Towards trustable software engineering...

Paul Sherwood 23 Oct 2018
@devcurmudgeon
www.devcurmudgeon.com

Intro: @devcurmudgeon

- CEO Codethink (.com)
- Stealing Spitfires (Spotify)
- Shut Up And Shoot Me (IMDB)
- Software Commandments (github)
- YBD: Yaml Build Deploy (gitlab)
- www.devcurmudgeon.com

- python/ruby/git/C and vi
- skeptical, opinionated and grumpy
- with trust issues
- insisting on honesty

Codethink

Intro: Codethink

Slipshod software practices...

… will turn our computers into spies, murderers & thieves

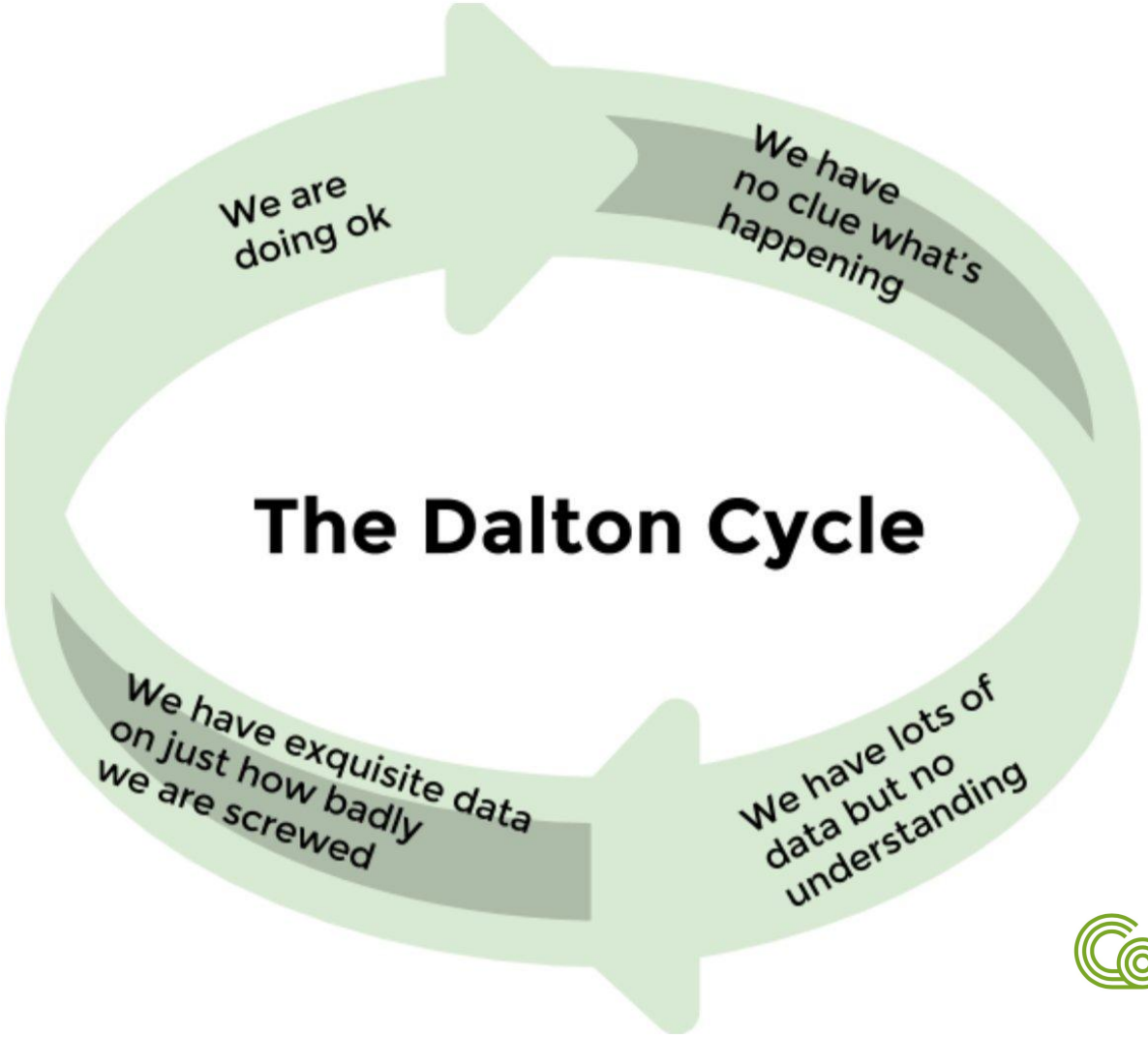
Slipshod software practices...

… will turn our computers into spies, murderers & thieves

- Most people using software don’t understand the risks
- Most people producing software don’t understand the job
- We have no consistent or reliable measures for software risks, quality, productivity, or costs

Maybe your company/project/software is better … but I doubt it

CodeThink


We are doing ok
We have no clue what's happening
The Dalton Cycle
We have lots of data but no understanding
We have exquisite data on just how badly we are screwed

CodeThink

The hole is so deep we ignore it

- Everybody has knowledge gaps, people still get jobs, they muddle through
- In mature critical industries there are norms, standards, regulations
  - Work is checked, both by the implementing organisation and independently
- Software has grown to be pervasive + critical without norms/regulations/ checks
- All software, including critical software, has bugs and vulnerabilities
- So we are all at risk of financial/emotional/physical harm from software issues
- When we suffer harm there are no normalised paths for redress/improvement

The nagging voice in my head...

