The Debugging Mind-Set

Communication of the ACM, JUNE 2017, VOL.60, NO.6

By Devon H. O'Dell

- Software developers spend a substantial amount of time, approximately 35%-50%, on validating and debugging software. This activity accounts for a significant portion of the total budget in software development projects.
- The article argues against the hyperfocus on eliminating bugs during development. Instead, it emphasizes embracing debugging as an integral part of problem-solving. Effective problem-solving skills, crucial for debugging, can be learned, taught, and mentored.
- The article argues against the hyperfocus on eliminating bugs during development. Instead, it emphasizes embracing debugging as an integral part of problem-solving. Effective problem-solving skills, crucial for debugging, can be learned, taught, and mentored.
- Debugging is essentially problem-solving. However, teaching and learning effective debugging is challenging. The literature suggests a focus on imparting strategies used by expert programmers, though this hasn't shown to significantly reduce the time novices spend on debugging.
- The author argues that debugging should be viewed as a science and suggests that educational institutions should offer courses specifically focused on debugging, a suggestion that has been made since the late 1980s.

Evaluation

I agree with the author on that debugging is one of the most critical skills in the industry, however; I beg to differ that offering debugging as a course wouldn't be as impactful as they believe it would be, despite being true in some domains, it would be better to focus on problem solving techniques, and having a wider knowledge base of mistakes that could happen in general, and let the debugging learning phase be an initial step in any project launch up.

Surviving Software Dependencies

Communication of the ACM, SEPTEMPER 2019, VOL.62, NO.9

By Russ Cox

- The article begins by noting the ubiquitous use of software dependencies today, contrasting it with the past when software reuse was more of a concept than a practice.
- Cox discusses the role of dependency managers (like NPM for Node.js) in facilitating the use of dependencies but also notes that these tools have made it too easy to incorporate external code without thorough scrutiny.
- The article advises a thorough inspection of dependencies, similar to the process of hiring a software developer, including checks on design, code quality, testing, debugging, maintenance, usage, security, and licensing.

- It stresses the importance of running a package's tests, writing new tests focused on the needed functionality, and isolating dependencies at runtime to limit potential damage from bugs.
- Cox suggests abstracting dependencies to make it easier to migrate to new ones if needed, thus avoiding being locked into a specific package.
- The article provides real-world examples, like the Equifax breach, to underline the consequences of inadequate dependency management.

Evaluation

While the recommendations, such as thorough evaluation of dependencies and continuous monitoring, are sound, their practical implementation in various scales of projects, especially smaller ones, or those with limited resources, may not be thoroughly addressed. The article could potentially overlook the constraints faced by smaller teams or projects with limited resources.

Automated Program Repair

Communication of the ACM, DECEMBER 2019, VOL.62, NO.12

BY CLAIRE LE GOUES, MICHAEL PRADEL, AND ABHIK ROYCHOUDHURY

- The article discusses various potential uses of APR, such as improving programmer productivity, fixing security vulnerabilities, self-healing software, and automatically generating hints for programming assignments.
- APR involves navigating a search space of program edits, using symbolic reasoning to synthesize code fragments, and learning from existing code and patches. A significant challenge in APR is dealing with weak specifications, as detailed formal specifications of intended program behavior are often unavailable.
- A key issue with APR is overfitting, where generated patches may fix the test suite used for repair but do not generalize well outside of it. This is due to the reliance on test suites as the primary correctness criterion.
- The article delves into various state-of-the-art techniques in automated program repair, including heuristic repair, constraint-based repair, and learning-based repair. Each approach has its strengths and limitations in addressing the challenges of APR.

Evaluation

the article relies heavily on theoretical discussion without sufficient empirical backing or diverse case studies, it could be critiqued for not adequately substantiating its claims with real-world evidence.