Mastering embedded Linux in 2025

A comprehensive guide
for development & maintenance

by The Embedded Kit

Dear reader,

Welcome to the synthesis of our 2025 study on embedded Linux system development! Embark on a journey with us as we delve into the ambitions, challenges, pains, and perspectives of equipment manufacturers (OEMs) in their pursuit of developing and maintaining industrial Linux-based devices.

This exploration has been driven by The Embedded Kit team with the desire to better understand the experiences and hurdles faced by OEMs in the realm of embedded Linux development. These insights will serve as the bedrock for adapting our products to bring more value to our users.

It took dozens of interviews with R&D managers, lead software engineers from companies of all sizes, and hundreds of responses to quantitative surveys combined with extensive online research to have a global overview, which we are excited to distill and share with you through this synthesis.

You will find in this report:

- An overview of the challenges encountered by peers, delineated by their level of maturity
- Solutions and best practices to streamline embedded Linux development and maintenance
- Decision matrices designed to help you select solutions pertinent to your unique context.
- Insights and prospects on the future of embedded Linux developments

Before we embark on our Linux journey, we'd like to thank, once again, all the people who shared their precious experience and insights. And to those who haven't had the chance to share their perspectives on this topic, feel free to reach out!

Enjoy the ride,

The Embedded Kit team

Summary

PART 1

State of embedded Linux system development - Where do your peers stand?

Our study led us to engage in conversations and gather experiences from various types of profiles, from embedded software developers to R&D managers, via tech leads and software architects, all working in companies manufacturing equipment (OEMs). Through these discussions, we have identified three different levels of maturity when it comes to embedded Linux system development and maintenance.

1

Teams with little to no in-house knowledge of embedded Linux: These teams often have very limited prior experience with Linux. They find themselves somewhat lost when developing their Linux-based system and aim to develop business applications that add value to their products.
This profile is prevalent among startups and small to medium-sized industrial enterprises with small development teams, in particular companies whose products have traditionally been based on microcontrollers (MCUs).

2

Mature teams with one or more Linux-based products on the market: Even if their Linux systems are already in place, these teams still face challenges related to long-term maintenance and cybersecurity.
This profile is mainly found in mid-sized enterprises and, in some cases, large corporations that have been working with microprocessors for a considerable time and have a dedicated, small-sized team.

3

Embedded Linux experts with large, well-trained teams capable of handling embedded Linux development and maintenance in-house.
This profile often corresponds to extensive teams within large corporations or mid-sized enterprises with a strong Linux culture.

We propose to examine the issues and challenges faced by each of these profile types, as well as the decision criteria used to select solutions that best suit their needs.

1 - Teams with little in-house knowledge of embedded Linux

Profile

- **Startups** entering the market with a new product and limited in-house expertise on Linux, as well as limited bandwidth,
- **Small to medium-sized industrial companies with compact development teams**, especially those whose products have traditionally been based on microcontrollers (MCUs).

Main objectives

Teams lacking embedded Linux experience usually have as their primary goal to **concentrate on their core business expertise**, namely their business applications. They aim to minimize the time spent on the foundational aspects of their product (such as the BSP, Linux distribution, and customization) and focus on developing their business applications while optimizing their costs and trying to meet their delivery deadlines. While some customers may request security features, they generally hold a medium priority due to limited expertise and budget constraints. More crucial to them is **delivering a product with long-term stability** over the next 10 years.

Main challenges

Manufacturers inexperienced in Linux-based systems face several substantial challenges. They suffer from a **shortage of in-house skills** and struggle to comprehend intricate frameworks like Yocto, which are pivotal in Linux system setup. Operating with **small, overburdened teams and tight launch schedules while adhering to budget constraints** further exacerbates their challenges. This often results in increased dependence on external parties, like chip and software providers, leading to communication difficulties due to the involvement of numerous stakeholders.
Furthermore, their lack of in-house expertise makes it challenging to **maintain system quality and security**, and they may miss out on key features such as robust, scalable update system. As well as balancing security and brand protection without overcomplicating an already complex system...

Preferred solutions

If startups often prefer to build their products and Linux layer entirely from the ground up or leverage open-source distributions in an effort to minimize costs, small companies with little in-house expertise may prefer:

- **Seeking assistance from external firms** to support their development efforts
- **Utilizing off-the-shelf distributions** such as Ubuntu or Debian

In terms of maintenance, their approach usually involves minimizing activities related to application updates and infrequent system updates, which may be handled by a service partner.

Finally, a few of these companies may opt to stick with MCU-based devices, as transitioning to Linux can be perceived as a steep learning curve, expensive, and offering minimal return on investment.

