Coccinelle: 10 Years of Automated Evolution in the Linux Kernel

Julia Lawall (Inria/LIP6)

June, 2018

Coccinelle

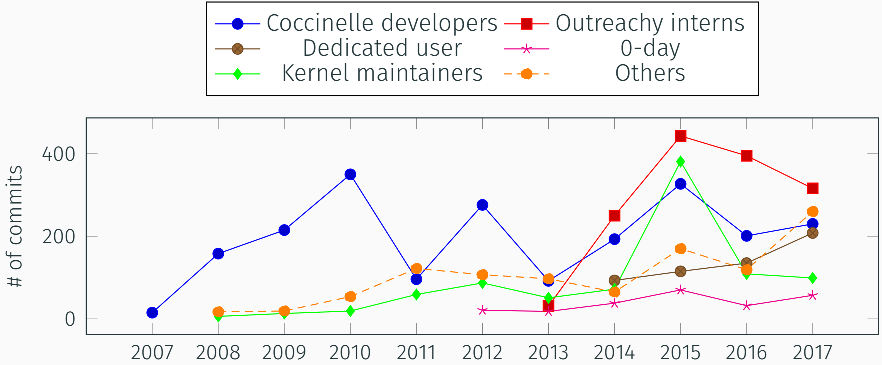
Goal: Automating bug finding and evolutions for Linux kernel developers.

- Development began in 2006.
- Goal to automate porting of Linux 2.4 drivers to Linux 2.6.

Requirements:

- Accessible to Linux developers.
- Reasoning about code as it appears to the developer.
- Treat a large subset of C.
- Ensure continuing maintainability.

Usage in the Linux kernel



How did we get here?

Coccinelle design: expressivity

Semantic patches: Patches with some abstraction.

- Remain close to the C level.
- A few extensions to control the level of abstraction.

```
@@
expression x,E1,E2;
@@
- x = kmalloc(E1,E2);
+ x = kzalloc(E1,E2);
  ...
- memset(x, 0, E1);
```

Coccinelle design: expressivity

Semantic patches: Patches with some abstraction.

- Remain close to the C level.
- A few extensions to control the level of abstraction.

```
@@
expression x,E1,E2,E3;
identifier f;
@@

- x = kmalloc(E1,E2);
+ x = kzalloc(E1,E2);
  ...  when != (<+...x...+>) = E3
       when != f(...,x,...)
- memset(x, 0, E1);
```

Coccinelle design: performance

Goal: Be usable on a typical developer laptop.

Target code base: 5MLOC in Feb 2007, 16.5MLOC in Jan 2018.

Choices:

- Intraprocedural, one file at a time.
- Process only `.c` files, by default.
- Include only local or same-named headers, by default.
- Use heuristics to parse macro uses.
- Provide best-effort type inference, but no other program analysis.

Coccinelle design: correctness guarantees

Ensure that outermost terms are replaced by like outermost terms

```
@@
expression x,E1,E2,E3;
identifier f;
@@
- x = kmalloc(E1,E2);
+ x = kzalloc(E1,E2);
  ...
- memset(x, 0, E1);
```

No other correctness guarantees:

- Bug fixes and evolutions may not be semantics preserving.
- Improves efficiency and expressiveness.
- Rely on developer's knowledge of the code base and ease of creating and refining semantic patches.

Coccinelle design: dissemination strategy

Show by example:

- June 1, 2007: Fix parse errors in kernel code.
- July 7, 2007: Irq function evolution
  - Updates in 5 files, in `net`, `atm`, and `usb`
- July 6, 2007: kmalloc + memset ⟶ kzalloc
  - Updates to 166 calls in 146 files.
  - A kernel developer responded “Cool!”.
  - Violated patch-review policy of Linux.
- July 2008: Use by a non-Coccinelle developer.
- October 2008: Open-source release.

