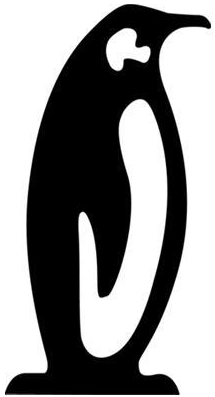


東京 2025 LINUX PLUMBERS CONFERENCE

TOKYO, JAPAN / DECEMBER 11-13, 2025

Crossing the Semantic Gap

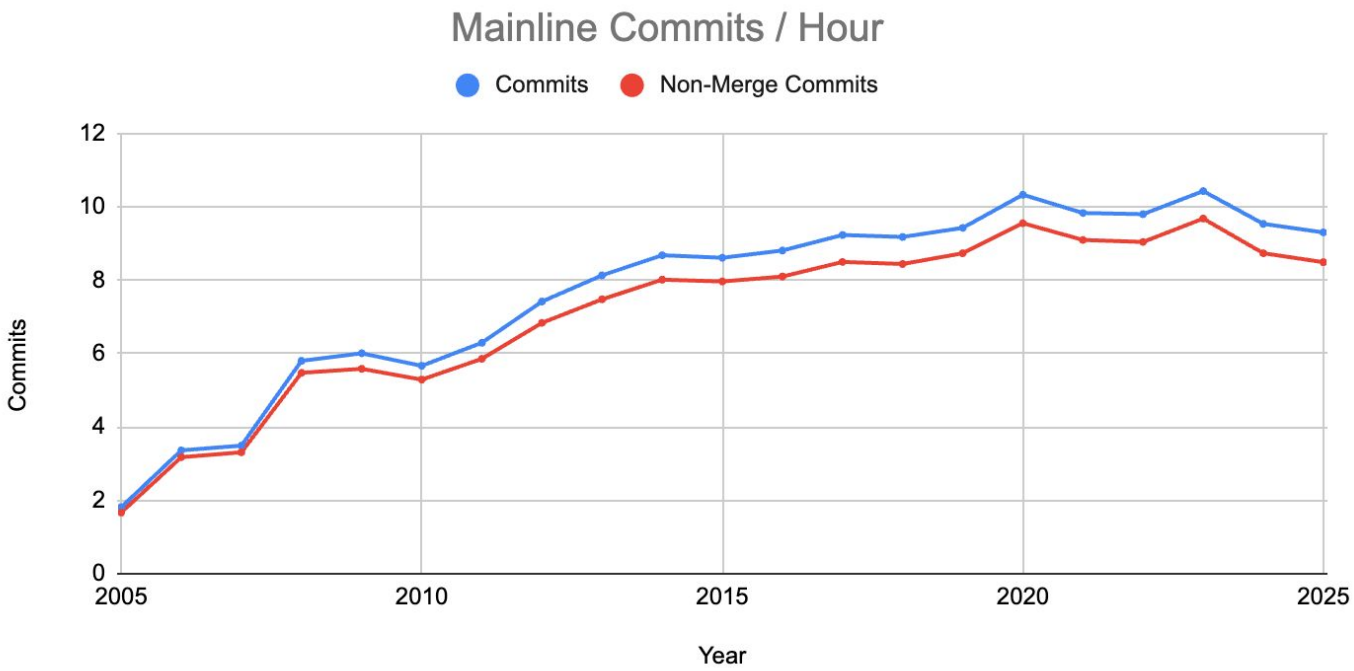
Chuck Wolber <chuck@wolber.net>
Gabriele Paoloni <gpaoloni@redhat.com>
Kate Stewart <kstewart@linuxfoundation.org>