Decision criteria

- **Development cost**
- **Time-to-market**
- **Ease of customization**
- **Support**

Expert advice #1 - **Do you need to migrate to Linux and MPU?**

You need to validate the needs of MPU and Linux by confirming which features (which also include your global product roadmap with future features) will be requested.
Do you need to get access to a better development framework? Do you plan to integrate multimedia usage? Advanced graphical UI? Or do we need more power for calculation or an AI algorithm?

Pierre GAL, Head of Product Development at The Embedded Kit

Expert advice #2 - **Build your own MPU team!**

Hiring a good Linux engineer is complex (but not impossible).
But ramping up your current team is another very interesting way to explore. It will empower your team, open new opportunities for them, and help reduce the attrition rate.
In that case, starting with an off-the-shelf distribution and receiving support from an external Linux expert in the beginning may be helpful. And we can help you with that at The Embedded Kit ;)

Samir BOUNAB, CEO of The Embedded Kit

2 - Mature teams with one or more Linux-based products on the market

Profile

- **Mid-sized enterprises & large corporations** that have been working with microprocessors for a some time and have a dedicated, small-sized team

Main objectives

At the forefront of these OEM teams' priorities is the willingness to **deliver new innovative products** on the market to bring more business to the company. They have enough insights to anticipate the development efforts and costs related to embedded Linux system development, validation, and delivery, and they'd love to **accelerate and rationalize these efforts to concentrate on their business applications**.
Another objective is the unwavering **commitment to long-term system stability**, often spanning well beyond one or two decades, ensuring consistent and stable performance over that extended period. This pursuit of stability is intrinsically linked to another high-priority concern: **security**. Safeguarding their products to earn and maintain their customers' trust are paramount, necessitating anticipation, protection, monitoring and patching activities against critical vulnerabilities and threats like CVEs (Common Vulnerabilities and Exposures).
Moreover, these teams also have to manage multiple platforms at the same time, resulting in intricate overall system architecture. As a result, it is crucial for some of these teams to **maintain flexibility within their software development processes and architecture**. This flexibility empowers them to promptly respond to evolving market demands and incrementally optimize their systems. This is why some prioritize retaining in-house embedded software expertise, aiming to **reduce dependence on external providers** and ensuring this sustained level of flexibility. This strategic approach allows them to uphold their commitment to delivering products with long-term stability, lasting for a span of 10 years.

Main challenges

One significant issue that connects various challenges is the difficulty in **averting service disruptions and abnormal behaviors**, particularly in the context of large-scale deployments. These disruptions not only have a detrimental impact on user experience but also incur substantial operational costs.
This cost factor is related with the challenge of managing Linux-based systems, which is known for being time-consuming. **Maintenance activities often fall outside the purview of core engineering teams, diverting resources, and consuming valuable engineering hours** that could otherwise be allocated to more strategic development efforts. Furthermore, the limited support from software and hardware providers compounds the complexity of maintaining system security, making an already intricate task even more daunting.
Additionally, **managing multiple product platforms**, each with its own Linux-based system, intensifies the time-consuming and costly nature of these operations. This complexity is particularly problematic for OEMs operating on limited budgets, who also fear becoming locked into dependencies with external providers. This budget constraint issue, in the grand scheme of things, may be exacerbated by senior management's limited understanding of the complexity of Linux-based systems.

Decision criteria

- **Quality & security**
- **Time-to-market**
- **Support**
- **Supported hardware ecosystem**
- **No vendor lock-in**

Preferred solutions

To tackle their challenges, OEMs will usually opt for one of those solutions:

- **Externalization**: Outsourcing the maintenance and cybersecurity activities to external service providers to reduce the burden on internal resources and ensure the system stability.
- **In-house development and maintenance**: to control the whole system from end-to-end even if it takes more time for the team. Recruiting new members is often useful.
- **Software tools**: using open source or proprietary software tools to accelerate developments and/or simplify maintenance activities.
- **Open Source and community support**: Leveraging support from open-source distributions for cost-effective maintenance and benefiting from community expertise.
- Finally, some chose not to maintain their system at all (and maintain only their application layer), leaving their systems vulnerable to security and stability risks.

Expert advice #1 - Team retention

Keeping your Linux teammates is a key success factor. Such profiles love tech, new developments and may be bored by a pure maintenance period. To manage that properly, keep knowledge as wide as possible, ramp up junior on maintenance activities, and keep high-value tasks for experts.

Julien BERNET, Cybersecurity team lead

Expert advice #2 - Platforming

When you start building a range of products, thinking about platforming is a key topic.
OEMs are already used to build the hardware side. That's, however, less common on the software side, but it's still as important to mutualize development efforts and anticipate maintenance efforts. Products like Welma Yocto Linux can help you with platforming, as it has been designed for this use case.

Pierre GAL, Head of Product Development

3 - Embedded Linux experts

Profile

- **Mid-size companies and corporations with large, well-trained teams** managing embedded Linux development and maintenance in-house. They have been developing a strong Linux culture for over a decade.