- Even with regulations, we will still be at risk
- It’s already too late - setting standards now won’t make any difference
- Most people don’t and won’t care, until their account gets emptied, their car decides to crash, or the power grid goes down, or their hospital gets hacked
- Many software people/organisations continue to pretend they know what they are doing, resist ‘standards’ and blame deadlines/pressure
- The pressure to innovate outweighs the pressure to provide guarantees
- Executives have no incentive to fix, so long as the problem is ‘software’
- Easiest strategy is to charge ahead til sh** happens, then blame someone else

trustable software engineering

Home | Short Guide | Contributing | Relevant Projects | Reading List

Trustable

cii best practices | in progress 76%

The Trustable Software project aims to improve the quality of enginering for software which most people would consider *important*, for ourselves and for our wider communities.

This is "a very big elephant". Our scope includes any software which satisfies some or all of the following criteria

- may cause harm to people either directly or indirectly
- may cause financial harm or damage productivity
- may cause harm to the environment, either directly or as a side-effect
- if hacked, could be exploited to cause any of the above

We consider that software can be considered trustable if

- we know where it comes from
- we know how to build it
- we can reproduce it
- we know what it does
- it does what it is supposed to do
- we can update it and be confident it will not break or regress

and perhaps most importantly...

- we have some confidence that it won't harm our communities and our children

LINKS

white paper

wiki

mailing list

irc logs

activities kanban

#trustable on freenode

code

POSTS

CodeThink

We need to raise the bar for everyone...

- pre-existing legacy systems/software
- open source communities
- hardware manufacturers
- cloud/infrastructure/IT providers
- operating systems vendors
- software platform providers
- applications developers
- non-technical users, managers and executives

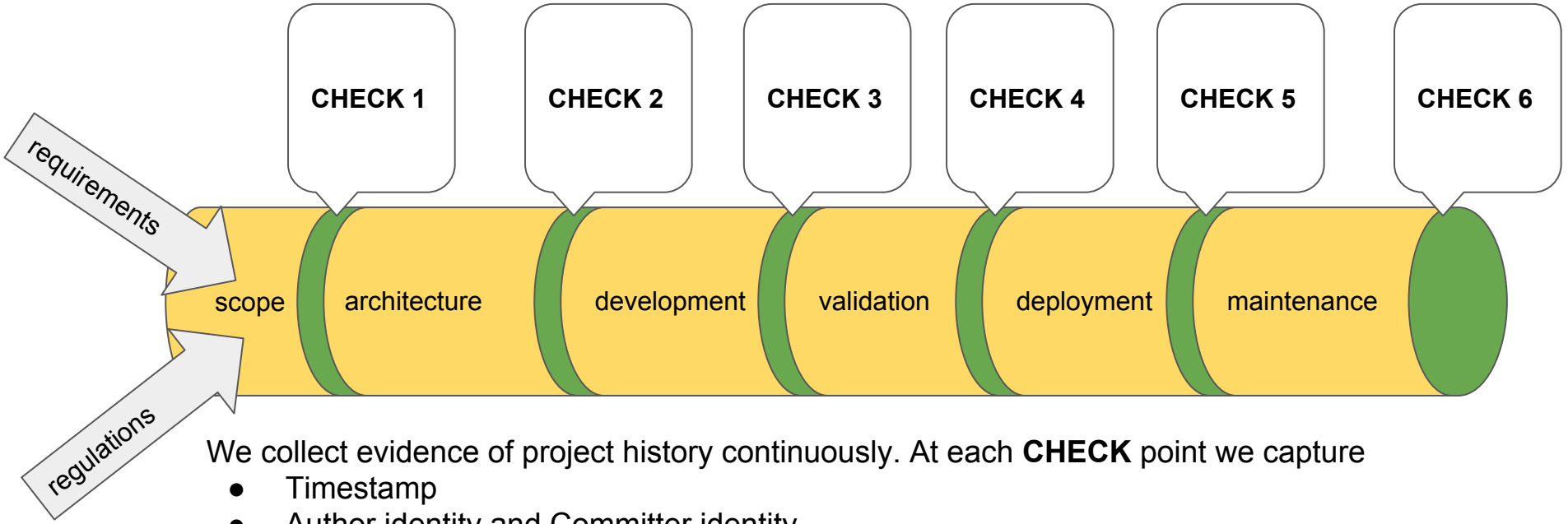
Towards trustable software ENGINEERING

Partial solution topics (from Trustable Mailing List discussion)

- training and education
- measurement and evidence of work processes
- pervasive version control with archived history, including identity and reviews
- proof of software provenance and reproducible builds
- fuzzing
- continuous integration for everything (requirements to deployment and mtce)
- automation of (continuous) compliance
- mathematical methods for formal verification of tools and software
- model-based engineering

