Chapter 3. Bug Detection and Code Review

Imagine paying the highest salaries in a company to software engineers to develop a product that will be responsible for the company's revenue, only to lose that revenue due to costly bugs in production. This is any business owner's worst nightmare, and sadly it happens every day. Software has automated whole industries, replacing lengthy manual processes and creating new ways to do previously impossible things. However, automation can't be effective when bugs detract from the underlying products' key functionalities.

To mitigate this fundamental concern, several job titles have been created over the years to guarantee proper quality assurance (QA), such as QA engineer, QA analyst, and test engineer. Processes, too, have been developed to detect bugs before they get deployed to production. Those processes boil down to two main categories:

Code reviews

This process is done during development, and it consists of team members reviewing each other's code before it is deemed ready to go live. Some teams mandate a minimum number of team members who must review and approve a pull request (PR) before it can be merged.

Quality assurance

This process is done after development as the last "gatekeeper" before code gets pushed to production. It consists of manual or automated tests done in an environment that closely matches production. These tests aim to mimic users' behavior to catch any bug that could have escaped a code review.

When either process finds any bugs, performance issues, security vulnerabilities, or other malfunctions, the code can be *regressed;* that is, it goes back to the software engineer who developed it, along with a comment containing the specific deficiencies that must be corrected.

These processes are critical to any software development team, yet they are often very lengthy and nondeterministic, introducing bottlenecks while not fully delivering on the vision of preventing bugs from showing up in production. As such, as AI tools have come into existence, the industry has seen a big focus on automating code reviews and making the process of detecting bugs much faster and more deterministic. Thousands of software engineering teams are already using AI-based automated code-review tools.

Types of AI Code-Review Tools

The AI tools reviewed for this chapter fall into three main categories, with slightly different usage in software development. Some of the tools reviewed offer more than one type of functionality.

IDE-based tools

IDE-based tools integrate directly into the software development environment that engineers use to write code, such as Visual Studio Code, IntelliJ IDEA, or Eclipse. These tools provide real-time feedback as developers write code: highlighting errors, suggesting improvements, and providing documentation links directly in the IDE. Of the three types of tools described here, this is the only type that provides feedback when the code is saved locally. This immediate feedback loop helps developers identify and fix issues on the spot, improving code quality and reducing the need for extensive reviews later.

Git-based tools

Git-based tools integrate with version-control systems, such as GitHub, GitLab, or Bitbucket, and operate within the Git workflow. Unlike IDE-based tools, Git-based tools can’t be triggered by local saves of a file, only by actions in the Git workflow. You can set them up to review code automatically whenever you push changes to a repository or create or merge a PR. These tools check the code against predefined rules and guidelines and can enforce coding standards across all branches of the codebase. They typically provide feedback in the form of comments in PRs or reports in a continuous integration pipeline, helping ensure code quality before merging changes into the main branch.

Browser-based tools

These tools are accessible through web browsers and typically integrate with online version-control platforms like GitHub, GitLab, and Bitbucket. Like Git-based tools, they can only be triggered by changes in the Git workflow, not local changes. You can use these browser-based tools to get automatic reviews of your PRs or code merges online. When you submit a PR, the tool reviews the code for errors, style violations, and security vulnerabilities, then provides feedback on that PR via the web interface in the browser. I find this the least convenient of the three types of tools presented here, since it requires you to use another platform besides the IDE and version-control tools you are already familiar with.

It's also important to differentiate between linters, static analysis tools, and AI-powered code-analysis tools:

Linters

These are the simplest of these tools, focusing primarily on enforcing coding standards and styles. They scan code to identify syntax errors, stylistic inconsistencies, and basic programming errors. Linters like ESLint for JavaScript and Pylint for Python are integrated into development environments, providing real-time feedback to correct issues like indentation, bracket placement, and line length.

Static analysis tools

These tools delve deeper, analyzing code without executing it in order to detect potential bugs, security vulnerabilities, and performance issues. Static analysis tools, such as SonarQube, understand control flow, data flow, and variable scopes, making them capable of identifying complex issues like memory leaks and concurrency problems. They are commonly integrated into CI/CD pipelines, helping to maintain code health across projects.

AI-powered code analysis

These tools use machine learning to analyze coding patterns across many projects, identifying complex issues and suggesting improvements. AI analysis tools, like DeepCode and Codacy, provide

context-aware suggestions and can predict the impact of code changes, offering optimization tips learned from vast datasets.

Due to the scope of this book, I'll only cover this last type of tool, AI-powered code analysis.

Use Cases

The millions of software engineers who are already using AI tools for automated code reviews and bug detection find that they bring obvious benefits across a range of daily use cases. These include:

Educating software engineers

Automated code-review tools provide software engineers, especially more junior ones, with a 24/7 pair programmer that points out bugs, offers suggestions, and above all gives context and reasoning for its suggestions. This is a great tool to hone your skills. Feedback loops are much more frequent with an automated tool than with normal code reviews by team members, increasing exposure to learning opportunities about the language, framework, or algorithm in question. This is extra beneficial for junior developers and for engineers switching to new tech stacks or working with a framework for the first time, since lack of experience makes mistakes more common. In code reviews, errors can be regressed with a message that helps the developer understand the mistake and avoid it next time.

