

Combining Process Simulation and Orthogonal Defect Classification for Improving Software Dependability

Ioana Rus

Fraunhofer Center for Experimental Software Engineering, Maryland
irus@fc-md.umd.edu

I. INTRODUCTION

Dependability can be regarded from multiple perspectives, depending on the stakeholder that defines it. Dependability can also have many facets (e.g. reliability, availability, security, or safety) depending on the application domain.

We will focus here on the “internal” perspective of dependability, that is a developer’s view of software dependability (predicted during development) and we will show how it can relate to the external perspective (perceived by the user and customer during system’s operation). Defects that are introduced and not detected during development, if activated in the delivered product, might impact on the behavior of the system with respect to dependability properties. For that reason, these defects must be identified and removed.

A manager has to achieve dependability requirements, dealing with schedule and cost constraints. By finding the right allocation of effort and resources to different activities in a project, in the specific context of an organization and project, the manager can reach the balance for achieving multiple goals. How to make these decisions is, however, difficult and relies to a great extent on personal experience. To support this decision-making task, we propose the use of process modeling and simulation, combined with historical data from the organization, provided by an Orthogonal Defect Classification (ODC) based defect data collection [1].

II. DEFECT MODELING AND SIMULATION

Modeling of a software development process can improve understanding of the process and communication between stakeholders. Process simulation can help managers predict the effect of changes on product and project parameters. A process simulator, once calibrated for a specific project can be used to support planning, tracking, and re-planning. By running *what-if* scenarios and sensitivity analyses, one can examine different alternatives and select the one that most suits the project at hand, and also see what

are the changes that have more impact - either positive or negative, and could lead to success or failure.

We developed a discrete event model of a software development process, focusing on the evolution of defects, i.e., their introduction, detection, and removal [2]. This modeling approach allows different attributes to be attached to the entities modeled - the defects in our case. For example, each defect can be characterized by its type, severity, and effort to be detected or fixed. The values of these attributes are set or changed during the simulation, when a related event happens (e.g. when a defect is discovered the *type* attribute is set to the appropriate value).

This model also captures the structure of the process, activities, resources, flows of artifacts and defects, parameters such as resource allocation, duration, effort and cost for each activity, and relationships between these parameters. The values for the product and process parameters can be deterministic or random (probabilistic distributions).

The simulator has to be first calibrated with values specific for an organization (such as expected productivity, quality of work expressed by defect generation, or effort for detecting and fixing a defect) and then executed with values specific for the project (such as size of the task and available resources). The outputs are cost, effort, schedule, and remaining defects, giving an indication about how dependable the software will be and how costly it is to reach that level of dependability.

III. ORTHOGONAL DEFECT CLASSIFICATION

Orthogonal Defect Classification (ODC) is a measurement concept for software development that uses the defect stream as a source of information on the product and the development process [1]. There are two classes of defect attributes that ODC suggests to be collected: one class associated with the discovery of the defect (activity, trigger, and impact) and another one with its removal (target, defect type, qualifier, source, and age).

IV. DISCRETE EVENT MODELING USING ODC ATTRIBUTES

Some of the parameters of our model overlap with the ODC attributes so, if ODC data are collected, then they could be used for calibrating the model, and also as expected values for the inputs of the simulator. Such ODC data are the *activity* and the *target*. We would need additional data that is not addressed by ODC, such as the expected number of defects introduced by the developers in the software and the effort for discovering and removing a defect (depending on its attributes). Also some projects record the severity of defects, attribute that is not addressed by ODC.

On the other hand, by looking at the ODC attributes that are not currently used by our model, we get an indication on how it could be refined and include factors that are of concern to projects. For example, 1) add more defect attributes, so that *triggers* could be captured if needed; 2) modify the assumptions and structure of the model to allow for a regeneration of defects that would capture the *age* ODC attribute; and 3) change the assumption that all software is developed from scratch, thus being able to capture and use the *source* ODC attribute.

