A SIR Model Approach to Software Bug Dynamics and QA Efficiency

Research Report

Yann Fonkoue
yann.fonkoue@facsciences-uy1.cm

January 11, 2026

Abstract

Over the past few decades, software has become an indispensable tool, solving complex problems across diverse sectors such as finance, education, and healthcare. This massive adoption has heightened the focus on Software Quality Assurance (QA). However, increasing system complexity leads to the emergence of bugs that degrade quality, leaving organizations struggling to define optimal strategies for their maintenance teams. We propose an approach based on the SIR (Susceptible-Infected-Recovered) mathematical model to evaluate bug propagation within software architectures and assess QA team performance. Experiments conducted on a synthetic dataset reveal a critical dynamic where the maximum infection rate reaches over 30%, significantly outstripping recovery capabilities capped at 15%. This imbalance mathematically illustrates a phase of software instability. Our results demonstrate the relevance of this framework in quantifying resolution efficiency over time. This study serves as a foundational step toward data-driven decision-making in software quality management.

1 Introduction

Over the past few decades, organizations across various sectors have massively adopted software systems as indispensable tools for solving complex problems. This growing integration now places Quality Assurance (QA) at the heart of software publishers' concerns, for both reliability and economic performance reasons.

However, the constant increase in the complexity of software architectures promotes the emergence of critical anomalies, commonly known as bugs. These failures directly impact the quality of the final product and complicate the strategic decision-making of organizations regarding the management of their maintenance teams. Understanding the dynamics of bug propagation within an architecture therefore becomes a crucial challenge for evaluating the effectiveness of correction processes.

The SIR (Susceptible-Infected-Recovered) epidemiological model, initially theorized by Kermack and McKendrick (1927) [2], offers a rigorous mathematical framework for analyzing these dynamics. The analogy between biological propagation and digital entities has been validated by pioneering works, notably those of Kephart and White (1991) [1], demonstrating that computer systems follow similar epidemiological laws.

In this research project, we propose to adapt this model to simulate bug propagation and quantify the performance of engineering and QA teams in resolving them. The objective is to identify critical thresholds of software stability through the analysis of infection and recovery rates.

The remainder of this document is organized as follows: Section 2 details the mathematical methodology and implementation, Section 3 presents the results obtained through simulation, Section 4 analyzes these results through a critical discussion, and finally, Section 5 presents the conclusion and perspectives of this work.

2 Methods

2.1 The SIR Model: From Biology to Software

The SIR model constitutes a fundamental mathematical framework initially introduced by Kermack and McKendrick (1927) [2]. Designed to model the spread of infectious diseases within a population, this model is based on a so-called "compartmental" approach in which the overall population is subdivided into three distinct and mutually exclusive categories. The first category comprises Susceptible individuals (S), defined as healthy subjects who are nonetheless liable to contract the disease. The second category concerns Infected individuals (I), referring to those who carry the disease and are capable of actively transmitting it. Finally, the third category consists of Recovered individuals (R), representing those who have recovered and have acquired lasting immunity.

This theoretical structure later found resonance in the technological domain through the work of Kephart and White (1991) [1]. These authors adapted this framework to computer science in order to model the propagation of software viruses, replacing social interactions with data exchanges occurring within a directed graph.

2.2 Application to Software Defect Dynamics

Building on these foundations, the present approach transposes the SIR model to the domain of software quality assurance. In this context, the variable S represents portions of code or modules that are still healthy but vulnerable to the emergence of defects due to coupling effects. The variable I denotes modules infected by active anomalies, while the variable R corresponds to modules whose bugs have been corrected, tested, and validated by the QA team. The dynamics governing the propagation and resolution of these defects are then driven by a system of interdependent differential equations.

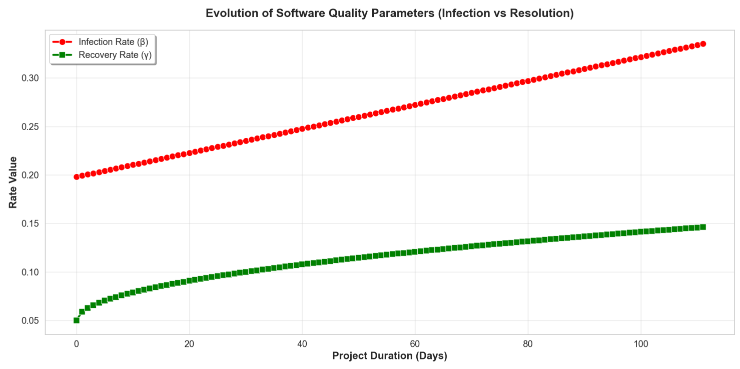
The first equation, $\frac{dS}{dt} = -\beta S \frac{I}{N}$, describes the flow of healthy modules becoming defective at time t. Here, the parameter β embodies the transmission rate, often reflecting software complexity or coupling, while N represents the total size of the system. The second equation, $\frac{dI}{dt} = \beta S \frac{I}{N} - \gamma I$, expresses the net variation in the number of active bugs, where the parameter γ represents the resolution rate and serves as a direct indicator of the maintenance team's efficiency. Finally, the evolution of the stock of validated fixes is given by $\frac{dR}{dt} = \gamma I$, thereby quantifying the cumulative effort devoted to stabilizing the system.