Trustable software engineering process pipeline

- Can we establish an accounting audit metaphor for software process?
- In accounting, we check
  - P&L
  - Balance sheet
  - Transaction logs
  - Bank statements
- There are checks and balances
  - double-entry bookkeeping - if the balances don't match up, something is wrong
  - Bank statements represent an external log - if they don't match up, something is wrong
- In software
  - If source SHAs don't line up, something is wrong
  - If built artifact cache-keys don't line up, something is wrong
  - If test results do not meet expectations something is wrong
  - Reproducible builds would be another checkpoint

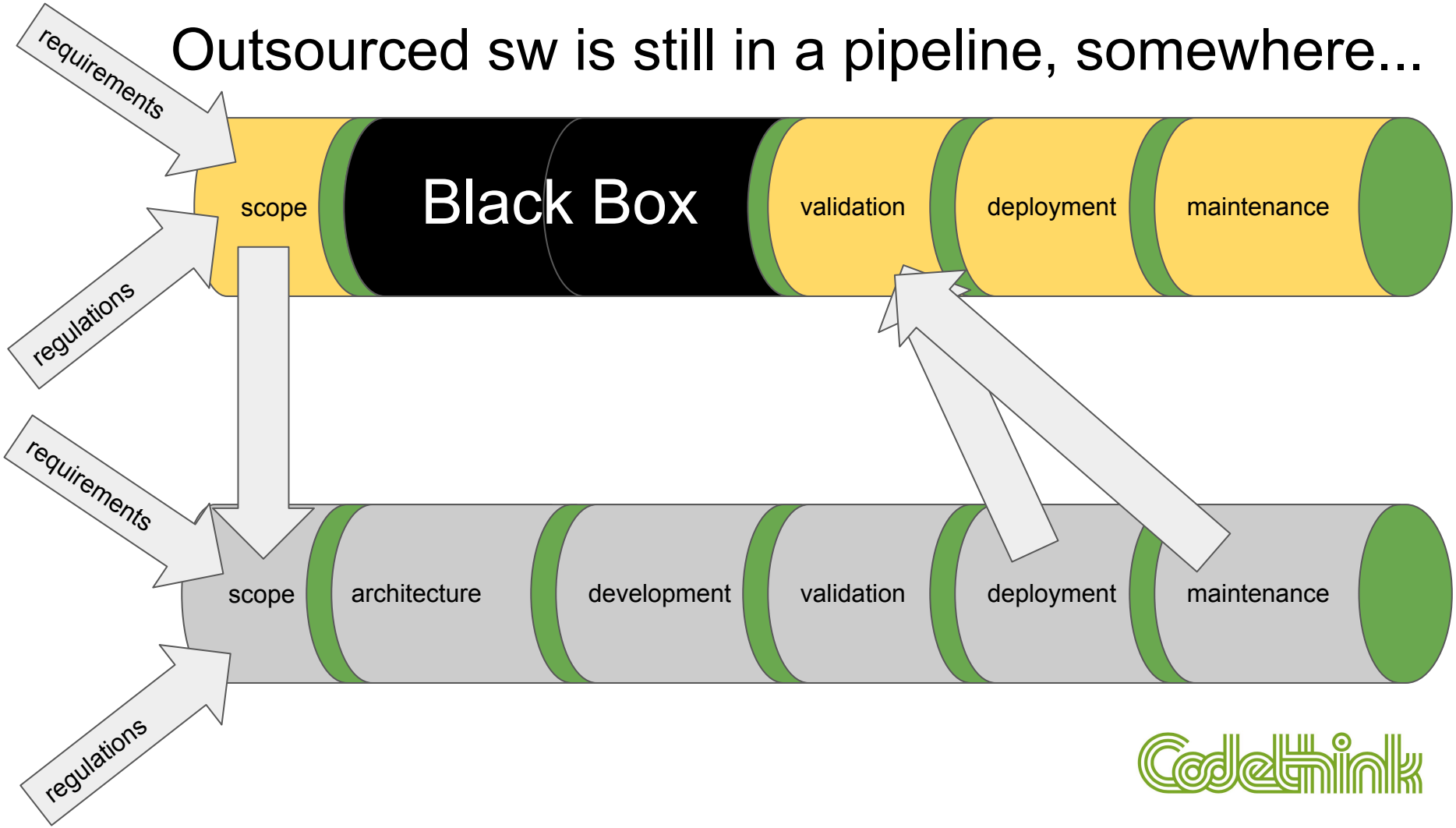


We collect evidence of project history continuously. At each **CHECK** point we capture

