Coccinelle, Prequel, and Spinfer: Automating Summarization and Application of Code Evolutions in the Linux Kernel

Julia Lawall, Lucas Serrano, Gilles Muller (Inria/LIP6)
January 20, 2020
Properties of the Linux kernel

- Code size is 18 MLOC
- Around 4000 contributors per year
- Need for frequent, rapid, large-scale changes
  - Security
  - Performance
  - New hardware features
Properties of the Linux kernel

Changes need to be propagated across multiple versions:

- 4.9.x
- 4.12.x
- 4.11.x
- 4.10.x

Mainline

Stables

Long term

Seems like an impossible challenge?
Properties of the Linux kernel

Changes need to be propagated across multiple versions:

- 4.9.x
- 4.12.x
- 4.11.x
- 4.10.x

Mainline

Stables

Long term

Seems like an impossible challenge?
Mitigating factor: Lots of similar code

<table>
<thead>
<tr>
<th></th>
<th>cxacru.c</th>
<th>speedtch.c</th>
<th>ueagle-atm.c</th>
<th>usbatm.c</th>
<th>xusbatm.c</th>
</tr>
</thead>
<tbody>
<tr>
<td>usbatm_usb_probe</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td>x</td>
</tr>
<tr>
<td>interface_to_usbdev</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>usb_submit_urb</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>usb_set_intfdata</td>
<td>-</td>
<td>x</td>
<td>-</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>request_firmware</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>wait_for_completion</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>mutex_lock</td>
<td>x</td>
<td>-</td>
<td>x</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>init_timer</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>kzalloc</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>-</td>
</tr>
</tbody>
</table>
Opportunities

- Possibility to script repetitive changes
Opportunities

- Possibility to script repetitive changes (Coccinelle)
Opportunities

- Possibility to *script* repetitive changes (Coccinelle)
- Possibility to *find* previous changes to *guide* new ones
Opportunities

- Possibility to script repetitive changes (Coccinelle)
- Possibility to find previous changes to guide new ones (Prequel)
Opportunities

- Possibility to **script** repetitive changes (Coccinelle)
- Possibility to **find** previous changes to **guide** new ones (Prequel)
- Possibility to **learn** how to perform changes from examples
Opportunities

- Possibility to **script** repetitive changes (Coccinelle)
- Possibility to **find** previous changes to **guide** new ones (Prequel)
- Possibility to **learn** how to perform changes from examples (Spinfer)
1. Scripting repetitive changes

Problem:

- Library interfaces change.
- Need to update all users.
1. Scripting repetitive changes

Problem:

- Library interfaces change.
- Need to update all users.

Example:

- init_timer replaced by setup_timer

- Over 400 occurrences in various Linux kernel versions.
**Examples**

**drivers/infiniband/hw/nes/nes_hw.c:**

- `init_timer(&nesadapter->mh_timer);`
- `nesadapter->mh_timer.function = nes_mh_fix;`
+ `setup_timer(&nesadapter->mh_timer, nes_mh_fix, (unsigned long)nesdev);`
  - `nesadapter->mh_timer.expires = jiffies + (HZ/5);`
- `nesadapter->mh_timer.data = (unsigned long)nesdev;`

**drivers/usb/atm/speedtch.c:**

- `init_timer(&instance->status_check_timer);`
- `instance->status_check_timer.function = speedtch_status_poll;`
- `instance->status_check_timer.data = (unsigned long)instance;`
+ `setup_timer(&instance->status_check_timer, speedtch_status_poll, (unsigned long)instance);`
  - `instance->last_status = 0xff;`
  - `instance->poll_delay = MIN_POLL_DELAY;`
- `init_timer(&instance->resubmit_timer);`
- `instance->resubmit_timer.function = speedtch_resubmit_int;`
- `instance->resubmit_timer.data = (unsigned long)instance;`
+ `setup_timer(&instance->resubmit_timer, speedtch_resubmit_int, (unsigned long)instance);`
**Examples**

