The Codethink Way

Whitepaper

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

Executive Summary

Productivity and reliability for embedded systems software development projects is too low. As a result, technology industries are failing to deliver new software for new and existing devices fast enough:

1

Projects are delivered late, or with features missing

2

Fixing security vulnerabilities takes too long, or proves to be impossible

This gap creates an economic imperative for major organisations to exploit Free and Open Source Software (FOSS) solutions. However engineering understanding and competence level is so low that projects still run late or fail, even with free access to billions of dollars worth of proven software.

Codethink is uniquely placed to understand the systemic problems involved in the use of FOSS for custom embedded systems, which requires a deep understanding across all of the following areas:

– Custom electronics

– Custom system software

– Embedded software

– Proprietary and open source operating systems

– Proprietary software development processes

– Open source software development processes

The lead engineers at Codethink have contributed extensively to widely-used FOSS community projects including Linux, as well as designing and developing custom proprietary IP on many hundreds of complex projects over several decades. As a result our core expertise spans a huge range of computer architectures, operating systems, programming languages and tools.

Codethink's solution combines workflow and tools for engineering teams to specify, design, develop and maintain software for large populations of complex system devices

– From data center servers to wearables

– From medical devices to aircraft

– From industrial equipment to cars.

We see an opportunity to establish thought-leader positioning and fill space vacated by Wind River, and others.

To that end, this document outlines our thinking about the problem, the environment we are in, our company and our approach to solving the problem so far, and the future.

[1] **Free and open-source software (FOSS)** is **computer software** that can be classified as both **free software** and **open source software**.That is, anyone is **freely licensed** to use, copy, study, and change the software in any way, and the source code is openly shared so that people are encouraged to voluntarily improve the design of the software. This is in contrast to **proprietary software**, where the software is under restrictive **copyright** and the source code is usually hidden from the users.

The Problem

As the performance and capabilities of computer-based electronic devices increase, so does the scale and complexity of the software required to run them.

Technology industries are failing to keep pace with this inexorable growth in demand, because their current understanding, tools and processes for system-level software production are inadequate.

The key implications of this are that:

- **Organisations try to save money by using low-cost resources - cheap people**
- **Organisations try to cut corners by reusing existing or 'free' software**
- **Projects overrun on cost, time or both**
- **Worst case projects fail completely**
- **Delivered software is incomplete, inadequately tested, unreliable, insecure, unsafe**
- **Worst case failures cause financial losses, injuries, fatalities, environmental damage**

Complexity, Connectivity and Chaos.

Most industries already depend on software-intensive systems:

- **Banks and retailers are online;**
- **Factories, industrial plants and power stations are automated; and**
- **Logistics and transportation networks are tracked 24/7.**

Planes, trains, automobiles, satellites and missiles are designed, manufactured, tested and ultimately piloted using complex software systems.

And our dependence on software in electronic devices will clearly increase - as shown by the hype around 'Cloud Computing', 'Internet of Things', 'Machine to Machine', 'eHealth', 'Home Automation', 'Intelligent Buildings', 'Connected Car' to name a few.

This will happen across all sectors where electronic technologies are widely used including science, education, telecoms, petrochems, transportation, automotive, defence, aerospace, media, entertainment, medical, finance, government.

As Carl Sagan said:

"We have also arranged things so that almost no one understands science and technology. This is a prescription for disaster. We might get away with it for a while, but sooner or later this combustible mixture of ignorance and power is going to blow up in our faces."

