THE LINUX FOUNDATION

OPEN SOURCE SUMMIT NORTH AMERICA

Engineering Quality in a Fast-Moving Open Source Project: WPE WebKit

Mario Sánchez-Prada

Minneapolis, May 18th 2026

igalia

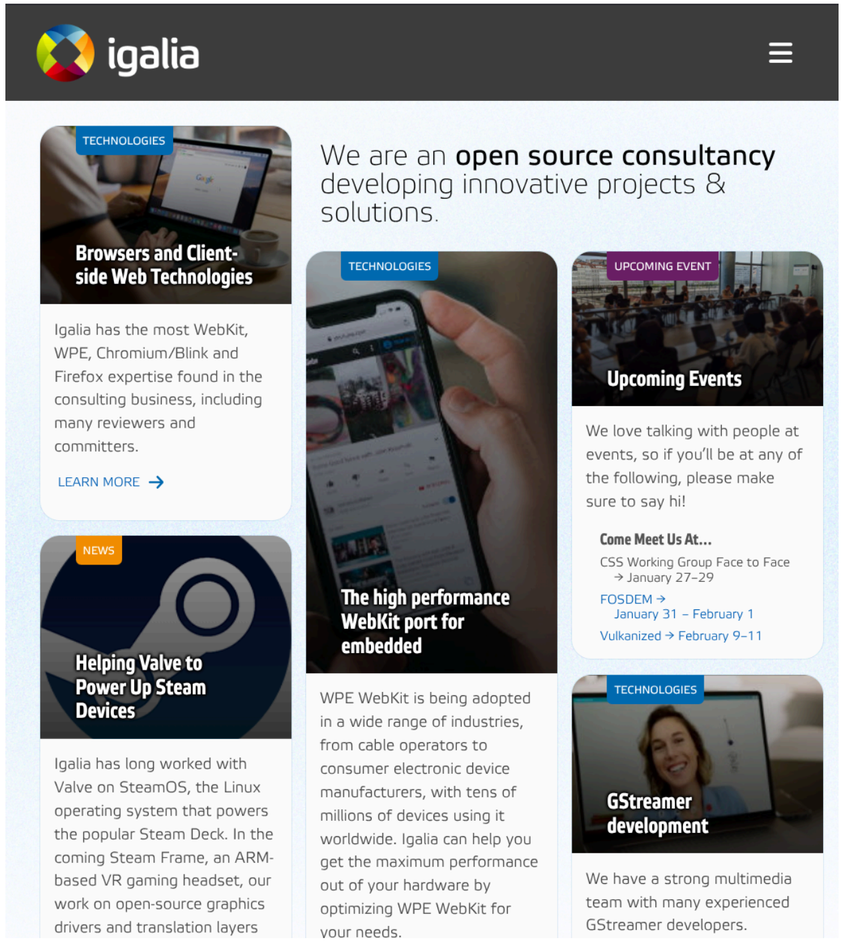
About Me

- **Software Engineer** and partner at **Igalia**
- **Open Source contributor**: GNOME, Chromium, WebKit
- **Other work**: Linux-based OSes (e.g. Endless OS), Flatpak, Samsung Smart TV, Maemo (Nokia)
- Currently **coordinating** Igalia's **WebKit team**

Contact me: mario@igalia.com

About Igalia

- **Specialized Open Source consultancy**
  - Founded in A Coruña (Spain) in 2001.
  - Fully remote and with a flat structure.
- **Top contributors** to the main **Web Engines**
  - Second-largest contributor to WebKit after Apple.
  - Maintainers of the two Linux WebKit *ports*.
- **Other OSS work**:
  - Graphics, multimedia, compilers, Linux kernel...
- **Members of different working groups**:
  - W3C, WHATWG, TC39, Test262, Khronos...



https://www.igalia.com

Slides in PDF:

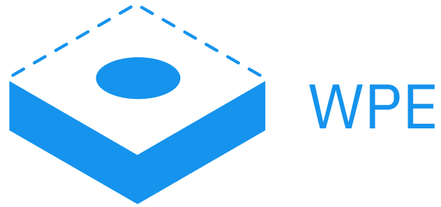
Outline

1. What Is WPE WebKit?
2. The Quality Challenge
3. Quality as Continuous Engineering
4. Testing Strategies
5. CI and QA Infrastructure
6. Stabilization Windows and Releases
7. Aligning Upstream and Downstream

What Is WPE WebKit?

What Is WPE WebKit?

Official **port of the WebKit Web engine** for Linux **embedded devices**.



