Discussion of Statistical Methods for SIL2LinuxMP

Nicholas Mc Guire <safety@osadl.org>

July 14, 2016

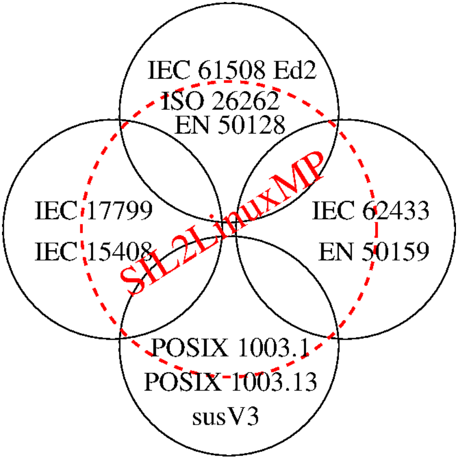
Open Source Automation Development Lab eG

Outline

- Introdution
- Overall approach to quantification
- Short overview of Linux kernel DLC
- Sttaistical process assessment
- Estimating bugs
- Analyzing exposure and coupling issues
- Handling Severity
- Conclusions

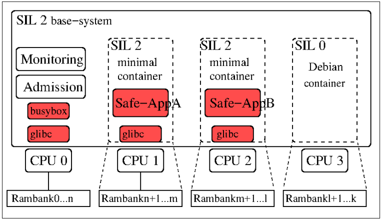
Nicholas Mc Guire <safety@osadl.

SIL2LinuxMP Context

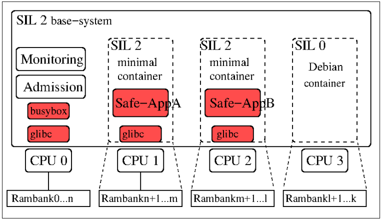


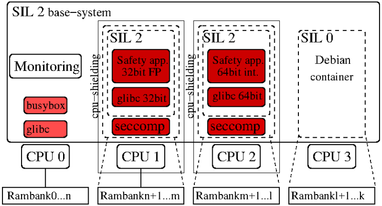
- Generic qualification approach for multicore systems
- Suitable for up to SIL2 (IEC 61508 Ed 2)

Specific context - Arch 4



Specific context - Arch 4





DLC Adjusted for software selection

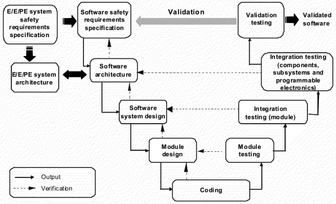


Figure 6 – Software systematic capability and the development lifecycle (the V-model)

DLC Adjusted for software selection

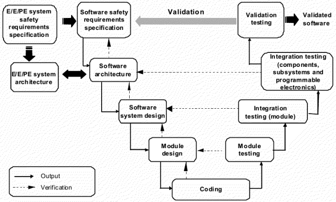
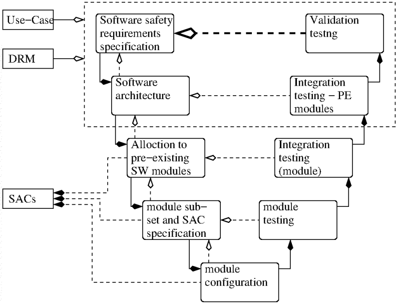
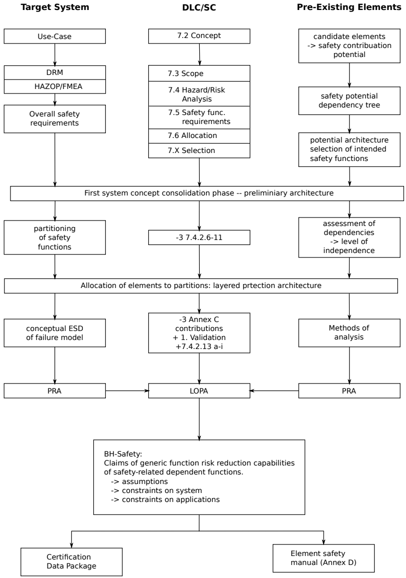


Figure 6 – Software systematic capability and the development lifecycle (the V-model)



Softwar systematic capability – V−model for pre−existing softwa

Big picture of DLC/SLC



Kernel DLC

Kernel DLC assessment of continuity

Nicholas Mc Guire
<safety@osadl.

-next/-linus 2nd/3ed stage

- Daily integration (...well 5 times a week on average)
- integration relative to linus tree

```
Non-merge commits (relative to Linus' tree): 1899
 1637 files changed, 67661 insertions(+), 46594 deletions(-)
```

- Merges 232 trees (incl Linus' and 35 trees pending for Linus)
- Build-log available at http://kisskb.ellerman.id.au/linux-next
- Daily report on
  - added/dropped trees
  - new conflicts
  - new build failures
  - any fixups inserted