Increasing software development velocity

Automating code review reduces the number of PR regressions. It also tremendously reduces the amount of time between the code being written and the review identifying issues to be fixed. Automatic code reviews at every change can point out vulnerabilities and improvements so that developers can fix them immediately. This eliminates the cycle of pushing faulty code only for other team members to find and regress it—a cycle of multiple regression loops that cost individual developers time and delay shipping features to production.

Reducing tech debt

Many times, code reviews overlook security vulnerabilities and performance issues because they don't usually impact functionality, which is objectively the biggest focus of any code review. Even when they are detected, they aren't often treated as cause for regression. Instead, they often go into a "nice to have" note, effectively adding the vulnerability or issue to the pile of tech debt. That pile usually accumulates for a long time, until it becomes unsustainable and requires extensive refactoring of the codebase.

Adding depth to code reviews

Most of the code-review tools mentioned in this chapter focus on security vulnerabilities and often point out occurrences of OWASP Top 10 vulnerabilities in code, along with suggestions for resolving them. Team code reviews rarely reach this level of depth; such vulnerabilities are often only detected much later (if ever), during professional security audits or penetration-testing reports. Using these tools allows teams to detect security vulnerabilities much earlier.

Keeping the Human Review

A common criticism of automated code-review tools is that they discourage (human) team members from performing code reviews in a timely manner. To be fair, code reviews were a dreaded activity in many teams long before AI tools came into existence. Software engineers frequently forget to review their peers' pull requests or leave a positive review message of "lgtm" (short for "looks good to me") just to unblock a feature deployment.

AI tools add tremendous immediacy to the code-review process. This reassures software engineers that their code has a high quality standard, but it also leaves them feeling less urgency to review their peers' code, believing the AI tool has already done that job for them.

This is a very fair criticism, in my opinion. *AI code reviews don't replace human code reviews*, especially those performed by senior engineers who know both a feature's technology and the business and use cases for it. This is the angle that is manifestly missing in AI code reviews. The AI tool misses the *context* behind the code being reviewed and the intent behind certain code segments. This can lead it to make irrelevant suggestions or fail to identify

context-specific issues that might be obvious to a human reviewer. This is a key reason why you should *never skip human code reviews*, even if you're also using automated code reviews.

It's also worth noting that the language used to market these automated code-review tools is quite different from that used for the code-generation tools reviewed in the previous chapter. Few of the tools discussed in this chapter mention AI in their marketing copy much (or at all), despite the fact that the products do use AI algorithms (e.g., Codacy).

There are two reasons for this. Several of these tools existed in the market for years before the recent popularity of AI. However, many position themselves as a backstop to issues found in AI-generated code. Sonar, for example, promises to minimize risk, ensure code quality, and derive more value from code created by both AI and humans. As the website copy states: "To maximize the advantages of generative AI in coding, developer teams need robust DevOps processes, reporting, and metrics that focus on code quality, security, and reliability."

Evaluation Process

I evaluated more than 20 automated code-review tools in order to shortlist the ones I highlight in this chapter. Every tool covered here meets the following criteria:

- It is a professional project with a competent team behind it.
- The code it generates has a high quality threshold.
- It offers some level of functionality for free or on a trial basis.
- It has a high level of adoption at the time of writing (mid-2025).

In order to select and compare AI tools for this chapter, I created a simple JavaScript program and introduced four issues into the code. You can review the full code in the book's GitHub repository, inside the folder named "Chapter 3." Example 3-1 provides the most relevant snippet, with each of the four issues commented for clarity. I ran the exact same code through each of the tools reviewed in this chapter, which discusses the results each tool provided.

Example 3-1. Code snippet for the tests of code-review tools

```
app.post('/submit', (req, res) => {
    const requestData = req.body;

    // 1. SQL Injection vulnerability
    const sqlQuery = `SELECT * FROM users
                      WHERE username = '${requestData.us
    db.all(sqlQuery, [], (err, rows) => {
        if (err) {
            console.error('Error executing SQL query:',
            res.status(500).send('Error in database oper
        } else {
            console.log('Query result:', rows);
            res.send('Data processed with SQL query resu
            + JSON.stringify(rows));
        }
    });

    // 2. Cross-Site Scripting (XSS) vulnerability
    const responseHtml = `
        <html>
            <body>
                <h1>User Profile</h1>
                <div>${requestData.userInput}</div> <!--
directly rendered into HTML -->
            </body>
        </html>
    `;
    console.log('Generated HTML for user:', responseHtml

    // 3. Potential memory leak in event listeners
    const listeners = [];
    for (let i = 0; i < 100; i++) {
        listeners.push(() => console.log('Event listener
    }
    console.log('Number of listeners created:', listener

    // 4. Inefficient loop
    let sum = 0;
    for (let i = 0; i < 100000; i++) {
        sum += i;
    }
    console.log('Sum of 0 to 99999:', sum);
```

```
});
```

Before we dive in, here is a brief explanation of each of the errors I introduced and why it would be important to catch them in a code review:

SQL injection vulnerability

This vulnerability arises from incorporating user input directly into a SQL query without any form of validation or sanitization. In the provided code, the variable `requestData.username` is directly concatenated into the SQL query string. This approach lets attackers craft user inputs that manipulate the SQL query to perform unauthorized actions, like accessing, modifying, or deleting data. For instance, an attacker could provide a username input like `' OR '1'='1`, which could alter the query logic to return all users in the system, thereby breaching data privacy.