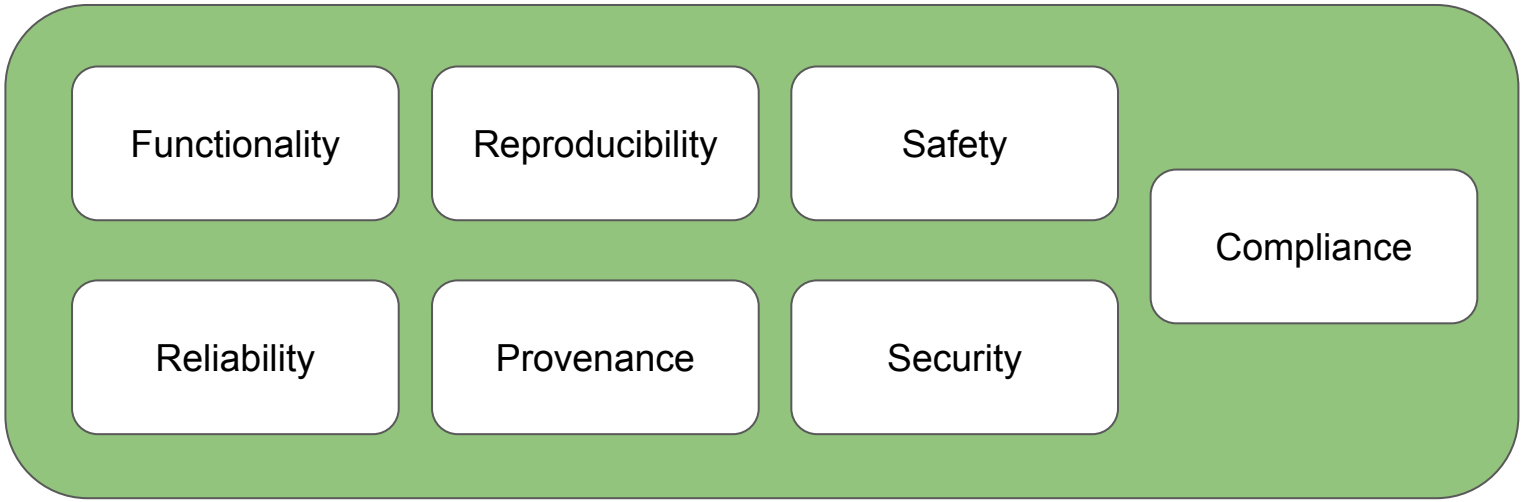
- Timestamp
- Author identity and Committer identity
- Independent review (with Reviewer identity)
- Appropriate signoff

CHECK 1: evidence of evolution of problem/scope (requirements, standards and verification criteria)
CHECK 2: evidence of evolution of solution/architecture
CHECK 3: evidence of selection/production of software (including tests)
CHECK 4: evidence of evolution of tests, test results and satisfaction of validation criteria
CHECK 5: evidence of traceability back through all previous phases
CHECK 6: evidence that maintenance changes/upgrades are being applied to the whole

Outsourced sw is still in a pipeline, somewhere...



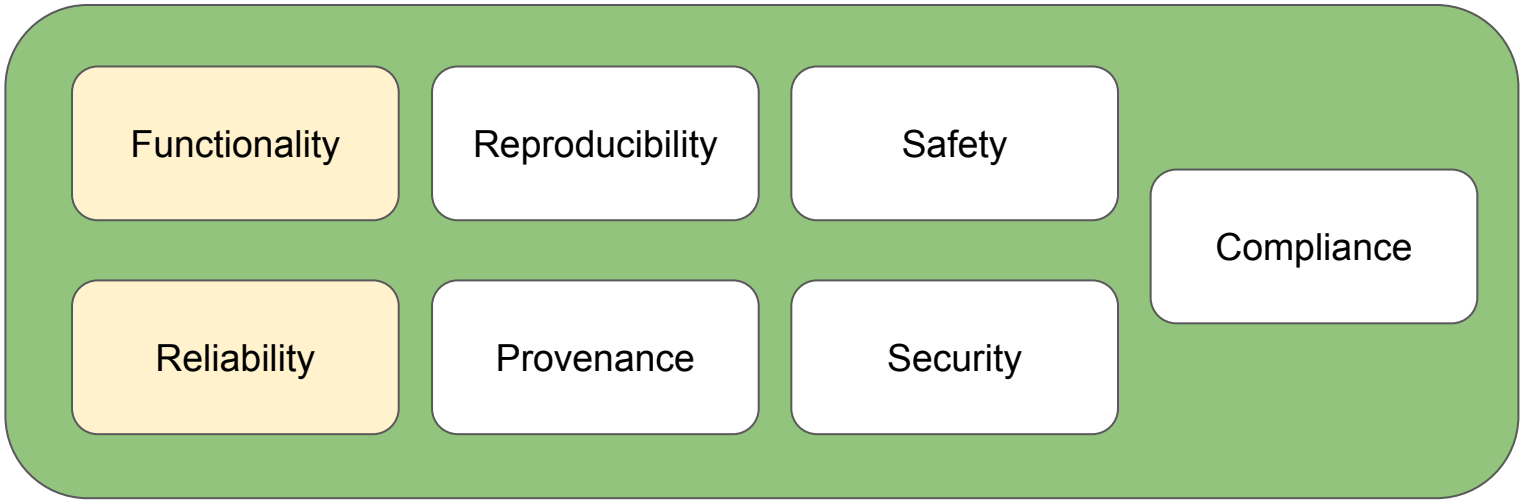
Working hypothesis: software trustability factors



