Chapter 2
Swarm Intelligence–Empowered Bug Prediction Strategy for Decision Support in Software Defect Prediction

Medhunhashini D. R.
Sri Krishna Arts and Science College, India

Jeen Marseline K. S.
Sri Krishna Arts and Science College, India

ABSTRACT

Swarm intelligence is inherent in many living things and is inspiring new ways of thinking among computer scientists. Scientists from all walks of life including software corporates are interested in it because of its ties to collective intelligence behaviour. Bugs are an expensive and quality killer in software development. The development of DP models was driven by the critical need to predict defects early on. Classifying modules as either defect-prone or non-defect-prone relies heavily on machine learning algorithms. Improving software quality relies on defect prediction. SI improves the accuracy and efficacy of bug predictions by modelling their actions after the social group behaviour of insect colonies. The objective of this chapter is to outline swarm intelligence-based bug prediction in order to assist software engineers and QA teams with increased accuracy.

1. INTRODUCTION

The success of any software application depends on making sure the quality has not been compromised in any way. Accomplishing quality requires a rigorous defect prediction mechanism. The continuous growth of the software features and its changing nature it makes the prediction challenging with traditional software defect prediction models. Software development is a complex procedure that raises bugs at the final stage making it the most pervasive part. These defects may jeopardize system functionality

DOI: 10.4018/979-8-3693-2073-0.ch002

or even put users' security at risk, among other serious consequences. The intricacy of defect prediction is increased by the dynamic nature of software systems, changing requirements, and frequent updated (Guo et al., 2023).

1.1 Software Engineering

The complex field of software engineering starts with a thorough examination of requirements, which is followed by stakeholder involvement in defining features and limitations. The software's architecture and structure are described in a high-level system design. The coding process takes this idea and turns it into code that can be executed by using languages such as Python or Java. Subsequently, comprehensive testing is conducted, including unit, integration, and system testing, to detect and fix errors and guarantee conformity with predetermined standards. The software is released for use after successful testing, and it is essential to perform continuing maintenance to fix bugs, add upgrades, and adjust to new requirements. Agile methodologies are commonly used in modern software engineering (Butt et al., 2023). These methodologies stress collaborative and iterative development, and they include practices such as continuous integration/deployment, documentation, security considerations, and version control systems like Git. Software systems that are dependable, scalable, and of high quality can be more easily created when team members effectively communicate, use collaboration tools, and follow best practices (Pereira et al., 2021).

1.2 Software Quality

The word "software quality" refers to the overall level of perfection of a piece of software, which is dependent on a number of factors. Software quality is a reflection of how effectively a software system satisfies the defined criteria and the expectations of its users. Structural quality is concerned with the inner workings of the system and its architecture, with an emphasis on things like scalability, maintainability, and modularity, whereas functional quality is concerned with how well the system does its designated functions. Overall quality is greatly influenced by non-functional elements like as performance, reliability, and security. These aspects guarantee that the program runs well in all kinds of situations. Software quality is heavily dependent on the development process, making it imperative to follow best practices, code standards, and thorough testing procedures (Prabha & Shivakumar, 2020).

There are many steps in the Software Development Life Cycle (SDLC) that must be carefully attended to to maintain quality standards. These steps include requirements analysis, design, coding, testing, and deployment. Unit, integration, system, and user acceptance testing are all parts of rigorous testing that help find and fix bugs so the product works as it should. Automated testing, code reviews, and routine inspections are all examples of continuous quality assurance procedures that help find and avoid errors early on. Maintenance, upgrades, and adaptation to changing requirements are also part of quality assurance, which goes beyond development. Because the perceived value of software is heavily influenced by factors like an easy-to-navigate interface and the capacity to meet or surpass expectations, customer satisfaction is a crucial criterion for assessing software quality. By including continuous improvement and feedback loops, which are common in agile approaches, developers can respond quickly to changing requirements and user needs, which in turn enhances software quality. In the end, achieving software quality isn't a one-and-done deal; it's a continuous process that calls for a comprehensive strategy that