2.3 Stability and Implementation

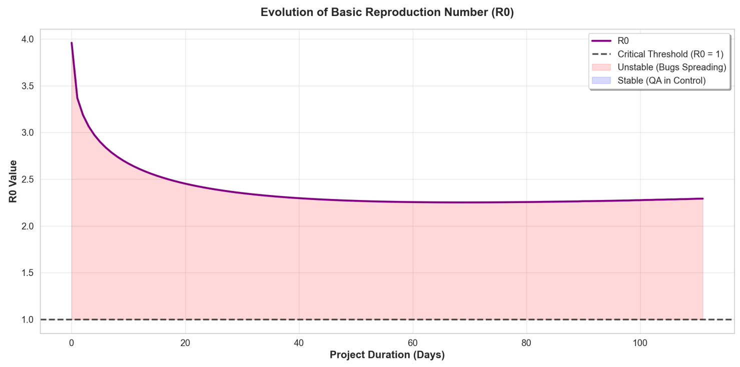
Within this modeling framework, a crucial metric emerges in the form of the basic reproduction number, defined by the ratio $R_0 = \frac{\beta}{\gamma}$. This coefficient expresses the intrinsic speed at which anomalies propagate relative to the team's capacity to resolve them. According to the stability theory developed by Kermack and McKendrick [2], bug propagation ceases once the density of susceptible modules falls below the inverse of the reproduction rate, that is, when $\frac{S}{N} < \frac{1}{R_0}$. This condition marks the tipping point toward definitive software stabilization.

To validate this approach, an implementation was developed in the Python programming language. It relies on a synthetically generated dataset produced via scripting, enabling the simulation of various quality assurance performance scenarios and the observation of the system's response to variations in the resolution rate.

3 Results



(a) Infection vs resolution



(b) Evolution of R_0

Figure 1: SIR model dynamics and QA efficiency indicators

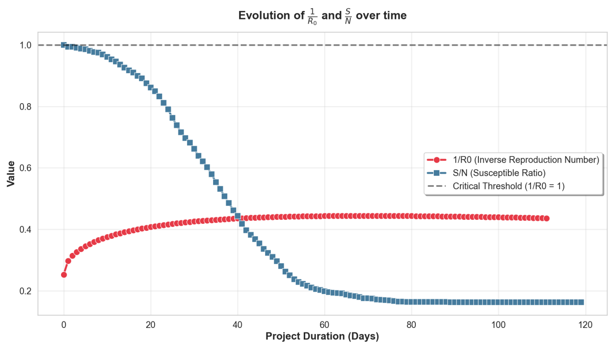


Figure 2: Evolution of $\frac{1}{R_0}$ and $\frac{S}{N}$ over time

4 Discussion

The analysis of the experimental results highlights the complex dynamics underlying the propagation of software defects. Figures 1a and 1b respectively present the temporal evolution of the infection rate relative to the resolution rate, as well as the trajectory of the R_0 indicator. Figure 1a shows a sustained growth in the infection rate, peaking at a value exceeding 30%. Simultaneously, the resolution curve, although increasing, plateaus at around 15%. This significant gap between the rate at which bugs emerge and the processing capacity of the QA team reflects a systemic instability of the software. This situation reveals a clear difficulty for the maintenance team in containing the spread of anomalies, leading to an accumulation of technical debt over time.

An examination of Figure 1b corroborates this interpretation through an analysis of the basic reproduction number. Although a certain decline in this indicator is observed due to corrective efforts, the value of R_0 remains consistently above the critical threshold of 1. Mathematically, this state confirms that each bug potentially generates more than one secondary infection within the system, thereby sustaining an active epidemic phase. Furthermore, Figure 2, which compares the ratios $\frac{1}{R_0}$ and $\frac{S}{N}$, shows convergence toward a normalized stability point. According to these projections, bug propagation will fully cease only when the system reaches an equilibrium point estimated at 42%, at which point the density of susceptible modules will be sufficiently low to halt contagion.

However, it should be emphasized that this approach entails certain limitations inherent to its generalist nature. Real-world software architectures exhibit coupling heterogeneities that the classical SIR model, due to its homogeneous mixing assumption, does not fully capture. A specific modular structure may therefore bias the results relative to uniform propagation. Nevertheless, the results obtained validate the relevance of the epidemiological analogy for software quality assurance, thereby providing software publishers with a robust statistical tool to guide maintenance strategies and anticipate phases of major instability.

Code Availability

The Python implementation of the SIR-based software defect model, along with all experimental scripts and synthetic datasets, is publicly available at: `https://github.com/yann-fk-21/-SIR-Model-Approach-to-Software-Bug-Dynamics-and-QA-Efficiency`.

5 Conclusion

At a time when software plays a central role in addressing complex organizational challenges, quality assurance has become a major strategic concern for software publishers. This research project aimed to propose an innovative approach, based on the epidemiological SIR model, to model the propagation of defects within a software system and, by extension, to evaluate the performance of testing (QA) teams.

The experiments conducted revealed a critical dynamic: the infection rate within the system reached peaks exceeding 30%, while the maximum resolution rate plateaued at 15%. This imbalance between bug generation and correction capacity confirms the relevance of our model in identifying phases of software instability. Although these results validate the coherence of our methodology, it should be noted that the approach remains generalist in nature. The current model may exhibit biases depending on the specific characteristics of the software architectures encountered, as it does not yet account for the exact topology of inter-module dependencies.

In conclusion, this work represents a first milestone toward a more rigorous and mathematically grounded decision-making process in the field of software quality assurance. It paves the way for future research aimed at integrating more complex graph structures, thereby refining the long-term predictability of software health.

References

[1] J.O. Kephart and S.R. White. “Directed-graph epidemiological models of computer viruses”. In: *Proceedings. 1991 IEEE Computer Society Symposium on Research in Security and Privacy*. 1991, pp. 343–359. DOI: `10.1109/RISP.1991.130801`.

[2] W. O. Kermack and A. G. McKendrick. “Contributions to the Mathematical Theory of Epidemics–II. The Problem of Endemicity”. In: *Bulletin of Mathematical Biology* 53.1–2 (1991), pp. 57–87. DOI: `10.1007/BF02464424`.