東京 2025
LINUX PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

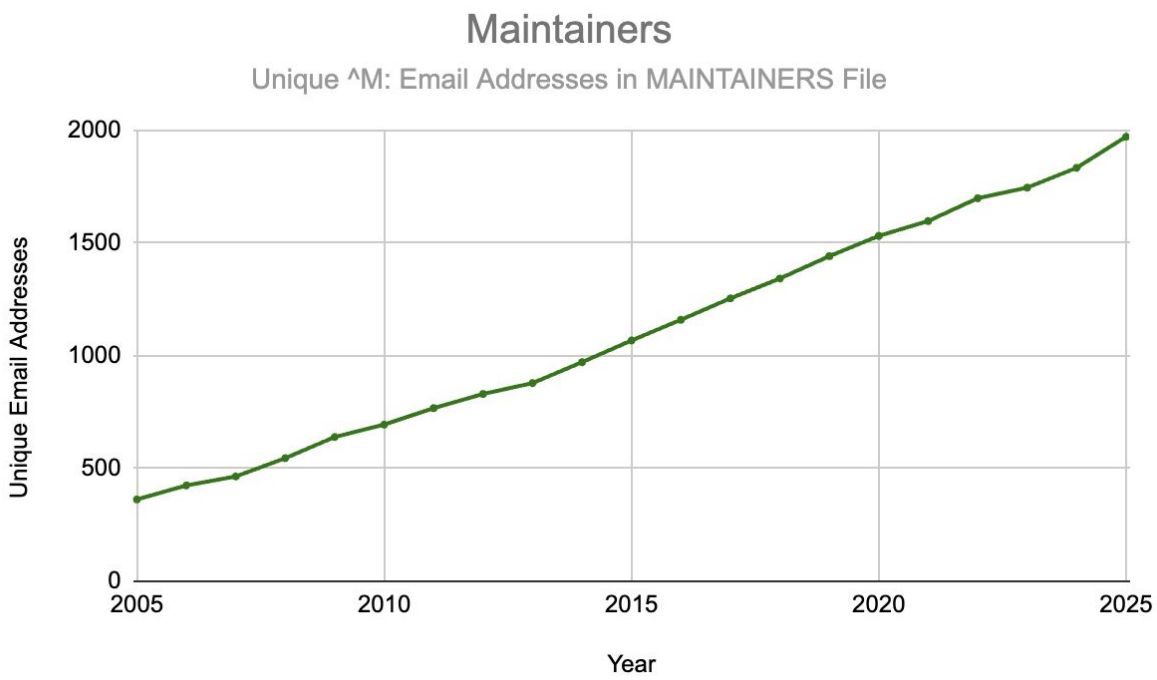
Mainline Kernel Development Activity

Year	Non-Merge Commits	Commits	Fixes	Maintainers
2005	14,589	15,857	2	363
2006	27,812	29,440	6	425
2007	28,978	30,573	10	465
2008	47,962	50,814	14	546
2009	48,911	52,614	13	640
2010	46,326	49,623	63	695
2011	51,298	55,136	69	768
2012	59,884	64,968	99	831
2013	65,499	71,227	230	879
2014	70,193	76,027	1,618	972
2015	69,765	75,419	3,152	1,068
2016	70,933	77,156	5,564	1,160
2017	74,425	80,860	8,470	1,255
2018	73,938	80,392	9,289	1,343
2019	76,500	82,561	11,527	1,442
2020	83,673	90,473	12,943	1,532
2021	79,672	86,111	13,610	1,598
2022	79,201	85,832	14,332	1,699
2023	84,777	91,339	13,957	1,746
2024	76,504	83,522	14,442	1,834
2025	74,368	81,481	13,618	1,972
Totals	1,305,208	1,411,425	123,028	