---

**drivers/infiniband/hw/nes/nes_hw.c:**

- `init_timer(&nesadapter->mh_timer);`
- `nesadapter->mh_timer.function = nes_mh_fix;`
  + `setup_timer(&nesadapter->mh_timer, nes_mh_fix, (unsigned long)nesdev);`
- `nesadapter->mh_timer.expires = jiffies + (HZ/5);`
- `nesadapter->mh_timer.data = (unsigned long)nesdev;`

**drivers/usb/atm/speedtch.c:**

- `init_timer(&instance->status_check_timer);`
- `instance->status_check_timer.function = speedtch_status_poll;`
- `instance->status_check_timer.data = (unsigned long)instance;`
  + `setup_timer(&instance->status_check_timer, speedtch_status_poll, (unsigned long)instance);`
  - `instance->last_status = 0xff;`
  - `instance->poll_delay = MIN_POLL_DELAY;`
- `init_timer(&instance->resubmit_timer);`
- `instance->resubmit_timer.function = speedtch_resubmit_int;`
- `instance->resubmit_timer.data = (unsigned long)instance;`
  + `setup_timer(&instance->resubmit_timer, speedtch_resubmit_int, (unsigned long)instance);`
Coccinelle

- Allows recurring changes to C code to be expressed using patch-like code patterns (semantic patches) using the language SmPL.
- Applies SmPL semantic patches to an entire code base, updating all relevant code sites at once.
- **Goal**: fit with the existing habits of the Linux developer.
Creating a semantic patch

Start with a typical example:

- init_timer(&nesadapter->mh_timer);
- nesadapter->mh_timer.function = nes_mh_fix;
+ setup_timer(&nesadapter->mh_timer, nes_mh_fix, (unsigned long)nesdev);
  nesadapter->mh_timer.expires = jiffies + (HZ/5);
- nesadapter->mh_timer.data = (unsigned long)nesdev;
Creating a semantic patch

Drop irrelevant code:

- \texttt{init\_timer(\&nesadapter->mh\_timer);}
- \texttt{nesadapter->mh\_timer.function = nes\_mh\_fix;}
+ \texttt{setup\_timer(\&nesadapter->mh\_timer, nes\_mh\_fix, (unsigned long)nesdev);}
...  
- \texttt{nesadapter->mh\_timer.data = (unsigned long)nesdev;}

Creating a semantic patch

Abstract over common subterms:

```plaintext
@@
expression timer, fn, d;
@@
- init_timer(&timer);
- timer.function = fn;
+ setup_timer(&timer, fn, d);
  ...
- timer.data = d;
```
Results

Updates 163 instances: 248 remaining.
Updates 163 instances: 248 remaining.

Another option: initialize data before function

```c
@@
expression timer, fn, d;
@@
- init_timer(&timer);
- timer.data = d;
+ setup_timer(&timer, fn, d);
  ...
- timer.function = fn;
```
Updates 163 instances: 248 remaining.

Another option: initialize data before function

```c
@@
expression timer, fn, d;
@@
- init_timer(&timer);
- timer.data = d;
+ setup_timer(&timer, fn, d);
... 
- timer.function = fn;
```

Updates 100 more instances: 148 still remaining.
Impact: Patches using Coccinelle in the Linux kernel

Almost 8000 patches overall (linux-next).
Assessment

+ Helps the developer express his knowledge of how a change should be done.
+ Also useful for metrics, bug finding, etc.
+ Supports C, some C++, and Java (prototype).
Assessment

+ Helps the developer express his knowledge of how a change should be done.

+ Also useful for metrics, bug finding, etc.

+ Supports C, some C++, and Java (prototype).

- Doesn’t help if one doesn’t know what to do.

- Doesn’t help understand what others have done.
2. Finding examples of repetitive changes

Problem:

- Library interface has changed.
- What should be done to update all users? How to find relevant examples?
2. Finding examples of repetitive changes

Problem:

- Library interface has changed.
- What should be done to update all users? How to find relevant examples?