V. USAGE OF THE MODEL

By using the simulator combined with ODC data, we can connect the external dependability perception with an internal perspective. That can be done by focusing only on defects that have as *impact* specific dependability attributes, e.g. *reliability*, or *security*. We can also examine how changes in the development process will impact the defects (internal view) and therefore the corresponding dependability properties (external view).

For example, for a given organization with historical knowledge about their capability in terms of ODC profile (such as effectiveness of V&V activities for different types of defects), plus defect introduction (by *type* or *trigger*), and defect detection and removal effort, the simulator will help answer questions such as: What changes in the process should be made in order to efficiently achieve the desired dependability objectives? More specifically: Should more resources or time be allocated to design reviews, to code inspections, or to unit testing? Should developers be trained for using new reading or testing techniques? Should test automation be introduced or increased? Should developer's coding skills be improved, or new

approaches be used (such as pair programming) so that fewer defects would be generated? How much each of these changes would cost, in terms of money and delays in product release? What changes would have the greatest impact on achieving the goal, or the best ROI? What improvements are worth making?

As an example, Figure 2 tracks the cost of V&V and rework activities throughout the development life-cycle, while Figure 1 shows how effective the V&V activities were for detecting defects that impact on the reliability and security of the system.

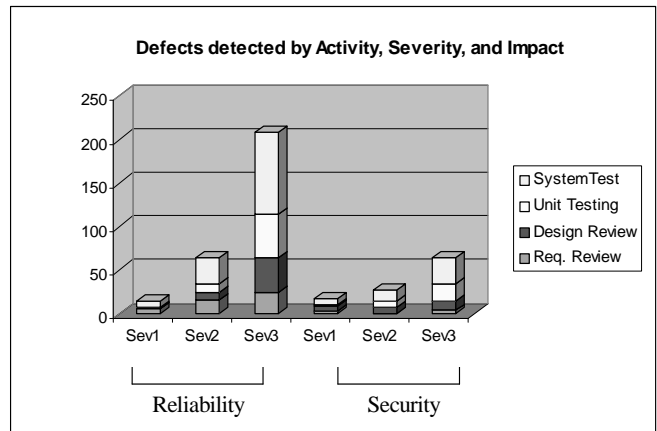


Figure 1. Profile of detected defects and their contribution to dependability properties

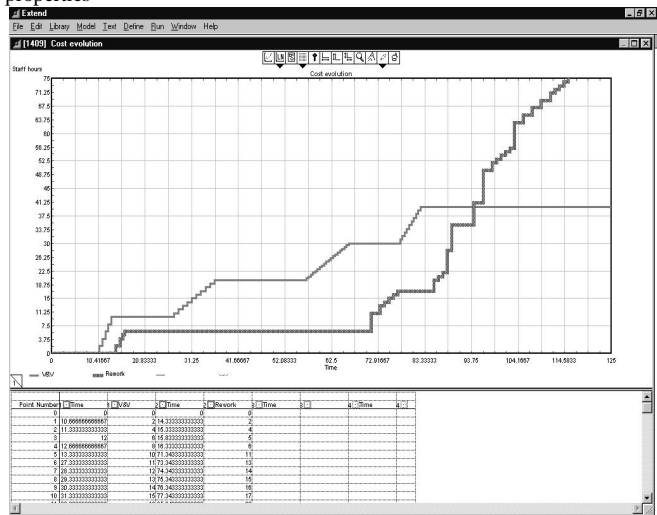


Figure 2. Staff hours spent for V&V and rework

REFERENCES

- [1]. Chillarege, Ram, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S. Moebus, B. K. Ray, M-Y Wong, "Orthogonal Defect Classification—A Concept for In-Process Measurements", *IEEE Transactions on Software Engineering*, Vol 18, No. 11, Nov 1992.
- [2]. Rus, Ioana, S. Biffi, and M. Halling, "Systematically Combining Process Simulation and Empirical Data in Support of Decision Analysis in Software Development", Workshop on Software Engineering Decision Support, in *Proceedings of SEKE2002*, Ischia, Italy, July 2002.