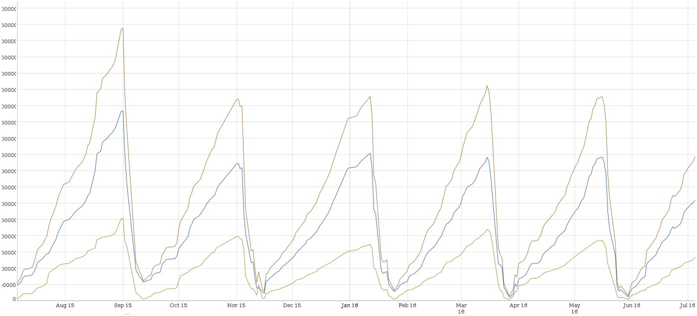
Statistics: http://neuling.org/linux-next-size.html

adding trees to -next


```
Thanks for adding your subsystem tree as a participant of linux-next.

As you may know, this is not a judgment of your code.  The purpose of
linux-next is for integration testing and to lower the impact of
conflicts between subsystems in the next merge window.

You will need to ensure that the patches/commits in your tree/series
have been:
  * submitted under GPL v2 (or later) and include the Contributor's
    Signed-off-by,
  * posted to the relevant mailing list,
  * reviewed by you (or another maintainer of your subsystem tree),
  * successfully unit tested, and
  * destined for the current or next Linux merge window.

Basically, this should be just what you would send to Linus (or ask
him to fetch).  It is allowed to be rebased if you deem it necessary.
--
Cheers,
Stephen Rothwell
sfr@canb.auug.org.au
```

No exceptions - Greg KH asks for a tree to be added - this message goes out !

Linux -next dynamics

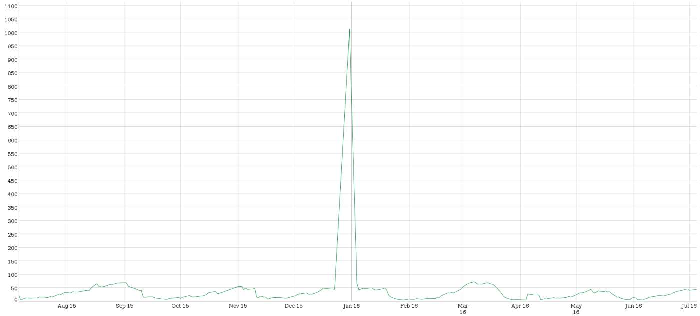


while(1) Integrate -> Consolidate

Usage of linux-next

- A large part of patches is against linux-next
- It is for integration - so it does not necessarily retain a seerate history
- Coordination of subsystem happens a lot in linux-next, asynchronous development can break things - linux-next is where it should break
- 0-day-build and arm-buildbot use linux-next for extended build-testing and boot-testing
- linux-next is a main input to the commit-window and has significantly lowered the conflict rate and conflict induced bugs (hot-fixes tend to be more buggy than continuous cleanup and polishing)

Linux -next merge conflicts



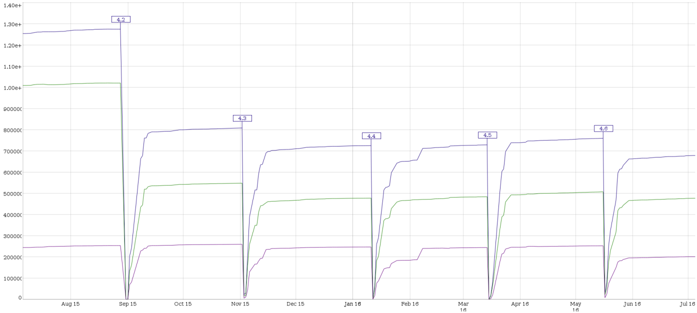
Note that it does hot hit 0 as Steven Rothwel carries some fixups in -next to resolve these issues.

Merge conflicts as 1st dependency indicator

Conclicts in -next during March and April 2016 cumulative

```
1 aio/vfs                    |  1 rdma/Linus
1 akpm-current/drm           |  1 rdma/crypto
1 akpm-current/gpio          |  1 rdma/net-next
1 akpm-current/powerpc       |  1 samsung-krzk/arm-soc
1 akpm-current/tile          |  1 samsung-krzk/omap
1 akpm/tip                   |  1 staging/staging
1 block/Linus                |  1 staging/staging.current
1 btrfs-kdave/Linus          |  1 staging/v4l-dvb
1 ceph/Linus                 |  1 staging/watchdog
1 clk tree/arm-soc           |  1 tip/arm64
1 crypto/net-next            |  1 tip/drm
1 current,staging/Linus      |  1 tip/mips
1 drm-misc/drm-intel         |  1 usb-gadget/usb-gadget-fixes
1 drm/Linus                  |  1 usb/tip
1 dt-rh/tegra                |  1 vfs/Linus
1 ext4/Linus                 |  1 watchdog/arm-soc
1 gpio/mfd                   |  1 xen-tip/arm64
1 gpio/tip                   |  1 xen-tip/tip
1 keys/crypto                |  1 xfs/ext4
1 kvm-ppc-paulus/kvm-arm     |  2 akpm-current/tip  <--
1 leds/nand                  |  2 drm-misc/Linus    <--
1 livepatching/Linus         |  2 overlayfs/ext4    <----  !!
1 mvebu/arm-soc              |  3 arm64/Linus       <--
1 net-next/Linus             |  3 tip/pm            <--
1 pm/renesas                 |  5 net-next/net
1 rcu/Linus                  |
```

Linux -linus consolidation



Estimating bugs

-stable search for explanatory parameters

- Bainstorm session in Bullendorf - what do we expect ?
  - Complexity of commits expressed as files-mode, lines-added, lines-del should be useful
  - Really old bugs should be small/simple so they can hide for a long time
  - Most patches are short lived -> strongly right scewed distribution -> old bugs should have been uncovered "by now"
  - big changes should be the ones with most bugs (complexity) -> bug-commits should have "bad-metrics"
  - rank of committer - a naive measure of developers quality should be a relevant factor (not yet implemented) - but I think it is not.
  - tools issues could also play in here, e.g. relevant gcc changes
- Derive the data templates and
- fire up R and filter out what we need

Basic data havesting options

To allow for such an analysis there are basically three (and a half) options

- Systematically analyze a series of bug-fix commits from -stable manually
- Randomly select a set of bug-fix commits from -stable and analyze manually
- Automatically extract the relevant information from git
- Correlated bug-fix density with some subset that represents the bug population faithfully -> "Fixes:" git tag and if correlation is high - take the age of "Fixes:" bug-fix commits

(anyone still following ?) - actually we are doing all three methods so that we can achieve resonable assurance.