- **Modern Web Platform** implementation, with a focus on:
  - **Performance**, **security**, **low resources footprint**, **flexibility**.
- **Backend-based I/O**: DRM, Wayland, headless or custom
- **Hardware acceleration** across many different SoCs
- **Ships in many commercial products**:

  TVs, set-top boxes, home appliances, digital signage...

ℹ️ **https://wpewebkit.org**

A Fast-Moving Codebase

📦 **Millions of lines** of C++, JS, tests and tooling

🔁 **Hundreds of patches** landing on `main` **every week**

👥 **Many contributors** from different organizations

🧩 **Multi-platform support**: Linux, Mac, iOS, Windows, PlayStation…

🚀 **Ships in production** while evolving daily

From Codebase to Products

- WPE WebKit's upstream code runs in **very different products**
- Each has its **own deadlines**, **certifications**, **HW** and **release cycles**
- **Upstream and downstream** are different but **share the same fate**
- **Quality is not optional**: Regressions can ship to millions of devices

The Quality Challenge

🤔 **Keeping quality** stable is **hard**

🤷 **Errors and regressions** are **unavoidable**

Keeping Quality Stable Is Hard

🔄 **Constant change**: hundreds of weekly changes on `main`

🧮 **Combinatorial explosion**: ports x archs x HW x configs

🕵️ **Subtle regressions**: perf, memory, rendering, flakiness

⏱️ **Late detection is risky**, delaying fixes complicates maintenance

🔀 **Downstream divergence** increases technical debt over time

😐 **Invisible work**: important, but mostly noticed when there are issues

Errors and Regressions Are Unavoidable

🚫 **Unrealistic goal**: prevent bugs in a project of this size

🎯 **Realistic goal**: minimize impact and time to react

- **Detect** regressions close to introduction.
- **Understand** scope and severity.
- **React** before further propagation.

Quality measured by how we react, not by the absence of bugs

- Organizing our infrastructure in different "**feedback loops**" crucial for detection

Quality as Continuous Engineering

Not a Final Validation Phase

🧪 **Traditional model**: develop → "QA phase" → release

⚠️ This **doesn't scale** when code evolves **every hour**

🔄 **Quality lives in everyday engineering**:

- Continuous triage of failures, flakes and regressions.
- Pre-commit checks, post-commit bots, perf dashboards.
- Stabilization windows tied to the release cadence.

👥 **Shared responsibility**, not a separate team's job

Layered Feedback Loops

⏱️ **Seconds**: local builds, unit tests, linters

🕐 **Minutes**: pre-commit Early Warning System (EWS) bots

🕐 **Hours**: post-commit builds running more tests across all ports

📆 **Days**: performance dashboards, longer test suites, fuzzers

🗓️ **Weeks**: stabilization, release branches, downstream deployments

Each loop catches a different class of issue

They **reinforce each other**

Testing Strategies

📚 A Layered Test Suite

🧩 Cross-Platform Coverage

⚠️ Flakiness Is the Real Enemy

A Layered Test Suite

🔌 **Unit tests** (API tests): isolated tests on the public API

🖼️ **Layout tests**: rendering, layout, behavior at engine level

🌐 **Web Platform Tests**: cross-engine conformance and interoperability

🧮 **JSC tests**: JavaScript correctness and performance

🚀 **Performance tests**: MotionMark, Speedometer, custom...

🛡️ **Fuzzers and sanitizers**: ASan, TSan, UBSan, libFuzzer

Different layers catch different kind of bugs

Cross-Platform Coverage

- **Each patch** must keep working across:
  - **Multiple ports**: Apple, GTK, WPE, PlayStation, Windows...
  - **Multiple architectures**: x86_64, ARM64, ARM32.
  - **Different HW/graphics stacks**: Mesa, Vivante, Mali, VideoCore...
- **Not every config can run every test** on every commit:
  - **Tiered coverage**: representative configs pre-commit, broader matrix post-commit.
  - **Special-purpose bots** for niche but critical configurations (e.g. WPE on Android).
- **Downstream CI** extends the matrix with vendor HW
  - Feeds useful information to the upstream project as well.

Flakiness Is the Real Enemy

- A **flaky test** is **worse than a failing one**
  - Erodes trust, and real regressions hide in the noise.
  - People learn to ignore CI, and real regressions hide in the noise.
- Treat **flakiness as a first-class bug**:
  - Quarantine fast, investigate seriously, fix the root cause.
  - Do not just mark them as *expected* and forget.
  - Track flakiness rates over time, per test and per platform.