Initial assessment

- Useful: By the Coccinelle developers to contribute to the Linux kernel.
- Usable: By outside developers to contribute to the Linux kernel.

Initial assessment

- Useful: By the Coccinelle developers to contribute to the Linux kernel.
- Usable: By outside developers to contribute to the Linux kernel.
- But some new needs emerged over time...

Expressivity evolutions

Original hypothesis: Linux kernel developers will find it easy and convenient to describe needed code changes in terms of fragments of removed and added code.

Expressivity evolutions

Original hypothesis: Linux kernel developers will find it easy and convenient to describe needed code changes in terms of fragments of removed and added code.

Confrontation with the real world:

- Many language evolutions: C features, metavariable types, etc.
- Position variables.
  - Record and match position of a token.
- Scripting language rules.
  - Original goal: bug finding, eg buffer overflows.
  - Used in practice for error reporting, counting, etc.

Position variables and scripts

```
@ r @
expression object;
position p
@@
(
drm_connector_reference@p(object)
|
drm_connector_unreference@p(object)
)

@script:python@
object << r.object;
p << r.p;
@@

msg="WARNING: use get/put helpers to reference and dereference %s" % (object)
coccilib.report.print_report(p[0], msg)
```

Performance evolutions

Original hypothesis: Limiting analysis scope via intraprocedural analysis, ignoring headers was good enough.

Performance evolutions

Original hypothesis: Limiting analysis scope via intraprocedural analysis, ignoring headers was good enough.

Confrontation with the real world:

- 1, 5, or 15 MLOC is a lot of code.
- Parsing is slow, because of backtracking heuristics.

Performance evolutions

Original hypothesis: Limiting analysis scope via intraprocedural analysis, ignoring headers was good enough.

Confrontation with the real world:

- 1, 5, or 15 MLOC is a lot of code.
- Parsing is slow, because of backtracking heuristics.

Evolutions:

- Indexing, via glimpse, id-utils.
- Parallelism, via parmap.

Correctness guarantee evolutions

Original hypothesis: Developer control over rules is good enough.

Correctness guarantee evolutions

Original hypothesis: Developer control over rules is good enough.

Confrontation with the real world: Mostly, developer control over rules is good enough.

Dissemination strategy evolutions

Original hypothesis: Show by example rather than attempting to impose.

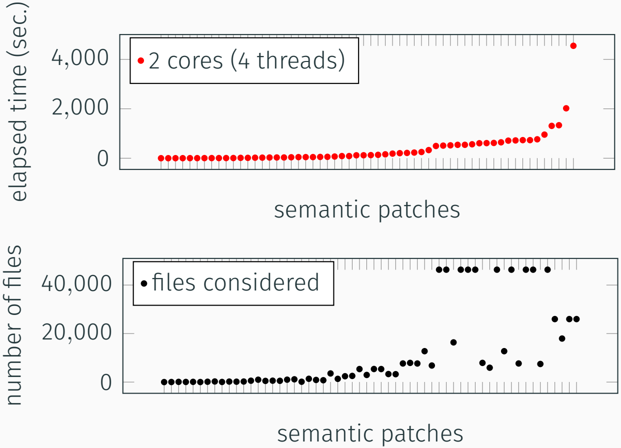
Dissemination strategy evolutions

Original hypothesis: Show by example rather than attempting to impose.

Confrontation with the real world:

- Showing by example generated initial interest.
- Organized four workshops: industry participants.
- Presentations at developer conferences: FOSDEM, Linux Plumbers, etc.
- LWN articles by kernel developers.

