Yaminabe Revisited

A Case Study for Linux Variants of Consumer Electronic Devices

Rainer Koschke, University of Bremen & Armijn Hemel
Tjaldur Software Governance Solutions

November 5, 2012

The authors

The original Yaminabe project

The Yaminabe research project was presented at LinuxCon Europe 2011 where Tsugikazu Shibata said the following:

- We expect there is a lot of non-upstream code in each industry product. However, is that true?
- For what part of the kernel, how many lines of codes and why?
- If we do not know anything, we are not be able to support them.
- So, we need some investigation.

Yaminabe scope

Yaminabe conducted a study of 15 “Gingerbread” smartphone from:

- HTC (12 devices)
- LG (1 device)
- Motorola (1 device)
- Samsung (1 device)

because of easy access to source code, a *real* “Gingerbread” kernel (instead of “Froyo” kernel with “Gingerbread” userland) and market share.

Due to sheer volume there will be quite a few HTC specific results.

Implementation of Yaminabe

1. create a database with checksums of files from many kernels from `kernel.org` and the Android project (193 in total). This includes kernels later than 2.6.35 because of backporting and forwardporting of patches.
2. for each handset kernel remove all files that can be found in any of the upstream kernels
3. for each file left determine if any of the other handset kernels has a file with the exact same name and different checksum

The result is a list of files that have the same name, are different, which can't be found in an upstream kernel and which are in at least two different devices, so likely to be duplicate effort!

Results of Yaminabe

- each kernel has about 5% to 10% files that are different from upstream (except the Motorola device: 0%)
- there was plenty of duplicate effort. Mostly these were expected changes (driver code, board specific code), but also quite a few changes in the *core kernel*.

The Yaminabe project gave enough reasons to start the "Long Term Support Initiative".

Drawbacks of the Yaminabe project

Yaminabe has a strong emphasis on checksums, not on actual differences. This is very fast, but it treats every change (including whitespace change) as equally important.

But not every change is equally important! Changed code is more important than whitespace changes (this is not Python!) or changed comments.

We need something more fine grained.

Introducing Yaminabe 2

Yaminabe 2 started at Dagstuhl Seminar 12071 in Germany ("Software Clone Management Towards Industrial Application") where prof. Rainer Koschke and I discussed using *code clone detection* for large-scale variance analysis.

In classic clone detection the goal is to discover copies of code.

Here not the clones themselves, but the *differences* are relevant.

Differences, especially duplicated work, could indicate cloning at a higher level, namely *cloning of work effort*. To relate non-identical yet similar files to each other we use clone detection.

Yaminabe 2 as a code clone detection problem

We combined the original Yaminabe approach with clone detection. Yaminabe acts as a “filter” to get rid of “uninteresting” files and *shrink the problem space* significantly.

Using a code clone detector from Axivion (`http://www.axivion.com/`) called “codematches” the remaining files were researched for similarity.

Results were presented at Working Conference on Reverse Engineering (WCRE) 2012.

Tokens & similarity

To compare the files they are first tokenized, where some information is discarded:

- whitespace
- comments
- identifier names (rewritten to generic names)
- values of literals
- and so on

then the tokens are hashed.

In short: it abstracts from the concrete names of identifiers and values of literals.

If two files have the same hash value, they are token-type identical, or short: t-identical.

Two t-identical files

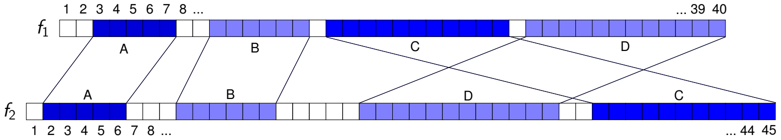
File f_1

```
int foo(int a, int i){
  while (i > 0) {
      a = a * i;
      i++;
  }
  return a;
}
```

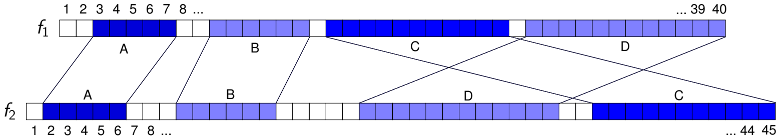
File f_2

```
int foo(int x, int i){
  while (i > 0) {
      x = x * i;
      i++;
  }
  return x;
}
```