we could base our trust on evidence for each/all of these

CodeThink

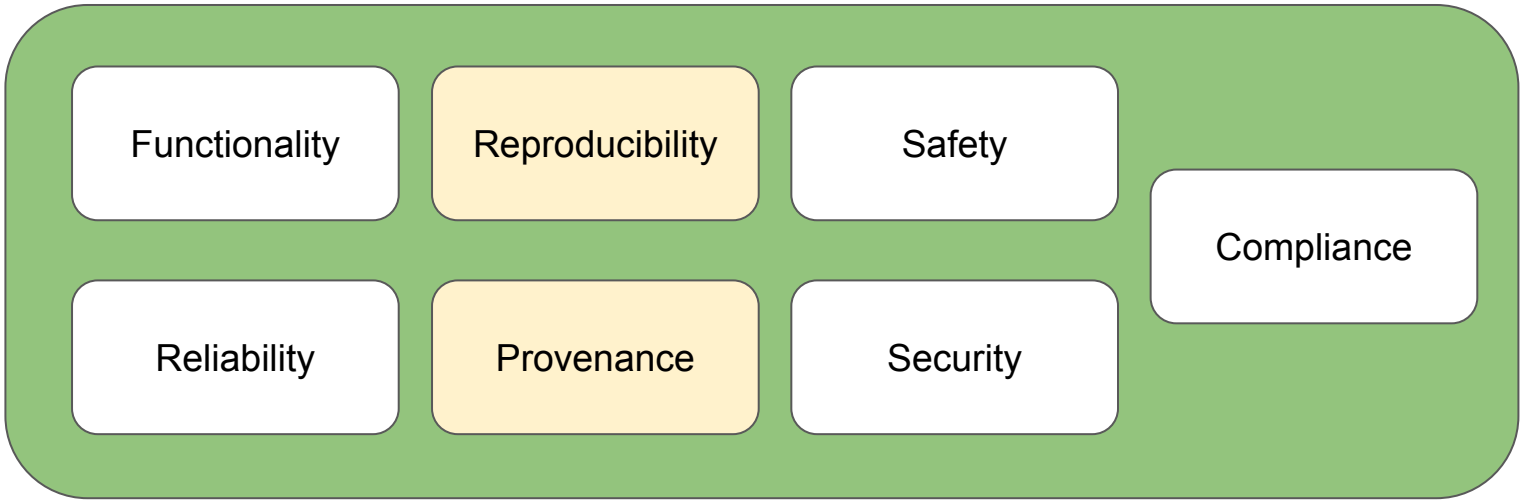
Working hypothesis: software trustability factors



(if we have requirements) we can run tests and collect evidence

CodeThink

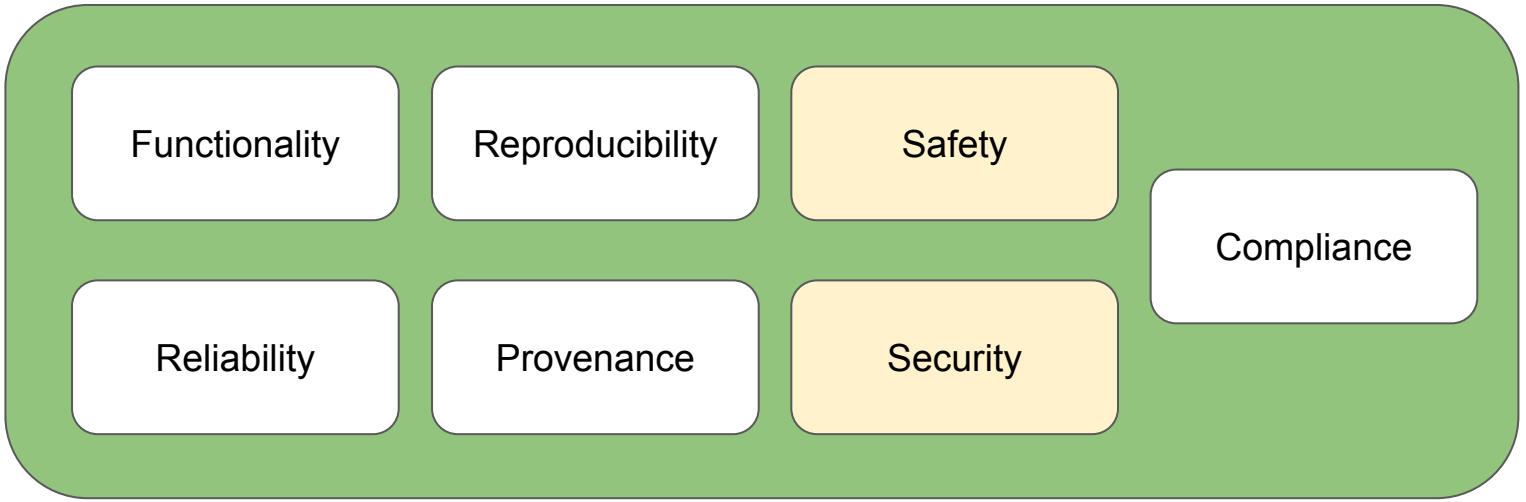
Working hypothesis: software trustability factors