Manual bug analysis process

- -stable changes are bug-fixes only -> use that as first source
- Generate overview `git diff --stat --stat-width=120 v4.5.1...v4.5.2`
- Identify relevant subsystems <- SIL2LinuxMP reference systems
- For each subsystem
  -> `git diff --stat v4.X.Y...v4.X.Y+1 subsystem`
- For each subsytem bug
  -> `git log -p v4.X.Y...v4.X.Y+1 file` analyze bug
  -> fill in bug template
- Extract suspected response parameters of interest over explanatory parameter(s)
- Determine suitable regression model
- Regression model -> signific., homosedasticity etc.
- Automate it all -> run over all kernel versions

Overview example

```
Statistic analysis of kernel devlopemnt 4.5.1 to 4.5.2
  git diff --stat --stat-width=120 v4.5.1...v4.5.2 | cut -f 1 -d "/" \
    | grep -v -e Makefile | sort | uniq -c | sort -n
    1  147 files changed, 1259 insertions(+), 557 deletions(-)
    1  crypto
    1  kernel
    1  mm
    2  Documentation
    5  sound
    9  include
   12  fs
   23  arch
   24  net
   68  drivers
```

Analyze kernel,mm,include,fs,arch,net and drivers selectively

Overview of fs - example

```
fs/btrfs/file.c        |   2 +-
fs/btrfs/tree-log.c    | 137 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
fs/dcache.c            |   5 ++-
fs/ext4/crypto.c       |  12 ++++--
fs/ext4/ext4.h         |  23 +++++++++++
fs/ext4/file.c         |  12 ++++--
fs/ext4/move_extent.c  |  11 +++++-
fs/ext4/super.c        |  47 ++++++++++++++++------
fs/nfs/dir.c           |   6 +--
fs/nfs/inode.c         |   2 +-
fs/nfs/nfs4file.c      |   4 +-
fs/overlayfs/super.c   |  33 ++++++++++++++++
12 files changed, 263 insertions(+), 31 deletions(-)
```

dcache is generic, ext4 and overlayfs on the wishlist -> descope brtfs and nfs

Document selection and descoping

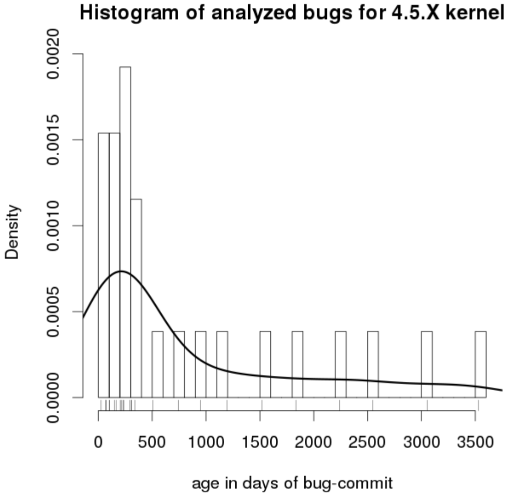
```
git diff --stat v4.5.1...v4.5.2 fs
12 files changed, 263 insertions(+), 31 deletions(-)
-> only dcache,ext4 relevant, overlayfs under consideration
  fs/dcache.c           |    5 +++-
  fs/ext4/crypto.c      |   12 +++++---
  fs/ext4/ext4.h        |   23 ++++++++++++++
  fs/ext4/file.c        |   12 +++++---
  fs/ext4/move_extent.c |   11 +++++--
  fs/ext4/super.c       |   47 +++++++++++++++++++++--------
  fs/overlayfs/super.c  |   33 ++++++++++++++++++++
```

The key point being that we are constraining our efforts at virtually every step we take, so we have to record these constraints so that any re-use can actually poin-point such limitations and re-open cases as need be. The format is really something we are still working on though...

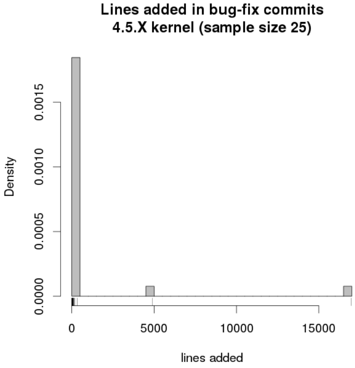
Teplate and example

```
git log -p vX...vX+1 file| git log -p v4.5.2...v4.5.3 kernel/events/core.c
 1)                      |  1) perf/core: Fix time tracking bug with multiplexing
 Commit:                 |  Commit: 1f861d0348fd
 Date:                   |  Date: Mar 29 2016/1462398554
 Stat:                   |  Stat: 1 file changed, 12 insertions(+), 2 deletions(-)
 Source:                 |  Source: Patchwork
 Consequence:            |  Consequence: wrong perf counts
 Severity:               |  Severity: minor
 Safety-related:         |  Safety-related: No (only would affect perf)
 Note:                   |  Note: could test-data but the error is visible reporting <not counted>
 Testcase:               |  Testcase: yes - perf see commit note
 Root-Cause:             |  Root-Cause: incorrect treatment of corner case time value of 0
 Reported:               |  Reported: LKML only
 Reported by:            |  Reported by: Stephane Eranian <eranian@google.com>
 Introduced:             |  Introduced: 3cbaa5906967 ("perf: Fix ctx time tracking by introducing EVENT
       Date:             |        Date: Feb 24 2016/1456386154
       Stat:             |        Stat: 1 file changed, 30 insertions(+), 12 deletions(-)
       From:             |        From: v4.5-rc6~1^2~4
       Offset:           |        Offset: 4
       Age:              |        Age: 6012400s/69days
       Rank:             |        Rank: TODO
       Note:             |        Note self fix
 Action:                 |  Action:
```