File Similarity

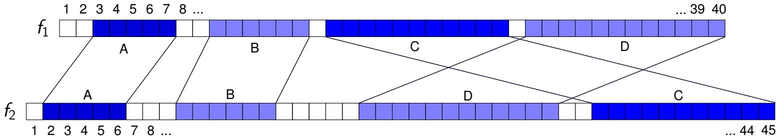


File Similarity



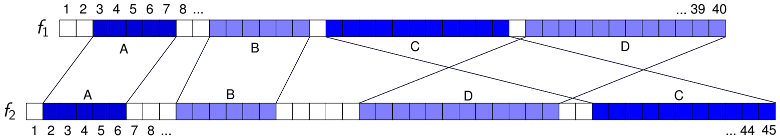
$$tsim(f_1, f_2) = \frac{|\{t | t \in f_1 \wedge t \in clone(f_1, f_2)\}|}{|f_1|}$$

File Similarity

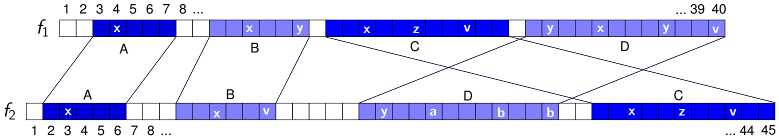


$$tsim(f_1, f_2) = \frac{|\{t | t \in f_1 \wedge t \in clone(f_1, f_2)\}|}{|f_1|} = \frac{34}{40}$$

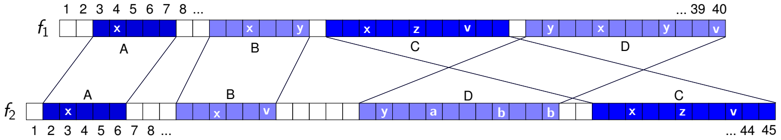
File Similarity



File Similarity

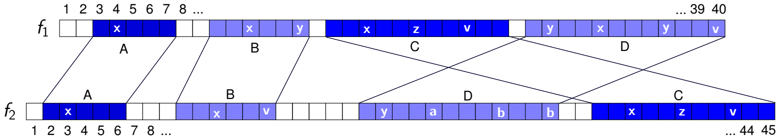


File Similarity



$$psim(f_1, f_2) = \frac{|parameters(f_1) \cap parameters(f_2)|}{|parameters(f_1) \cup parameters(f_2)|}$$

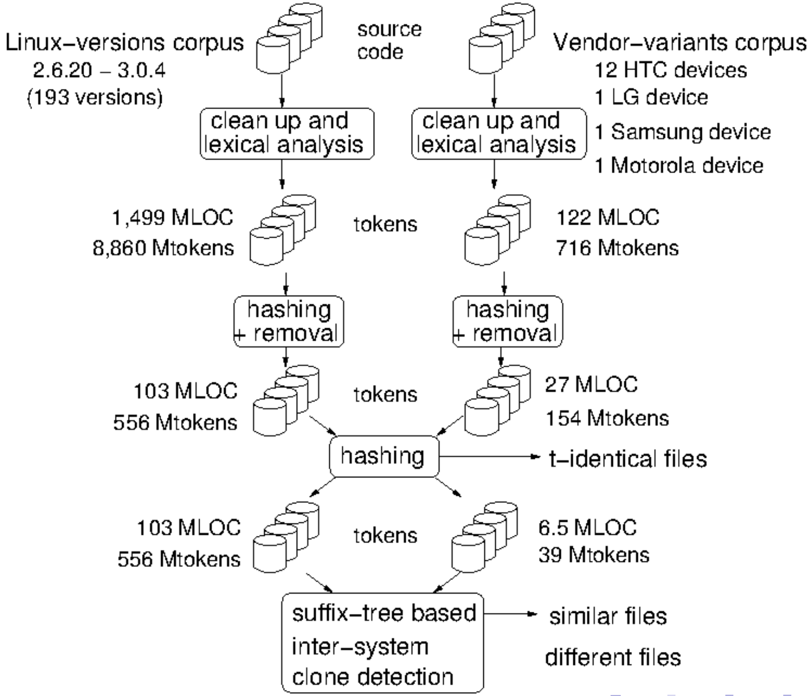
File Similarity



$$psim(f_1, f_2) = \frac{|parameters(f_1) \cap parameters(f_2)|}{|parameters(f_1) \cup parameters(f_2)|} = \frac{|\{x, y, z, v\}|}{|\{a, b, x, y, z, v\}|}$$