we can gather evidence of input changes, output artifacts, reviews etc

CodeThink

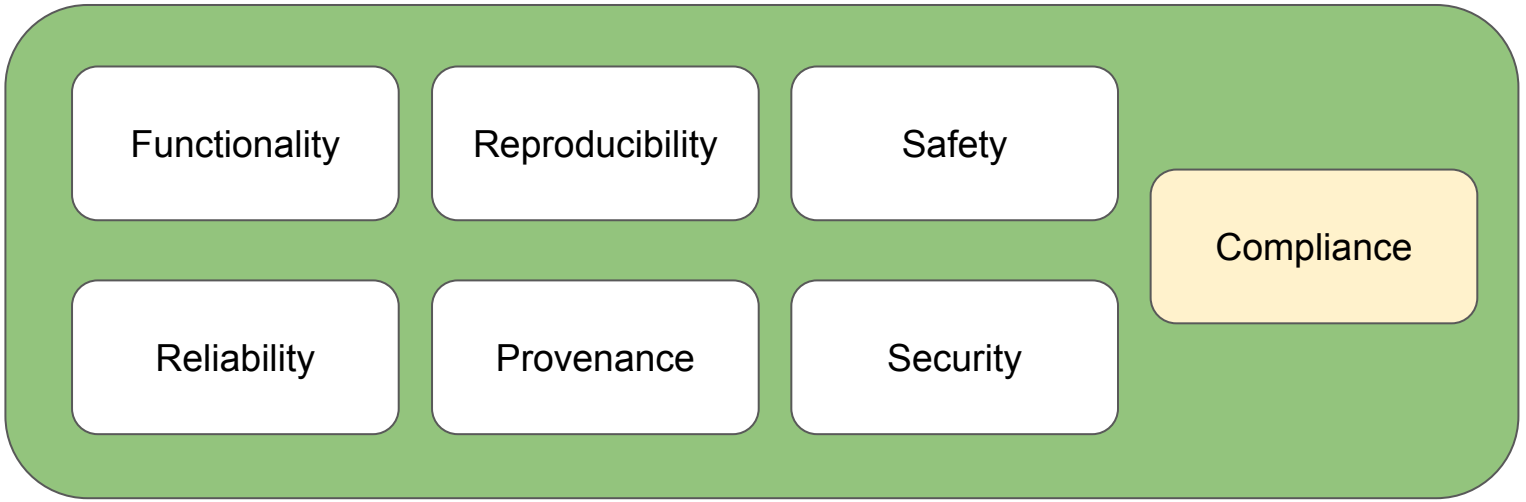
Working hypothesis: software trustability factors



safety and security are (emergent) system properties, not just software

CodeThink

Working hypothesis: software trustability factors



we can collect evidence of (non)compliance vs regulations/standards

CodeThink

Can we trust software for **Security**?

Security: we need to get much better at this...

```
Date: Mon, 22 Oct 2018 15:07:55 +0000
From: Andrew Sandoval <ASandoval@...root.com>
To: "oss-security@...ts.openwall.com" <oss-security@...ts.openwall.com>
Subject: GCC Compiler Induced Vulnerability - affects programs compiled with
 GCC 7 and 8 containing nested functions

Introduction to GCC Compiler Induced Vulnerability
==================================================

Hal Lonas
11 October 2018

INTRODUCTION
Webroot engineers recently discovered a vulnerability with Linux and Windows
executables produced by the Gnu C Compiler, commonly known as GCC.

Technical Description of the vulnerability
When nested C functions are compiled by GCC, code is generated which causes the
call stack of the currently executing thread to be made executable prior to the
call to a nested function and for the duration of the thread's lifetime.  This
is essentially the equivalent of disabling Data Execution Prevention (DEP).
A stack overflow, etc., that is able to place instructions on the page(s) of
memory made executable has the potential of gaining execution and running
malware, etc.  This places the process at substantial risk of being exploited.
```

Some interesting examples over the last year or so

- **NIST** (https://www.riskcontrolstrategies.com/2018/01/08/new-nist-guidelines-wrong/)
- Deloitte (https://www.theregister.co.uk/2017/09/25/deloitte_email_breach/)
- **Equifax** (https://www.theregister.co.uk/2018/05/08/equifax_breach_may_2018/)
- Google Mini touch (https://www.androidpolice.com/2017/10/10/google-nerfing-home-minis)
- Hyatt (https://www.securityweek.com/hyatt-hotels-hit-another-card-breach)
- **WPA2 KRACK** (https://en.wikipedia.org/wiki/KRACK)
- **Infineon RSA** (https://en.wikipedia.org/wiki/ROCA_vulnerability)
- **FreeRTOS/SafeRTOS** (https://www.theregister.co.uk/2018/10/22/freertos_iot_platform_security_flaws/)