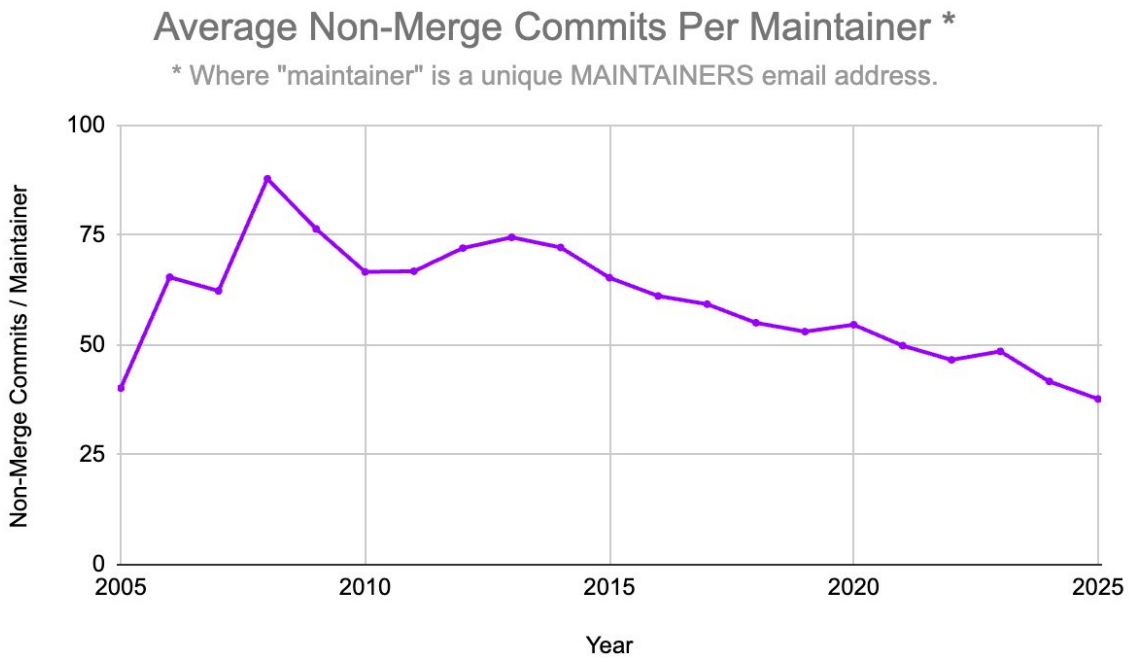
Mainline Commits Per Hour



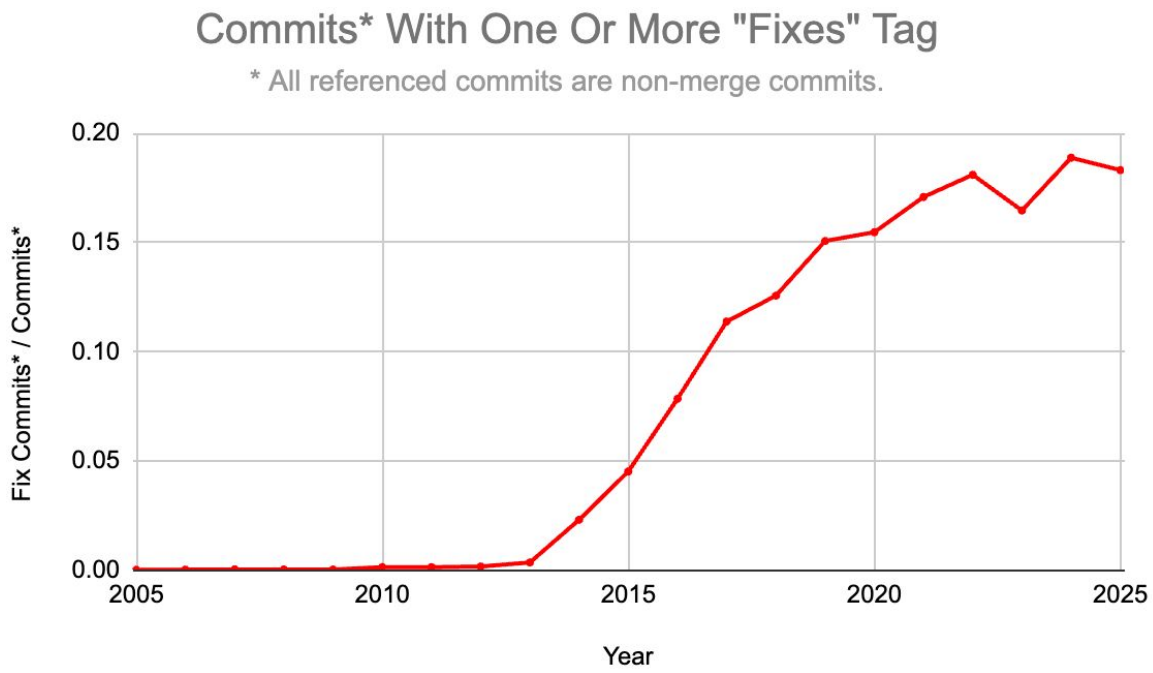
Maintainers



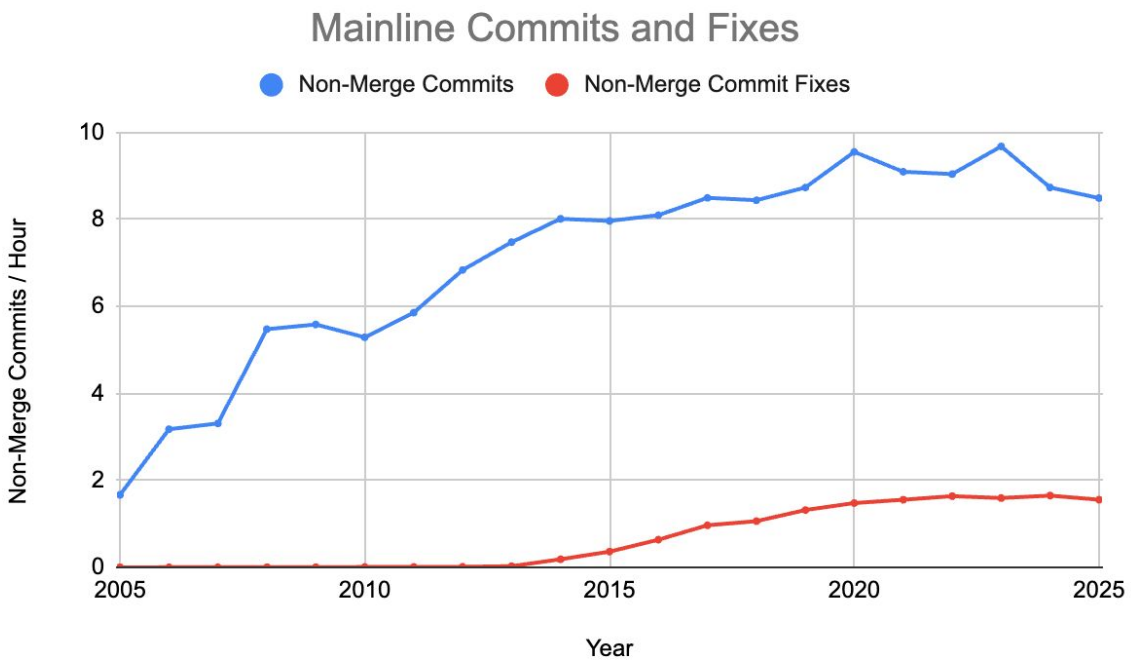
Commits Per Maintainer



Fix Commit Rate



Hourly Rate: Mainline Fixes and Commits



Thank You For Helping Us

Steven Rostedt - Ftrace Maintainer

- 2024 LPC - Safe Systems with Linux MC
  - Homework: Develop a machine readable template.
- 2024 ELISA Goddard Workshop
  - Homework: Reverse engineer the design of ftrace.
- 2025
  - In person meeting with A/B design review.
  - Multiple patch reviews, detailed feedback, code merge.

Ftrace Analysis

- Gabriele Paoloni analyzed portions of `kernel/trace/trace_events.c`
- One fix accepted into mainline.
- One “document behavior” bug that cannot be fixed without a redesign.
- Kernel-doc patch submissions:
  - trace_set_clr_event:

    https://lore.kernel.org/linux-trace-kernel/20250620085618.4489-1-gpaoloni@redhat.com/
  - event_enable_write/read (v4 currently pending review)

    https://lore.kernel.org/linux-trace-kernel/20250814122206.109096-1-gpaoloni@redhat.com/