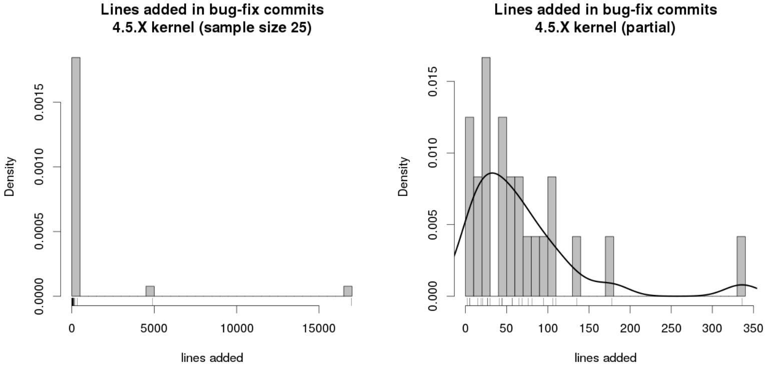
v4.5...v4.4.7 bug-commit age - manual analysis



v4.5...v4.4.7 lines added - manual



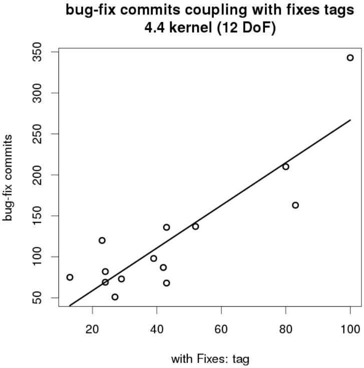
v4.5...v4.4.7 lines added - manual



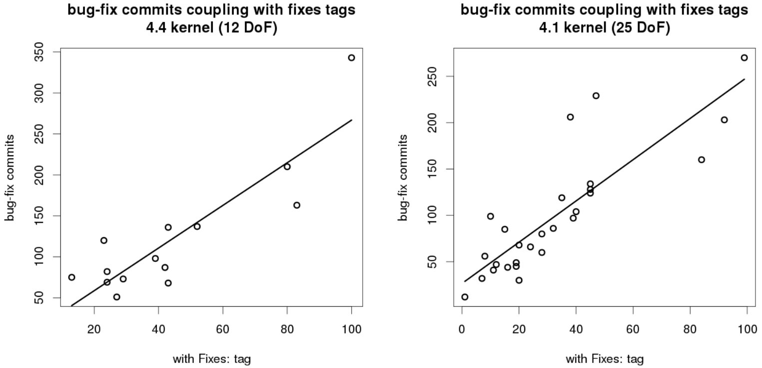
First automation steps

- Review of -stable patchlevel commits -> confirm they are bugs only
- Filter out those that contain a "Fixes: " tag and record the hash
- Extract all bug-fix commits
- Extract the bug-source commits via "Fixes: " tag
- Correlate buf-fix-commits with and without "Fixes: " tag over patchlevels -> verify that we are still looking at the same population
- Extract explanatory parameters of interst from git
- Note that this will not be able to fill in issues like severity or root-cause !
- Check tendencies and distributions against manually obtained results (graphically and statistically)

Asessing suitability of Fixes: tag



Asessing suitability of Fixes: tag



Note that the Fixes: tag is not yet that well established (more or this later)

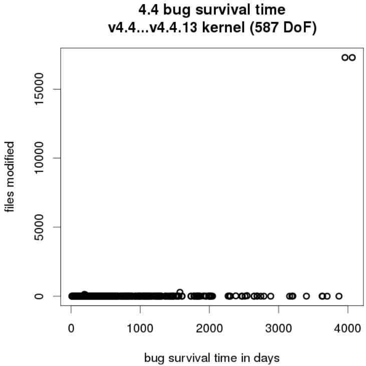
Analyze strength of coupling

A linear model could be a problem - the use of "Fixes:" is steadily increasing - but the increse within a single SUBLEVEL does not seem to be critical.

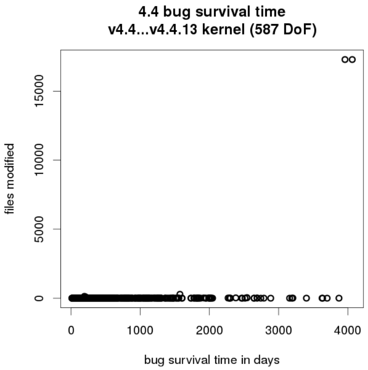
```
Call: lm(formula = bugs ~ fixes, data = d)

Residuals:
    Min      1Q  Median      3Q     Max
-59.627 -22.001  -4.902  16.287  76.149

Coefficients:
            Estimate Std. Error t value Pr(>|t|) Signif
(Intercept)   6.7077    21.1989   0.316    0.757
fixes         2.6014     0.4158   6.257 4.21e-05 0.001

Residual standard error: 38.92 on 12 degrees of freedom
Multiple R-squared:  0.7654,    Adjusted R-squared:  0.7458
F-statistic: 39.15 on 1 and 12 DF,  p-value: 4.21e-05
```