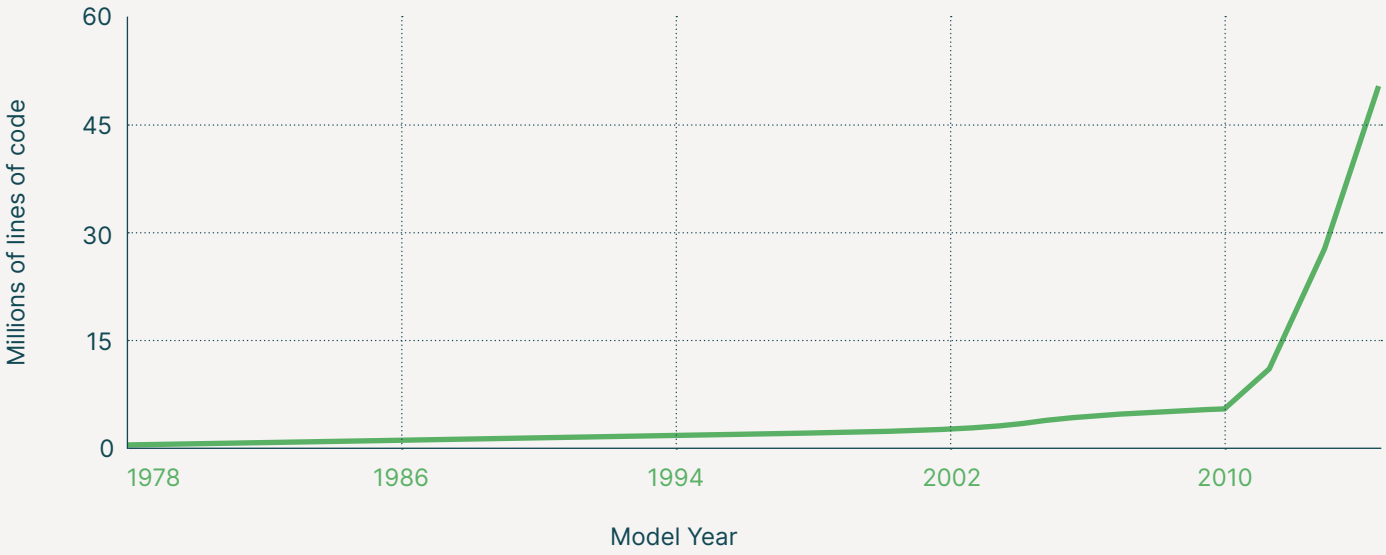
So too with these complex software systems. We are building ever more complex systems that our engineering community is hard-pressed to understand, let alone the executives and regulators charged with developing and executing business strategies.

Complexity

Over recent years there has been a dramatic increase in the complexity and capabilities required from electronic devices, leading to the need for much more comprehensive software solutions - hence much more code.

A clear example of this is the automotive sector; up to 2010 most of the software was small-scale controller code for Engine Control Units (ECUs) - each separate controller had perhaps a few tens of thousand Lines of Code (LOC). But with the introduction of In-Vehicle Infotainment and provision of applications and connectivity services, typical vehicles in development today already carry more than one hundred million LOC when they hit the market.

Growth of codebase in vehicle



Source: internet + anecdotes

Connectivity

More and more electronic systems are being connected to the internet. An unavoidable consequence is that these devices will be increasingly probed, attacked and ultimately hacked.

Security holes are discovered in major software components every day. They are regularly exploited by individuals, criminal organisations, companies and governments.

Clearly the longer a vulnerability remains unfixed, the more opportunity for malicious exploitation. Failure to patch systems promptly is negligence, with potentially disastrous consequences.

As security specialist Bruce Schneier has said, **"You can't defend. You can't prevent. The only thing you can do is detect and respond."**

Chaos ensues...

- Hacker backdoors Linksys, Netgear, Cisco and other routers
- London firm at centre of hack redirecting 300000 routers
- Volkswagen sues UK university after it hacked sports cars
- CanSecWest Presenter Self-Censors Risky Critical Infrastructure Talk
- Replicant Hackers Find and Close Samsung Galaxy Back-door
- Critical crypto bug leaves Linux, hundreds of apps open to eavesdropping
- Why Toyota's Oklahoma Case Is Different | EE Times
- Embarrassing stories shed light on US officials' technological ignorance ...
- Senator's Letter To Automakers Demands Info On Cyber Security...
- EU has secret plan for police to 'remote stop' cars
- Hacker pwns police cruiser and lives to tell tale
- Snowden leak: GCHQ DDoSed Anonymous & LulzSec's chatrooms
- New Flash vuln exploited (again). Adobe posts emergency fix (again)
- Pondering the X client vulnerabilities
- Hackers gain 'full control' of critical SCADA systems
- Samsung retries botched update to Galaxy S3 smartphone
- Researcher hacks aircraft controls with Android smartphone
- Update your iThings NOW: Apple splats scary SSL snooping bug in iOS
- Knight Shows How to Lose $440 Million in 30 Minutes
- Biting into Apple
- Malware designed to take over cameras and record audio enters Google Play
- HTTPS More Vulnerable To Traffic Analysis Attacks Than Suspected

The Rise of FOSS and Linux in embedded electronic devices

With high-volume devices, a key commercial driver is reduction of the overall Bill of Materials (BoM) cost. Since traditionally software has been licensed on a per-unit royalty basis, the software license cost is added to the BoM. This was the model for Symbian, Nucleus, VxWorks and is still the commercial approach for Microsoft Windows, Blackberry QNX and Green Hills Integrity.