Main objectives

Teams fitting in this category focus mainly on **achieving greater control over their systems, ensuring compliance with industry regulations, and maintaining long-term stability while remaining flexible**.
Firstly, they opt to develop their Linux systems in-house to ensure seamless integration with their overall company architecture and legacy applications. This approach allows them to create custom solutions that align with the specific requirements of their products and services. By doing so, they can also tailor their systems to meet the compliance requirements of their respective industries and reduce the risk of legal issues.
Flexibility is also a key objective for OEMs with embedded Linux expert teams. By developing their Linux systems, they can be independent enough to be able to adapt to changing market dynamics, technology trends, and business needs without being tied to a vendor's solutions.
While they develop their systems in-house, they recognize the importance of external support. They remain open to leveraging support services provided by hardware providers or external partners when necessary. This approach ensures that they can access expertise and resources outside of their organization to address complex issues or enhance system performance while maintaining control over their systems.
Ensuring the long-term maintenance of their Linux systems is another objective for them. This includes tasks like regularly switching kernel versions and monitoring CVEs (Common Vulnerabilities and Exposures) to keep an up-to-date, secure, and reliable system over the years.

Main challenges

As these teams already have internal expertise regarding embedded Linux system development and maintenance, they'll face challenges mainly focused on **specific technical aspects of their custom systems**. For instance, they grapple with connecting with legacy applications, ensuring the security and compatibility of custom drivers, and optimizing their Linux systems to achieve greater performance.
Furthermore, they face what we could call a **"platforming challenge"**. This challenge involves the management of diverse platforms, each with unique hardware, software, and application requirements. This management becomes complex when it comes to maintenance, as they have to also keep up with the LTS versions for each platform, a task that can be time-consuming.

Decision criteria

- Quality & security
- Time-to-market
- Support
- Customization & flexibility
- Costs optimizations

Preferred solutions

To tackle their challenges, OEMs with this profile will usually opt for one of those solutions:

- **In-house development and maintenance**: to control the whole system from end-to-end, even if it takes time, as they already have the team to do it.
- **Software tools**: using open source or proprietary software tools to accelerate developments and/or simplify maintenance activities.
- **Open Source and Community Support**: Leveraging support from open-source distributions for cost-effective maintenance and benefiting from community expertise.
- **Hardware provider support or external partner services**: to address very specific and technical issues with their team.

Expert advice #1 - Team organization

Managing a team comprised of multiple Linux experts poses challenges, as each expert holds their unique perspectives on the system, quality standards, future-proof solutions, and personal habits. To address this, it's crucial to establish a well-structured organization that promotes shared responsibilities and actively involves all team members.

Pierre LECOMTE, Solution Director & Product Management Lead

Expert advice #2 - Avoiding dependency and vendor lock-in

Many large organizations utilize renowned software solutions that may result in a vendor lock-in effect, leading to long-term costs. Consider the level of dependency and the associated risks when engaging with a new supplier or adopting a new product.

Pierre GAL, Head of Product Development

PART 2

Solutions & best practices

Off-the-shelf or tailor-made Linux distro?

Linux standard distribution
Ubuntu and Debian are the most classical standard Linux distributions we can find in the embedded world.

Pros:
- ✅ A standard way of working that resembles your Linux usage, making it familiar on the desktop world.
- ✅ Large ecosystem: These distributions have extensive repositories and precompiled packages, making it easier to manage expected features in the system.
- ✅ Regular updates are provided from a patch & vulnerability point of view.

Cons:
- ❌ The package distribution system is not adapted to the embedded world. If you are using Secure Boot, you cannot use the package manager to get updates.
- ❌ Image sizes are usually larger, and the level of customization is limited.
- ❌ Not as flexible as needed for an embedded device. For example, the implementation of advanced security features like secure boot is not easy.

Build our own Linux distribution
A reliable build system is the foundation of embedded Linux development, providing a structured environment for configuring, building, and customizing your embedded Linux image. Yocto Project and Buildroot are two prominent build systems that streamline the development process.

- **The Yocto Project** offers extensive customization options and supports a wide range of hardware architectures. Simplifies the creation of custom Linux distributions, allowing OEMs to fine-tune every aspect of their system. Its layer-based approach and wide community make it a valuable tool for complex embedded projects.
- **Buildroot** is known for its simplicity and ease of use. A great choice for smaller projects or when you need a minimal, fast-booting system. Lightweight, making it ideal for resource-constrained embedded devices. However, Buildroot seems to be less and less supported by silicon vendors and SOM makers.

Pros:
- ✅ Allows fine-grained customization of system images based on specific project requirements, enabling the creation of lightweight and optimized embedded systems.
- ✅ Eases integration on multiple hardware.
- ✅ System configuration is centrally managed using recipes and layers, facilitating version control and reproducibility.
- ✅ Automates the build process, making development and maintenance more efficient.

