Coccinelle: 10 Years of Automated Evolution in the Linux Kernel

Julia Lawall, Gilles Muller (Inria/LIP6)

July 12, 2018

Context

The Linux kernel:

- Open source OS kernel, used in smartphones to supercomputers.
- 16MLOC and rapidly growing.
- Frequent changes to improve correctness and performance.

Issues:

- How to perform evolutions in such a large code base?
- Once a bug is found, how to check whether it occurs elsewhere?

How to better maintain large code bases?

Patches: The key to reasoning about change in the Linux kernel.

```
@@ -1348,8 +1348,7 @@
- fh = kmalloc(sizeof(struct zoran_fh), GFP_KERNEL);
+ fh = kzalloc(sizeof(struct zoran_fh), GFP_KERNEL);
  if (!fh) {
    dprintk(1,
      KERN_ERR "%s: zoran_open(): allocation of zoran_fh failed\n",
      ZR_DEVNAME(zr));
    return -ENOMEM;
  }
- memset(fh, 0, sizeof(struct zoran_fh));
```

A SmPL idea: Raise the level of abstraction to semantic patches.

From:

```
@@ -1348,8 +1348,7 @@
- fh = kmalloc(sizeof(struct zoran_fh), GFP_KERNEL);
+ fh = kzalloc(sizeof(struct zoran_fh), GFP_KERNEL);
  if (!fh) {
    dprintk(1,
      KERN_ERR "%s: zoran_open(): allocation of zoran_fh failed\n",
      ZR_DEVNAME(zr));
    return -ENOMEM;
  }
- memset(fh, 0, sizeof(struct zoran_fh));
```

Coccinelle

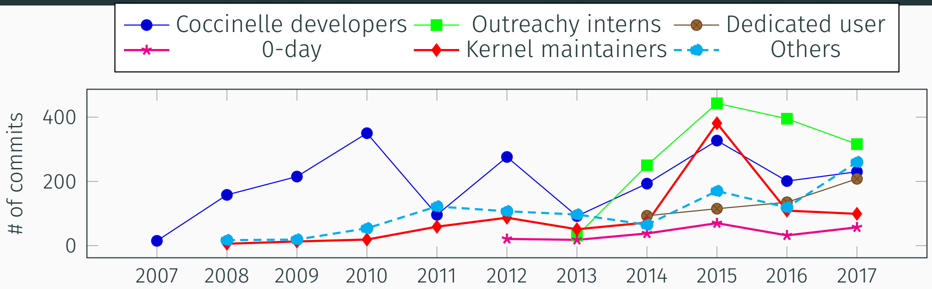
A SmPL idea: Raise the level of abstraction to semantic patches.

To:

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

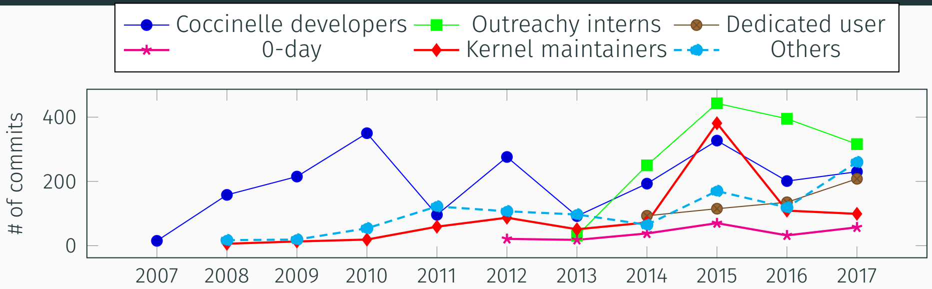
- SmPL = Semantic Patch Language
- Coccinelle applies SmPL semantic patches across a code base.
- Development began in 2006, first released in 2008.

Usage in the Linux kernel



- Over 5500 commits.
- 59 semantic patches in the Linux kernel, usable via make coccicheck.

Usage in the Linux kernel



- Over 5500 commits.
- 59 semantic patches in the Linux kernel, usable via make coccicheck.
- 44% of the 88 kernel developers who have at least one commit that touches 100 files also have at least one commit that uses Coccinelle.

How did we get here?

Design dimensions

- Expressivity
- Performance
- Correctness guarantees
- Dissemination

Did we make the right decisions?

Coccinelle design: expressivity

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.

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

Expressivity evolutions

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)
```

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.

Coccinelle design: performance

Goal: Be usable on a typical developer laptop.

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

Original design choices:

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

Performance evolutions

Confrontation with the real world:

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

Performance evolutions

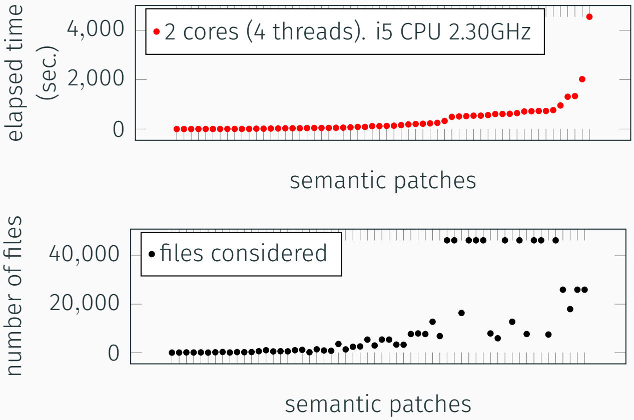
Confrontation with the real world:

- 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.

Status: Performance



Based on the 59 semantic patches in the Linux kernel.

Coccinelle design: correctness guarantees

Ensure that outermost terms are replaced by like outermost terms

```
@@
expression x,E1,E2,E3;
@@
- 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 expressiveness and performance.
- Rely on developer's knowledge of the code base and ease of creating and refining semantic patches.

