Identifying Stable Kernels - top-down model

Nicholas Mc Guire <safety@osadl.org>

August 1, 2017

Open Source Automation Development Lab eG

Outline

- Context
- A few notes on statistics
- Data preperation
- The kernel as a whole
- Looking at subsystems
- Conclusions

Why GNU/Linux for Safety

- Satisfy Demands:
  - **multicore** there is no real alternative
  - **security** demands of C2C/C2X
  - **performance** demands in cognitive systems -> multi-core
  - functional requirements of e.g. autonomous systems (HAD)
- Satisfy Certification
  - standardization (notably requirements)
  - established concepts
  - breadth of deployment
  - development model (DLC data)

Using well-selected FLOSS for safety has clear technical advantages - developing the reusable generic arguments is a key element in SIL2LinuxMP.

Complex software and Safety

"*As far as practicable the design shall keep the safety-related part of the software simple*" [61508-3 Ed 7.4.2.6]

Complex software and Safety

”*As far as practicable the design shall keep the safety-related part of the software simple*” [61508-3 Ed 7.4.2.6]

”*This Standard does not discourage the use of software in safety-critical systems. When designed and implemented correctly, software is often the first, and sometimes the best, hazard detection and prevention mechanism in the system.*” [NASA NPR 8719.13B 1.2]

Complex software and Safety

"As far as practicable the design shall keep the safety-related part of the software simple" [61508-3 Ed 7.4.2.6]

"This Standard does not discourage the use of software in safety-critical systems. When designed and implemented correctly, software is often the first, and sometimes the best, hazard detection and prevention mechanism in the system." [NASA NPR 8719.13B 1.2]

61508 Ed 2 essentially is guidance on complexity management at all levels of the DLC

What changed in the safety busines ?

- Performance demands
  - UP -> SMP/MP
  - 32bit -> 64bit
  - Large caches
  - Large memory footprint
- Non-determinism: caches, BTB, dynamic wait-states/retries
- Power-management: frequency-scaling, variable latency
- Complex OS - migration, pinning, per-core ops, physical concurrency
- Security - all the fun you can have in life in one problem
- Automotive industry discovered functional safety !

Re-use not re-write

- Rewriting complex software is not the solution - the problem is **not** the code but requirements and design.
- Violates security best practices (use common core libraries, use high level languages)
- Significantly changing pre-existing elements invalidates field data
  - -> risk of breaking the process
  - -> risk of more latent bugs (code level)
- Rewriting Linux would be a 60B USB waste
- 3_S is not a fall back, it is the more adequate approach for complex systems if the elements selection is properly handled.

Level of complexity: Simple system calls

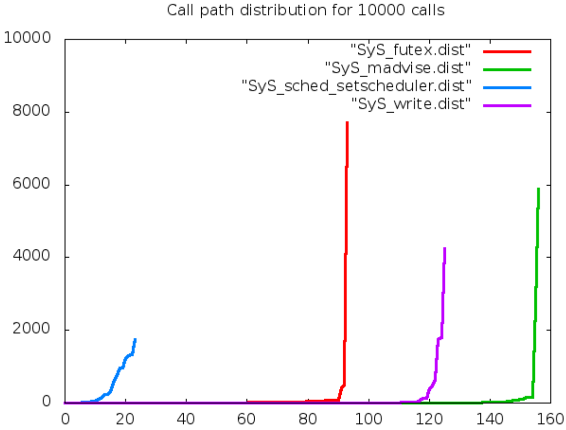
10000 runs - single call-tree

```
sys_arch_prctl              SyS_brk
sys_geteuid                 sys_getppid
SyS_getrlimit               sys_getuid
SyS_lseek                   SyS_read
SyS_rt_sigaction            SyS_rt_sigprocmask
SyS_sched_get_priority_max  SyS_sched_get_priority_min
SyS_set_robust_list
```

Calls are simple:

- => Code Review doable
- => Testing doable with small test-samples
- => Field data may be quite relevant

Mid complexity system calls



- => Code review effort excessive
- => Need significantly more than 10k samples

Statistics

Goal:

- Covering residual systematic faults
- Enabling architectural mitigations
- Managing/quantifying uncertainty -> recalls are costly !
- Address the limitations of highly-complex systems
- Provide key evidence for pre-existing elements
- Address busines risks

"Assessment of non-compliant development", that is the demonstration of the adequacy of an element "as-is" and the process behind it, that was **not** initially intended for high-assurance/safety-related systems.

Some Basics

- Software exercised without change should show an exponential decrease of faults found
- Its a count -> poisson distribution
- But its highly overdispersed -> switch to negativ-binomial distribution
- Only view data-points -> borrowing strength over kernel versions.

Clean approach:

The problem of valid/relevant metrics

- postulate your hypothesis
- design test metric
- sample -> statistic
- confirm/reject based on quantified criteria (e.g. α level)
- Not realistic unfortunately because we do not yet know what we are looking for in many cases and complex systems rarely have simple cause-consequence models that are relevant.

The "clean" solution is the reference - but we know we must divert from this.

Prediction on kernel - Data

What do we want to predict ?

Prediction on kernel - Data

What do we want to predict ? **residual bugs in the kernel**

Prediction on kernel - Data

What do we want to predict ? **residual bugs in the kernel**

- Ideal input: time of bug-discovery

Prediction on kernel - Data

What do we want to predict ? **residual bugs in the kernel**

- Ideal input: time of bug-discovery
- Real input:
  - time of bug-fix in -stable
  - -stable releases -> bug-fix only
  - Fixes: tag -> indication of bug source
  - