hours) is achieved, a much lower valued than the corresponding value for the base scenario (127.4 working hours). The Average Defect Cost (ratio between number of defects closed and total time spent) is displayed in Figure 10 and Figure 11, showing similar trends and final values of 6.2 working hours (base scenario) and 5.6 hours (optimized scenario).

To finish, a comparison of the Process Efficiency is provided in Figure 12 and Figure 13. Process Efficiency at the end (89.0%) is marginally lower than in the base scenario (90.0%), since this optimization is intended to minimize the average defect life, but it begins growing at earlier stages of the testing project.

Other simulations can help find the best input values for project schedule, resource allocation and quality objective from among all that lead to the optimization of the key process outputs. Moreover, the results of optimizations presented in this paper have been performed separately, but this makes room for future explorations in multi-objective optimizations. For example, in order to balance the maximization of variables like Process Efficiency as well as the minimization of variables like Average Defect Life.

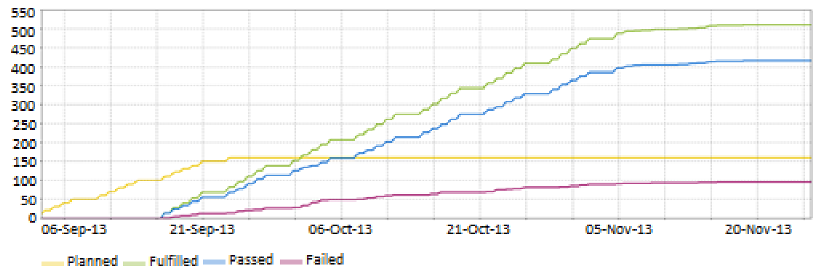

Figure 4. Time evolution of the number of test cases in each state (base scenario).

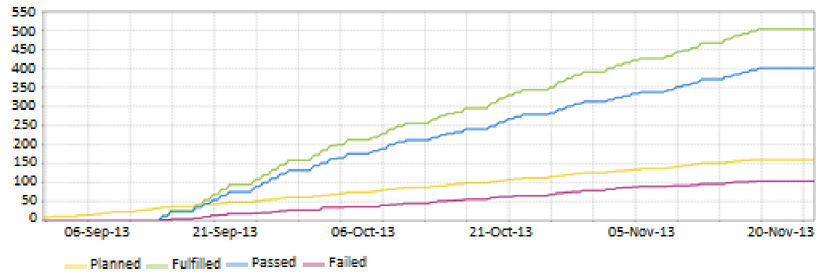

Figure 5. Time evolution of the number of test cases in each state (optimized scenario).

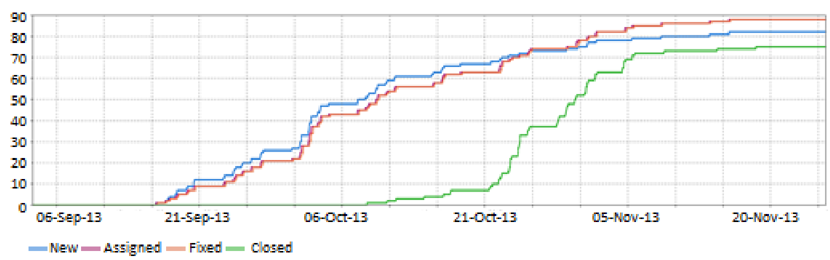

Figure 6. Time evolution of the number of defects in each state (base scenario).

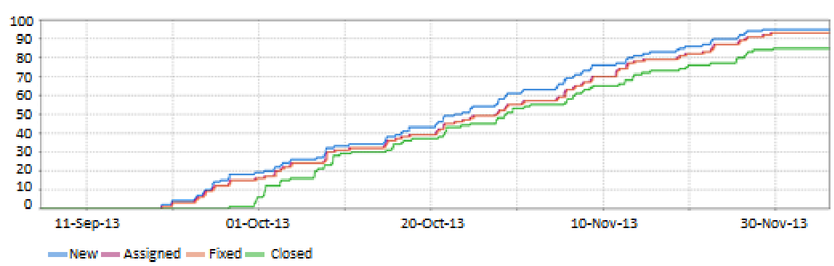

Figure 7. Time evolution of the number of defects in each state (optimized scenario).