Example:

- “Don’t use init_timer, use setup_timer instead.”
Analogy between:

- Performing changes (semantic patch):
  - Describe code to match and remove
  - Describe code to add

- Matching changes (patch query):
  - Describe removed code
  - Describe added code
Analogy between:

- Performing changes (semantic patch):
  - Describe code to match and remove
  - Describe code to add

- Matching changes (patch query):
  - Describe removed code
  - Describe added code

Query language for patches in a git history
Prequel

Analogy between:

- Performing changes (semantic patch):
  - Describe code to match and remove
  - Describe code to add

- Matching changes (patch query):
  - Describe removed code
  - Describe added code

Query language for patches in a git history

**Goal:** fit with the existing habits of the Linux developer.
init_timer → setup_timer patch query

@@
@@
- init_timer(...);
+ setup_timer(...);
init_timer → setup_timer patch query

@@
@@
- init_timer(...);
+ setup_timer(...);

Results:

- fcb58a033271: 100% (157/301)
- acc6539fe629: 100% (205/301)
- e9c43a75cda0: 100% (163/301)
- 2443c6cc92ed: 100% (114/301)
- ...
- aff55a3638a2: 88% (137/301)
- 03f23fc51dc9: 83% (222/301)
- b9eaf1872222: 80% (1/301)
- 96ff2c11c5e8: 75% (116/301)
- 01e77e13fc5a: 75% (291/301)
- ...
Assessment

+ Gives examples of how to make a change.
+ Also useful for metrics.
+ Supports C, some C++, and Java (prototype).
Assessment

+ Gives examples of how to make a change.
+ Also useful for metrics.
+ Supports C, some C++, and Java (prototype).
- Human intervention required to understand and perform the illustrated changes.
3. Inferring semantic patches from examples

Problem:

- Library interface has changed.
- How to understand changes done by others?
- How to perform the changes fully automatically?
3. Inferring semantic patches from examples

Problem:

- Library interface has changed.
- How to understand changes done by others?
- How to perform the changes fully automatically?

Example:

- “Don’t use init_timer, use setup_timer instead.”
From patch examples, infer a semantic patch that generalizes the changes.
From patch examples, infer a semantic patch that generalizes the changes.

Main challenges:

- Identify change instance boundaries
- Identify constraints between fragments in a change instance
  - Control-flow constraints.
  - Data-flow constraints.
- Recognize multiple change variants
- Recognize noise

Principally the work of Lucas Serrano
Methodology

- Collection of common code fragments (clustering).
- Identification of common control-flow graph structures (…).
- Generalization of common code fragments into patterns.
  - Metavariables connecting common terms within matched code fragments.
  - Metavariables connecting terms from the matched code to the constructed code.

To avoid high complexity, address these one by one.
- init_timer(&nesadapter->mh_timer);
- nesadapter->mh_timer.function = nes_mh_fix;
+ setup_timer(&nesadapter->mh_timer, nes_mh_fix, (unsigned long)nesdev);
    nesadapter->mh_timer.expires = jiffies + (HZ/5);
- nesadapter->mh_timer.data = (unsigned long)nesdev;

- init_timer(&instance->status_check_timer);
- instance->status_check_timer.function = speedtch_status_poll;
- instance->status_check_timer.data = (unsigned long)instance;
+ setup_timer(&instance->status_check_timer, speedtch_status_poll, (unsigned long)instance);
    instance->last_status = 0xff;
    instance->poll_delay = MIN_POLL_DELAY;
- init_timer(&instance->resubmit_timer);
- instance->resubmit_timer.function = speedtch_resubmit_int;
- instance->resubmit_timer.data = (unsigned long)instance;
+ setup_timer(&instance->resubmit_timer, speedtch_resubmit_int, (unsigned long)instance);

- init_timer(&dev->tx_watchdog);
- dev->tx_watchdog.data = (unsigned long)ndev;
- dev->tx_watchdog.function = ns83820_tx_watch;
+ setup_timer(&dev->tx_watchdog, ns83820_tx_watch, (unsigned long)ndev);
## Clustering