Similarity Criterion

$$(\mathit{tsim}(f_1, f_2) \geq 0.7 \vee \mathit{tsim}(f_2, f_1) \geq 0.7) \wedge \mathit{psim}(f_1, f_2) \geq 0.75$$


Linux–versions corpus
2.6.20 – 3.0.4
(193 versions)
source code
Vendor–variants corpus
12 HTC devices
1 LG device
1 Samsung device
1 Motorola device
clean up and lexical analysis
clean up and lexical analysis
1,499 MLOC
8,860 Mtokens
tokens
122 MLOC
716 Mtokens
hashing + removal
hashing + removal
103 MLOC
556 Mtokens
tokens
27 MLOC
154 Mtokens
hashing
t–identical files
103 MLOC
556 Mtokens
tokens
6.5 MLOC
39 Mtokens
suffix–tree based inter–system clone detection
similar files
different files

Top-14 files with highest number of variants

path	modified	similar
drivers/mmc/core/core.c	12	4
drivers/usb/gadget/android.c	12	1
drivers/usb/gadget/composite.c	12	2
drivers/mmc/card/block.c	11	4
drivers/usb/gadget/f_mass_storage.c	11	2
drivers/mmc/core/mmc.c	10	9
drivers/mmc/host/msm_sdcc.c	10	8
include/linux/mmc/host.h	10	10
arch/arm/mach-msm/devices_htc.c	9	0
drivers/cpufreq/cpufreq_ondemand.c	9	8
drivers/mmc/host/msm_sdcc.h	9	8
drivers/net/wireless/bcm4329_204/wl_iw.c	9	1
drivers/video/msm/msm_fb.c	9	1
kernel/power/wakelock.c	9	1

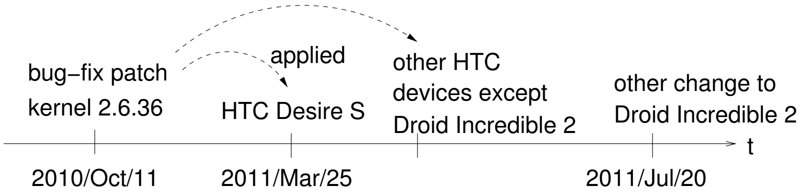
Top-14 files with highest number of variants (resorted)

path	modified	similar
arch/arm/mach-msm/devices_htc.c	9	0
drivers/usb/gadget/android.c	12	1
drivers/video/msm/msm_fb.c	9	1
drivers/net/wireless/bcm4329_204/wl_iw.c	9	1
kernel/power/wakelock.c	9	1
drivers/usb/gadget/composite.c	12	2
drivers/usb/gadget/f_mass_storage.c	11	2
drivers/mmc/core/core.c	12	4
drivers/mmc/card/block.c	11	4
drivers/mmc/host/msm_sdcc.c	10	8
drivers/cpufreq/cpufreq_ondemand.c	9	8
drivers/mmc/host/msm_sdcc.h	9	8
drivers/mmc/core/mmc.c	10	9
include/linux/mmc/host.h	10	10

Most-Varied Subsystems

- MultiMediaCard (MMC)
- USB gadget driver
- Touchscreen input drivers
- Others: limited differences in other input drivers, memory management, YAFFS2, network drivers

Missed Defect Correction



Future research (1)

Yaminabe 2 gave much more fine grained results, but it is not enough:

- same criteria used for all parts of the kernel: for some subsystems (like the core kernel) changes are much more significant than for other parts. What are good thresholds?
- "dead code" (code that might have been changed, but is not used in the final binary and just left to bitrot) is not removed. Changes that are irrelevant might be reported.

Future research (2)

Other open research questions are:

- How much have vendors improved? Is more code being upstreamed?
- Are there any vendor specific patches that are ported between generations of devices that are not upstreamed?

Conclusions

- many variances introduced by handset manufacturers
- some expected (handset specifics), some unexpected (core kernel)
- combination of file hashing and zooming in with inter-system clone detection finds:
    - highly dissimilar/variant path-identical files: integration difficulties
    - highly similar path-identical files: local patches or missed patches
- handset manufacturers fail(ed) syncing changes in the mainline
- better communication and collaboration among kernel and headset developers needed

Questions?