Cross-site scripting

Cross-site scripting (XSS) occurs when an application includes untrusted data, typically from user inputs, within the content of its web pages without proper validation or escaping. In the script, `requestData.userInput` is directly included in an HTML response structure sent back to the client. If this user input includes malicious JavaScript code, the browser could execute that unauthorized script, leading to session hijacking, personal data theft, or malicious redirection.

Memory leak

Memory leaks in web applications can occur when memory that is no longer needed is not released back to the system. In the example, a large number of event listeners are created within a loop but are never removed. Each listener retains a closure scope that may consume more memory. Over time, especially in long-running applications like servers, these listeners accumulate, occupying an increasing amount of memory. This can potentially exhaust available resources and lead to performance degradation or crashes.

Inefficient loop

The loop in the example code inefficiently performs a large number of iterations to compute the sum of all integers from 0 to 99,999. Each iteration involves performing arithmetic operations and updating a local variable. Although these actions are relatively simple, they are unnecessarily repeated many times. This not only consumes CPU cycles, but it could also block the event loop in a Node.js environment, leading to delays in processing other incoming requests or operations.

Now let's dive into the top-performing AI code-review tools I tried.

Codacy

Codacy is a startup based in Portugal that launched an automated code-review tool in 2012. The product has evolved significantly over the years and is now a market-leading solution that leverages AI to "help developers identify and fix issues within their code, improving code quality and reducing technical debt, with support for more than 40 programming languages and seamless integrations with GitHub, Bitbucket, and GitLab," as per the copy on its website.

Codacy's AI tool analyzes code for potential errors, style violations, security vulnerabilities, and performance issues, and it provides software engineers with suggestions for improvement. The tool is designed to learn from past reviews, adapting to the specific standards and practices of each development team.

By automating the code-review process, Codacy helps developers focus more on building features rather than fixing issues, ultimately speeding up the development cycle and enhancing code maintainability.

Practical example

I created an account with Codacy using my GitHub account and ran the tool on the code shown earlier in this chapter (which you can review in full in the book's GitHub repository).

Codacy correctly identified issue number 1, the SQL Injection vulnerability, and labeled its severity as "Critical," the highest level in its ranking, as seen in Figure 3-1.

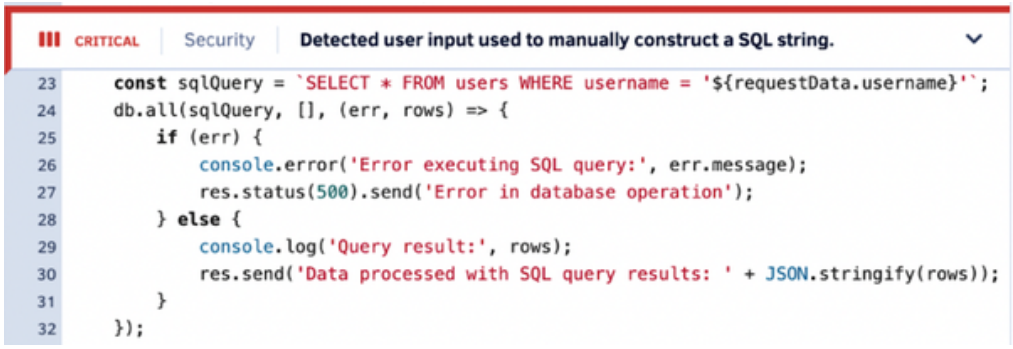

Figure 3-1. Codacy identified the SQL injection vulnerability

Codacy provides an expandable section with an explanation of what the error is, why it's dangerous, and how to solve it (Figure 3-2).

Figure 3-2. Codacy explained the SQL injection vulnerability

Codacy also correctly identified issue 2, the XSS vulnerability, and labeled it as "Medium" severity (Figures 3-3 and 3-4).

Figure 3-3. Codacy identified the XSS vulnerability

As seen in Figure 3-4, Codacy clearly explained this XSS vulnerability.

