Javier Ron

Introduction to research area

Software reliability can be described as a measure related to errors in a software system. A system is said to be fully reliable when users do not perceive any incorrect behavior, even in the presence of system faults [2]. The main focus of my research is to improve reliability of software artifacts. To achieve this, we identify specific application-level properties which can be leveraged to increase reliability. As an example, in application domains that are protocol-driven (e.g. web browsers, blockchains, etc.), many different but inter-operable implementations of said protocols might exist. In that case we say that there is a natural diversity of implementations, which do not share design concerns, software stacks, or software supply chains. A way to take advantage of diversity is to build n-version execution units [1]. An n-version execution unit can be seen as a single artifact which internally executes several implementations simultaneously. Assuming that the used implementations have different fault profiles, the outputs of the internal implementations are compared and evaluated for integrity and correctness, and then presented to the user. This in turn allows to detect and hide any erratic behavior of single implementations to the system's user.

Topics on Software Engineering

Automated software testing

Automated software testing is a process to automatically and methodically validate the correct behavior of a program. Automatic testing is tightly related to modern software engineering methodologies and processes (DevOps, test driven development, fault injection testing, etc).

Automated testing is also a central part of continuous integration (CI). The use of version control systems allows for every feature or change introduced to the code base to be tested. This ensures not only its correct functionality but also prevents breaking previously existing features.

Naturally, the resulting program's correctness would heavily depend on the of the tests' correctness. This opens research opportunities which leverage AI such as automatic test generation, and automatic program repair. Programming language models such as OpenAI Codex can be used to achieve this goals.

Related to my research on software reliability, automated test suites are used to test specific hypotheses and scenarios. This in turn is critical to methodically validate reliability enhancing techniques.

Continuous deployment

Continuous deployment is the process of automatically deploying changes to production environment. It is closely related to CI and automatic software testing, given that every change to the system must ideally comply with the testing pipeline before being released.

Continuous deployment is not to be confused with continuous delivery, which produces artifacts (executables or libraries) ready for deployment, but does not handle deployment itself.

In the area of continuous deployment, AI techniques can be researched to solve deployment specific issues such as resource allocation optimization. Related to software reliability, continuous deployment can be coupled to the monitoring via metrics and logs. AI models can be trained to recognize patterns in

metrics and logs, and therefore detect anomalies, incorrect behavior, or regressions caused by deployments.

Maintenance and evolution

Software maintenance is a process that aims to improve and update software artifacts to preserve both their correct functionality and relevance. This is achieved e.g. through code refactors of dependency upgrades, with aims to fixing vulnerabilities/bugs or improving performance.

This means that software is in constant evolution. Every day, newer versions of software artifacts are released to the public. Coupled with version control systems and CI/CD pipelines, developers are able to frequently push new features and fixes to their customers.

In large projects with many dependencies, evolution and maintenance are very important issues. Not only these projects need to take care of their own maintenance, but also of their software supply chain (SSC). The SSC is the collection of the projects dependencies, and those dependencies' dependencies. Every day, optimizations, fixes, and new functionality are introduced for many popular libraries, which projects are happy to take advantage of. However, new versions of dependencies may contain errors, vulnerabilities, or breaking changes. Thus, making the reliability of SSCs a compelling research direction.

Related to this area, many tools exist to automate updates of dependencies, such as DependaBot. However, automatic fixes of breaking changes, and automated detection of vulnerabilities remain as very relevant research topics.

Discussion on future trends

ML and AI are already having a notable impact in software engineering, however, I believe that instead of "AI eating software engineering", AI techniques are becoming tools for software engineering, which need to be used according to the problems that need to be solved.

For example, within AI for software engineering, models like OpenAI's Codex (which powers GitHub's Copilot) are of great relevance. This type of tools allow for greater productivity and for easier introduction to programming. However, I do not think that they contribute to a scenario where "AI will eat software engineering". Traditional methods might be better suited to solve many problems, specially when taking into account the high amount of data and compute resources needed to train powerful models. Furthermore, these kind of tools do not account for other areas of software engineering than writing code. Interesting directions would be to research new AI tools that handle other parts of software engineering, such as translating requirements into system designs, automatically generating tests from existing code, or helping developers understand existing code. It is also relevant to mention AI-powered cloud services. These services are specialized and can be used as the back-end for many types of applications. I believe that these services will grow even more specialized and diverse.

Related to my research topic, compelling directions would be to evaluate and quantify how reliable is the software produced by AI tools under different scenarios, programming languages, and across application domains. It is not unthinkable that code created by blind use of AI tools is already deployed in production environments, and therefore it is important that said tools also provide some form of reliability guarantees. Another direction would be to determine the reliability of cloud AI services, and devise possible ways to enhance them.

Regarding software engineering for AI, I believe that software engineering practices will continue to evolve under the influence and needs of AI systems. This means that methodologies and processes to methodically create, validate, deploy, and monitor AI models will continue to be refined. In this area, automation is key, mirroring modern software engineering practices like DevOps.

Overall, it is quite difficult to predict the future. But, the trends I have described are the ones that make most sense to me. Software engineering and AI are increasingly coupled, however I do not believe that in the future, AI would be used as a black box for every engineering problem in need of a solution (software-related or not).

References

[1] Liming Chen and Algirdas Avizienis. N-version programming: A fault-tolerance approach to reliability of software operation. In *Proc. 8th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-8)*, volume 1, pages 3–9, 1978.

[2] Anthony Iannino and John D. Musa. Software reliability. volume 30 of *Advances in Computers*, pages 85–170. Elsevier, 1990.