Cons:
- ❌ Learning curve: These tools can have a steep learning curve due to their complexity. Users need to understand concepts such as recipes, layers, and bitbake to effectively create images.
- ❌ Build time: generating an image may take time due to source-based compilation, which can be a drawback for rapid development.

Select the right Linux distribution for your embedded system

	In-house development	debian	Ubuntu Core	WNDRVR	The Embedded Kit	Development via a service provider
Technical features for embedded systems	☑ Perfectly adapted to your needs	~ May require additional developments	☑ Feature-rich	☑ Feature-rich	☑ Feature-rich; built on 20+ years of experience with OEMs	☑ Perfectly adapted to your needs
Price	☑ Free	☑ Free	☒ High; lack of transparency	☒ High; lack of transparency	~ One time fee to get the source code	~ Reasonable to expensive depending of the project complexity
Time-to-market	☒ Long	~ Medium	☑ Rapid	☑ Rapid	☑ Rapid	~ Medium
Ease of use	☒ Complex	~ Learning curve	~ Learning curve	~ Learning curve	~ Learning curve	~ Learning curve
Flexibility	☑ High	~ Medium	~ Medium	~ Medium	☑ High	☑ High
Quality & security	☒ ~ ☑ Low to high depending on teams' skills	~ Medium	☑ High	☑ High	☑ High	☑ High
Supported hardware ecosystem	☑ Adapted to your needs	~ Medium (not specialized in embedded Linux)	☑ High	☑ High	☑ Partnerships with SOM & Silicon vendors	☑ Adapted to your needs
Absence of vendor lock-in	☑ Independence	☑ Independence	☒ Vendor locked via brand store	☒ Vendor locked	☑ Independence with full source code provided	~ Dependence on skills
Customer support	☒ None	~ Community-based	~ Medium	☑ Good	☑ Direct contact with dev team (EU & US); technical documentation	☑ Dedicated team

The essential of a best-in-class embedded Linux system

1 OTA update

Remote software updates play a crucial role in maintaining and enhancing the security and functionality of connected devices. By prioritizing reliability, implementing the A/B update schema, and addressing potential challenges, we can ensure that these updates are seamless and robust, providing an improved experience for both users and equipment manufacturers.

Besides the established A/B update model, various update schemes, such as the golden image and differential incremental update, exist. However, we won't delve into those details today.

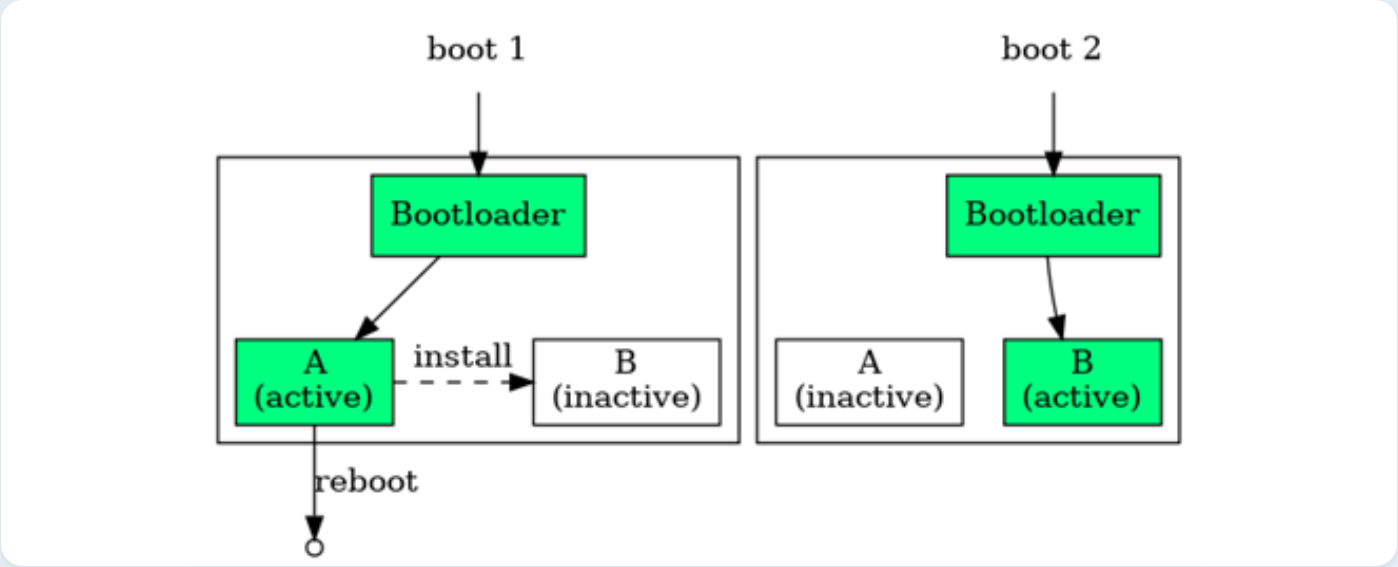
What is OTA update?
OTA update, which stands for "over-the-air updates," refers to the capability to remotely update the root file system, firmware, bootloader, and/or applications and services through a wireless network (like Wi-Fi or mobile telecommunications). Due to language convenience, the term OTA has become standard, even for non-wireless update cases.

