How to Minimize Firmware Bugs?

WhitePaper v0.1

Outline:

Ensuring minimal bugs in firmware development is crucial for maintaining the reliability and performance of your devices. This white paper outlines effective strategies for minimizing bugs, emphasizing a balanced approach to coding, automated testing, and manual testing throughout the development lifecycle. The advice provided here is tailored to CTOs, Engineering Managers, Technical Project Managers, and Product Managers in Western Europe, working for companies that manufacture their own devices, particularly in the Life Sciences and Smart Home/Smart City sectors.

What is most important to reduce bugs in software in general:

What are the five main steps to make sure the custom software development is produced with as few bugs as possible?

To ensure custom software development is produced with minimal bugs, you can follow these five main steps:

1. **Requirement Clarity**: Ensure all requirements are well-documented, clear, and understood by all stakeholders.
2. **Code Reviews**: Regularly conduct thorough code reviews to catch errors early and ensure best practices.
3. **Automated Testing**: Implement automated unit tests, integration tests, and end-to-end tests to continuously check for issues.
4. **Continuous Integration**: Use continuous integration (CI) to integrate code frequently and run automated tests with each integration.
5. **User Acceptance Testing (UAT)**: Perform UAT with real users to identify any issues in real-world scenarios before final deployment.

These steps can significantly reduce the likelihood of bugs in custom software development.

What are frequent questions regarding bugs in firmware?

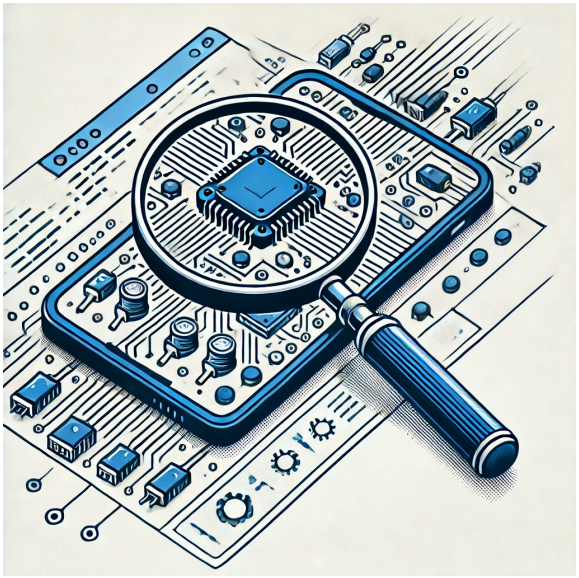
1. Where does the manual tester role fit into these five points?

The manual tester role fits primarily into the User Acceptance Testing (UAT) step. They are crucial in:

- Performing exploratory testing to identify issues that automated tests might miss.
- Conducting detailed functional testing based on user scenarios.
- Providing feedback on usability and overall user experience.
- Ensuring that the software meets the end users' needs and requirements.

Manual testers can also be involved throughout the process, including during requirement clarity and code reviews, by providing insights and identifying potential issues early on.

2. What if we're talking about custom firmware software for a device? Automated testing is tough because the hardware abstraction layer is costly to create.



For custom-made firmware software where automatic testing is challenging due to the hardware abstraction layer, manual testing becomes even more crucial. Here's how you can adapt the five main steps:

Clear Requirements: Ensure detailed, clear requirements for how the firmware should interact with hardware.

Code Reviews: Regularly review code to catch errors early, especially in hardware interaction parts.

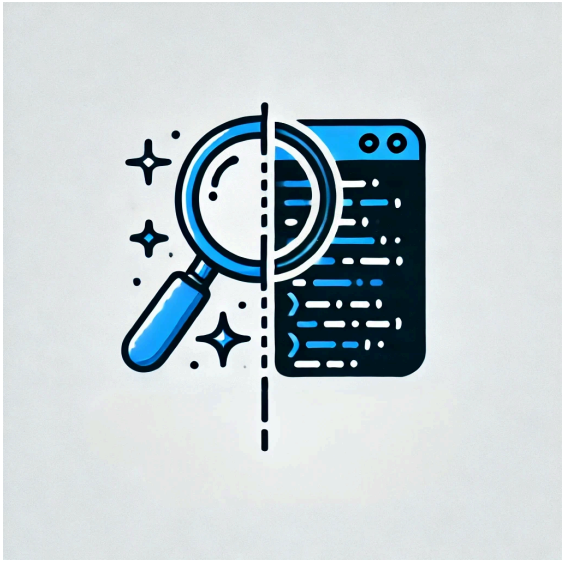
Manual Testing: Rely on manual testers to test firmware on actual devices.

Simulations in CI: Use hardware simulations in continuous integration to find issues before testing on real hardware.

Real-World UAT: Test firmware on actual devices in real-world scenarios to ensure it works as expected.