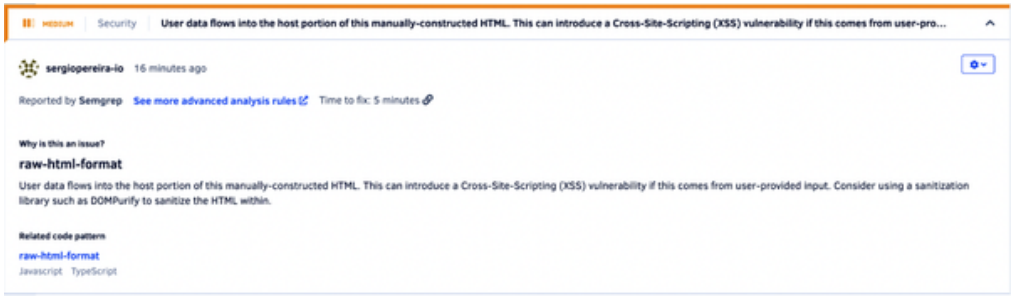

Figure 3-4. Codacy explained the XSS vulnerability

Codacy's analysis didn't identify issues 3 and 4, which are more related to performance than to security.

All of this feedback was provided on Codacy's website immediately after I connected my GitHub account and selected the repository I wanted to have analyzed. However, after I opened a PR on that same repository, Codacy performed a second level of analysis directly in the repository.

Most of the errors it identified reiterate those it found in the previous analysis, which I expected, since the code is the same. However, on GitHub, Codacy also offers a "commit suggestion" to fix each issue along with a brief explanation. This makes it very convenient for software engineers to simply accept the suggestion and merge the PR with one click (Figure 3-5).

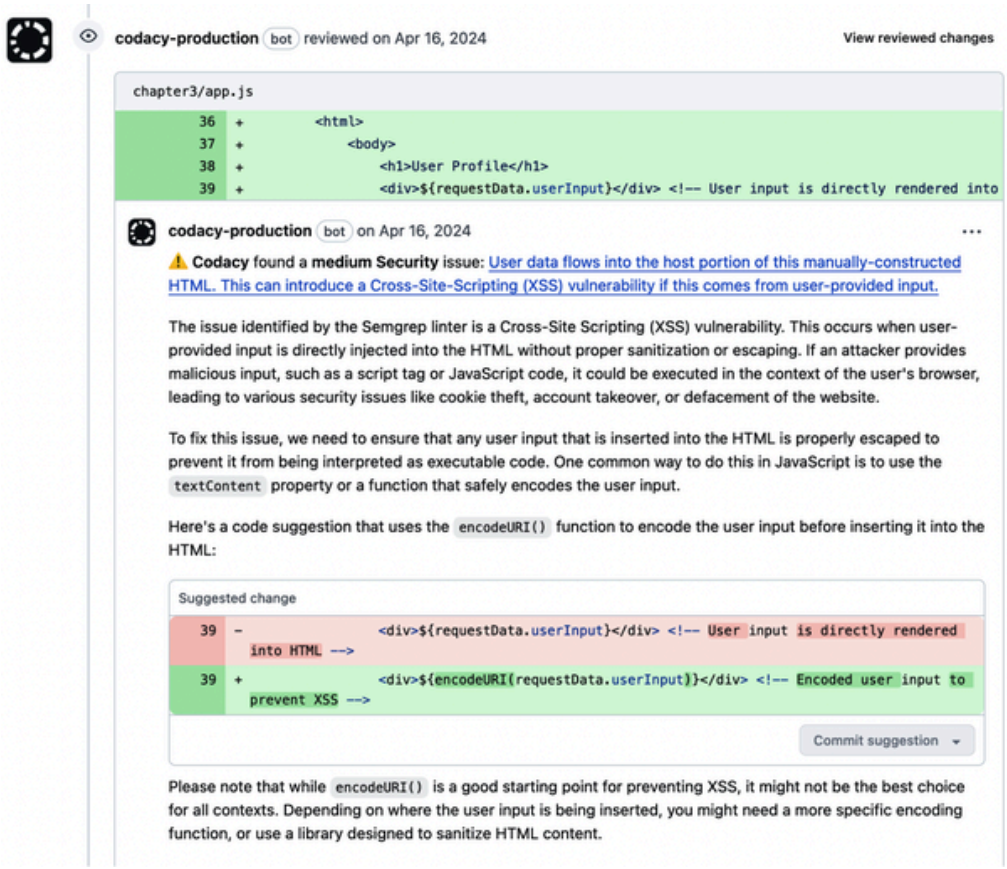

Figure 3-5. Codacy suggested a fix for the issue it found

For all these reasons, I rate Codacy's tool an 8/10. It found two of two security issues, but it didn't find either of the two performance issues. For the issues it did find, it offered very comprehensive explanations and proposed fixes that could be accepted in the actual repository with one click.

DeepCode (by Snyk)

DeepCode began as an independent startup based in Zurich, Switzerland, as a spinoff from ETH Zurich University.[1] It was acquired by the cybersecurity behemoth Snyk in September 2020.[2] Since then, the product was marketed

first as “DeepCode by Snyk” and more recently as “DeepCode AI,” and has been integrated into Snyk’s broader suite of products and services.

As Snyk described it in 2020, DeepCode includes “sophisticated interpretable machine learning semantic code analysis. The technology scans code 10–50x faster than alternatives, enabling real-time workflows within the development process, and dramatically reduces both false negatives and false positives using a custom machine learning platform that is able to quickly learn from huge volumes of code.”

