International Journal of Computational and Experimental Science and ENgineering (IJCESEN)

Vol. 11-No.3 (2025) pp. 4170-4175
http://www.ijcesen.com

ISSN: 2149-9144

Research Article

Enhancing Reliability in Complex Embedded Systems Software: The Critical Role of Unit Testing in the Age of AI and Machine Learning

Kishore Ranjan[1]*, Reena Chandra[2], Karan Lulla[3]

[1]Independent Researcher, Trumbull, CT, USA.
*** Corresponding Author Email:** kishoreranjan@gmail.com- **ORCID:** 0009-0001-5372-9490

[2]Independent Researcher, San Francisco, CA, USA.
Email: reenachandra11@gmail.com **- ORCID:** 0009-0001-8061-1084

[3]Independent Researcher, San Francisco, CA, USA.
Email: kvlulla16@gmail.com **- ORCID:** 0009-0007-7491-4138

Article Info:

DOI: 10.22399/ijcesen.2633
Received : 21 March 2025
Accepted : 25 May 2025

Keywords

Embedded Systems,
AI,
CI/CD,
Unit tests,
integration tests,
software quality.

Abstract:

Qualification of a large code base embedded software systems, like Lithography machines, Aerospace Embedded Systems, High-End Medical Embedded Systems & Automotive Embedded Systems, is an important and challenging aspect. The goal of this article is to provide a brief description of large code base software qualification in embedded systems and how "Unit Tests" can make a huge difference. This research addresses the importance of qualification and automation in embedded system devices to improve the software quality of source code by introducing appropriate test strategy at right place to improve the quality of final product and at a low maintenance cost. This article will focus more on the importance of unit tests in the software development lifecycle, and how the advance AI tools and CI/CD concepts are used across industry is enabling it.

1. Introduction

The integration of Machine Learning (ML) and Artificial Intelligence (AI) technologies into the embedded software has led to the development of more intelligent embedded systems such as autonomous vehicles, smart homes, IoT and lithography machines. The size of the global embedded system market is expected to reach 110.46 billion USD by 2026 [1].

In the Software development lifecycle, testing plays a vital role. Especially in the embedded system development, the role of software testing becomes more critical, as it might also impact and influence the re-design of hardware and can jeopardize the entire product delivery time and the cost.

Usually, the software testing is the last phase of the product development lifecycle, however, in the embedded systems development, it becomes more crucial that testing has to be done at the earlier phases together with the software development. This article is a practical guideline for professionals in embedded software development who are eager to improve the quality of large code bases and complex embedded software systems development to develop Unit testing using advance AI tools and take advantage of CI/CD.

2. Software development life cycle

Software development life cycle (SDLC, as shown in figure1) is a structured process to effectively design, build, test, and maintain software. It offers a methodical way to guarantee software quality, cut expenses, and satisfy user needs.[2]

Phases of SDLC

Planning

- Specify the project's resources, timetable, viability, and scope.
- Determine the project's goals and potential hazards.
- Perform technical, financial, and operational feasibility studies.
- Create a project schedule.

Requirements Analysis
- Compile and record business and user requirements.
- Interview the stakeholders.
- Specify both functional and non-functional needs.
- Make a document called a Software Requirement Specification (SRS).

Design
- Create system components, data models, and architecture.
- The overall system architecture is defined by high-level design (HLD).
- Detailed component-level design is specified by low-level design (LLD).
- Database schemas, security considerations, and UI/UX wireframes.

Implementation
- Writing and developing the actual software code is the aim of implementation (coding and development)
- Choose programming languages, tools, and frameworks.
- Adhere to coding standards and best practices.
- To monitor changes, use version control (like Git).

Testing & Integration
- Confirm that the program satisfies specifications and operates as intended.
- Unit Testing: Qualify the highest granular part of software (module/function)
- Integration testing: Verify that various components are compatible.
- System Testing: Verify the entire program.
- User Acceptance Testing (UAT): Software is validated by end users.

Maintenance & Support
Address bugs and security flaws. Disseminate fixes and feature updates. Keep an eye on user input for upcoming enhancements.

3. Insight of Unit Test

Unit testing is a software testing methodology that involves evaluating individual software application modules, functions, or components separately to make sure they operate as intended. It is usually carried out together with the software development (coding) phase and is a crucial component of the Software Development Life Cycle (SDLC). [4]

3.1 Importance of Unit Testing

a) Finding early bugs in software lowers the expense of debugging at the later phase and assists in the early identification of problems during the development stage.
b) Enhances Code Quality: Makes sure every function works as intended, which lowers production flaws.
c) Enables Refactoring: Developers can alter code with assurance without affecting already-existing functionality.
d) Improves Maintainability: Software that has been well tested is simpler to maintain, change, or expand.
e) Elevates Agile & CI/CD: Test-driven development (TDD), agile approaches, and continuous integration (CI/CD) all depend on this.