However, Linux and other free and open source (FOSS) solutions can be used with zero license cost addition to the BoM, so long as users abide by the applicable licenses and are willing to bear the cost of understanding the software and integrating it.

This in part explains the dramatic uptake of Linux and BSD for consumer devices and mobile handsets. Google's Android is a Linux-based operating system. Apple's iOS and MacOS have been built on the work of BSD and many other FOSS projects.

Why would anyone agree to pay $10 per unit royalty on a device destined to ship a million or more units if there's a zero royalty option that can be made to work?

The economic advantage of this zero BoM cost argument seems unassailable for high volume devices, from cellphones, to cars. This also applies for hyperscale datacenter systems, smart metering systems, defence systems, medical equipment, wearable devices and so on.

The only commercially viable counter to Linux is that the risks and/or costs of adoption, development, integration and test may make the overall business case for a FOSS approach worse. In other words if the Non-Recurring Engineering (NRE) to implement the specific FOSS solution are too high, or the project will take too long.

In fact, the costs, risks and time required for implementation of custom FOSS systems **are extremely high, as with all embedded projects.**

Codethink's frustration at this situation is what caused us to start down our own path in the first place. We have been seeing and dealing with the same mistakes in project after project, customer after customer, market after market with a multitude of embedded OSes.

In our view, the lack of understanding and measurability in the overall software engineering process makes it impossible for organisations to consider objectively whether the TCO for any particular approach will be better or worse than FOSS. Arguably this is what makes it possible for companies such as Microsoft, QNX, Green Hills among others, to continue with the royalty-based model, particularly where systems require hard real-time, deeply secure or safety-critical performance.

However, it is clear that most vendors of proprietary approaches have already been compelled to move towards Linux. Wind River, Mentor Graphics and ENEA have all seen the writing on the wall, and now offer Linux development platforms in addition to their proprietary solutions.

Unfortunately this does not appear to be leading to increased efficiency and reduced costs. For example, one of our automotive customers recently noted that after Mentor acquired MontaVista's automotive business, effectively reducing the number of recognised automotive software 'platform' vendors from three down to two, both Mentor and Wind River "**virtually doubled their prices**".

[2] Total Cost of Ownership

We can't even measure this

There is almost no reliable scientific research about large-scale software projects, so it is quite difficult to support this overall argument with facts.

Many widely cited 'truths' for software engineering turn out be be based on inadequate science, hidden agendas and/or misinterpretations.

For example:

- **Barry Boehm's widely quoted Software Engineering Economics was based on a very limited sample of projects, and over the years his findings have been overvalued;**
- **The widely adopted 'Waterfall' model was actually based on a fundamental misunderstanding of the original author's intentions;**
- **The champions of so-called Agile & Lean practices for software claimed to draw heavily on Toyota's innovations for automotive manufacture, yet to the limited extent that this is true, it must be clear that system software development is very unlike traditional manufacturing;**
- **Moreover, the recent Unintended Acceleration lawsuit demonstrates that Toyota has itself struggled to deliver software to meet the quality necessary for safety-critical systems;**
- **'Scrum' has been widely adopted over the last decade in industries and organisations where its use is clearly inappropriate (The rules of 'Scrum' do not scale beyond small localized project teams of 5-9 people, nor can 'Scrum' work for projects with fixed-price commercial constraints.);**
- **Consultants have been happy to facilitate huge-scale 'Scrum' and 'Agile' initiatives in Nokia, Vodafone, UK Government and many more organisations, often with disastrous results;**
- **The Linux Foundation published a white papers entitled "The Economic Value of the Long Term Support Initiative (LTSI)" and "Value of Collaborative Development" With no hard data to go on, the authors made a series of questionable assumptions, leading to conclusions which are not credible.**

[3] Boehm, Barry W. "Software engineering economics." (1981).

[4] "The Leprechauns of Software Engineering - Leanpub." 2012. 22 Feb. 2014 <https://leanpub.com/leprechauns>

[5] "Waterfall model - Wikipedia, the free encyclopedia." 2004. 22 Feb. 2014 <http://en.wikipedia.org/wiki/Waterfall_model>

[6] "Acceleration Case: Jury Finds Toyota Liable | EE Times." 2013. 22 Feb. 2014 <http://www.eetimes.com/document.asp?doc_id=1319897>

[7] "Vodafone turns its back on '360 • The Register." 2011. 22 Feb. 2014 <http://www.theregister.co.uk/2011/10/18/vodafone_kills_360/>

[8] "Francis Maude plays down universal credit IT row ... - The Guardian." 2014. 22 Feb. 2014 <http://www.theguardian.com/politics/2014/jan/08/francis-maude-universal-credit-it-row-dwp>

[9] LTSI Documents

[10] LF Value of Collaborative Development

The ugly truth about ‘software engineering’

We argue that system-level software production practices are lagging a long way behind other engineering disciplines.