DeepCode uses machine learning algorithms to learn from millions of publicly available open source software-development repositories. This large dataset allows DeepCode to provide highly accurate suggestions and find potential issues that human reviewers might overlook.

DeepCode can be used on an IDE or directly in a Git repository. It points out security vulnerabilities on the spot, as alerts in the IDE tool tip or as comments to the pull request in the repository. As the company’s website puts it:

> *It combines symbolic and generative AI, multiple machine learning methods, and the expertise of top security researchers to offer accurate vulnerability detection and tech debt management. DeepCode AI is purpose-built for security, supporting 11 languages and over 25 million data flow cases to find and fix vulnerabilities efficiently. This AI technology enhances developer productivity by offering one-click security fixes and comprehensive app coverage while ensuring the trustworthiness of the AI through training data from millions of open-source projects. DeepCode AI stands out for its hybrid approach using multiple models and security-specific training sets to secure applications effectively.*

Practical example

Just like I did for Codacy, I created an account with Snyk/DeepCode using my GitHub account and ran it on the code in Example 3-1 within the book’s repository.

DeepCode correctly identified issue 1, the SQL injection vulnerability, and labeled it with “H” (High), the highest level in its ranking system. It even

provides a score (Figure 3-6), though I could not find specific information about what this score means. This issue's score of 830 is the highest score received by my code.

Figure 3-6. DeepCode identified the SQL injection vulnerability

Snyk/DeepCode provides two expandable sections for each error. One provides a deeper explanation of the issue, resembling a stack trace rendered in the browser UI (Figure 3-7).

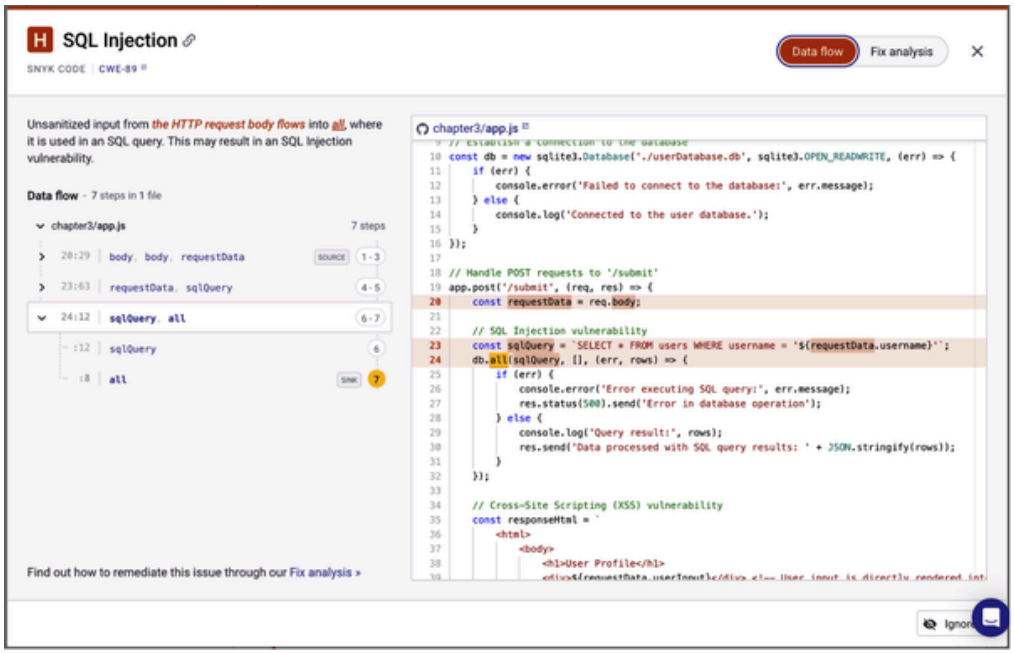

Figure 3-7. DeepCode explained the SQL injection vulnerability

The second expandable section suggests a fix for the issue (Figure 3-8) and gives pointers to avoid using concatenated SQL statements as strings stored directly from user-entered parameters. This is a best practice in defensive programming.