CI and QA Infrastructure

What CI Does for WPE WebKit

🏗️ **Builds** every change across ports and configs

🧪 **Runs test suites**: API tests, layout tests, WPT, JSC...

🚦 **Pre-commit EWS bots** detect issues before merging

🤖 **Post-commit bots** track `main` after each patch

📊 **Dashboards** make failures visible and actionable

🔁 **Bisection and triage tooling** localizes regressions

Managing the CI Infrastructure

🔧 **Infrastructure as code**: it has bugs, regressions and tech debt

🧰 Needs **reviewers**, **owners**, **tests**, **on-call rotations** and **monitoring**

🏷️ Failures must be **clearly attributed** (e.g. regression? flake? infra?)

📈 **Capacity planning matters** (e.g. build farms, device labs, queues)

Example: Performance Dashboard

MotionMark 1.3 improvements on the 64-bit Raspberry Pi 4

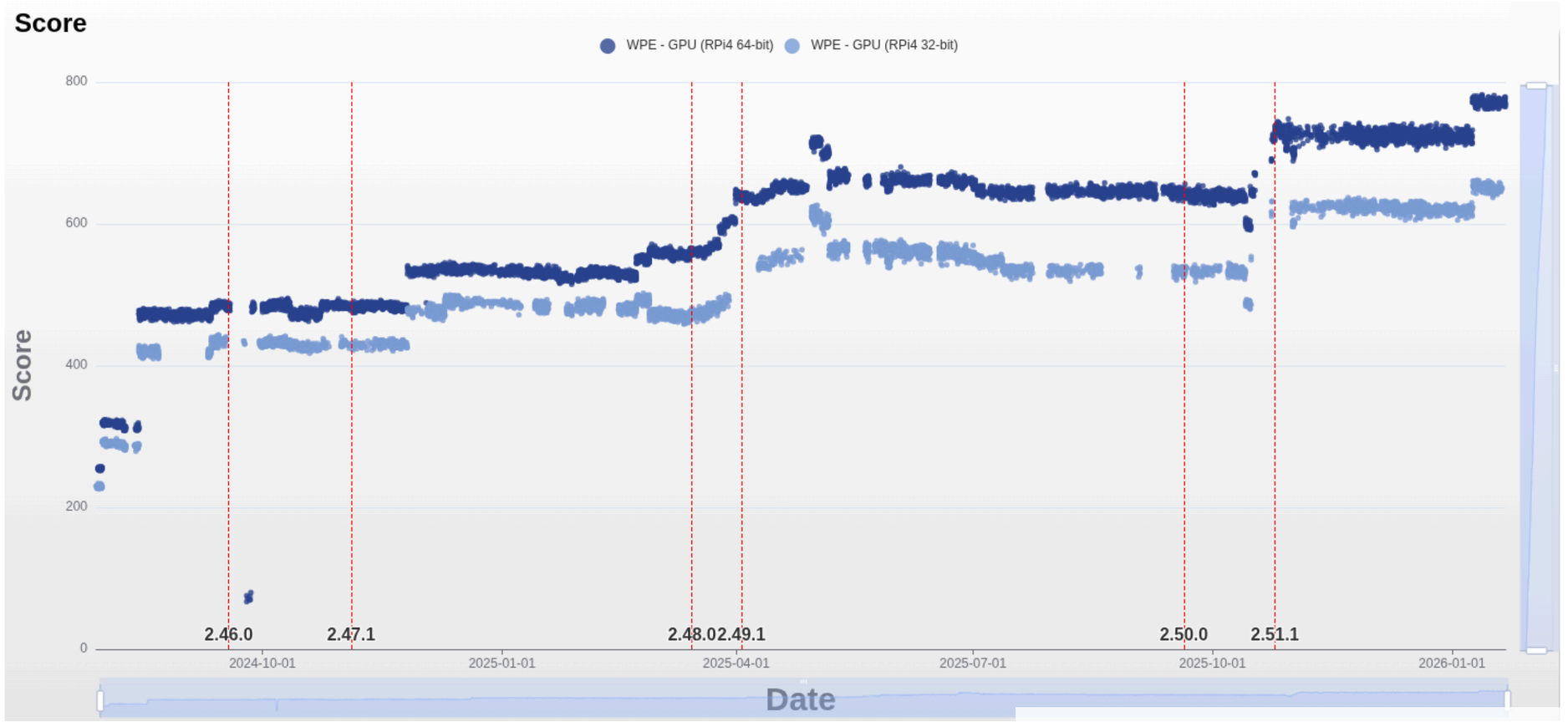
Test	Score July 2024	Score April 2025	Score October 2025	Score January 2026
Multiply	501.17	710.75	697.15	678.93
Canvas arcs	140.24	820.64	859.68	859.48
Canvas lines	1613.93	3025.16	4648.54	7508.43
Paths	375.52	4268.87	3953.83	4288.59
Leaves	319.31	480.19	684.72	673.94
Images	162.69	265.14	263.19	267.88
Suits	232.91	444.55	388.62	399.03
Design	33.79	63.99	114.09	100.24
OVERALL	**254.15**	**634.49**	**737.56**	**778.99**

Continuous tracking turns one-off gains into sustained improvements.