A tremendous amount of uncertainty and ignorance still remains:

– ‘software engineering’ is not really an engineering discipline;

– most software developers are self-taught, learning from Google and colleagues;

– there are no accurate metrics for analysing and comparing the scale, complexity and quality of software projects;

– there is no reliable way to measure or to compare software developer productivity for different people, languages, tools, processes or projects;

– there is no accepted way to distinguish software maintenance from software development;

– there is no way to estimate accurately the effort required to develop software;

– there is no ‘methodology’ that works predictably across the range of projects which the technology industries are now required to deliver;

– most ‘methodologies’ significantly overstate their benefits;

– most ‘methodologies’ are described incompletely and in vague terms, to the extent that practitioners can adjust their positions on a case-by-case basis if necessary;

– those ‘methodologies’ which are strictly defined tend to be so prescriptive and long-winded that they are unworkable in practice, so teams end up ignoring them;

– most organisations are unable to manage complex software projects effectively;

– most managers allocating people, resources and tools for software projects do not have enough reliable information or understanding on which to base their decisions;

– most executives have never been involved in system-level software projects or embedded software projects;

– many executives with responsibility for electronic systems products and projects delivery do not even know what the activity of programming involves;

– most developers would agree that code review is a good thing, yet do not develop the habit and practice of reviewing other people’s code, and most code is not actually reviewed;

– most software developers are not very good at the job and create software which is unreliable, insecure or both;

– most developers overstate their own abilities and do not properly understand the tools and techniques that they claim to use; and

– given the lack of understanding and measurability, these overstatements and failures in understanding routinely go unnoticed.

[11] Programmers should not call themselves engineers

[12] Pants on fire: 9 lies that programmers tell themselves

The Codethink Way

Our broad scope is to dissect, understand and optimize the whole process of specifying, creating, modifying, testing, deploying, maintaining and updating large-scale custom FOSS-based software systems. We have already proved that we can standardize the workflow and tooling for system-level software development, using an integrated and self-sufficient set of fully open source components.

In short, Codethink aims to establish 'software engineering' as a genuine engineering discipline. We want processes and tools which are:

1. Efficient
2. Reproducible
3. Predictable
4. Traceable
5. Reliable
6. Secure
7. Safe

In order to achieve this, we are creating a cohesive approach to handle all of the following

- Requirements capture;
- System specification and design;
- Engineering work breakdown, estimating and work scheduling;
- Embedded software development environment and target operating systems;
- Source code and work products review;
- Team workflow;
- Integration and build system;
- Rigorous bootstrap for new devices and architectures;
- Linux software distribution;
- Virtualization;
- Whole system configuration management and change control;
- Continuous integration and continuous delivery; and
- Atomic updates for running production systems

Our work involves developing and integrating open source components to form an end-to-end solution.

Our established implementation includes a complete toolchain and Linux operating system, and we expect that most of our target users will be Linux-based.

Much of our work can be also applied for non-Linux systems:

- Our preferred operating systems, tools and build envirtonments can be used to create non-Linux targets, for example BSD, bare-metal RTOS and firmware.
- Our preferred workflow components can be deployed on Linux, Windows, Mac, whatever.

Proof Points

Codethink's approach and software tools already allows us to work significantly faster and with much less engineering effort than other approaches, particularly for OS bringup on new architectures and hardware. Key proof points over the last 18 months include:

In short, Codethink aims to establish 'software engineering' as a genuine engineering discipline. We want processes and tools which are:

- First big-endian Linux OS for server class ARM systems, delivered in three months;
- Big-endian Linux OS for ARM VE TC2, delivered in two weeks flat;
- Creation and bringup of full GENIVI Baseline on i.MX6 and x86;
- Bringup of custom infotainment stack on NVIDIA Tegra; and
- Full OS implementation on IBM POWER, delivered in six weeks.

Updating Components

A common problem for real-life projects is the cost and time involved in updating the Linux kernel and other system components. Typically this takes significant integration effort by many engineers over a period of months. In the worst case, updating can be 'a nightmare' or ultimately prove to be impossible.