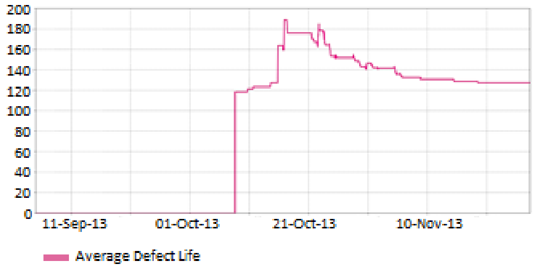

Figure 8. Time evolution of average defect life (base scenario).

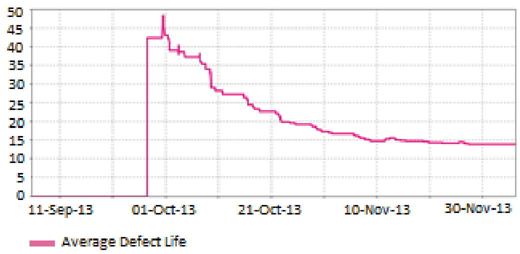

Figure 9. Time evolution of the average defect life (optimized scenario).

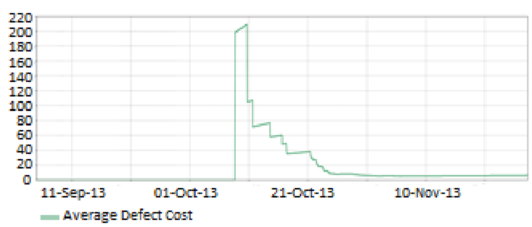

Figure 10. Time evolution of the average defect cost (base scenario).

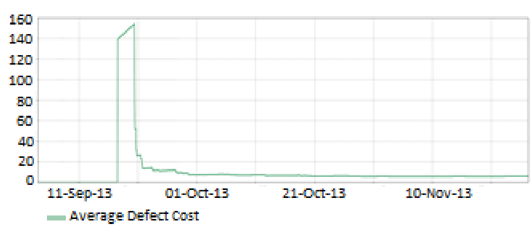

Figure 11. Time evolution of the average defect cost (optimized scenario).

VI. CONCLUSION AND FURTHER WORK

This paper presented a simulation model for the dynamic testing processes that allows a seamless integration between the testing and development processes. The model is devised as a multi-paradigm model composed by a discrete event simulation model, to simulate the execution of the dynamic test processes, and an agent-based simulation model, to in-depth simulate the defects life cycle. The model has been first used to simulate a base scenario. The results of the simulation runs were then used to design two simulation

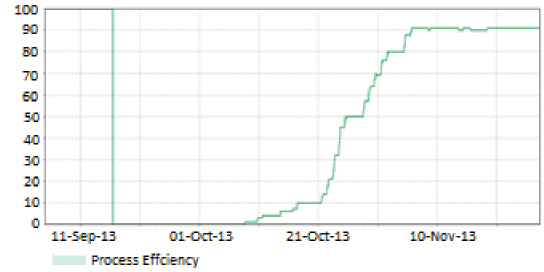

Figure 12. Time evolution of the process efficiency (base scenario).

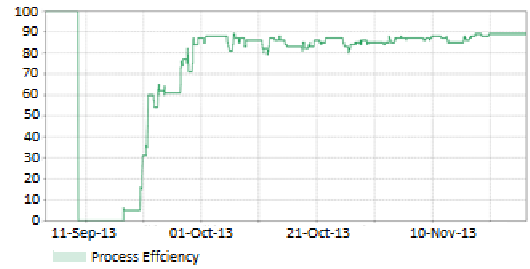

Figure 13. Time evolution of the average process efficiency (optimized scenario).

optimization scenarios. By merging simulation and optimization it is possible to use the model to find the best testing team configuration so that key process metrics are optimized. Results show that the simulation model can be effectively used to optimize different process metrics (Test Process Efficiency and Average Defect Life) and then help managers to achieve a trade-off between cost, schedule and quality.