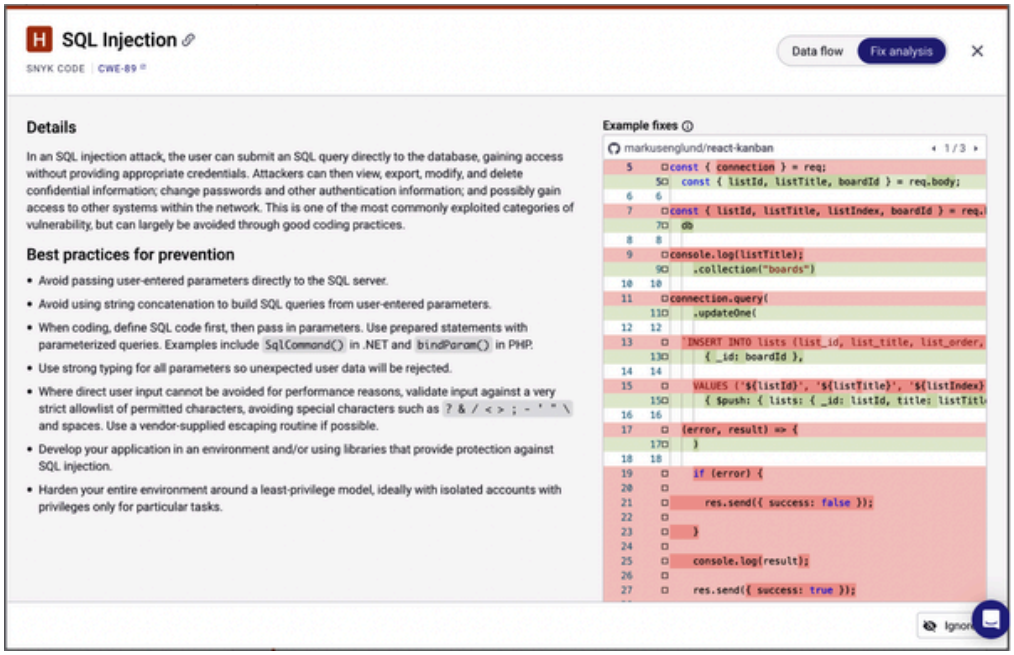

Figure 3-8. DeepCode suggested a fix for the SQL injection vulnerability

These suggestions are provided "as is" from an open source repository in the training dataset. This is very nice in terms of transparency, as a software engineer should always want to know where the code comes from. However, it adds some extra cognitive load in terms of actually solving the problem, since this is just a proposed solution to help the software developer fix the issue, not an actual proposed solution to be adopted with the click of a button.

Despite this deep level of detail for issue 1, DeepCode didn't find issues 2, 3, or 4. It did find some lower-severity issues in some libraries I used (inside `node_modules`), which were irrelevant to this book's exercise.

I rate DeepCode a 6/10. It found one of two security issues and didn't find either of the performance issues. For those issues it found, it provided very comprehensive explanations; however, the help it offers for each issue is lacking in comparison to that offered by Codacy and CodeRabbit. DeepCode provides information about the issue, but it doesn't offer proposed solutions that are easy to adopt with one click.

CodeRabbit

CodeRabbit is an automated code-review platform launched in September 2023, amid the generative AI buzz. It gained significant popularity very fast, especially on Twitter/X as some tech influencers did thorough reviews of the product and promoted it in their networks (here's one example). The official number of CodeRabbit users had not been publicly disclosed at the time of writing (mid-2025).

CodeRabbit leverages AI capabilities to enhance the quality, performance, and efficiency of code reviews. It delivers its code recommendations through comments in the repository.

Practical example

Like I did for the other tools, I created an account with CodeRabbit, allowed it access to my GitHub account, and selected the repository I wanted to give it access to. Unlike Codacy and DeepCode, CodeRabbit won't statically analyze code that's already in a repository. Instead, I needed to open a pull request; CodeRabbit then posted comments to that PR with its code-review items and suggestions. CodeRabbit promotes this as a much more interactive tool that aims to mimic a team member commenting on a PR seconds after it's opened on GitHub. However, my experience on CodeRabbit's website was greatly inferior to my experiences with the competitors analyzed here.

CodeRabbit correctly identified issue 1, the SQL Injection vulnerability (Figure 3-9). It doesn't provide any indication of severity level: all issues it reports look alike in that regard. It did a good job pointing out the faulty code snippet and offered a brief explanation about why it contains a vulnerability. I believe most software engineers will enjoy this simple UI, since it's exactly the type of interaction they get from human colleagues who review their PR.