Why should you use OTA update?
OTA update is a crucial aspect of modern device management as it enables seamless security and device enhancements without the need for physical intervention on your device, as was the case a few decades ago when using a hard disk or a flash drive to manually download new firmware on each device.

Resisting power and network loss: a top priority
One of the primary goals of OTA updates is to ensure that devices can withstand power and network interruptions, which can lead to abrupt, uncontrolled software shutdowns. In scenarios where thousands or even hundreds of thousands of devices are in use, avoiding a 10% failure rate due to updates is paramount. To achieve this, we prioritize resistance to power and network losses.

How does OTA update work? The A/B update schema, a classical and proven approach
To achieve robustness in remote updates, we employ the A/B update schema. This approach involves having two copies of updateable software components: Part A and Part B, which are crucial for maintaining system integrity during the update process.

When a device boots up, it initially runs a bootloader responsible for managing the starting process. The bootloader's role is to hand over control to one of the partitions, either Part A or Part B, typically containing the kernel and Root File System (Root FS). This becomes the active partition.



During an update, the new software version is installed on the inactive partition (usually Part B). If the update process is interrupted, the active partition (Part A) remains unchanged, ensuring continued functionality. A subsequent reboot directs execution to the updated partition, making it the new active partition. This approach guarantees that the device can roll back to a known, functioning state if the update on the inactive partition fails to validate correctly.

A key feature of this update strategy is the ability to maintain two different versions of software on each partition, ensuring the system can always revert to a previous functional version in case the new one encounters errors. Initially, in version N of the software, partition A is active while partition B is inactive, running on version N-1. However, upon updating the system to version N+1, partition A becomes inactive with version N, while partition B takes over, becoming active with the brand-new version N+1. This cycle continues with subsequent updates, with partition A becoming active under version N+2 when updating to version N+2, while partition B remains inactive under version N+1. This pattern repeats, ensuring a reliable and smooth update process for end-users while maintaining service continuity.

If you need to implement an OTA update solution on your system, the A/B update schema is a reliable mechanism to maintain system integrity.

OTA update solutions

OTA update solutions such as SWUpdate, RAUC, and Mender offer a simplified and efficient approach to handling firmware updates on embedded Linux systems. These tools make it easier for device manufacturers to ensure the security and functionality of devices in the field.

This is why we have seamlessly incorporated them as OTA update options within Welma, our Linux distribution. This integration provides device manufacturers with a ready-to-use, robust method to keep their devices up to date.

While we have focused on the embedded part of this solution, there is another aspect to consider: the update service and fleet management solution that assists in managing and deploying these updates remotely. Mender.io stands out as one of the only solutions on the market that, in addition to the local update system, provides a cloud-based device management platform for handling update campaigns.

It's evident that different strategies and technical solutions exist for updating embedded Linux systems. It's crucial to define the right strategy based on specific needs and the system environment. For example, a platform in a regulatory context may not be suitable for remote updates. On the other hand, a highly connected system with higher cybersecurity risks should allow for quick and robust updates. More specific scenarios may require a customized approach, such as in a constrained connectivity network where an incremental update may be necessary to optimize the amount of data exchanged over the network.

2 Continuous integration

Incorporating continuous integration (CI) practices into your development workflow ensures that changes are continuously tested and integrated. Automated testing helps identify issues early in the development process, reducing debugging and validation time. CI tools and platforms make it easy to set up testing pipelines for your embedded projects.