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>init_timer(&amp;nesadapter-&gt;mh_timer);</code></td>
<td><code>nesadapter-&gt;mh_timer.function = ...;</code></td>
</tr>
<tr>
<td><code>init_timer(&amp;instance-&gt;status_check_timer);</code></td>
<td><code>instance-&gt;status_check_timer.function = ...;</code></td>
</tr>
<tr>
<td><code>init_timer(&amp;instance-&gt;resubmit_timer);</code></td>
<td><code>instance-&gt;resubmit_timer.function = ...;</code></td>
</tr>
<tr>
<td><code>init_timer(&amp;dev-&gt;tx_watchdog);</code></td>
<td><code>dev-&gt;tx_watchdog.function = ...;</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>nesadapter-&gt;mh_timer.data = ...;</code></td>
<td><code>setup_timer(&amp;nesadapter-&gt;mh_timer, ...);</code></td>
</tr>
<tr>
<td><code>instance-&gt;status_check_timer.data = ...;</code></td>
<td><code>setup_timer(&amp;instance-&gt;status_check_timer, ...);</code></td>
</tr>
<tr>
<td><code>instance-&gt;resubmit_timer.data = ...;</code></td>
<td><code>setup_timer(&amp;instance-&gt;resubmit_timer, ...);</code></td>
</tr>
<tr>
<td><code>dev-&gt;tx_watchdog.data = ...;</code></td>
<td><code>setup_timer(&amp;dev-&gt;tx_watchdog, ...);</code></td>
</tr>
</tbody>
</table>
Abstractation of clustered examples

<table>
<thead>
<tr>
<th>init_timer(&amp;X0);</th>
<th>X0.function = X1;</th>
</tr>
</thead>
<tbody>
<tr>
<td>init_timer(&amp;nesadapter-&gt;mh_timer);</td>
<td>nesadapter-&gt;mh_timer.function = ...;</td>
</tr>
<tr>
<td>init_timer(&amp;instance-&gt;status_check_timer);</td>
<td>instance-&gt;status_check_timer.function = ...;</td>
</tr>
<tr>
<td>init_timer(&amp;instance-&gt;resubmit_timer);</td>
<td>instance-&gt;resubmit_timer.function = ...;</td>
</tr>
<tr>
<td>init_timer(&amp;dev-&gt;tx_watchdog);</td>
<td>dev-&gt;tx_watchdog.function = ...;</td>
</tr>
<tr>
<td>X0.data = X1;</td>
<td>setup_timer(&amp;X0,X1,X2);</td>
</tr>
<tr>
<td>nesadapter-&gt;mh_timer.data = ...;</td>
<td>setup_timer(&amp;nesadapter-&gt;mh_timer, ...);</td>
</tr>
<tr>
<td>instance-&gt;status_check_timer.data = ...;</td>
<td>setup_timer(&amp;instance-&gt;status_check_timer, ...);</td>
</tr>
<tr>
<td>instance-&gt;resubmit_timer.data = ...;</td>
<td>setup_timer(&amp;instance-&gt;resubmit_timer, ...);</td>
</tr>
<tr>
<td>dev-&gt;tx_watchdog.data = ...;</td>
<td>setup_timer(&amp;dev-&gt;tx_watchdog, ...);</td>
</tr>
</tbody>
</table>
Semantic-patch-rule graph construction

Ingredients:

- init_timer(&X0);
- X0.function = X1;
- X0.data = X1;
+ setup_timer(&X0,X1,X2);

Current rule:

Pending rule:
Generate and test as the rule is constructed
Ingredients:

- `X0.function = X1;`
- `X0.data = X1;`
+ `setup_timer(&X0, X1, X2);`

Current rule:

```
init_timer(&X0);
```
Semantic-patch-rule graph construction

Ingredients:

- X0.data = X1;
+ setup_timer(&X0, X1, X2);

Current rule:

```
init_timer(&X0);
```

```
X0.function = X1;
```

Dominance and post-dominance required
Semantic-patch-rule graph construction (splitting)