Ftrace: Mainline Fix

```
commit 0c588ac0ca6c22b774d9ad4a6594681fdfa57d9d
Author: Gabriele Paoloni <gpaoloni@redhat.com>
Date:   2025-03-21 18:08:21 +0100

    tracing: fix return value in __ftrace_event_enable_disable for TRACE_REG_UNREGISTER

    When __ftrace_event_enable_disable invokes the class callback to
    unregister the event, the return value is not reported up to the
    caller, hence leading to event unregister failures being silently
    ignored.

    This patch assigns the ret variable to the invocation of the
    event unregister callback, so that its return value is stored
    and reported to the caller, and it raises a warning in case
    of error.

    Link: https://lore.kernel.org/20250321170821.101403-1-gpaoloni@redhat.com
    Signed-off-by: Gabriele Paoloni <gpaoloni@redhat.com>
    Signed-off-by: Steven Rostedt (Google) <rostedt@goodmis.org>
```

Ftrace: Document Behavior

Initial patch submission: https://lore.kernel.org/linux-trace-kernel/20250612104349.5047-2-gpaoloni@redhat.com/

```
+       /*
+        * A requested cnt less than strlen(buf) could lead to a wrong
+        * event status being reported.
+        */
+       if (cnt < strlen(buf))
+               return -EINVAL;
```

Final Disposition: https://lore.kernel.org/linux-trace-kernel/20250701175826.429a7b4b@batman.local.home/

"[...] It's not something that's ever been reported as an issue. I rather not touch it if it hasn't been reported as broken because there's some hypothetical use case that can see it as broken.

Documenting its current behavior is perfectly fine with me.

-- Steve"

Ftrace: Conclusions

- Analyzed about 10% of `kernel/trace/trace_events.c`
- Eight discrete threads on `linux-trace-kernel`.
  - Gabriele Paoloni: 21 Emails
  - Steven Rostedt: 9 Emails
  - Masami Hiramatsu: 4 Emails
- Mainline fix and behavior documentation is evidence of untested and/or undocumented functionality.

PREEMPT_RT Scheduling

When PREEMPT_RT developers needed to understand scheduler fairness, they had to reverse engineer kernel behavior.

The Rust Discussion...

From: Kees Cook <kees@kernel.org>

"[...] I don't see any reason to focus on replacing existing code -- **doing so would actually carry a lot of risk**. [...] Old code is more stable and has fewer bugs already, and yet, we're still going to continue the work of hardening C, because we still need to shake those bugs out."

https://lore.kernel.org/rust-for-linux/202502191026.8B6FD47A1@keescook/

General Conclusions

- Reverse engineering intended behavior of kernel code is hard.
- Questions of intended behavior require precious maintainer time.
- Nothing exists to...
  - uniformly capture this valuable expenditure of maintainer time and attention.
  - ensure subsequent code change reliably reflects original design intent.
  - reliably tie automated testing to intended behavior.
  - alert automated testing to potentially new and changed design intent.
  - trace maintainer signoff to collections of intended behavior.

So what?

```
Date: Wed, 26 Nov 2025 14:55:11 +0100
From: Greg KH <gregkh@linuxfoundation.org>
To: Chuck Wolber <chuckwolber@gmail.com>
Cc: [...]

On Fri, Nov 07, 2025 at 04:29:13 PM +0000, Chuck Wolber wrote:
> On Wed, Oct 22, 2025 at 5:13 PM Greg KH <gregkh@linuxfoundation.org> wrote:
[...]
> > For some reason Linux has succeeded in pretty much every place an
> > operating system is needed for cpus that it can run on (zephyr for those
> > others that it can not.)  So why are we suddenly now, after many decades,
> > requiring basic user/kernel stuff to be formally documented like this?
>
> You are correct, the kernel has succeeded over many decades, and will
> continue succeeding for many decades to come.
>
> With the exception of some very narrow situations, the emergent design (or
> "nuanced complexity" if you prefer that term) of the Linux kernel is not
> communicated in a broadly consistent way. This affects the way the kernel is
> tested, and also the way it is developed. Even veteran kernel maintainers are
> tripping over nuance and complexity.

We all trip over nuance and complexity, but I do not believe that adding more
formal comments is going to solve that (hint, the proof is on you...)
```

https://lore.kernel.org/lkml/2025112631-repeated-crafty-207d@gregkh/

What we currently do...