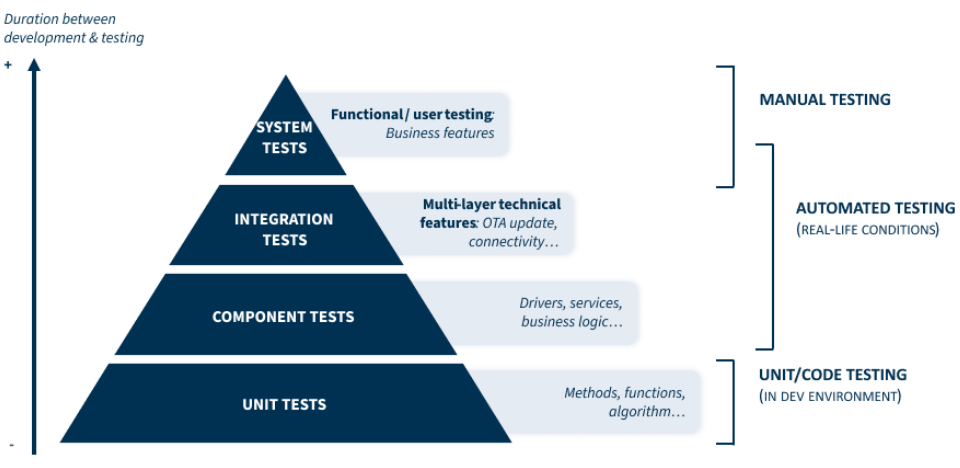
Focus on reproducible builds
Ensuring that your system's build process is consistent and reproducible is crucial for maintaining uniformity and reducing errors during development. To achieve this, using version control (e.g., Git) and pipelines for automated builds (with platforms like GitLab, GitHub, or Jenkins) is a key success factor. This approach allows implementing regular automatic builds to receive prompt feedback, validate pull requests, and generate archives for release builds.

While this way of working has become a standard in software development processes, implementing such a continuous integration system in an embedded Linux context to build a complete operating system remains complex to deploy properly.

3 Automated testing

In the realm of embedded products, automated testing is pivotal. It streamlines development, enhances quality, and minimizes errors, making it essential for reliability and functionality.

What is the pyramid of tests?
Testing embedded products can be challenging, but it is essential to ensure the reliability and functionality of these complex systems. To comprehend the hierarchy of tests, we turn to the well-known Pyramid of Tests, which is a framework for organizing testing efforts in an IoT/embedded project.



- **Unit testing**: At the base of the pyramid lies unit testing. It involves the automated validation of specific components, algorithms, or functions within the code. Unit tests evaluate individual parts of the software with various parameters to ensure they function as expected.

- **Component testing**: Moving up the pyramid, we encounter component testing. This level involves testing larger components or subsystems, such as an embedded system driver.

- **Integration testing**: Integration tests examine how different components behave when assembled. This level of testing focuses on the interaction between various layers and services in an embedded solution, such as OTA updates, which require coordination between cloud platforms, firmware uploads, and data reporting.

- **System testing**: At the peak of the pyramid is system testing. This level employs end-to-end testing, simulating user interactions with the complete IoT system. System tests validate all business features, providing a holistic view of the system's functionality.

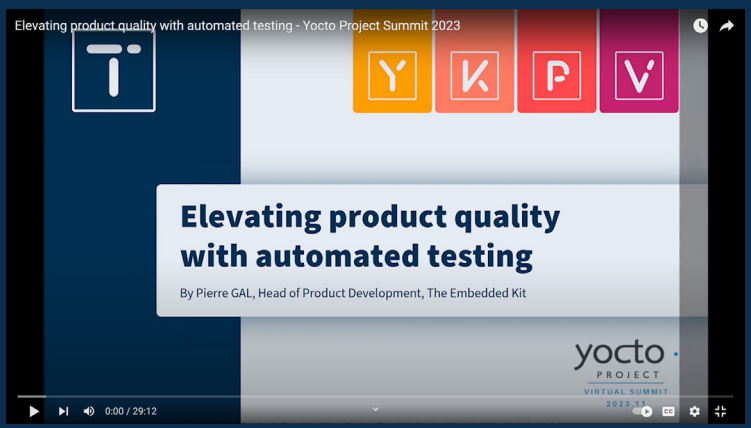
What does on-board automated testing mean?
Automated testing in embedded comes in two flavors: "on-board" and "off-board". On-board automated testing refers to testing on the real hardware or in the actual cloud environment. It is performed automatically but takes place within the real integration environment. Onboard tests are ideal for checking system behaviors that are specific to the actual deployment environment.
What's the difference between manual testing and automated testing?
It's crucial to differentiate between manual testing and on-board automated testing. Manual testing remains essential for various aspects of product development. While on-board automated testing offers numerous benefits, manual testing is irreplaceable for specific tasks that require human intervention and expertise. The key is to strike a balance between manual and automated testing.

What are the benefits of automated testing?
Reducing development-to-testing lag: One significant challenge in embedded development is the time gap between feature development and testing. In traditional testing methods, this gap can be quite substantial, making it more difficult to fix bugs or issues that emerge during testing. The aim of automated testing is to close this gap, allowing teams to discover issues earlier in the development process, making bug fixing quicker and more efficient.