3.2 Tools for Unit Test Framework (based on coding language)
- Java – JUnit, TestNG
- Python – unittest, pytest
- C/C++ – Google Test (GTest), CppUnits
- JavaScript/Node.js – Jest, Mocha, Jasmine
- C#/.NET – NUnit, MSTest, xUnit

3.3 Test Driven development
Test-Driven Development (TDD) is the most recent iteration of unit testing. In reality, this is a test-driven development technique. In order to use the tests to validate your code in real time, you write them concurrently with the code as the developer. The approach produces unit tests and working code as outputs.
The TDD workflow follows a simple cycle:
- Write a small test to test a behavior.
- Build and run the test suite to see the new test fail, possibly not even compile yet.
- Make the CUT changes needed to pass the test.
- Build and run the test suite to see the new test pass.
- Refactor to remove any duplication or cleanup the test or CUT.

Since TDD essentially entails advancing test development into code development, it will be simpler to implement if you already have off-target unit tests. Because it moves receiving feedback to the left on the development timetable, this is known as "shift left." You should continue to use your current infrastructure for unit testing. The primary shift will be that the developers that write the CUT will also write the unit tests rather than hiring different test developers. To enable test isolation,

you might need to modify your software in addition to spending time building your unit test infrastructure. This endeavor has the advantage of exposing the code to testing. [4]

3.4 AI/ML in Unit Testing

Artificial Intelligence (AI) and Machine Learning (ML) are being widely used in transforming unit testing by automating test generation, prioritization, and also it' maintenance. These technologies proactively analyze the entire codebases, revision histories, and also takes account into the runtime behavior to identify the potential high-risk areas, generate intelligent test cases, and predict defects early in the development cycle. Some key highlights on using AI/ML in unit tests are:

- Unit Test Generation: ML models can automatically generate unit tests by learning from existing code patterns, function behaviors, and historical defects.
- Unit Test Optimization & enhancements: AI helps analyzing the code changes and historical failure data, to improve the regression test cases. Advance ML algorithms can also detect unusual patterns in test executions, highlighting hidden bugs or regressions.
- Code Coverage Improvement: AI can identify untested or under-tested code paths and recommend targeted unit tests.
- Flaky Test Detection: By analyzing historical test results, ML can identify non-deterministic or flaky tests and help in stabilizing them.
- These AI-driven enhancements reduce manual effort, improve testing accuracy, and foster a "shift-left" approach, enabling earlier defect detection and faster delivery cycles.

3.5 Role of Unit test in CI/CD

Continuous integration and continuous development (CI/CD) are widely used across the industry for a sustainable maintenance of large code base software were adding more low level "Unit Testing" in pipeline is enabling the robustness of software development lifecycle. Unit testing is a cornerstone of a reliable CI/CD (Continuous Integration / Continuous Deployment) pipeline. Here's are highlights on it's importance

- Early Bug Detection in CI/CD pipeline
- Faster Development Cycles: Automated unit tests run quickly and often (on every commit or pull request).
- Code Quality and Reliability: CI pipelines with high unit test coverage tend to have more robust, predictable software releases.
- Enables Continuous Integration: CI/CD relies on automated checks to validate every change before integration. Unit tests are the first line of defense in that automation stack, ensuring that only working code progresses through the pipeline.
- Reduces Risk in Deployment: Combined with integration and system tests, unit tests form a test pyramid. This layered validation helps reduce deployment risks and ensures production stability.

4. Importance of UT in large code base Embedded Systems

A perfectly "unit tested" code reduces the risk of finding defects at the latter phase of qualification and hence reduces the implementation cost. Large-scale embedded systems are performance-constrained, and extremely complicated, which makes unit testing (UT) essential. Because embedded software interacts with hardware with real-time limitations, and external surroundings in a different way than typical software applications, thorough unit testing becomes essential.

The key advantages of Unit testing for an embedded software system are as follows:

a) Faster execution time: Since UTs are done at the smallest unit of software (function/module), their development and execution time is very fast.

b) Easy to find and fix bug: Since UT is done as a part of software development, therefore it's easy to change the design of software and resolve the bugs.

c) Platform independent testing: Usually UT is done in a simulated environment, therefore it does not need any hardware to test and hence it becomes very critical in embedded systems.

d) Shift Left: Since UT exposes potential issues at the smallest level of software module, and hence it reduces the chances of finding bugs at the latter phase of qualification on real hardware which is more time consuming, both in terms of testing time and fixing bugs.

e) Enhances system performance, particularly in contexts with limited resources. Facilitates strong integration with hardware components in real time.