- Long Form Project Documentation
- Mailing Lists
- Commit History
- Seasoned Intuition
- Hallway Track
- Testing
  - Syzkaller
  - kselftest
  - KUnit
  - Vendor Test Suites

But Testing Today is Fundamentally Guesswork

Testing is driven by interpretations of behavior, not maintainer confirmed design intent.

And Precious Maintainer Time is Wasted

Valuable insights are not reliably captured in a uniformly reusable way that communicates to the broad community of developers and testers.

The Semantic Gap

Code is declarative, intent is inferred.

Therefore...

Coding is a lossy process.

Intent is lost when ideas are converted to declarations.

Developer Skill

Relying on developer skill to write code whose full and unambiguous intent is reliably inferred is not a viable solution*. Complexity and size will quickly overcome available resources.

* This is not an excuse to write (or tolerate) spaghetti code. Please take pride in your work and demand that others take similar pride in theirs. But recognize that there are practical limits to what design intent can be reliably inferred from even the most beautifully written code.

The Question We Asked

Is there a small and maintainable way to uniformly record design intent as it is discovered?

Back to Ftrace

On April 4, 2025, Steven Rostedt graciously agreed to join us for an in-person (Zoom) meeting to show off an A/B test of proposed methods.

Ftrace Option A

```
/*
 * [...]
 *
 * The enable parameter shall be evaluated:
 * if it is set to disable:
 *  1) evaluate soft_disable: if set decrement the soft-mode reference counter
 *  and return if the same has not reached zero. If it has reached 0:
 *    a) clear the SOFT_MODE flag;
 *    b) if the SOFT_DISABLED flag is set, prepare to disable tracing the
 *       event;
 *    c) disable buffering the events.
 *  If soft_disable is not set but the SOFT_MODE flag is set, just set
 *  the SOFT_DISABLED flag without taking any action on the event tracing, else
 *  prepare to disable tracing the event;
 *
 *  2) If the ENABLED flag is set (hence tracing the event is actually enabled):
 *     a) clear the flag
 *     b) if the RECORDED_CMD flag is set, clear it and stop tracing command
 *     line at sched switches
 *     c) if the RECORDED_TGID flag is set, clear it and stop tracing tgids at
 *     sched switches
 *     d) invoke the event class callback to unregister the event tracepoint
 *     e) clear the SOFT_DISABLED flag
 *
 * if the enable parameter is set to enable:
 * [...]
 *
 * NOTE: in order to invoke this code in a thread-safe way, event_mutex shall
 * be locked before calling it.
 * NOTE: the validity of the input pointer file shall be checked by the caller
 *
 */
static int __ftrace_event_enable_disable(struct trace_event_file *file,
                                         int enable, int soft_disable)
```

Ftrace Option A: Feedback

"Uhhh, ok. I guess... Looks like pseudo-code though. I am not sure how useful that is." - Steven Rostedt (paraphrased)

We agree, and so do a lot of other people:

> "In an [reverse engineering] process, the [intent] may be developed from source code without due regard to the difference in the abstraction level and the resulting [testable expectations] being too similar to the code **(e.g., pseudocode)**. Because such translations may be performed with **little intellectual effort or understanding of the code's intent**, these practices should not be permitted on [reverse engineering] projects because they do not provide the same level of confidence as a forward engineering process." - DOT/FAA/TC-15/27