- **Behavioral testing**: Automated testing covers standard behavioral tests, akin to what manual testing accomplishes. This includes verifying that the software meets its expected functional requirements.
- **Edge case management**: Automated testing is capable of managing edge cases that can be difficult and time-consuming to address manually. This ensures that the software is robust and reliable under various conditions.
- **Non-regression testing**: While some non-regression testing can still be performed manually, automated testing can handle a substantial portion of these tests, ensuring that new changes do not break existing functionality.
- **Endurance testing**: This type of testing examines how the system performs over time under normal usage. Automated testing is particularly valuable for such long-duration tests.
- **Robustness testing**: Testing the system's robustness, especially in edge cases, is best achieved with automated testing.
- **Performance measurement**: Automated testing allows for comprehensive performance measurements, which are difficult to achieve manually. This includes assessing how well the system performs under various conditions and loads.

Select the right automated testing tool for your system

	In-house tool development	ROBOT FRAMEWORK	LAVA	The Embedded Kit
Technical capabilities for embedded systems	✓ Perfectly adapted to your needs	~ Feature-rich but not for embedded testing	~ Feature-rich but only for embedded Linux	✓ Feature-rich and optimized for embedded testing
CI integration	✓ Adapted to your needs	✓ Yes	✓ Yes	✓ Yes
Syntax	✓ Adapted to your needs	~ Verbose	~ Ramp up required	✓ Accessible
Cost	✓ Free	✓ Free	✓ Free	~ One time fee
Off-the-shelf	✗ No	✗ No embedded testing features available	~ Ready-to-use but missing interface testing features	✓ Set of embedded testing features provided
Flexibility	✓ High	✓ High	✓ High	✓ High
Customer support	✗ None	~ Community-based	~ Community-based	✓ Access to dev team



Watch our webinar on "How to elevate product quality with automated testing"

4 Cybersecurity

System security

Security measures at the system level are essential to protect against unauthorized access or tampering. They include vulnerability monitoring, providing a device identity using X.509 certificates, and using secure elements for pre-provisioned identities when using cloud services based on robust certificate management.

OS security

Robust security measures must be implemented at the operating system level to protect against vulnerabilities and threats. Start with creating a secure image. Disable unnecessary interfaces to reduce potential attack vectors. Implement secure boot mechanisms to help verify the authenticity of critical components like the bootloader and kernel, as well as an authenticated read-only file to prevent unauthorized writes and system or application modifications. Safeguard secure keys to ensure recovery in case of failure. Finally, employ encryption for data storage.

Application security

Security practices should extend to the application layer to protect sensitive data and functions. Use the principle of least privilege and restrict applications using tools like SELinux, AppArmor, or containers to isolate them from the system.

See our article on "How to build a secure by design Linux-based product"

5 Anticipate your long-term maintenance

The maintenance process in software development is a dynamic and crucial phase that extends beyond the initial release of a product. It involves a meticulous strategy to ensure the ongoing stability, security, and performance of the software. Long-Term Support (LTS) versions, periodic updates, and the management of Common Vulnerabilities and Exposures (CVEs) are integral components of this process.

Why keeping your embedded Linux system healthy for the long run is important
As devices become more and more part of global systems—like high-end devices connected directly or indirectly to IoT hubs, business apps, ERPs, and industrial tools—it's super important to make sure they don't have any weak points. That's why long-term maintenance activities are crucial to prevent vulnerabilities and make sure your product stays top-notch throughout its whole life.

The challenges of embedded Linux system maintenance
Despite the importance of long-term maintenance activities and processes, equipment manufacturers often face challenges in implementing effective methodologies. Challenges include the need for a flexible system versioning strategy, a platforming approach to optimize the time spent on maintenance for multiple products, CVE and LTS update follow-up, and the perception that maintenance is not always an added value activity. To address these challenges, a robust methodology and adapted tooling are essential.

Building products with security in mind to simplify your maintenance
It's a must to weave cybersecurity into the very fabric of your solution for keeping your product in good shape over the long run. Don't wait until your project is out in the wild—make security decisions right from the start (cf previous page).

Key concept: LTS versions

LTS, or long-term support, is a key concept to grasp in software development.
It denotes the extended support provided for all the software layers in your system, akin to what the Yocto Project offers on various kernel versions. This support spans several years, simplifying the task of maintaining stability and reliability in embedded systems. With a maintenance period lasting three to four years, LTS versions serve as the rock-solid foundation for uninterrupted software operation, delivering crucial updates, security patches, and bug fixes. This stability is especially vital in fields like medical devices, where software maintenance may stretch over a decade or more.
The migration strategy from one LTS version to another, involving routine minor updates and periodic shifts to major LTS versions, strikes a balance between embracing new features and upholding a robust foundation.

Build a reference DevOps platform from the get go
At The Embedded Kit, we've mapped our reference DevOps platform in our commitment to ensuring the long-term health and security of our customers software projects.
Anchored in Yocto BSP source code management, it orchestrates a seamless build

process, incorporating SonarQube analyses for source code integrity. At its core is CVE scan, our Linux vulnerability scanner that meticulously detects vulnerabilities in the SBOM. The platform facilitates a sophisticated annotation process, allowing for nuanced analysis and differentiation between actual vulnerabilities and false positives.
Automated testing, with Pluma, ensures that every release undergoes rigorous validation, mitigating the risk of major regressions in the system.