Correctness guarantee evolutions

Confrontation with the real world:

Mostly, developer control over readable rules is good enough.

Coccinelle design: dissemination strategy

Show by example:

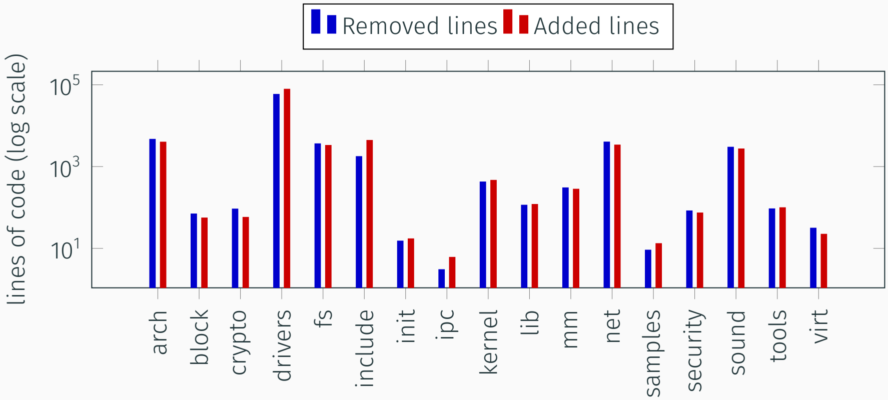
- June 1, 2007: Fix parse errors in kernel code.
- July 7, 2007: Irq function evolution
  - Updates in 5 files, in `net`, `atm`, and `usb`
- July 6, 2007: kmalloc + memset $\longrightarrow$ kzalloc
  - Updates to 166 calls in 146 files.
  - A kernel developer responded “Cool!”.
  - Violated patch-review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

Dissemination strategy evolutions

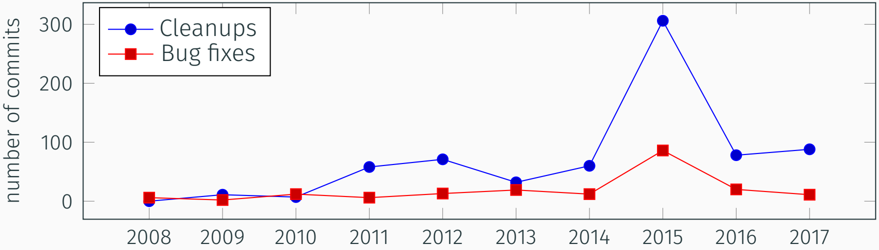
Confrontation with the real world:

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

Impact: Changed lines



Impact: Maintainer use



Impact: Maintainer use examples

TTY. Remove an unused function argument.

- 11 affected files.

DRM. Eliminate a redundant field in a data structure.

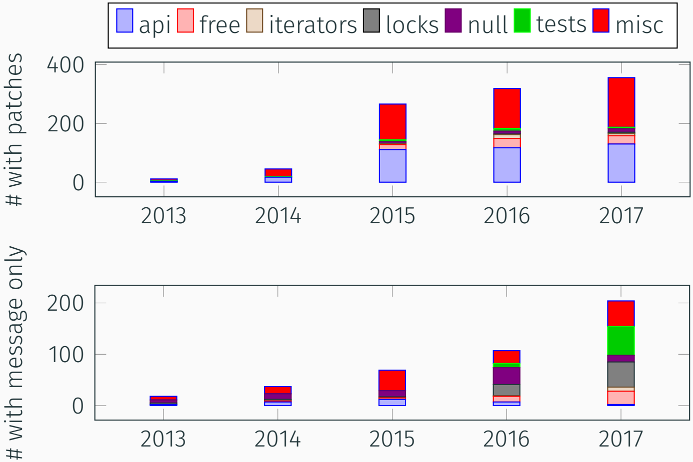
- 54 affected files.

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 from the Coccinelle team, due to use of OCaml and PL concepts.
- Active mailing list (cocci@systeme.lip6.fr).

Availability

- Packaged for many Linux distros.

Use outside Linux

- RIOT, systemd, qemu, etc.

Conclusion

- Initial design decisions mostly remain valid, with some extensions.
  - Take the expertise of the target users into account.
  - Avoid creeping featurism: Do one thing and do it well.
- Tool should be easy to access and install, and easy to use and robust.
- Success measure: Over 5500 commits in the Linux kernel based on Coccinelle.

Conclusion

- Initial design decisions mostly remain valid, with some extensions.
  - Take the expertise of the target users into account.
  - Avoid creeping featurism: Do one thing and do it well.
- Tool should be easy to access and install, and easy to use and robust.
- Success measure: Over 5500 commits in the Linux kernel based on Coccinelle.
- Probably, everyone in this room uses some Coccinelle modified code!

Conclusion

- Initial design decisions mostly remain valid, with some extensions.
  - Take the expertise of the target users into account.
  - Avoid creeping featurism: Do one thing and do it well.
- Tool should be easy to access and install, and easy to use and robust.
- Success measure: Over 5500 commits in the Linux kernel based on Coccinelle.
- Probably, everyone in this room uses some Coccinelle modified code!

`http://coccinelle.lip6.fr`