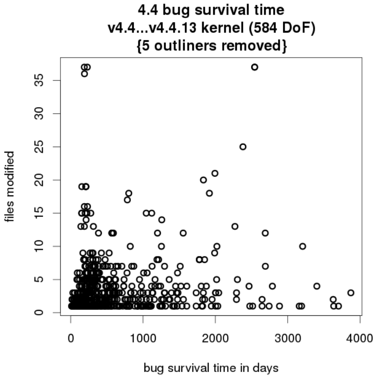
Conclusion: using the Fixes: tags should be a fairly robust representative of the full population (of bugs)

Bug introduction time (Fixes: tag)

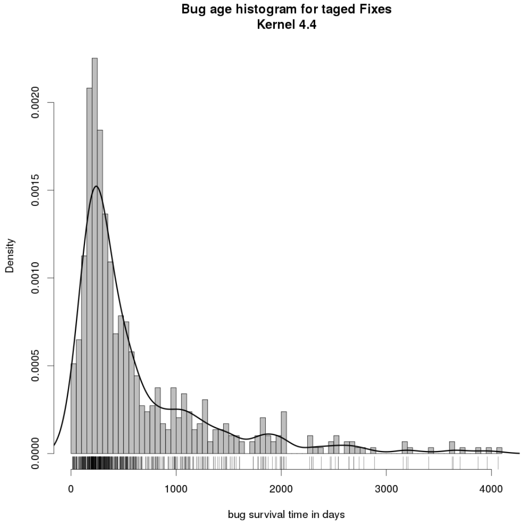


Bug introduction time (Fixes: tag)

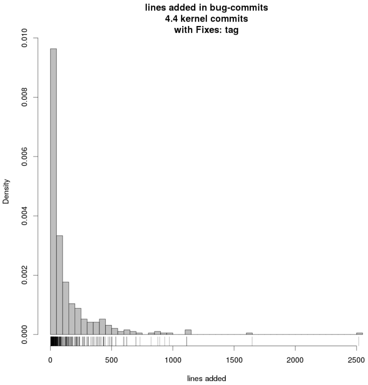




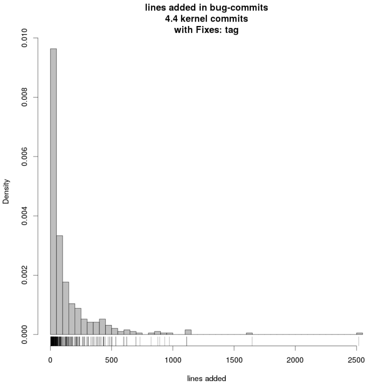
Distribution of bug survival in v4.4...v4.4.13

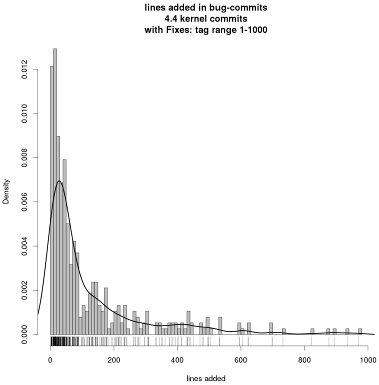


Distribution of lines-addded in bug-commits v4.4...v4.4.13



Distribution of lines-addded in bug-commits v4.4...v4.4.13





Development of bugs in v3.10


```
Call: lm(formula = bugs ~ lv, data = d)

Residuals:
    Min      1Q  Median      3Q     Max
-70.934 -19.782  -3.365  18.876 106.955

Coefficients:
            Estimate Std. Error t value Pr(>|t|) Signif
(Intercept)  76.5588     6.0540  12.646  < 2e-16 0.001
lv           -0.4530     0.1041  -4.353  3.3e-05 0.001

Residual standard error: 30.04 on 98 degrees of freedom
Multiple R-squared:  0.162,     Adjusted R-squared:  0.1535
F-statistic: 18.95 on 1 and 98 DF,  p-value: 3.302e-05
```

Development of bugs v3.2...

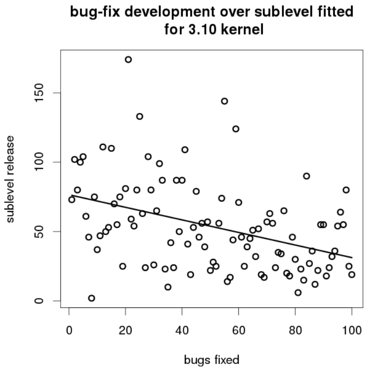
Nicholas Mc Guire
<safety@osadl.

bugs-fixed vs sublevel

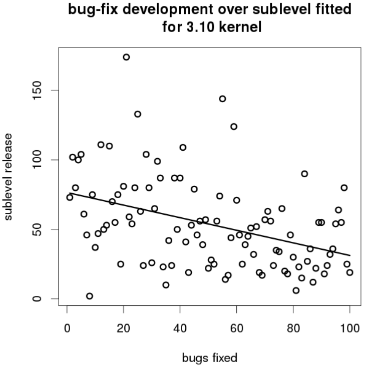
ver	intercept	slope	p-value	R^2	DoF
3.2	66.7359	0.5143	0.023	0.05185	79
3.4	53.38385	-0.02928	0.834	insig	110
3.10	76.5588	-0.4530	3.3e-05	0.1535	98
3.12	96.0521	-0.8508	0.00117	0.1284	70

Intercepts were very shaky with respect to significance, slope shows some level of significance and some explanatory power...