Our normal demonstration is to start from an existing complete embedded system, and update it to a new version of Linux.

The whole process, including building the new system with the latest never-before-seen kernel, can normally be done in under fifteen minutes:

https://vimeo.com/88970773

The Codethink Way

Codethink's overall approach to the 'software engineering' problem described here involves broadly:

- Philosophy and guidance describing how to do software engineering 'the Codethink way';
- A breakdown of the system software engineering lifecycle and analysis to determine the information, techniques and tools required at each stage;
- Software tooling and infrastructure which support engineers working the Codethink way; and
- Data definitions to create complete software systems reproducibly using Codethink recommended tooling.
- These four areas are considered in the following sections.

[13] One provider had quoted '$10M and five years' for this project

[14] This was an urgent requirement after other organisations had been promising a solution for several months

[15] GENIVI is a non-profit consortium whose goal is to establish a globally competitive, Linux-based operating system, middleware and platform for the automotive in-vehicle infotainment industry

Codethink Philosophy

- We are seeking processes and tools that are efficient, reliable, repeatable and secure.
- We aim for a culture of collaboration, excellence, honesty and transparency.
- We need engineers to share knowledge and co-operate in an effective way so that
  - all work is documented;
  - all work can be independently reviewed; and
  - all reviews can be considered and discussed, and improvements adopted.
- We know that mistakes are unavoidable and therefore, our people and processes need to be tolerant and cope robustly when things go wrong, so we want
  - all source code to be available so we can diagnose problems as effectively as possible; and
  - all workflow and development history to be traceable, so that we can track back to why a specific function or line of code was implemented and by whom.
- We know that all software engineering ultimately requires frequent ongoing repetitions of 'change, test and release' loops, so we want to
  - understand and improve how loops are done;
  - optimise the time and effort required for loops; and
  - provide visibility and feedback on loops and encourage further optimisations.

We know that engineer uncertainty and delay in communications are key causes of wasted time and effort, so we need

- fast and effective channels for information requests, discussions and reviews between engineers, teams and organisations; and
- standardisation of processes so that interested participants can become adept and know what to do.

We cannot satisfy all requirements or work with all technologies, so we must choose our weapons and minimise our workload by

- using and integrating the best available open source solutions to meet our needs;
- aligning with the upstream developers to minimise our deltas;
- improving existing tools where the cost of change is justifiable (and in the case of OSS tools, contributing these changes back to the community); and
- introducing new solutions where current offerings do not satisfy our overall objectives as described herein.

System Software Engineering Lifecycle

In order to become efficient we need to break down, understand, standardise and improve the whole lifecycle of system software engineering. Across this broad area we aim to provide workflow, tooling and infrastructure to minimise the pain, risk, effort and time required to build complex system software.

Activity	Example established players	Codethink	*
Project management, work planning, work co-ordination	MS Project, Pivotal, Jira, Trello, Google	Hobokan Internet Relay Chat (IRC) Mailing Lists (ML)	(2)
Issue tracking	Fogbugz, Jira, Trac, RT, Mantis		(3)
Configuration management	Rational, Github, Gitlab	Trove	(1)
Code and work review	Gitlab, Gerrit, Crucible	Trove, Mustard, ML	(2)
Requirements capture	Rational (Doors)	Mustard	(2)
System architecture design	Enterprise Architect, Visio, Rational	Mustard	(2)
Initial board bringup	???	Repeatable process	(2)
Device driver development	???	???	(2)
Operating system bootstrap	???	Repeatable bootstrap	(1)
Middleware development	???	???	(1)
Applications development	IDE eg Eclipse, Visual Studio		(3)
Overall system integration	ad-hoc	Baserock	(1)
Continuous integration	Jenkins, Travis	Baserock Mason	(2)
Internal system deliveries	tarballs	Baserock deploy	(2)
Production systems rollout	Red Bend		(3)
WV	Red Bend		(3)

(*) Efficiency factor as outlined below:

(1) We have a solution which is already effective, traceable and demonstrably better than other solutions

(2) We have incomplete/imperfect tools - our efficiency is the same or better than other approaches

(3) It is currently easier/better/faster/cheaper to do this without our existing R&D solution

System Definitions

We maintain a public set of declarative data definitions of a set of example systems at

http://git.baserock.org/cgi-bin/cgit.cgi/baserock/baserock/definitions.git/

These include example implementations for x86 (64-bit and 32-bit), ARMv7 and IBM POWER architectures, and include pre-integrated software sets for a wide range of components including

- Linux system-level components including kernel, systemd, openssh
- Vase-level developer tools including GCC, make, autotools
- Advanced file systems including BTRFS and Ceph
- Graphics and multimedia
- Virtualization
- Bluetooth and wifi
- Network connectivity and telephony
- Qt
- GTK+
- Enlightenment and XFCE desktops
- OpenStack

Clearly these components and many more are already integrated in other Linux distributions and build systems including Debian, Ubuntu, RHEL, SUSE, Gentoo, Yocto, Buildroot etc.

