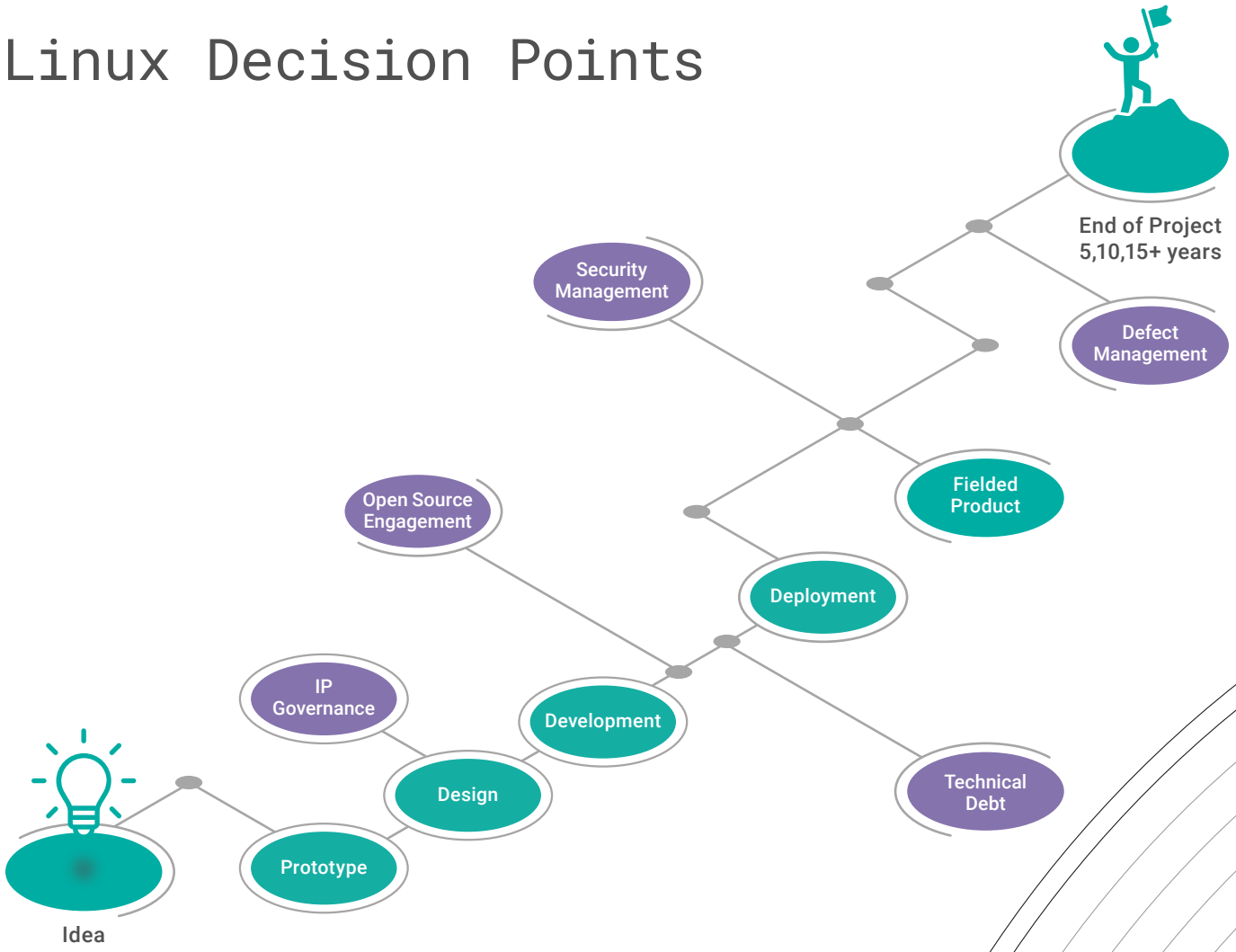
/From Prototype to Post Deployment: Linux Decision Points

Developing embedded solutions can be a journey. Not all applications start at the same place in the journey or take the same development path, but a common theme applies to the steps involved. First there's the idea or concept, followed by prototyping; then design, development, deployment, and post-deployment maintenance.

Linux Decision Points



This e-book will walk the reader through this journey and touch on key considerations throughout development. But before starting the journey, we should ask ourselves: What is our business strategy for open source leadership? Unique code contribution? Delivering a service? Or building an innovative product?