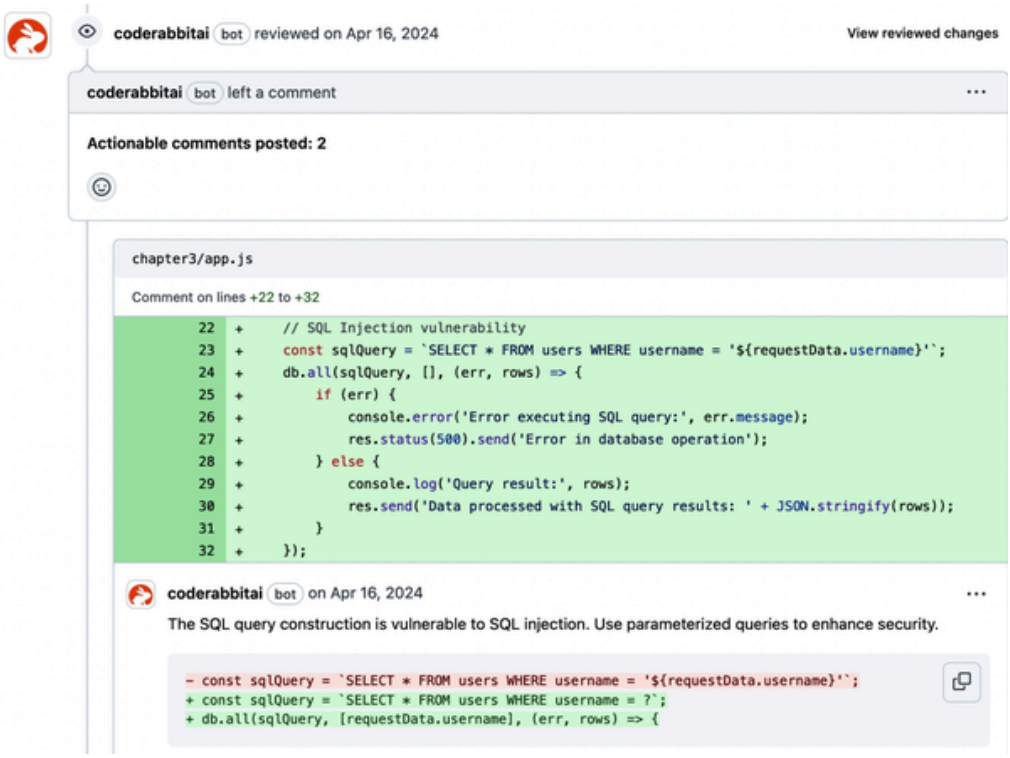

Figure 3-9. CodeRabbit identified the SQL injection vulnerability

Along with the explanation, CodeRabbit offers an expandable section called "Committable suggestion" that contains a suggested fix for the issue (Figure 3-10). While CodeRabbit displays a noticeable warning to review the offered solution thoroughly, committing it is only one convenient click away.

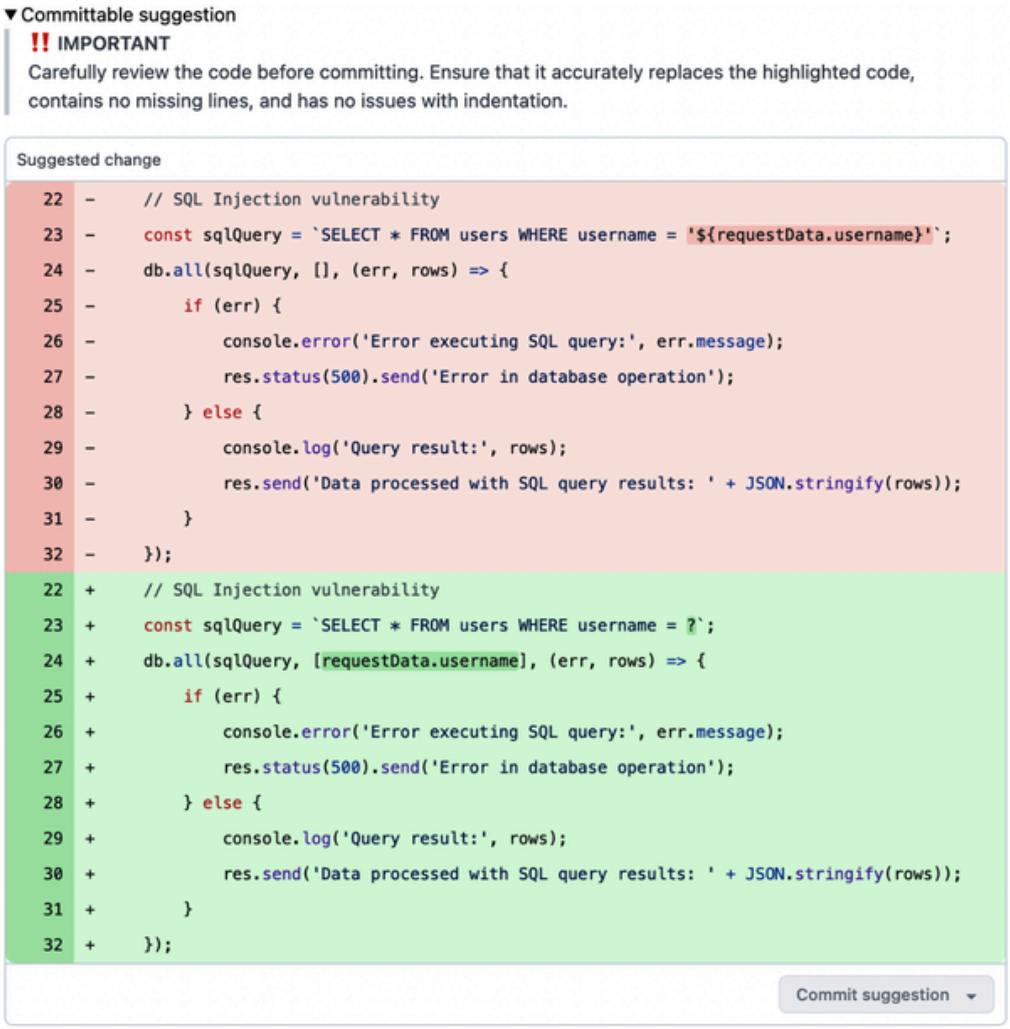

Figure 3-10. CodeRabbit suggested a fix

CodeRabbit also detected issue 2, the XSS vulnerability (Figure 3-11), but just like the other tools analyzed, CodeRabbit didn't find performance issues 3 and 4.