Ftrace Option B

```
/*
 * [...]
 *
 * - If soft_disable is 1 a reference counter associated with the trace
 * event shall be increased or decreased according to the enable parameter
 * being 1 (enable) or 0 (disable) respectively.
 * If the reference counter is > 0 before the increase or after the decrease,
 * no other actions shall be taken.
 *
 * - if soft_disable is 1 and the trace event reference counter is 0 before
 * the increase or after the decrease, an enable value set to 0 or 1 shall set
 * or clear the soft mode flag respectively; this is characterized by disabling
 * or enabling the use of trace_buffered_event respectively.
 *
 * - If soft_disable is 1 and enable is 0 and the reference counter reaches
 * zero and if the soft disabled flag is set (i.e. if the event was previously
 * enabled with soft_disable = 1), tracing for the trace point event shall be
 * disabled and the soft disabled flag shall be cleared.
 *
 * - If soft_disable is 0 and enable is 0, tracing for the trace point event
 * shall be disabled only if the soft mode flag is clear (i.e. event previously
 * enabled with soft_disable = 0). Additionally the soft disabled flag shall be
 * set or cleared according to the soft mode flag being set or clear
 * respectively.
 *
 * - If enable is 1, tracing for the trace point event shall be enabled (if
 * previously disabled); in addition if soft_disable is 1 and the reference
 * counter is 0 before the increase, the soft disabled flag shall be set.
 *
 * [...]
 *
 */
static int __ftrace_event_enable_disable(struct trace_event_file *file,
                                         int enable, int soft_disable)
```

Ftrace Option B: Feedback

"Oh wow, this is great! This is exactly what I need. I even have some of my own code that is so complex that I have to relearn it every time I have to work on it. This would save me so much time!"

- Steven Rostedt (paraphrased)

But Wait...

"We all trip over nuance and complexity, but I do not believe that adding more formal comments is going to solve that (hint, the proof is on you...)"

-Greg k-h

We agree.

What this is Not

- A slippery slope into formal verification.
- Repeating the code in a different language.
- Rigid language policing based on arcane rules.
- A new process mandate.
- In any way involved, related to, or supported by AI.
- Something meant specifically for a regulated industry.

What this Is

- **Capture:** A voluntary method for uniformly collecting lost design intent.
- **Follow:** A method for synchronizing code change with intended behavior.
- **Test:** A method that reliably binds code and test to intended design.
- **Trace:** A simplified approach to maintainer sign-off of intended behavior.

We call them “Testable Expectations”.

Fine, but I see some problems here...

- Comments do not compile, how do you keep this in sync?
- Who is going to write these things?
- How long will it take?
- How do we even know they are complete or correct?

Comments do not compile...

Machine readable template* compatible with existing kernel-doc.

```
echo -nE "${PROJECT}${FILE_PATH}${TEMPLATE}${CODE}" | sha256sum
```

Tag Name	Cardinality	Mutability	Location(s)
SPDX-Req-ID	(1,1)	Immutable	Inline, Sidecar
SPDX-Req-Text	(1,1)	Mutable	Inline
SPDX-Req-End	(1,1)	N/A	Inline
SPDX-Req-Ref	(0,*)	Immutable	Inline
SPDX-Req-Note	(0,1)	Mutable	Inline
SPDX-Req-HKey	(1,1)	Mutable	Sidecar
SPDX-Req-Child	(0,*)	Mutable	Sidecar
SPDX-Req-Sys	(1,1)	Mutable	Sidecar

*** Template is still in the proposal stage. Template originally discussed at LPC 2024, and recommended by Thomas Gleixner and Steven Rostedt as a necessary element in the design.**

Who is going to write these things?

Anyone who wishes to capture design intent.

- Anyone who worked hard to unearth (complex) design intent and wants to capture that information in a useful way.
- Anyone tired of regressions.
- Anyone tired of relearning their own code.
- Anyone tired of fragile unmaintained code.
- Newcomers trying to learn the code.
- Testers trying to understand the code.
- Industry participants who want better testing and traceability.

How long will it take?

It is not a question of how long.

It is a question of respecting everyone's time.

When hard work goes into learning something about the design intent, this is a method for uniformly capturing that information and making it useful to a broad audience.

- Kernel Maintainers
- Testers
- Newcomers
- Anyone who uncovers subtle bugs or tricky edge cases.
- Yes, even industry consumers.