This becomes important, as it can be used as a guiding light for making big decisions.

Why Linux?

So why is Linux appealing at the edge? The edge is a natural extension of the cloud, which was built on Linux and open source technologies and practices. The Linux Foundation openly cites more than 235,000 developers, more than any number you'll find at a single company. In addition, its open source contributions consist of more than 19,000 contributing companies as well as 1.15 billion lines of code.

ON THE ADOPTION SIDE, HERE ARE SOME OTHER FACTS ABOUT THE LINUX PLATFORM:

100% of supercomputers use Linux.

95% of public cloud providers use Kubernetes.

70% of global mobile subscribers use devices run on networks built using ONAP.

50% of Fortune Top 50 blockchain deployments use Hyperledger.

50% of all global auto shipments supported by OEMs use Automotive Grade Linux.

At the end of the day, how does a company decide whether to use a COTS distribution or a specific vendor (either a semiconductor vendor, board vendor, or commercial OS vendor)? The answer ties back to the company's business strategy and the factors critical to its success. Is a competitive control point needed to level the playing field? Or is innovation the primary focus? This should help guide you on your journey, and the answers may evolve and change over time.

Prototype

We begin by putting an idea or concept to the test, as in making a prototype. Developers will use whatever is readily available to them and learn, fail, recover, and create whatever they need. This could involve ordering development kits (such as Raspberry Pi), downloading free distributions, monitoring forums, grabbing interesting code from GitHub, or emulating hardware or kernel subsystems — whatever is needed to prove a concept. This is an important phase in which developers learn as much as possible to have deeper insights for future decision-making.

