Open Source as Supply Chain:

The Need for Provenance, Traceability, Upstream Alignment

Whitepaper

www.codethink.co.uk
sales@codethink.co.uk
+44 161 660 9930

If you are delivering complex systems or technology projects in 2017, it is extremely likely your work involves open source software to some degree. You may or may not be fully aware of your organisation's dependence on open source, but given the unavoidable economic advantages of free versus paid-for, it is almost certainly increasing.

Major players like Apple, Intel, ARM, IBM, Oracle, HP, Samsung, Cisco, Facebook and Google are all proactive contributors to and exploiters of open source projects. Even Microsoft is making its solutions work with Linux, and has open sourced core technologies such as the .NET framework.

Here we consider use of open source from the perspective of companies and teams working to bring new or updated products or services to market, and to maintain them over extended production lifetimes. We draw on Codethink's experience across a wide range of organisations; IP and silicon vendors, electronics companies, system integrators, OEMs, ISVs, OSVs, service providers and industry initiatives.

We aim to provide insight and answers for the following questions:

1

Where does open source come from, and why does that matter?

2

What happens if we treat open source like our own IP?

3

How can we minimise our costs and risks when using open source?

The Key Lessons

The key lessons we highlight from the case studies here are:

1

Organisations can and should treat open source with the same diligence as they apply to elements of their supply chain. This includes

- Understanding Bill of Materials, which means establishing which components are we using, and where they come from, and terms associated with their use
- How much are we paying an integrator/supplier for these components vs the cost of going direct. Taking open source direct from upstream is free, but with it comes the overhead of direct interaction and more suppliers overall
- Dealing with new revisions/versions of components on an ongoing basis
- What happens when components reach end-of-life
- Who carries liability for problems associated with components. In many cases it is simply impossible to assign or delegate this risk, due to the 'as is, no warranty' nature of open source

2

An ongoing, reliable and effective process for establishing provenance and reproducibility of open source software is a must for major projects. It is unsafe to

- Rely on the continuity of community-provided infrastructure
- Use open source code without the ability to demonstrate provenance and licence compliance

3

Proactive engagement with the upstream open source communities gives better visibility of where projects are going, including early warning about long-term threats and technical opportunities

4

Any project with heavy dependence on specific open source projects should consider aligning with the upstream directions of those projects. This reduces overall maintenance costs and avoids the trap of being isolated on old software which will become increasingly insecure and hard to fix.

Case Study

1

Free software? No, our code is commercial quality. Open source? No sir!

An experienced software production strategist joined an international vendor of high-end electronics equipment. A key objective for his role was to help them increase their effectiveness and reduce costs for development and maintenance of software in the company's products. Each unit is multifunction and contains many microprocessors. The cost and complexity of the software required for each device is a growing problem.

When asking his new colleagues about the company's approach to open source he was surprised to discover that many senior engineering managers did not even understand the concept. In general the established company perspective wrote off open source as unreliable, insecure, untrustworthy.

Further investigation established that most of the software in the units was actually being provided by subsystem suppliers as part of their deliveries. The official line from management in those organisations also played down open source. Each supplier was offering custom solutions leveraging their own internal IP to satisfy the needs of particular products and projects.

But by this time the strategist was smelling a rat. Discussions with actual software engineers at his company had established that they were, in fact, using open source quite heavily in many cases. This was unavoidable given the deadline pressures they faced, plus the time, cost and friction associated with procuring commercial equivalents. Management was either unaware of this, or turning a blind eye.

Having established that company leadership was not properly aware of the spread of open source throughout the organisation and its products, the strategist worked to quantify the problem by running a research project. His team procured one of the company's products in the market, stripped it down in the lab, and used engineering forensics to establish what software was actually running on the various microprocessors.

More than half of the total code on the devices was found to be open source.

Given that the company had no corporate knowledge or understanding of this, it had no established practices for compliance with applicable licenses, nor did it have any capability to establish provenance of the software it was using.

Just saying 'stop using open source' was clearly not an option.

In many cases its contractual commitments with suppliers neglected to address the issue of open source entirely, since most agreements assumed by default that the systems were proprietary. In other cases the specific terms (eg 'must agree use of open source in advance') had been disregarded, often through ignorance on both sides. Worse, the company had no cost-effective or timely way to fix the situation. This would ultimately require root and branch overhaul and training for both its internal engineering and supplier management approach.

For long-lifetime products whose components were no longer supported directly by suppliers, no solution could be found. This time bomb remains unexploded, as far as we can tell.

Case Study 2

Oops, there goes the community.

A multi-year project was launched, involving custom electronics design with an integrated software application stack. The software was to be developed by hundreds of software engineers spread internationally across several organisations (chipset vendor, board design, device manufacturer, system integrator, end customer). After discussion between the various participants, the project chose a popular sponsored open source software distribution as its architecture platform.

Approximately 18 months into the development, the open source distribution was cancelled, after being abandoned by its sponsors. From a high level this is similar to what happens when a commercial vendor pulls the plug on a product - it's a pain, but we need to transition to an alternative.