CodeThink

Security: Identity is fundamental to trustability

- Most of our identifying data is already online
- Once the data has leaked, it can't be un-leaked
- Yet this will still be the content of 'security questions'
- Passwords are a ridiculous way to secure systems
- Pin codes are worse
- **Biometrics are even worse**
  - **not secret**
  - **easily stolen**
  - **not protected by law**
  - **hard to repudiate**

Multi-factor authentication is the only way to go

Legals and insurance for this topic are nowhere near 'done' yet

CodeThink

Safe-secure C studygroup

- Deep expert community
- Highlights the friction/politics between ISO and MISRA
- So much software (e.g. Linux, Android) doesn’t involve either approach
- Most engineers don’t read the standards, let alone apply them

Arguably most of the problems are **outside the domain the group is discussing**

Most C is neither Cert nor MISRA compliant. Most software is not C

https://lists.trustable.io/cgi-bin/mailman/listinfo/c-safe-secure-studygroup

Can we trust software for **Safety**?

Subaru Destroys 293 Ascent SUVs After Coding Error Leads to Unsafe Cars

A coding error led robots to miss welds on 293 of Subaru's Ascent 2019 SUVs.

By Jessica Miley

September, 27th 2018

462 shares

ANDY GREENBERG SECURITY 07.21.15 06:00 AM

HACKERS REMOTELY KILL A JEEP ON THE HIGHWAY—WITH ME IN IT

You thought Dieselgate was over? It's not.

The scandal of Volkswagen caused political turmoil in Germany

By Wolfgang Kerler | Sep 18, 2018, 5:46pm EDT

SHARE

The number of recalls linked to electronic failures has risen by 30 per cent a year since 2012, compared with an average of 5 per cent a year between 2007 and 2012, according to data from consultancy AlixPartners.

Safety:

Our current tools are all 40-65 years old but our technology is very different today

1940 1950 1960 1970 1980 1990 2000 2010 2020

FMEA

FTA HAZOP ETA

Bow Tie (CCA) FTA + ETA

- Introduction of computer control
- Exponential increases in complexity
- New technology
- Changes in human roles