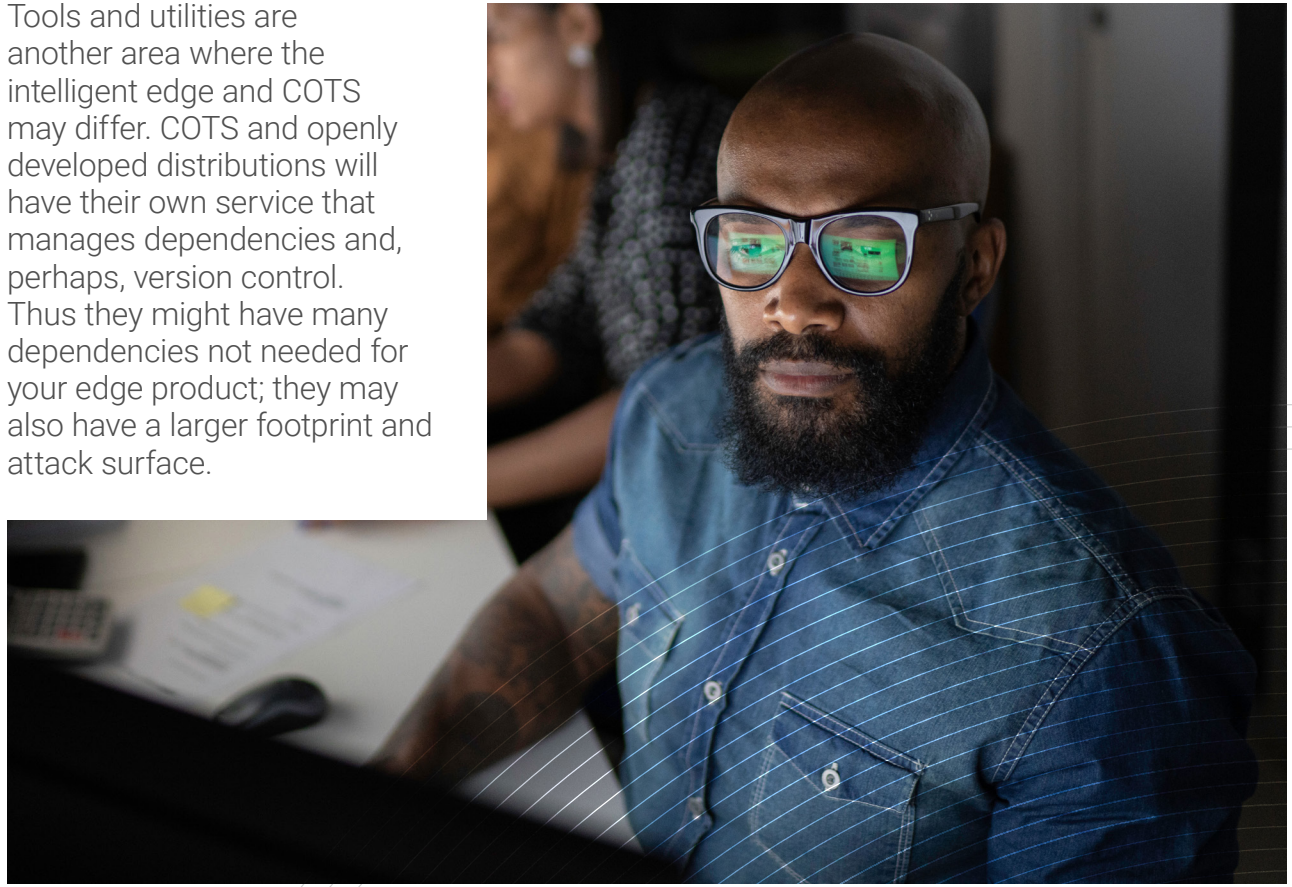
Design

The design phase involves assessing requirements and technologies. If we look at the requirements at the intelligent edge, they vary widely depending on the edge solution. It may be tempting to start with an enterprise commercial off-the-shelf (COTS) Linux distribution and be finished. But consider COTS vendors and how their distributions deliver against the general needs for a larger population, and the way in which requirements and expectations tend to be more specific closer to deployment. Those with more specific edge application needs will likely not intersect on all requirements for a COTS distribution, which in turn might not apply enough toward the edge. Examples of these requirements are:

- **/ Nonstandard hardware**
- **/ Certification (in some cases)**
- **/ Performance, real time, and low latency**
- **/ Small footprint**
- **/ Security**
- **/ Artificial intelligence and machine learning**

In many cases, edge hardware is nonstandard, with unique components and industry-specific device certifications that are unique and likely not supported by COTS hardware. COTS solutions may offer certifications that are aimed at a broad audience and not of particular interest to an open source community.

Tools and utilities are another area where the intelligent edge and COTS may differ. COTS and openly developed distributions will have their own service that manages dependencies and, perhaps, version control. Thus they might have many dependencies not needed for your edge product; they may also have a larger footprint and attack surface.



IP Governance

Governance is about protecting your intellectual property (IP) and protecting the contributions of communities with licensing. From a corporate perspective, a company's IP is at risk without a strong policy that includes some of the following key principles:

- **Developers must receive approval before integrating open source into the product code base.**
- **Third-party software, including open source, requires auditing.**
- **Approval for code in one product doesn't mean it's acceptable for use elsewhere.**
- **This protective policy must be in place before shipping to a customer.**

A key emerging trend in open source software is the creation and use of a software bill of materials, or SBOM. The SBOM is shipped with your product; it identifies all the software components in your product and the licenses they use. Another common practice is to create an Open Source Program Office to create the policy and execute the governance practice. The Open Source Program Office would provide oversight for all upstream contributions as well as licensing and governance for all projects in the organization.

Open Source Engagement

Engagement with open source communities must be grounded in your business priorities and business strategy. The Linux Foundation has dozens of embedded projects, and you will engage on the subset of those projects (including the Yocto Project) that is consistent with your business vision. Engagement can be viewed in four categories:

Consume

In this category, developers simply take what they can and leverage what is available, with minimal or no engagement with the community.