Status: Performance

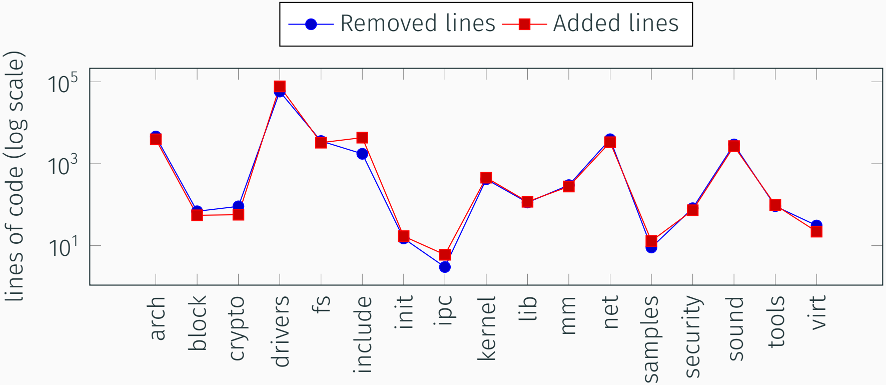


Based on the 59 semantic patches in the Linux kernel.

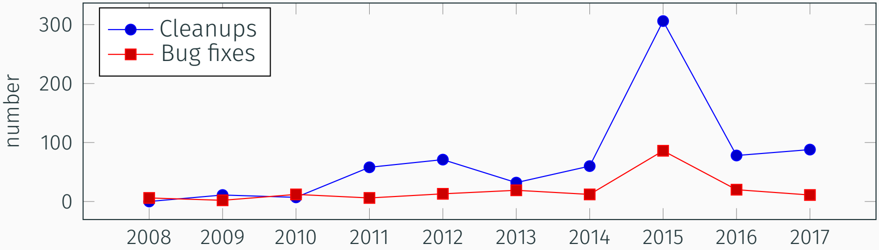
Status: Use of new features

- 3325 commits contain semantic patches.
- 18% use position variables.
- 5% use scripts.
- 43% of the semantic patches using position variables or scripts are from outside the Coccinelle team.
- All 59 semantic patches in the Linux kernel use both.

Impact: Changed lines



Impact: Maintainer use



45% of maintainers who have at least one commit touching at least 100 files have at some point used Coccinelle.

Impact: Maintainer use examples

TTY. Remove an unused function argument.

- 11 affected files.

DRM. Eliminate a redundant field in a data structure.

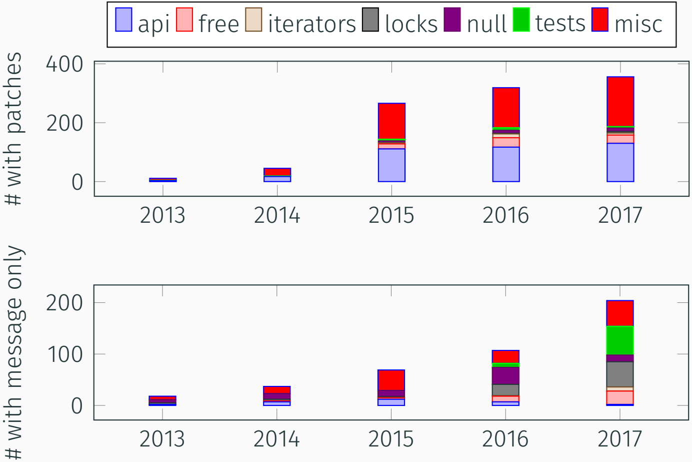
- 54 affected files.

Interrupts. Prepare to remove the irq argument from interrupt handlers, and then remove that argument.

- 188 affected files.

Impact: Intel’s 0-day build-testing service

59 semantic patches in the Linux kernel with a dedicated make target.



Coccinelle community

25 contributors

- Most at Inria, due to use of OCaml and PL concepts.
- Active mailing list.

Availability

- Packaged for many Linux distros.

Use outside Linux

- RIOT, systemd, qemu, etc.

Conclusion: Lessons learned

- Visibility is necessary.
- Tool should be easy to access and install.
- Tool should be easy to use and robust.
- Interleaving pattern matching and scripts is very powerful.
- Avoid creeping featurism: Do one thing and do it well.

`http://coccinelle.lip6.fr`