This work is a first step in the use of multi-paradigm simulation models for testing management. Further work will include, although not limited to, the consideration of agent-based models to simulate parts of the dynamic test processes, the integration into a more complex project development simulation model [36], multi-objective optimization and experimentation in different projects using different lifecycle models and including different test levels of testing. After calibrating and validating the model with historical data from the industry, it will be also possible to exploit it as an operating tool for decision-making in the industrial domain.

ACKNOWLEDGMENTS

This work has been partially supported by the Spanish Ministry of Science and Technology with ERDF funds under grants TIN2010-20057-C03-03, TIN2010-20057-C03-01, TIN2013-46928-C3-2-R and TIN2013-46928-C3-1-R.

REFERENCES

[1] M. Ruiz, J. Tuya, and D. Crespo, "Simulation-based management for software dynamic testing processes," Proceedings of the 7th International Conference on Software Engineering Advances (ICSEA 2012), IARIA 2012, Lisbon, pp. 630-635.

[2] E. van Veenendaal (ed), Standard glossary of terms used in software testing, Version 2.1, International Software Testing Qualifications Board, Oct. 2010.

[3] RJ. Madachy, Software process dynamics. John Wiley & Sons, Inc., 2008.

[4] K. Nogeste and DHT. Walker, "Using knowledge management to revise software-testing processes," Journal of Workplace Learning, 2006, 18(1), pp. 6-27.

[5] R. Abdullah, ZD. Eri, and AM. Talib, "A model of knowledge management system in managing knowledge of software testing environment," 5th. Malaysian Conference in Software Engineering, (MySEC 2011), 2011, pp. 229-233.

[6] J. Andrade, J. Ares, M. Martínez, J. Pazos, S. Rodríguez, J. Romera, and S. Suárez, "An architectural model for software testing lesson learned systems," Information and Software Technology, 2013, vol. 55, no. 1, pp, 18-34.

[7] X. Liu, G. Gu, Y. Liu, and J. Wu, "Research and implementation of knowledge management methods in software testing process," WRI World Congress on Computer Science and Information Engineering, (CSIE 2009), 2009, pp. 739-743.

[8] Z. Bluvband, S. Porotsky, and M. Talmor. "Advanced models for software reliability prediction," Proceedings - Annual Reliability and Maintainability Symposium, 2011, pp. 1-5.

[9] L. Lazić and N. Mastorakis, "The COTECOMO: COnstractive test effort COst Model," N. Mastorakis and V. Mladenov (Eds) Proceedings of the European Computing Conference, vol. 2, Series: Lecture Notes in Electrical Engineering, 2009, vol. 27, pp. 89-110.

[10] SD. Kanawat, A. Pandey, A. Singh, and A. Maloo, "Software testing model for quality," Advanced Materials Research, 2001, vol. 4507, pp. 403-408.

[11] L. Xin-Ke and Y. Xiao-Hui, "A goal-driven measurement model for software testing process," WRI World Congress on Software Engineering, (WCSE 2009), 2009, vol. 4, pp. 8-12.

[12] L. Lazić, "Software testing optimization by advanced quantitative defect management," Computer Science and Information Systems, 2010, 7(3), pp. 459-487.

[13] J. Saldaña-Ramos, A. Sanz-Esteban, J. García-Guzmán, and A. Amescua, "Design of a competence model for testing teams," IET Software, 2012, 6(5), pp. 405-415.

[14] A. Farooq, K. Georgieva, and RR. Dumke, "A meta-measurement approach for software test processes," Proceedings of the 12th. IEEE International Multitopic Conference (IEEE INMIC 2008), 2008, pp. 333-338.

[15] X. Li and W. Zhang, "The PDCA-based software testing improvement framework," Proceedings of the 2010 International Conference on Apperceiving Computing and Intelligence Analysis, (ICACIA 2010), 2010, pp. 490-494.

[16] L. Han, "Evaluation of software testing process based on bayesian networks," Proceedings of the 2010 International Conference on Computer Engineering and Technology, (ICCET 2010), 2010, 7, pp. V7361-V7365.