5. Focused Case study: Importance of UT in Lithography Embedded Systems

Lithography machines, which prints tiny circuit patterns onto silicon wafers, are essential to the production of semiconductors. These devices

employ extremely intricate embedded software systems to manage several subsystems, guarantee accuracy, and maximize throughput. In order to fabricate new integrated circuits with the accuracy and efficiency needed, lithography machines—which are crucial to the semiconductor industry—heavily rely on complex embedded software systems. These machines are very complex and expensive. Each unit of the latest High NA EUV machines costs approximately $370 million [Ref]. The operating cost of these machines are very high. In semiconductor fabrication plants, unscheduled downtime of these lithography machines can cost up to $1 million every day. Therefore, keeping these machines up and running becomes very critical. [5]

5.1 Key Challenges in Lithography Embedded Software Development:

a) Software must adhere to stringent latency requirements (microsecond-level response) in order to function in real-time.

b) Exceptional Accuracy: Wafer placement error margins must be within nanometers.

c) System Complexity: A lithography machine has hundreds of interdependent subsystems.

d) Integration with Manufacturing Execution Systems (MES): Facing automation systems is a need.

e) Security & IP Protection: Guarding against reverse engineering and software manipulation.

The operating costs of these sophisticated machines are huge and hence the machine downtime is very expensive. Therefore, any defect reported from FAB, operating these machines could be very expensive. Considering these facts, more and more emphasis is given to qualifying the software at UT level to find early defects and qualify them on a simulated environment without the need of an actual machine for the qualification.

Depending on the severity of the issue and the stage at which it is found, repairing bugs in the embedded software of a lithography machine might be expensive. With real-time embedded software managing incredibly precise motions, optics, and exposure parameters, lithography machines are extremely complex devices.

5.2 Factors Affecting Bug Fixing Cost:

5.2.1 Stage of Detection

Early (Development/Testing Phase): Less expensive because software can be changed prior to implementation.

Due to manufacturing delays in semiconductors, late (after deployment/production downtime): Costs are exponentially greater.

Complexity of the Issue

Minor UI or parameter changes could be resolved with a $10,000–$50,000 software patch. Critical Real-time Control Bugs: These require in-depth analysis, impact machine accuracy, and can cost anywhere from $100,000 to $1 million or more.

Machine Downtime & Yield Loss

In semiconductor fabs, or fabrication plants, unscheduled downtime can cost up to $1 million every day.Defective chips can result from a flaw that affects wafer alignment or exposure precision, which would raise costs even more.

Software Validation & Regulatory Compliance

Re-certification, regression testing, and QA validation may be necessary to fix a bug, which would increase costs.

Field Service & Deployment Costs

The cost of travel, labor, and hardware replacements might increase significantly if a hardware-related fault necessitates engineers to be on-site.

Best Practices to Minimize Bug Fixing Costs:

- Thorough simulation and software unit testing during deployment.
- Continuous Integration & Automated Unit Testing (CI/CD) for instant feedback.
- Predictive maintenance powered by AI to identify problems before they become more serious.
- To reduce downtime, use remote debugging and patch deployment.

Early defect detection is always in-expensive, as the bug can be fixed in software prior to final qualification in embedded platforms(hardware). Due to manufacturing delays in semiconductors, bugs caught at latter phase (after deployment/production downtime), are exponentially expensive.

4. Conclusions

Unit testing drives embedded system code development and qualification in the absence of hardware by using tiny, fine-grained automated tests of particular behavior that can be done off-target. It gives quick feedback and code confidence before deploying the software in complex embedded systems. This becomes even very critical for high-end and expensive embedded system machines, like Lithography machines, Military & Aerospace Embedded Systems, High-End Medical Embedded Systems & Automotive Embedded