However in this case there were some factors which differ from normal business:

– Because the distribution was 'free' the project had no commercial commitment from anyone to provide support while transitioning to an alternative

– The project was relying on 'community' resources to maintain the source code of the distribution itself

– Although the cost of maintaining public documentation, source code repositories and build infrastructure was very small versus marketing and other activities, the sponsors took down the community websites without notice

– As a result the project found itself suddenly without access to documentation and source code, and its ability to maintain its own build processes was irreparably damaged.

Ultimately there are only a few approaches to protect against this kind of risk

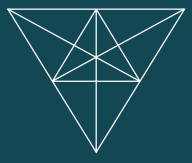
– Restrict use of open source - but this is increasingly hard to do (see CASE STUDY ONE)

– Engage with commercial providers - but in the non-Linux case mentioned above, for example, the leading commercial provider was destroyed when sponsors pulled the plug

– Participate in the community so you can understand and ideally influence the agenda, and get advance warning of trouble ahead. In the Linux case above, Codethink was aware that sponsored engineering was being diverted away from the distribution more than six months before the project was publicly killed.

Note that this is a recurring situation which has hit many organisations hard. The wording above actually describes two completely different projects, with different technologies, in different markets.

One involved a Linux distribution, the other did not.



Case Study 3

What do you mean open source? We wrote it, it's ours, pay up!

A software vendor wanted to expand its offering, by including functionality similar to capabilities developed by a startup. To reduce its time and cost to market, the startup had heavily leveraged existing open source, and in fact was publishing the core of its software solution under applicable open source licenses, on GitHub.

The software vendor engaged in commercial discussions with the startup, but a mutually acceptable deal could not be reached. As a result, the vendor decided to implement its own solution, and after careful consideration decided to adopt the startup's published open source software as the basis of the work.

When the startup later became aware of this, they removed their code from GitHub, and later attempted to seek legal redress with the vendor.

However, the vendor was working with Codethink and was unsurprised by the startup's attempt to derail their project. The whole development had been established with clear understanding of the rights and restrictions inherent in open source.

The project reliably maintained a complete mirror copy of everything which had been published by the startup. This included all the history for modifications made available in public by the startup right up to the point at which they attempted to move the goalposts.

As a result the vendor could establish clear provenance and traceability for the work done. They were able to demonstrate the whole process by which they sourced, adapted and re-used the software in accordance with the applicable licenses.

Case Study 4

To upstream, or not to upstream

Our customer had selected an advanced new SoC from a well-known vendor, but for technical reasons needed custom work on the Board Support Package, including kernel drivers, to fit the specifics of the project.

The SoC vendor was committed to supporting, but its preferred route was to make the modifications against its standard BSP, which at the time of the decision was based on LTSI (Long Term Support Initiative) Linux 3.4. LTSI was established with the help of the Linux Foundation to provide a standard support approach for consumer electronics. Each LTSI release comes to end-of-life in around two years.

Codethink argued that doing the work on LTSI was the wrong approach, given our customer's expected time horizon for the project of five to fifteen years. Our proposal, accepted by the customer, involved working to get the BSP drivers upstream into the mainline kernel.

Approximately one year later, the SoC vendor noticed that our customer was delivering exciting solutions on their silicon by exploiting advanced virtualisation features which were not possible with the vendor's standard BSP. They were very pleased to see their chips in such advanced products, but wondered how this was being achieved, without the software they were providing.

The explanation is simple - by alignment with mainline Linux and Codethink's work to upstream support for the SoC, the customer has been able to exploit features from recent kernels. At the time of writing their system can run Linux 3.19, while the SoC vendor's official BSP is still at 3.10.

Case Study

5

We're not using open source tools - you need to use what our IT gives you

Codethink was engaged to help an international electronic systems provider develop its next generation product, based on Linux with custom hardware.

At the beginning of the project, Codethink discussed software engineering process, tools and approach. The customer had standardised on a range of practices and tools, primarily using Windows-based technologies.

Given the nature of the work, our team would be designing a custom Linux stack for the customer's new hardware. This would require building (and rebuilding) and integrating many tens of open source software components, as well as developing new IP for the customer's product.

We convinced the customer to let us get underway using the normal tools and practices for Linux-based development - Linux itself, Git, IRC, mailing lists.

In spite of this, members of the customer team insisted that our engineers would need to adopt their mandatory corporate tools once the project was properly underway. This would mean Windows by default, with Linux in a Virtual Machine, plus tools such as Skype and Outlook.

At one point we were told that the project would be forced to drop Git, and adopt a corporate version control system specified by the customer's central IT team. We had to demonstrate to them that this would make the work impossible.

The project was delivered successfully, but our team faced some resistance from customer engineers who continued to use Windows tooling by default. Ultimately our customer had to recruit different staff (with Linux tools experience) since the Windows-based team seemed unable to acquire deep Linux knowledge for themselves.

Open Source as Supply Chain:

The Need for Provenance, Traceability, Upstream Alignment

Whitepaper

www.codethink.co.uk
sales@codethink.co.uk
+44 161 660 9930