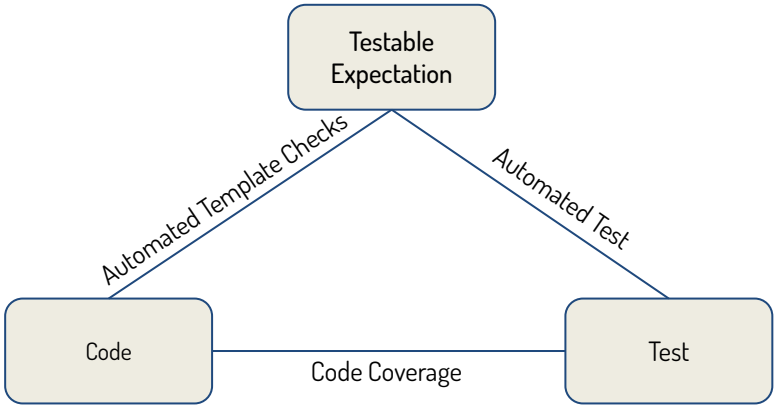
How do we even know they are complete or correct?

Other than some documented basics like conciseness, not making negative (unprovable) statements, and being precise with your words, correctness and completeness is not a necessary initial goal.

Think “eventual consistency”.

Correctness and completeness happens organically through a “Virtuous Cycle”.

Virtuous Cycle



Bug Discovery (Validation Failure)

- Invalid or missing testable expectation.
- Invalid or missing test case.

Test Failure (Verification Failure)

- Invalid testable expectation.
- Code bug.

Verification: Does it do the thing right?
Validation: Does it do the right thing?

With requirement, test, and coverage in equilibrium, it becomes possible to assert that the code is an accurate reflection of developer intent.

Ok, so walk me through the process...

- Code is always written first.
  - Maintainer review, sign-off, merge, mainline, etc.
  - *Note: Kernel-doc Testable Expectations (TE) always lag code. This is a feature.*
- CI/Automation notes checksum mismatch on existing kernel-doc TE.
- Any interested party is free to submit or update kernel-doc TE.
  - kernel-doc TE patch, maintainer sign-off attesting to reasonable accuracy.
  - Possible/optional TE maintainer sign-off path to help educate on basics.
- CI/Automation notes checksum kernel-doc TE checksum change.
  - Triggers test case review using “before and after” TE.

Call to Action

- Push back, point out flaws, give feedback, engage in constructive dialog.
- Look for kernel-doc TE patch submissions and provide feedback.
- Ask us to reverse engineer your subsystem.
- Contact us if you want to participate:
  - Template design and tool automation development.
  - Kernel-doc TE development.
  - Work with other groups (e.g. KernelCI) to close the virtuous cycle.

Thank You For Listening

Q & A

Backup Slides

Definition: Testable Expectation

A concise testable human readable description from which a minimal amount of source code can be directly implemented without further information.

Threads: linux-trace-kernel

https://lore.kernel.org/linux-trace-kernel/20250314125725.6425-1-gpaoloni@redhat.com/

https://lore.kernel.org/linux-trace-kernel/20250321170821.101403-1-gpaoloni@redhat.com/

https://lore.kernel.org/linux-trace-kernel/20250521124829.123720-1-gpaoloni@redhat.com/

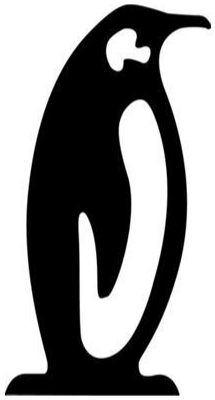
https://lore.kernel.org/linux-trace-kernel/20250620085618.4489-1-gpaoloni@redhat.com/

https://lore.kernel.org/linux-trace-kernel/20250612104349.5047-1-gpaoloni@redhat.com/

https://lore.kernel.org/linux-trace-kernel/20250630135519.11429-1-gpaoloni@redhat.com/

https://lore.kernel.org/linux-trace-kernel/20250630142032.3322-1-gpaoloni@redhat.com/

https://lore.kernel.org/linux-trace-kernel/20250814122206.109096-1-gpaoloni@redhat.com/



東京 2025
LINUX PLUMBERS CONFERENCE

TOKYO, JAPAN / DECEMBER 11-13, 2025