With a holistic approach that encompasses development, analysis, and testing, our reference DevOps platform serves as a robust foundation, empowering developers to navigate the complexities of long-term maintenance with efficiency and precision.

6 Select the right hardware + software combination

Embedded product development is a complex interplay between selecting the right hardware components and crafting software that harmonizes seamlessly with them.

Selecting the right hardware

Choosing the perfect hardware for your embedded product is a nuanced process. Here are some key steps:

- Define hardware requirements based on your product scope. Consider factors such as processing power, memory, connectivity options, and any specialized components needed for your application.
- Assess compatibility: Ensure that the chosen hardware components are compatible with each other and can be easily integrated into your system. Compatibility is key to avoiding later complications.
- Future-proofing: Consider your product's expected lifespan. It's wise to choose hardware that allows for future upgrades or enhancements to meet evolving needs.
- Energy efficiency: In many embedded systems, power efficiency is critical. Choose components that strike a balance between performance and energy consumption.

Developing the right software on top

Pairing the right software with your hardware is equally important. Here's how to do it effectively:

- Choose the right Linux distribution to start with.
- Develop or select application software tailored to your product (and overall company choices!). Consider factors like the programming language, libraries, and tools that best align with your chosen hardware.
- Ensure compatibility with drivers and middleware. Your software stack's harmony depends on the seamless integration of drivers and middleware with your hardware components. It's essential to ensure that these vital components are not only compatible but also optimized for your specific hardware board. This optimization minimizes development time and ensures robust functionality.
- Integrate security from the start (cf section on cybersecurity).

At The Embedded Kit, with Welma, we pre-integrated several hardware and SOMs (iMX8, STM32MP1, etc.) to kick-start your software developments. See the list of pre-integrated hardware here.

PART 3

Perspectives

In the rapidly evolving landscape of embedded Linux system development and maintenance, several key considerations shape our approach and outlook for the future.

Yocto Project: the best solution for tailor-made embedded Linux systems

We firmly believe that Yocto remains the optimal choice for equipment manufacturers' embedded Linux developments, offering unparalleled advantages in custom configuration, optimization, cybersecurity, documentation, and strong community support. Nevertheless, it's important to acknowledge that Yocto can present complexity in its management and implementation. While the benefits are substantial, addressing the intricacies of Yocto may require dedicated expertise and resources to ensure a smooth and successful integration into your projects.

Ownership and control

The need for ownership and control over embedded systems is a fundamental principle guiding more and more OEM strategies. Having complete control over their systems provides the flexibility to adapt to changing requirements and technological advancements. In a world where the lifespan of embedded systems can extend for 10 to 20 years or more, this long-term view is critical. It ensures that systems remain viable and secure throughout their extended lifecycles without the need to migrate to another system because of a provider deficiency.

Cybersecurity is a priority

We are convinced that the landscape of embedded Linux development and maintenance will soon undergo a transformation due to newly published cybersecurity regulations. These new regulations, such as the Radio Equipment Directive (RED) and the Cyber Resilience Act (CRA), make security restrictive for embedded systems, depending on their level of criticism. As such, it's important to start integrating robust cybersecurity measures into embedded systems, safeguarding them against evolving threats and regulatory compliance demands.

Off-the-shelf hardware + software packages

Looking ahead, we see the future of embedded systems as a convergence of hardware and software packaged offers. OEMs will increasingly seek comprehensive solutions that include both hardware and software components, allowing them to accelerate their development processes. This trend empowers OEMs to focus on their core business expertise while relying on integrated hardware and software packages to handle the low-level intricacies.
That's exactly what The Embedded Kit is trying to achieve in partnership with renowned silicon vendors like STMicroelectronics, NXP, Intel, MediaTek, and SOM makers like Ezurio, Aaeon, and more.

PART 4

Conclusion

This exploration helped us identify how to simplify embedded Linux development & maintenance challenges faced by equipment manufacturers.

That's why we created The Embedded Kit, a comprehensive toolkit for embedded systems. **The Embedded Kit offers everything device-makers need to build, connect, test, and secure a custom embedded Linux system**. Its four off-the-shelf products (Welma Yocto Linux, Kamea IoT, Pluma automated testing and CVE Scan) were designed specifically for equipment manufacturers, to ease and accelerate their product development while giving full source code control.

Want to see a demo? Or just share some feedback? We'll be happy to chat with you :)

This report is based on community contributions from all over the world. We would like to thank, once again, the people who took part in our exploration, and the witekians who have shared their knowledge, who have been working side by side with equipment manufacturers for years to build, customize and maintain their connected solutions.

The Embedded Kit

theembeddedkit.io