Chapter 81
A Game Theoretic Approach for Quality Assurance in Software Systems Using Antifragility-Based Learning Hooks

Vimaladevi M.
Pondicherry Engineering College, India

Zayaraz G.
Pondicherry Engineering College, India

ABSTRACT

The use of software in mission critical applications poses greater quality needs. Quality assurance activities are aimed at ensuring such quality requirements of the software system. Antifragility is a property of software that increases its quality as a result of errors, faults, and attacks. Such antifragile software systems proactively accepts the errors and learns from these errors and relies on test-driven development methodology. In this article, an innovative approach is proposed which uses a fault injection methodology to perform the task of quality assurance. Such a fault injection mechanism makes the software antifragile and it gets better with the increase in the intensity of such errors up to a point. A software quality game is designed as a two-player game model with stressor and backer entities. The stressor is an error model which injects errors into the software system. The software system acts as a backer, and tries to recover from the errors. The backer uses a cheating mechanism by implementing software Learning Hooks (SLH) which learn from the injected errors. This makes the software antifragile and leads to improvement of the code. Moreover, the SLH uses a Q-Learning reinforcement algorithm with a hybrid reward function to learn from the incoming defects. The game is played for a maximum of K errors. This approach is introduced to incorporate the anti-fragility aspects into the software system within the existing framework of object-oriented development. The game is run at the end of every increment during the construction of object-oriented systems. A detailed report of the injected errors and the actions taken is output at the end of each increment so that necessary actions are incorporated into the actual software during the next iteration. This ensures at the end of all the iterations, the software is

DOI: 10.4018/978-1-6684-3702-5.ch081

immune to majority of the so-called Black Swans. The experiment is conducted with an open source Java sample and the results are studied selected two categories of evaluation parameters. The defect related performance parameters considered are the defect density, defect distribution over different iterations, and number of hooks inserted. These parameters show much reduction in adopting the proposed approach. The quality parameters such as abstraction, inheritance, and coupling are studied for various iterations and this approach ensures considerable increases in these parameters.

1. INTRODUCTION

Software applications are becoming more complex day by day and it is difficult to maintain code quality and manage the cost of the software development. Some of the factors that make this quality-cost balance a challenging task needs further discussion. They are the growing pressure on the software organizations, rise of the developmental costs, need to get the product to market quickly and accelerated development schedules. The most effective way to keep the development cost down is the minimization and the introduction of defects. The software bug cost of United States economy has increased from $59.5 billion to $1.1 trillion from 2002 to 2016. This increase in cost is due to the loss in revenue due to the software being unusable, payments to developers for bug fixing, loss in shareholder value, etc. Also, there are some indirect financial costs arising due to the problem of brand reputation and customer loyalty. The bug fixing process even interferes with other developments and enhancements for new functionality addition that ultimately affect the project schedule. It is critical to catch the defects early since, the cost of fixing the defects increases exponentially as the software progresses through the life cycle phases. From the report of National Institute of Standards and Technology (NIST), the increase in the bug fix follows the trend as shown in Table 1 (National Institute of Standards and Technology, 2002). Here, X is the normalized unit of cost and can be expressed in terms of person-hours.

Table 1. Cost of defect fixing

Design	1X
Implementation	5 X
Integration Testing	10 X
Customer Beta Testing	15 X
Post Product Release	30 X

Hence, there is an important need for proactive approaches to improve the overall quality and decrease the software development cost. This research work discusses such an arrangement to proactively detect defects by building antifragile characteristics into an object-oriented software within the existing software development framework. But this defect prevention is a challenging task. The operating environment and the kinds of failure and recovery of a software system are highly uncertain and are open ended. For example, an information report states that the Eclipse development environment runs on at least 5 million different machines. The developer foreseeing all possible failures is nearly impossible,