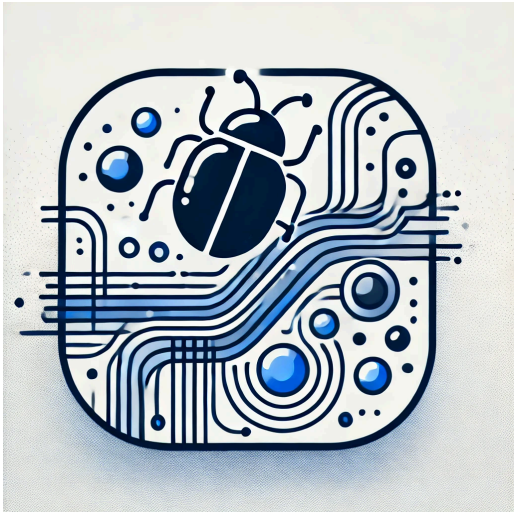
By emphasizing manual testing and integrating hardware simulations, you can ensure a robust firmware development process even when automatic testing is limited.

3. How much time should be spent on testing compared to coding?



Typically, you should allocate around 30-50% of your project time to testing compared to coding. This ensures thorough verification and validation of the software, especially important in firmware development where hardware interactions are critical. Adjust based on project complexity and criticality.

4. Is it possible to produce bug-free software?



It's practically impossible to produce completely bug-free software. However, you can minimize bugs significantly through thorough testing, code reviews, and good development

practices. The goal is to reduce bugs to a level where they are rare and not critical, ensuring reliable and stable software.

5. Can we produce satisfactory firmware with minimal bugs without using manual testers?

It is challenging but possible to produce satisfactory firmware with minimal bugs without a manual tester. This requires:

1. **Extensive (100%) Automated Testing**: Use as many automated tests as feasible, including unit tests and hardware-in-the-loop simulations.
2. **Rigorous Code Reviews**: Ensure thorough and frequent code reviews by experienced developers.
3. **Strong Requirements and Documentation**: Clear and detailed requirements help catch potential issues early.
4. **Developer Testing**: Developers must perform extensive testing on actual hardware to identify and fix issues.

However, having manual testers usually provides an additional layer of assurance and helps catch issues that automated tests might miss.

6. How much time does it take to automate testing in firmware

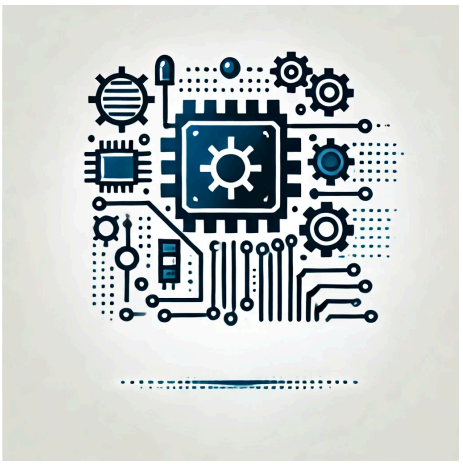
projects compared to manual testing in the short term?

In the short term, automating tests in firmware projects typically requires more time compared to manual testing. This is because:

1. **Setup Effort**: Initial setup of automated test environments, scripts, and tools can be time-consuming.
2. **Complexity**: Creating robust tests for hardware interactions and ensuring accurate simulations can be challenging and time-intensive.

Manual testing, on the other hand, can be quicker to start as it requires less initial setup, but it's less scalable and consistent in the long run. In summary, automation requires significant upfront time investment but pays off with efficiency and consistency over time, whereas manual testing is quicker initially but less sustainable for extensive testing needs.

7. What's the practical extent of automated testing in typical software projects?



In average software projects, the practical level of automated testing typically includes:

1. **Unit Tests**: Automated unit tests for individual components or functions, covering most of the codebase.
2. **Integration Tests**: Automated tests for interactions between components, ensuring they work together as expected.
3. **Regression Tests**: Automated tests to ensure new changes don't break existing functionality.
4. **Continuous Integration (CI)**: Automated testing integrated into the CI pipeline, running tests with each code change.

While full automation is ideal, many projects achieve partial automation due to resource constraints, focusing on the most critical areas. The goal is to automate as much as possible while balancing the effort required for setup and maintenance.

8. What's the practical ratio of time spent on writing code versus automating tests in typical software projects?

In average software projects, a practical ratio of time spent on writing code versus automating tests is often around 70-30 or 60-40. This means:

- **60-70% on Development**: Time spent on writing and reviewing code, implementing features, and fixing bugs.

- **30-40% on Automated Testing**: Time spent on creating, maintaining, and running automated tests, including unit, integration, and regression tests.

This ratio can vary depending on project complexity, team experience, and the criticality of the software. Investing in automated testing upfront often leads to higher quality and easier maintenance in the long run.

9. With this level of automated testing, what is the average test coverage of features?

With the level of automated testing effort you mentioned (30-40% of time spent), average test coverage typically ranges from 60% to 80% of features. This means:

- **60-80% Test Coverage**: Most critical and frequently used features are covered by automated tests, ensuring they work as expected.
- Some less critical or harder-to-automate areas might have lower coverage or rely more on manual testing.