Example: Performance Dashboard

MotionMark 1.3 improvements on the 64-bit Raspberry Pi 4

Last processed revision - RPi4 32-bit: **305815@main·RPi4** 64-bit: **305924@main**



Stabilization Windows and Releases

Why Stabilization Windows?

- **Main** branch is **always moving**, but **products** need **stable cut points**
- **Stabilization windows** act as a **bridge**, providing a period where:
  - Risky changes are deferred.
  - Focus shifts to bug fixing, performance, and triage.
  - Test failures are taken extra seriously.
- Produce a **release branch** of known, characterized quality
- **Regular cadence**, tied to the release schedule

Inside a Stabilization Window

🚦 **Triage** the open bugs: what blocks the release, what doesn't

🧯 **Fix** the must-fix only, defer the rest with clear owners

🔎 **Run extra suites** beyond the regular CI budget

📉 **Compare** against previous releases: perf, memory, stability

🔧 **Backport** selected fixes to active stable branches

📝 **Document** known issues, expected failures, risks

Aligning Upstream and Downstream

Downstream Is Tightly Coupled with Upstream

- Downstream lives on **a moving upstream**, underlying code is the same
  - Quality of downstream products reflects upstream's, with **delay and delta**.
  - This is always the case, regardless of the size of the delta with upstream.
- **Bigger delta ⇒ harder integration** and **bigger maintenance cost**
  - Harder and more complex rebases with more patches to maintain.
  - Harder to report and fix issues, slower to get improvements from upstream.
  - Harder to collaborate with upstream in general (i.e. heavily modified baseline).

Practices That Keep Both Sides Healthy

- **Develop on top of `main` branch**, not on stable branches.
- **Contribute back** instead of forking quietly
  - **Report bugs upstream** with reproducers.
  - **Upstream-friendly patches**: small, well-described, tested.
- **Frequent rebases** against newer upstream versions
- **Downstream CI** mirroring and extending upstream coverage
  - Same tests, plus product-specific HW and configurations.
- **Communicate roadmaps** for better alignment with upstream

Closing the Loop

- Downstream often sees issues **upstream cannot reproduce**
  - Specific SoCs, drivers, content, real-world workloads.
- That info is **gold** for upstream:
  - New tests, platforms, specific performance-related scenarios.
  - Not sharing this information with upstream is a common mistake.
- **Closing the loop** with upstream **benefits everyone**:
  - **Report** issues with **enough context** to be acted on.
  - **Contribute fixes** and **tests** that protect the fix.
  - **Share performance data** from real devices.

Higher upstream quality ⇒ lower downstream cost

Conclusion

🛠️ **Quality is a continuous engineering process**, not a final phase

- Errors and regressions are inevitable; what matters is detecting and reacting fast.

📐 **Layered feedback loops and test suites** reinforce each other

- Pre-commit, post-commit, dashboards, unit tests, layout tests, performance tests...

🧪 **CI and QA infrastructure** are first-class products

- Build farms, dashboards, bots, triage tooling. All need owners and care.

🗓️ **Stabilization windows** enable cutting releases out of the main branch

- Enable a process to decide what goes in a release, run more tests, document issues...

🤝 **Aligning upstream and downstream** is the key multiplier

- Frequent rebases, shared CI signal, contributing back... benefit quality on both sides

Further Resources

WebKit:

- Website: **https://webkit.org**.
- Documentation: **https://docs.webkit.org**.
- Mailing list: **https://lists.webkit.org/mailman3/lists/webkit-dev.lists.webkit.org**.

WPE WebKit:

- Website: **https://wpewebkit.org**.
- Mastodon: **https://floss.social/@WPEWebKit**.
- Bluesky: **https://bsky.app/profile/wpewebkit.org**.
- Mailing list: **https://lists.webkit.org/mailman3/lists/webkit-wpe.lists.webkit.org**.
- Matrix: **#wpe:matrix.org**.

Slides in PDF:

Questions?

Mario Sánchez-Prada
mario@igalia.com

WPE


igalia