Figure 1. *SDLC*

Figure 2. *CI/CD & importance of UT in CI/CD*

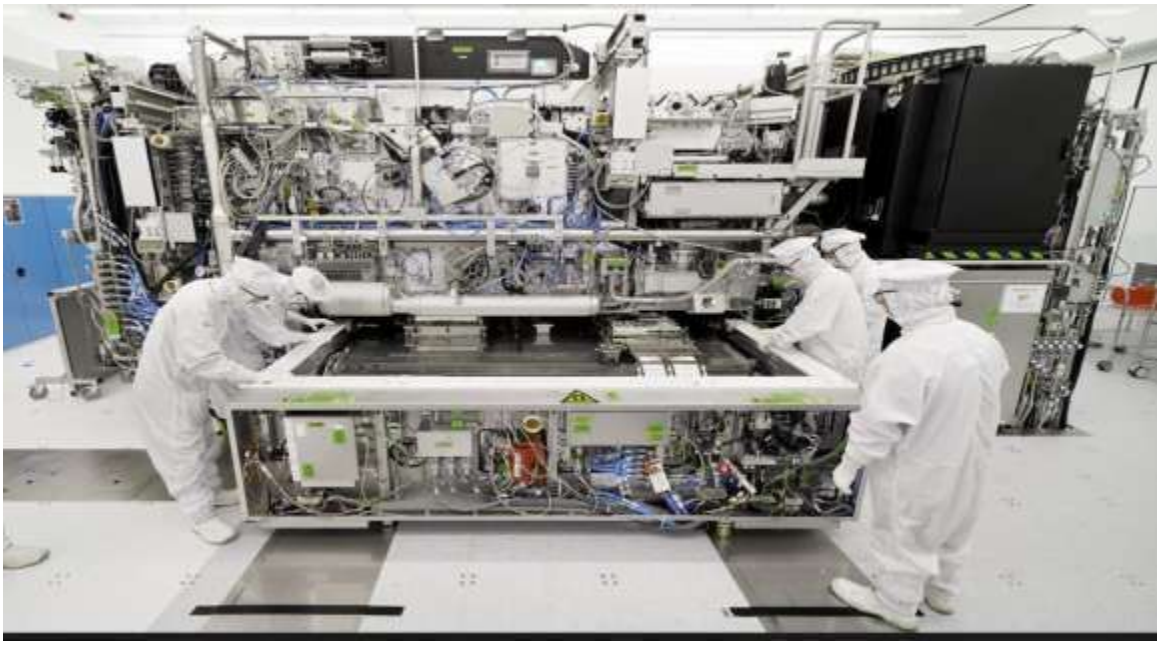

Figure 3. *ASML High NA EUV Machine*

Systems to find the bugs at the earlier phase of software development through Unit Testing. With the leverage of advance AI tools and CI/CD concepts, Unit test will add more value and process/cost efficiency in time ahead.

Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.

References

[1] Embedded System Market. Accessed: Apr. 1, 2023. [Online]. Available: https://www.marketsandmarkets.com/Market-Reports/embedded -system-market-98154672.html

[2]https://iaeme.com/MasterAdmin/Journal_uploads/IJARET/VOLUME_11_ISSUE_12/IJARET_11_12_019.pdf?utm_source=chatgpt.com

[3] Unit Testing For Embedded Software Developmenthttps://dojofive.com/blog/unit-testing-for-embedded-software-development/

[4] For TDD on embedded systems, see Test Driven Development for Embedded C, by James Grenning.

[5] :https://www.datacenterdynamics.com/en/news/tsmc-to-receive-first-high-na-euv-lithography-machine-from-asml-in-q4/#:~:text=It%20is%20unclear%20how%20many,advanced%203nm%20and%205nm%20chips

[6] Q. Liao, Modelling CI/CD Pipeline Through Agent-Based Simulation,"2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Coimbra, Portugal, 2020, pp.155-156, doi: 10.1109/ISSREW51248.2020.00059

[7] D. V. Landuyt, L. Sion, W. Philips and W. Joosen, "From automation to CI/CD: a comparative evaluation of threat modeling tools," 2024 IEEE Secure Development Conference (SecDev), Pittsburgh, PA, USA, 2024, pp. 35-45, doi: 10.1109/SecDev61143.2024.00010.

[8] A. Saxena, S. Singh, S. Prakash, T. Yang and R. S. Rathore, "DevOps Automation Pipeline Deployment with IaC (Infrastructure as Code),"2024 IEEE Silchar Subsection Conference (SILCON 2024), Agartala, India, 2024, pp. 1-6, doi: 10.1109/SILCON63976.2024.10910699

[9] Reinforcement Learning from Automatic Feedback for High-Quality Unit Test Generation Benjamin Steenhoek, Michele Tufano, Neel Sundaresan, Alexey Svyatkovskiy https://doi.org/10.48550/arXiv.2412.14308

[10] VALTEST: Automated Validation of Language Model Generated Test Cases Hamed Taherkhani, Hadi Hemmati https://doi.org/10.48550/arXiv.2411.08254