Participate

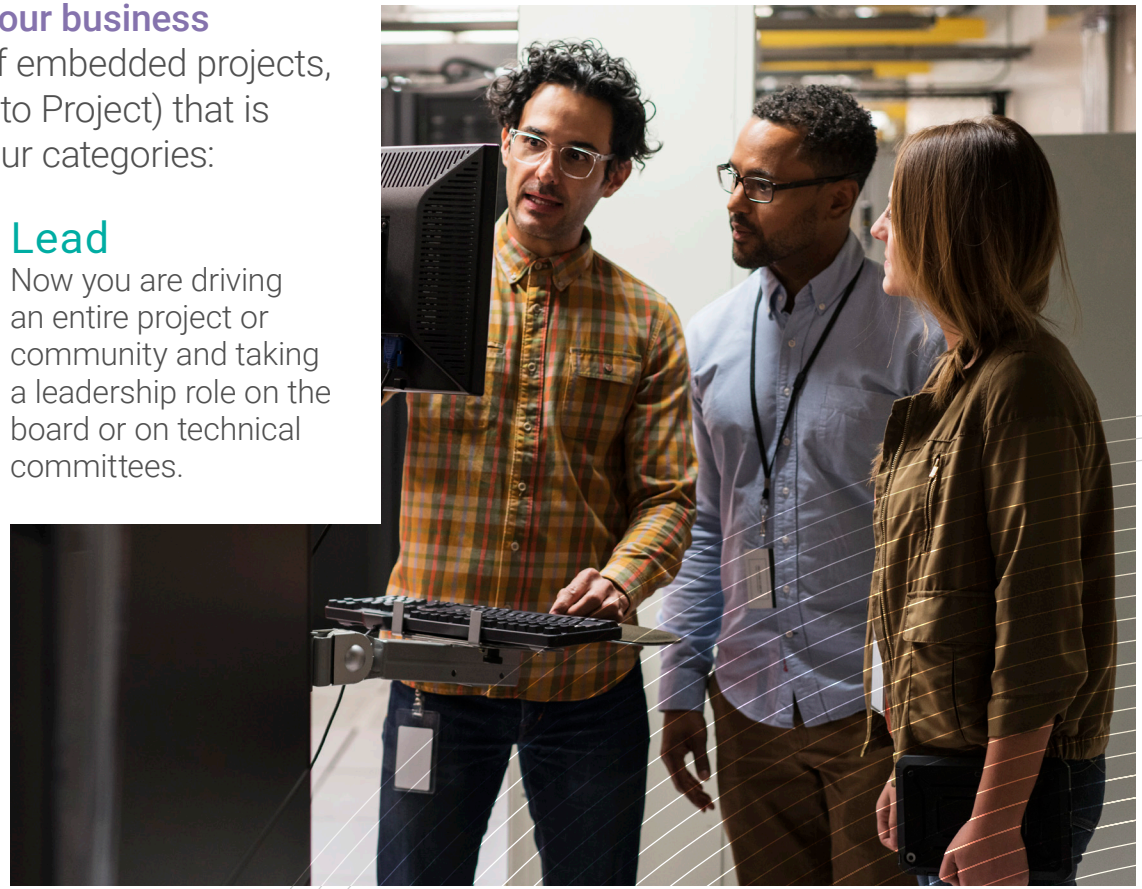
This category represents a stronger two-way engagement and aligns more with the community's goal. Projects may have enough similarities with your own that you'd want to track or evaluate using one or more of them, potentially purchasing an introductory-level membership.

Contribute

Here your participation becomes more influential and represents your core values. This is often done with key projects that your product is dependent on, when you might wish to have more influence in getting code upstreamed.

Lead

Now you are driving an entire project or community and taking a leadership role on the board or on technical committees.



Technical Debt

Technical debt is the cost of maintaining source code caused by a deviation from the main branch where joint development happens. It's the enemy of the pursuit of business strategy and innovation, burdening it over time with the weight of unnecessary and distracting maintenance. Developers are faced daily with the need to solve critical bugs without timely community support. Often they are forced to decide whether to wait and push their innovations upstream or move forward and release as early as possible. If the innovations or patches are not pushed upstream, the result may be long-term technical debt. The debt must be managed aggressively and pulled through to make it mainstream again. Up front, this can be costly and resource heavy, but in the long term it can be the cheaper solution.

If developers or leaders find themselves taking on technical debt without a long-term plan, and there seems to be no way to avoid this situation, it's best to find a partner or service to take it on instead. Some typical symptoms of technical debt:

- **An inability to include upstream innovations and features in your code is one signal.**
- **Increased security risks — who is monitoring your code for CVEs? If it is upstream, the community will help monitor those plus your code's induced issues.**
- **Increased code maintenance time and effort — the complexity of getting technical debt to work correctly with upstream code increases over time.**
- **Increased onboarding time; this is subtle but important and is due to the need for insider developer training on the code base.**
- **Similarly, difficulty in hiring or maintaining open source developers is another sign.**