Achieving higher coverage often requires additional effort and resources, but 60-80% is a practical and common target for many projects, balancing thoroughness and efficiency.

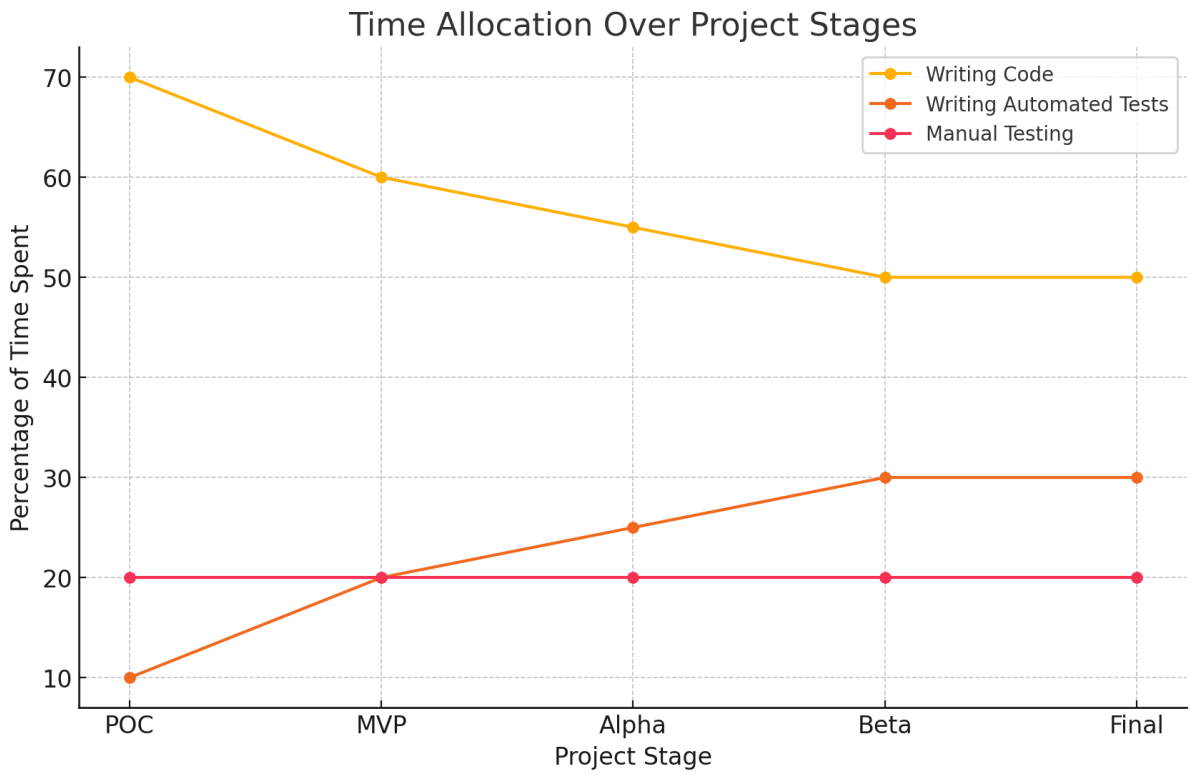
10. How much time should be spent on writing code, writing automated tests, and doing manual testing in a firmware?

Here's a summary of how time should be allocated percentage-wise in a firmware project:

1. **Writing Code**: 50-60%
    - Developing new features and functionality.
    - Fixing bugs and refactoring code.
2. **Writing Automated Tests**: 20-30%
    - Creating and maintaining unit tests.
    - Developing integration tests.
    - Setting up hardware-in-the-loop simulations (where possible).
3. **Manual Testing**: 20-30%
    - Conducting exploratory testing.
    - Performing detailed functional testing on actual hardware.
    - Ensuring usability and reliability in real-world scenarios.

These percentages can vary based on the project's specific needs and the complexity of the hardware interactions involved. This balanced approach aims to ensure thorough testing and high-quality firmware development.

Here's a graph showing the percentage of time allocated to writing code, writing automated tests, and manual testing across different project stages (POC, MVP, Alpha, Beta, Final).



This visualization helps illustrate how the focus shifts from primarily coding in the earlier stages to increased emphasis on automated testing and consistent manual testing as the project progresses.

Conclusion

Minimizing firmware bugs requires a comprehensive approach that includes clear requirements, rigorous code reviews, extensive automated testing, and thorough manual testing. By following these strategies, companies can significantly reduce the likelihood of bugs, ensuring their firmware is reliable, efficient, and user-friendly.

For WizzDev's typical customer persona, adopting these best practices will enhance the quality and reliability of your products, positioning your company as a leader in the Life Sciences and Smart Home/Smart City sectors. By leveraging a balanced approach to firmware development, you can deliver high-quality products that meet and exceed user expectations.