v3.10 bug development over sublevel



v3.10 bug development over sublevel



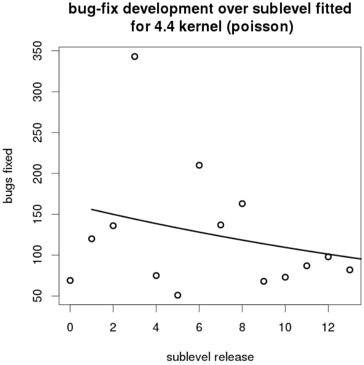
- So roughly at patchlevel 170 the expected bug-fix rate will hit 0 ... and then it will turn negative :)
- And incresed sublevels "explain" removed bugs ?

patchlevels - a convoluted parameter

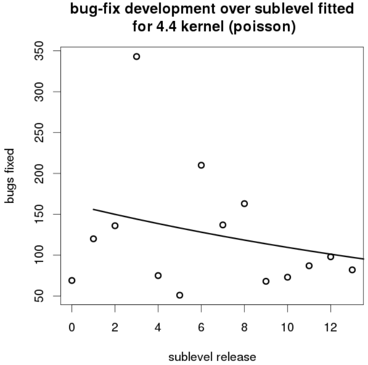
- Patchlevels as a configurable number of course have no impact on bug
- They represent the convolution of time dependent activities of the community, like:
  - Writing new bugs
  - Analysis and testing
  - Evaluation of indirect parameters e.g. performance
- So the model really should be from the exponential family as it is to be expected that the 0 bugs intercept is asymptotic at INF.

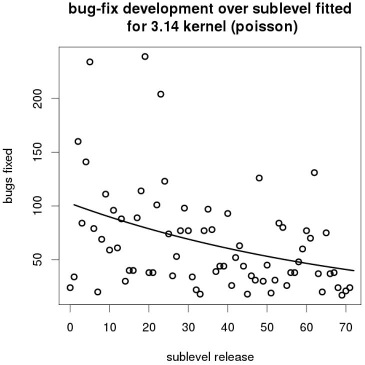
We actually were using a linear model for some time until we realized that the model made no sense - even if the linear model might be quite close to the tangent of the exponential model - its wrong.

Development of bugs over SUBLEVELS



Development of bugs over SUBLEVELS





- Iteration over sublevels -> confidence in the fitted lines
- Constance or decreasing bug-fix rates also would be a process stability indicator -> trending

"Improvement" Trend v3.2 - v4.4

Bugs-fixed over sublevel for -stable kernels simple poisson model

ver	intercept	slope	p-value	DoF	AIC
3.2	4.2233783	0.0059133	$< 2^{-16}$	79	2714.8
3.4	3.9778258	-0.0005657	0.164**NA**	110	4488
3.10	4.3841885	-0.0085419	$< 2^{-16}$	98	2147.1
3.12	4.7146752	-0.0014718	0.0413	58	1696.9
3.14	4.6159638	-0.0131122	$< 2^{-16}$	70	2124.8
3.18	4.671178	-0.006517	7.34^{-5}	34	1881.2
4.1	4.649701	-0.004211	0.09	25	1231.8
4.4	5.049331	-0.039307	7.69^{-11}	12	571.48

3.16 reappered as stable at 3.16.35 but is not considered here as there is no adeqate data for 3.16.8...3.16.35.

v4.4.13 -stable release - bug estimation

subsys	fix-commits 1643	with Fixes: 589 (35.8%)	rel Fixes:	i7 min config LoC
kernel	3.89%	4.75%	43.7%	60774 (37.9%)
mm	1.82%	2.17%	53.3%	33594 (47.4%)
block	0.36%	0.84%	83.3%!	15749 (63.6%)
fs	8.76%	4.92%*	20.1%*	54795 (8.6%)
net	9.31%	12.56%	48.3%	98617 (15.1%)
drivers	47.96%	49.23%	36.8%	124953 (1.4%)
include	6.87%	19.18%	28.3%*	141240 (31.4%)
arch/x86	4.50%	12.56%	33.7%	72942 (29.8%)

Reduced to selected subsystems, for a particular config -> a runable i7 config would have been potentially affected by roughly 9% of the bugs

Estimation of bugs

- Analyzing the stability/consistency of the process allows to use trending over sublevels
- We think that the data we have establishes such stability of the overall process (except for 3.16... which is a bit of a mess it seems)
- To put it together now in a fist step, we look at the exposure take the estimates (we trust due to trending) and estimate the bugs.
- If we had used 4.4.13, we would estimate the bugs that would affect us for **our given i7_min_config** at 9% of 103 bugs - or an expected value of aproximately $9 + / - 6$ bugs.

Limits of (current) data

- Some of the Assumptions
  - Normal distributions of bugs is assumed (questionable)
  - We do not yet have a model to estimate the hiden bugs (working on this)
  - The severity analysis is based on the specifics of SIL2LinuxMP and can not be generalized
  - Trigger conditions and thus probability of manifestation has not yet been analyzed
- Current state of data
  - Of the 1643 fixed bugs 8.9% would have affected our i7 config -> 145 bugs
  - Of 25 analyzed bugs 1 was potentially safety related
  - The 1 of 25 is unfortunately not sufficient to make an estimation of how many safety critical bugs would have been in v4.4
  - Bug occurance rates $\neq$ failure probabilities - these are systematic faults !

So there is quite some work still to do.

Exposure and Coupting

Nicholas Mc Guire
<safety@osadl.

Layering model of the kernel

- The kernel is a layered model (subsystems, internal interfaces, module interfaces, external interfaces)
- How is it layered and how independent are these layers ?
- How to quantify coupling ?

This is work in progress - but some first results are starting to look usable so we will review the process along with the applied methods and techniques.