Ingredients:

- init_timer(&X0);
- X0.function = X1;
- X0.data = X1;

+ setup_timer(&X0, X1, X2);

Pending ingredients:

- X0'.data = X1';
+ setup_timer(&X0', X1', X2');

Current rule:

```
init_timer(&X0);
X0.function = X1;
X0.data = X1;
```

Pending rule:

```
init_timer(&X0');
X0'.function = X1';
```
Semantic-patch-rule graph construction

Ingredients:

- init_timer(&X0);
- X0.function = X1;
- X0.data = X1;
+ setup_timer(&X0,X1,X2);

Pending ingredients:

- X0'.data = X1';
+ setup_timer(&X0',X1',X2');

Current rule:

- init_timer(&X0);
- X0.function = X1;
- X0.data = X1;
- setup_timer(X0,X1,X2);

Pending rule:

- init_timer(&X0');
- X0'.function = X1';
In completed rule graphs introduce metavariables according to some constraints:

- Metavariables used in removed or context code should reflect relationships common to all instances.

- Metavariables used in added code must be instantiated by removed or context code.

- Respecting the constraints on added code may entail more splitting.
Generation of semantic patch code and assignment of metavariables

```
@@ expression timer, fn, d; @@
- init_timer(&timer);
- timer.function = fn;
+ setup_timer(&timer, fn, d);
... 
- timer.data = d;
```
@@expression T,F,D;@@
- init_timer(&T);
+ setup_timer(&T, F, D);
...
- T.data = D;
- T.function = F;

@@expression T,F,D;@@
- init_timer(&T);
+ setup_timer(&T, F, D);
...
- T.data = D;
- T.function = F;
...
- T.data = D;
+ setup_timer(&T, F, D);

@@expression T,F,D;@@
- T.function = F;
- T.data = D;
+ setup_timer(&T, F, D);

@@expression T,F;@@
- init_timer(&T);
+ setup_timer(&T, F, 0UL);
...
- T.function = F;
How did we address the challenges?

Identify change instance boundaries:

- Generate and test: smaller rules may cover all instances.

Control-flow constraints:

- Dominance/postdominance requirement

Data-flow constraints:

- Metavariant constraints
How did we address the challenges?

Recognize multiple change variants:

- Iterative rule construction. Splitting.

Noise:

- Distance metric used in clustering - distant terms are dropped.
Evaluation results

Completes in at most a few minutes for most examples.
Pipeline for fully automating summarizing and performing of evolutions:

- Prequel to find examples.
- Spinfer to infer semantic patches.
- Coccinelle to apply the results to the code.
Pipeline for fully automating summarizing and performing of evolutions:

- Prequel to find examples.
- Spinfer to infer semantic patches.
- Coccinelle to apply the results to the code.
- Semantic patches as an intermediate step allow user customization.

Potential uses:
- Backporting of complete services to older kernel releases (e.g., device drivers)
- Backporting of patches over API changes.
- Upstreaming of out-of-tree code.
- Make Coccinelle more accessible to new or infrequent users.

Conclusion

Pipeline for fully automating summarizing and performing of evolutions:

- Prequel to find examples.
- Spinfer to infer semantic patches.
- Coccinelle to apply the results to the code.
- Semantic patches as an intermediate step allow user customization.

Potential uses:

- Backporting of complete services to older kernel releases (e.g., device drivers)
- Backporting of patches over API changes.
- Upstreaming of out-of-tree code.
- Make Coccinelle more accessible to new or infrequent users.
Conclusion

Pipeline for fully automating summarizing and performing of evolutions:

- Prequel to find examples.
- Spinfer to infer semantic patches.
- Coccinelle to apply the results to the code.
- Semantic patches as an intermediate step allow user customization.

Potential uses:

- Backporting of complete services to older kernel releases (e.g., device drivers)
- Backporting of patches over API changes.
- Upstreaming of out-of-tree code.
- Make Coccinelle more accessible to new or infrequent users.