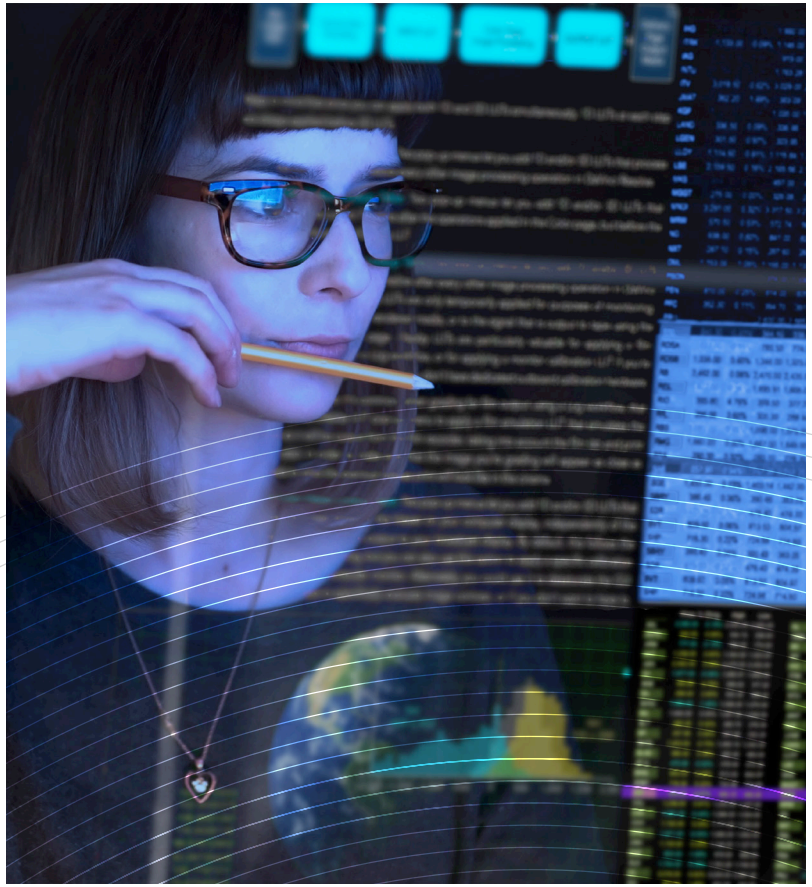
Linux Security and Defect

With a recent executive order on "Improving the Nation's Cybersecurity," it has now become more important than ever to implement protection against aggressive new threats and perform security assessments. The Linux Foundation has announced the formation of the Open Source Security Foundation (OpenSSF), which is committed to improving the overall security of open source software. However, awareness of the processes, resources, and talent requirements to manage security responses may not be straightforward and can easily drop off. If security management isn't fully aligned with your business strategy consider that it may not be a differentiator and may also require a full-time team of experts.

As the complexity of the open source code base grows, so does the number of common vulnerabilities and exposures (CVEs). As the developer leverages open source code, it's important to consider that it might have flagged CVEs, but not all will be relevant to your implementation or represent a high enough risk to prioritize and fix immediately. And as community patches become available, it's important to verify they're acceptable for your product or if backporting is needed. If not, CVEs can also add to your technical debt.

Managing defects is similar to managing CVEs, but there's a twist: If you've positioned yourself as a contributor in an open source community, you can influence the direction and adoption of a patch. But you'll need to work it upstream and implement it in a way that benefits everyone in the community.

If engagement on either CVEs or defects is at a minimum, partners (such as Wind River®) are available to help manage your distribution defects over the long term.



Wind River Can Help

Ultimately, everyone will face an obstacle that impacts speed, resources, and quality. Below are some of the elements associated with most of the challenges developers face on their Linux journey:

- **Platform architectural assessments, software design, and implementation**
- **Security vulnerabilities alerts, analysis, and mitigation**
- **Long-term Linux platform security and defect maintenance and support**
- **Performance and reliability requirements**
- **The latest industry-specific features and standards**
- **IP compliance audits and remediation recommendations**

Wind River can be a partner to you, no matter where you are in the product lifecycle. Available support includes solution assessments and implementation, security compliance, analysis, remediation, and lifecycle performance assurance services. Know your business strategy, reduce your technical debt, and reach out for help when you need it.