Assumes accidents caused by component failures

Source: Nancy Leveson, MIT

Software for Safety: 80s/90s

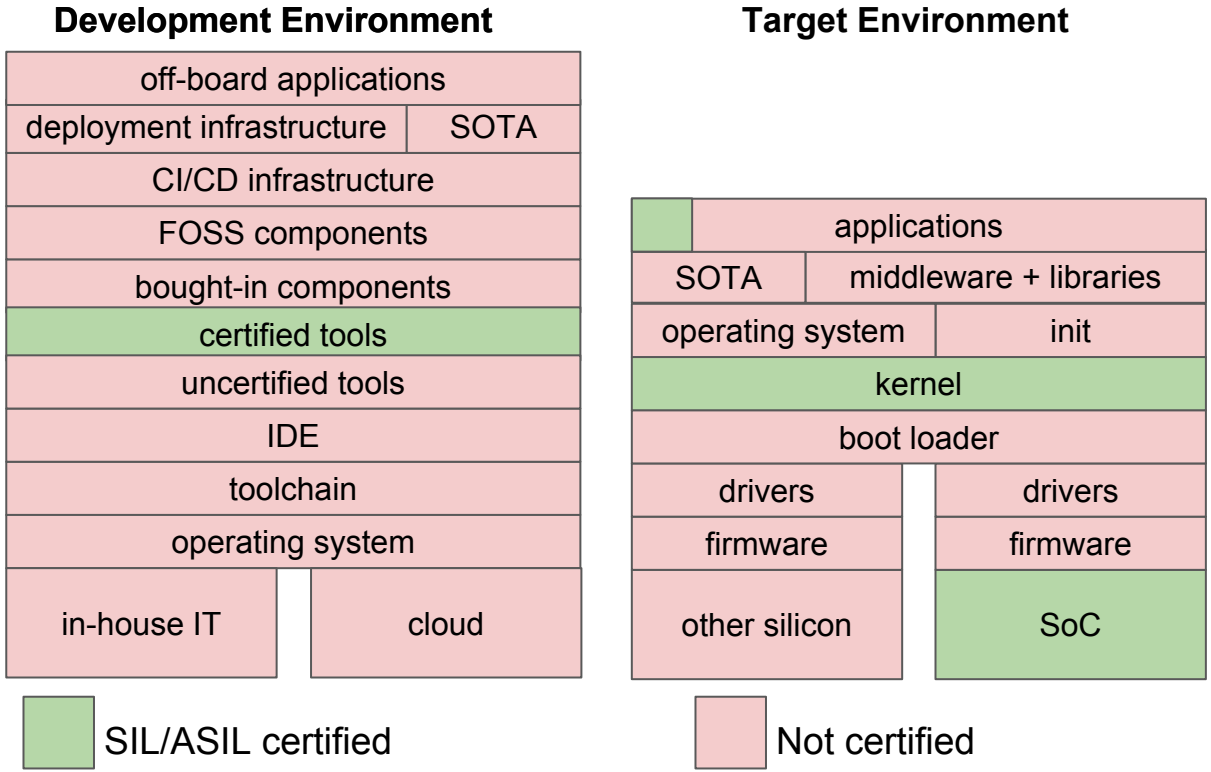
Development Environment

Target Environment

SIL/ASIL certified

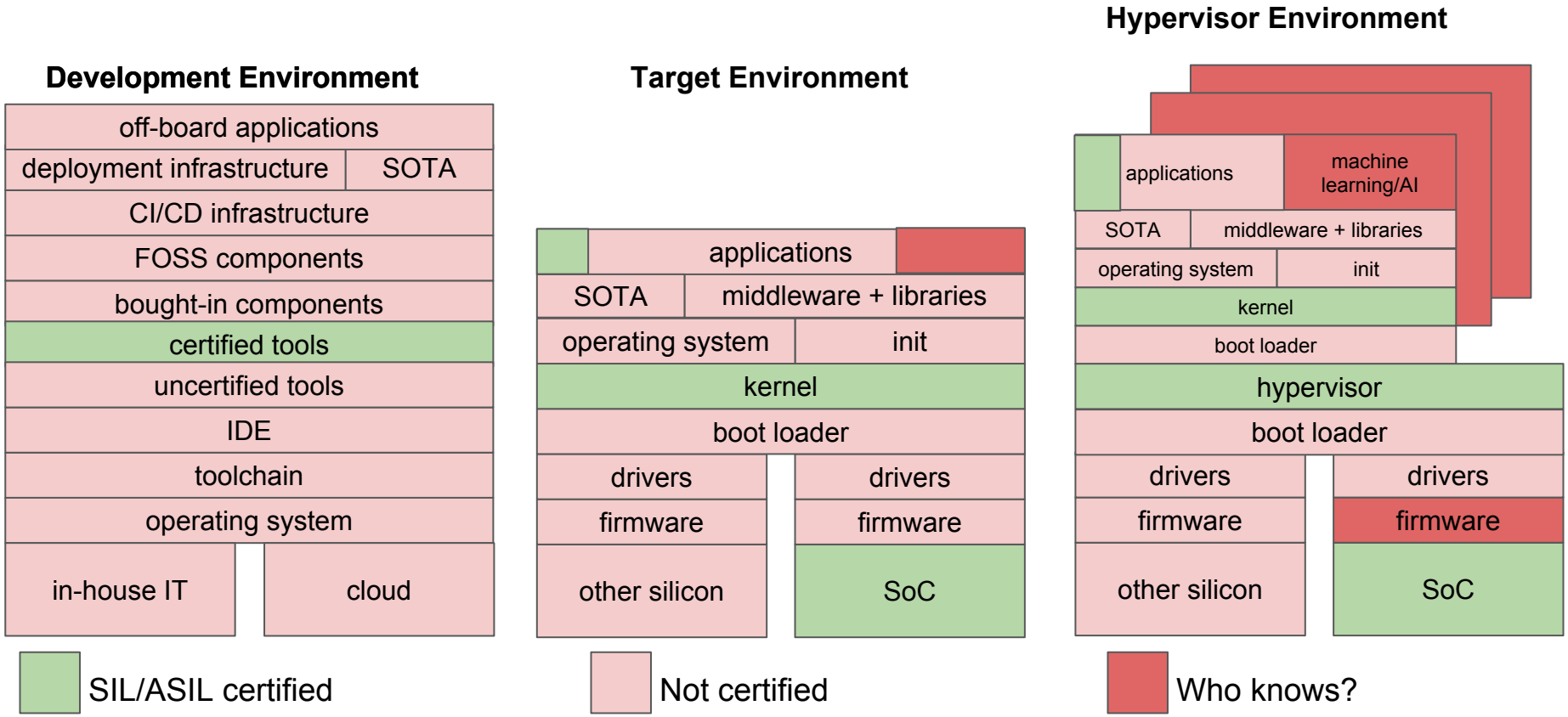
Software for Safety: as time goes by...

(we need to think about all of parts, not just the kernel and some MISRA C)



Software for Safety: 2018

(safety for connected devices involves security, obviously...)



Safety has to evolve to handle complex software...

Electromechanical **safety and reliability** requirements (functionality of seatbelts, airbags, brakes, steering, lights etc)	**Simple** electronics and software **safety and reliability** requirements	**Complex** electronics and software **safety and integrity** requirements

We **can't** guarantee behaviour of software at scale.
So designs need to **expect** misbehaving software

Some Goals

- Ability to reproduce outputs so we can guarantee the target **is** the target
- Ability to trace what happened so we can at least figure out what went wrong
- Ability to update so we can repair and/or replace
- Evidence of identity and history so we can establish who did what
- Business level traceability so executives can't just blame 'evil engineers'

… in a trustable engineering process.

And above/beyond/outside the software engineering, I think we need to

- outlaw systems that rely only on passwords, pins, biometrics, security questions
- design multiply redundant systems with multiple locks and diverse algorithms
- defeat the trade/standards ponzi schemes, be public

trustable.io
trustablesoftware.com
https://lists.trustable.io/cgi-bin/mailman/listinfo

Institute for Strategy
Resilience & Security
University College London

CodeThink