The key differentiator in our approach is that we have streamlined and simplified the process for integration to reduce the time and effort dramatically. As a result the total amount of information required to describe our systems relative to upstream is much reduced vs other approaches.

Comparing our definition files against Yocto/ OpenEmbedded bitbake recipe files, for example:

- morph files are significantly smaller and simpler than bitbake recipes
- morph files are designed to be machine parseable - as a result we can manipulate whole systems via software to update our design. The complexity and irregularity of bitbake recipes means this is not really possible in Yocto
- more than half of the components we integrate require no description by us - they are used directly as released by their upstream developers.

[16] Our kanban tool **https://github.com/CodethinkLabs/Hobokan**

[17] Our approach for managing source code for the full system software stack - e.g. **http://git.baserock.org**

[18] Our requirements capture and system architecture design tool: **https://github.com/CodethinkLabs/Mustard**

[19] Integrated Development Environment - GUI-based solution combining editor, debugger, make, version control

Competitive Landscape

Many organisations offer methodologies and tools to reduce their customers' software engineering costs and time to market. Similarly, many technology organisations develop and improve tools and techniques for their own use. Thus, there is a wide range of organisations with interest and/or offerings around software productivity:

- Consulting services eg IBM, Accenture, Symphony Teleca, Capgemini
- Software 'platforms' and tools eg Mentor Graphics, Wind River, ENEA
- Operating systems eg Red Hat, Canonical, SUSE, Microsoft
- Not-for-profit industry organisations including Linaro, GENIVI, and the Linux Foundation
- Design of whole systems eg LG, Visteon, Continental, Robert Bosch, BAe Systems
- Software tools to sell hardware or intellectual property eg HP, Intel, ARM, NVIDIA, Atmel
- Device companies eg Apple, Ford, Micron, Nokia, Samsung, Volkswagen
- Service providers eg Vodafone, BSkyB, BT, Bloomberg

Codethink's people have worked with most of these organisations at various times, and our broad conclusion is that everyone struggles with the software quality and productivity issues we have identified. The problems have remained unsolved for decades, and everyone is too busy and focused on the day job to consider how to clean up the overall mess.

In many cases, the business models above may be structurally opposed to improving things, for example:

- Consulting services charge for engineer time, so higher productivity is not in their interest;
- Tool vendors' revenues often come from user/seat volumes and fees - the more engineers their customers need, the better;
- OS vendors need to justify their license/support/subscription fees - demystifying their processes would be bad for business;
- Hardware and IP vendors typically see system software as a necessary sweetener to win customers, so software is rushed to meet a demo or release date - quality is not priority; and
- Systems and device companies usually prioritize costs and revenues associated with hardware, and fail to recognise how expensive their software efforts ultimately are.

To the best of our knowledge, no organisation, apart from Codethink, is actively working towards an overall solution for the 'software engineering' problems highlighted here.

In discussions at conferences and with customers, prospects and partners, we find that in general, people are either ignorant of, in denial of, or resigned to acceptance of most of the issues we raise.

Organisation	Offers	
IBM	Doors, Team Concert, Red Bend	
Intel	Yocto, Tizen	
Wind River	Wind River Linux	Yocto-based
Mentor Graphics	Mentor Embedded Linux, Sourcery	Yocto-based
ENEA	Enea Linux	Yocto-based
Linaro	Upstreaming, LEBs, LAVA, Support	Member services
Linux Foundation	LTSI, Yocto	
Atlassian	Jira, Confluence, Bamboo, Stash	
Google	Android, ChromeOS, Gerrit, Repo,	
Canonical	Ubuntu, Launchpad, Bzr	
Red Hat	RHEL	
Gitlab	Gitlab	
Gitlab	Gitlab	
Gitorious	Gitorious	
Gitorious	ptxdist	
Elektrobit	Software Factory	

The Codethink Way

Whitepaper

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