Analysis of Layering

There are heuristics models available

- http://www.makelinux.net/kernel_map
- based on queries to kernel developers
- But - while quite usable - it is not config specific
- ...and we can not quantify effects

What we need is a mapping and a quantification of potential interaction.

Analysis of Layering - process oerview

- Configure the target kernel
- Run the minimize tool to extract the actual target code
- Classify all functions as
  - empty functions (typically config dependency handling)
  - wrapper functions that provide interfaces only (typically one or liners)
  - real functions (more or less everything else)
  - called functions
- Generate each classification for each subsystem and then set intersect them.

Coccinelle classifier

Sample data - i7 minimum config

```
w:dl_time_before:kernel/sched/cpudeadline.c:33
w:left_child:kernel/sched/cpudeadline.c:23
w:parent:kernel/sched/cpudeadline.c:18
w:right_child:kernel/sched/cpudeadline.c:28
f:cpudl_change_key:kernel/sched/cpudeadline.c:73
f:cpudl_cleanup:kernel/sched/cpudeadline.c:241
f:cpudl_exchange:kernel/sched/cpudeadline.c:38
f:cpudl_find:kernel/sched/cpudeadline.c:103
f:cpudl_heapify:kernel/sched/cpudeadline.c:48
f:cpudl_init:kernel/sched/cpudeadline.c:212
f:cpudl_set:kernel/sched/cpudeadline.c:136
c:WARN_ON:kernel/sched/cpudeadline.c:75
c:WARN_ON:kernel/sched/cpudeadline.c:121
c:WARN_ON:kernel/sched/cpudeadline.c:141
```

Sample data - sort and reference

```
...
workqueue.c calls  1  functions from  linux_sched.h
workqueue.c calls  1  functions from  sched_completion.c
workqueue.c calls  1  functions from  workqueue_internal.h
workqueue.c calls  1  wrappers from  linux_mutex.h
workqueue.c calls  1  wrappers from  linux_rcupdate.h
workqueue.c calls  1  wrappers from  linux_topology.h
workqueue.c calls  1  wrappers from  linux_wait.h
workqueue.c calls  1  wrappers from  rcu_tree.c
workqueue.c calls  1  wrappers from  sched_completion.c
workqueue.c calls  2  wrappers from  linux_sched.h
workqueue.c calls  3  functions from  linux_rcupdate.h
workqueue.c calls  3  functions from  locking_mutex.c
workqueue.c calls  3  wrappers from  linux_list.h
workqueue.c calls  4  functions from  sched_core.c
workqueue.c calls  5  wrappers from  linux_spinlock.h
workqueue.c calls  6  functions from  linux_list.h
...
```

intermediate all-to-all mapping of subsystems

```
arch calls  1     functions from       crypto
arch calls  4     functions from       init
arch calls  5     functions from       fs
arch calls  37    functions from       lib
arch calls  48    functions from       drivers
arch calls  50    functions from       mm
arch calls  117   functions from       kernel
arch calls  101   functions from       include
arch calls  1555  functions from       arch
...
security calls  4   functions from       arch
crypto calls    6   functions from       arch
block calls     17  functions from       arch
net calls       27  functions from       arch
init calls      21  functions from       arch
fs calls        44  functions from       arch
lib calls       46  functions from       arch
mm calls        65  functions from       arch
include calls   85  functions from       arch
drivers calls   89  functions from       arch
kernel calls    103 functions from       arch
...
```

preliminary layering results i7_min_config

```
init fs block crypto net security

mm lib drivers

kernel - arch

include
```

- This gives us some level of failure propagation estimation
- We also can estimate independence of fault response qualitatively
- The goal is a quantification - not there yet - e.g. based on interface with/usage.

Stability of Bug-locality (stable releases)

if bugs are well localized in general then there should be a strong correlation between number of bugs and the number of files being modified and the relation should be very close to 1. The assumption being that well localized bugs are "acting" on a ideally single local function through a well defined interface - so thie locality of bug does not limit the potential for the bugs severity or probability of being triggert in any way, but it allows a statement about:

- the stability of interfaces (bug-fix does not mandate wide-spread changes all over the place if the interfaces held)
- the bugs are assessible in the local scope - which is a key quality metric to allow for a impact analysis for a discovered bug in the first place given the complexity of the Linux kernel
- The constance (within the margin of the statistics

5-number overviews v3.18-v4.4


```
v3.18-stable (36 sample)        v4.1-stable  (27 sample)
      file              bugs          file             bugs
 Min.   :  2.00   Min.   :  3.00  Min.   : 13.0   Min.   : 12.00
 1st Qu.: 54.75   1st Qu.: 47.00  1st Qu.: 57.5   1st Qu.: 48.00
 Median : 97.00   Median : 78.00  Median :101.0   Median : 85.00
 Mean   :111.67   Mean   : 95.53  Mean   :111.0   Mean   : 99.04
 3rd Qu.:168.75   3rd Qu.:151.00  3rd Qu.:139.5   3rd Qu.:126.00
 Max.   :262.00   Max.   :231.00  Max.   :284.0   Max.   :270.00

v4.4-stable  (14 sample)
      file             bugs
 Min.   : 65.0   Min.   : 51.0
 1st Qu.: 88.0   1st Qu.: 73.5
 Median :110.0   Median : 92.5
 Mean   :134.1   Mean   :122.3
 3rd Qu.:149.8   3rd Qu.:136.8
 Max.   :348.0   Max.   :343.0
```

explanatory power and trend v3.2...v4.4

bugs-found vs fils-change (worst outliner removed)

patch level	intercept(0.1)	slope(0.001)	R^2 adj.	DoF	
3.2	1.15550	0.90490	0.9514	78	
3.4	-1.65521	0.93359	0.9602	109	
3.10	1.60677	0.86643	0.9237	97	
3.12	4.03145	0.87922	0.9341	57	
3.14	-3.19233	0.94505	0.9668	69	
3.16	NA	NA	NA	NA	
3.18	0.14564	0.90963	0.9709	32	
4.1	-3.70289	0.92558	0.9661	25	
4.4	-14.3038	1.0182	0.9809	12	