Figure 3-11. CodeRabbit identified the XSS vulnerability

Thus, I rate CodeRabbit a 7/10. It found both security issues but neither of the performance issues. It also proposed a solution for one of the issues it found, but not the other one. However, its explanation for the issues was very superficial compared to the other two tools. Finally, it lacks a website interface that would let users research issues in more depth and provide some historical perspective of changes and improvements made on the codebase, which the other tools have.

Tool Comparison

All three of these AI code-review tools take different approaches to blocking my pull request from being merged, as shown in Figure 3-12:

- Codacy blocks the PR merge until I fix the issues it identified (which, to be fair, I can do using its suggested fixes).
- Snyk/DeepCode doesn't block the PR merge, despite the issues found.
- CodeRabbit only posts comments; it doesn't run actual checks, and thus it would never block a PR merge regardless of any issues it finds.

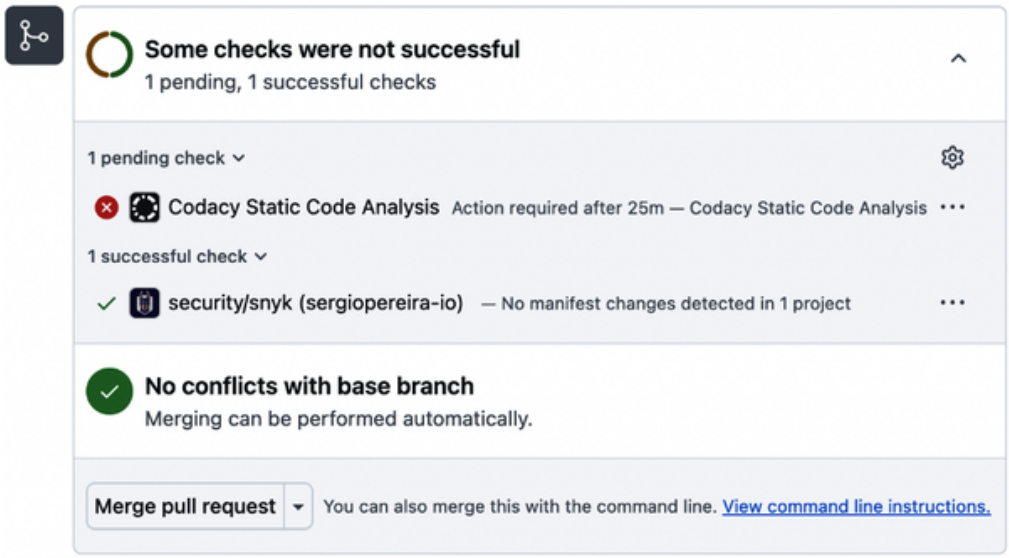

Figure 3-12. Codacy and Snyk/DeepCode show up in the checks section for the PR merge

If I were to select a single tool, Codacy would be my go-to tool. As Table 3-1 indicates, it had the highest score.

Table 3-1. AI code-review tools overview

Tool	UX	Test performance
Codacy	Browser + Repository	8/10
Snyk/DeepCode	Browser + Repository	6/10
CodeRabbit	Repository	7/10

Conclusion

Code reviews have been one of the biggest frustrations in my software development teams over the years. People are naturally more inclined to pick up new tasks assigned to them than to stop their own thread of work to review a colleague's PR. This default behavior has delayed features from being moved to QA and ultimately going live. It has also created situations where we fast-track some urgent features even with a less-than-ideal level of code review, resulting in bugs showing up in production. In general, the biggest casualty of these common code-review frustrations is team morale, with team members feeling like they're constantly switching context and losing focus.

I began using several forms of automated code review in my teams, like linters, static code analysis, and test coverage dashboards, long before the recent generative AI hype. Any team with robust engineering standards has probably done likewise.

However, after 15 years in the industry, I can tell that the recent wave of evolution adds more depth to these tools—especially the seamless way they integrate with your software development workflow, and the option to accept suggested fixes with one click. Having a very capable code reviewer who's available 24/7 to provide thoughtful feedback on issues in your code is a massive help to anyone. It's something I could only have dreamed of when I started out as a software engineer myself.

However, I believe that software engineers should leverage these tools as learning opportunities before anything else. They can and do make mistakes, as the tools themselves note in very visible warnings, and I can only underline that. *Always* have a human being review and test the suggested fixes. As with code-generation tools, I recommend a high level of diligence when reviewing any code or fixes suggested by these tools. Make it yours before you open a PR or merge to master.

[1] This book's author was part of the DeepCode team prior to the company's acquisition by Snyk, but has no contractual relationship, equity, or any other vested interest in DeepCode at the time of writing.

[2] McKay, Peter. September 23, 2020. "Accelerating Our Developer-First Vision with DeepCode". *Snyk* (blog).