[17] D. Zhang, C. Nie, and B. Xu, "Cross-entropy method based on Markov decision process for optimal software testing," Ruan Jian Xue Bao/Journal of Software, 2008, vol. 19, no. 10, pp. 2770-2779.

[18] Q. Li, Y. Yang, M. Li, Q. Wang, BW. Boehm, and C. Hu, "Improving software testing process: Feature prioritization to make winners of success-critical stakeholders," Journal of Software: Evolution and Process, 2012, vol. 24, no. 7, pp. 783-801.

[19] K. Cai, Z. Dong, and K. Liu, "Software testing processes as a linear dynamic system," Information Sciences, 2008, vol. 178, no. 6, pp. 1558-1597.

[20] HKN. Leung, "Improving the testing process based upon standards," Software Testing Verification and Reliability, 1997, vol. 7, no. 1, pp. 3-18

[21] L. Lazić and N. Mastorakis, "RBOSTP: Risk-based optimization of software testing process. Part 1," WSEAS Transactions on Information Science and Applications, 2005, vol. 2, no. 6, pp. 695-708.

[22] L. Lazić and N. Mastorakis, "RBOSTP: Risk-based optimization of software testing process. Part 2," WSEAS Transactions on Information Science and Applications, 2005, vol. 2, no. 7, pp. 902-916.

[23] L. Lazić and N. Mastorakis, "Integrated intelligent modeling, simulation and design of experiments for Software Testing Process," Proceedings of the International Conference on Computers, 2010, vol. 1, pp. 555-567.

[24] L. Lazić and N. Mastorakis, "The use of modeling & simulation-based analysis & optimization of software testing," WSEAS Transactions on Information Science and Applications, 2005, vol. 2 no. 11, pp. 1918-1933.

[25] L. Lazić and D. Velasěvić, "Applying simulation and design of experiments to the embedded software testing process," Software Testing Verification and Reliability, 2004, vol. 14, no. 4, pp. 257-282.

[26] WM. Zhang, BS. Zhou, and WJ. Luo, "Modeling and simulating of sequential iterative development processes," Jisuanji Jicheng Zhizao Xitong/Computer Integrated Manufacturing Systems, (CIMS 2008), 2008, vol. 14, no. 9, pp. 1696-1703.

[27] C. Lizhi, T. Weiqin, B. Zhou, and J. Zhang J, "Modeling software testing process using HTCPN," Fourth International Conference on Frontier of Computer Science and Technology, (FCST 2009), 2009, pp. 429-434.

[28] K. Saurabh, "Modeling unit testing processes: a system dynamics approach," Proceedings of the 10th International Conference on Enterprise Information Systems (ICEIS 2008), 2008, vol. ISAS 1, pp. 183-186.

[29] JS. Collofello, Y. Zhen, JD. Tvedt, D. Merrill, and I. Rus, "Modeling software testing processes," Proceedings of the International Phoenix Conference on Computers and Communications, 1996, pp. 289-293.

[30] ISO/IEC/IEEE 29119-2:2013 Software and Systems Engineering - Software Testing – Part 2: Test processes. August 2013.

[31] MI. Kellner, RJ. Madachy, and DM. Raffo, "Software Process Modeling and Simulation: Why, What, How?," Journal of Systems and Software, April, 1999, Vol. 46, no. 2/3.

[32] R. Black, "Managing the testing process: practical tools and techniques for managing hardware and software testing," Wiley Publishing, 2002.

[33] XJ Technologies. Anylogic™. http://www.anylogic.com/ [retrieved: May, 2014] .

[34] OpTek Systems, Inc. OptQuest®. http:// www.opttek.com/ [retrieved: May, 2014].

[35] T. Koomen, L. van der Aalst, B. Broekman, and M. Vroon, "TMap Next for result-driven testing," UTN Publishers, 2007.

[36] D. Crespo and M. Ruiz, "Decision making support in CMMI process areas using multiparadigm simulation modeling," 2012 Winter Simulation Conference (WSC 2012), 2012, pp. 1-12.