Alternative models

A mutivariat linear model could do better maybe...

```
Call: lm(formula = bugs ~ add + file, data = d)

Residuals:
    Min      1Q  Median      3Q     Max
-18.230  -6.656   2.449   7.091  22.248

Coefficients:
             Estimate Std. Error t value Pr(>|t|) Signif
(Intercept) -1.859695   4.033385  -0.461   0.6489
add          0.008890   0.003328   2.672   0.0133 0.05
file         0.809396   0.053090  15.246 7.62e-14 0.001

Residual standard error: 10.86 on 24 degrees of freedom
Multiple R-squared:  0.9749,    Adjusted R-squared:  0.9728
F-statistic: 465.6 on 2 and 24 DF,  p-value: < 2.2e-16
```

So lines added could have some say in explaining the bug-fix count (note that add and file are not independent though!)

Alternative model decisions

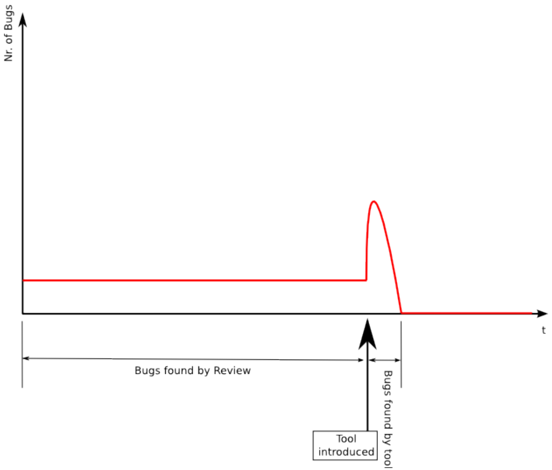
```
> anova(fm1,fm2,fm3)
Analysis of Variance Table

Model 1: bugs ~ file
Model 2: bugs ~ file + add
Model 3: bugs ~ file + add + del
  Res.Df    RSS Df Sum of Sq      F  Pr(>F)
1     25 3674.1
2     24 2831.9  1    842.16 6.9220 0.01494 *
3     23 2798.3  1     33.68 0.2768 0.60383
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' '
```

Starting at the bottom comparison and reading up, Model 2 is the most suitalbe "explaination" for bugs.

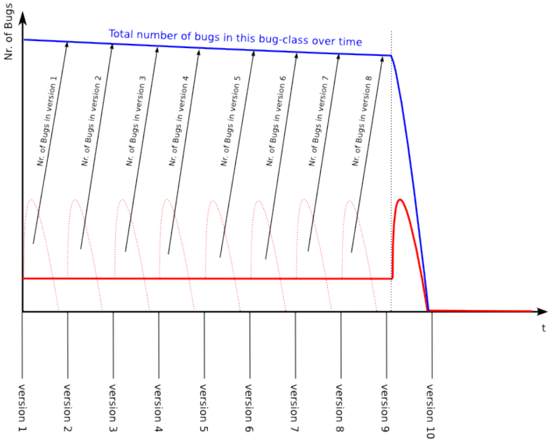
Estimating hidden bugs

- If the source is large enough bugs behave like large populations - statistically well behaved.
- A detected bug is a representative of a possible bug-class
- Based on a resonably large set of uncovered bugs we can backtrack in git and determine the rate of undisclosed bugs
- Together with trending we can model the hiden bugs
- The math is not yet sound - so this is still at a very speculative level.
- What we want to get from this is a quantitative estimate of the residual bugs in highly complex code like the linux kernel.

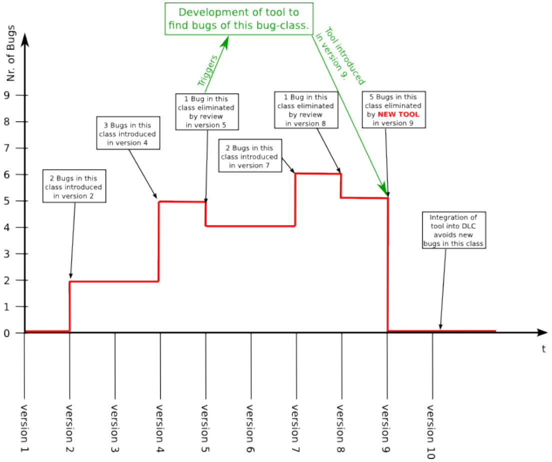
Elimination of detected bugs



Life-time and population estimate



Speculativ example



Discussion of Statistical Methods for SIL2LinuxMP

Nicholas Mc Guire
<safety@osadl.

Overall approach

Kernel DLC

Process Assessment

Estimating bugs

Exposure & Coupling

Handling severity

Conclusions

- If you want to utilize FLOSS -> fix the processes first
- IEC 61508 was not really conceived with selection as primary strategy in mind - but it **is** doable.
- IEC 61508 is robust enough to provide a solid foundation for formalizing element selection (Route 3_S) as primary strategy
- The process adjustments are in review by TueV Rheinland and look feasible.
- Process statistics are going to play a key role.
- Applying this to GNU/Linux RTOS will not be trivial - but looks doable

Thanks !

http://www.osadl.org/SIL2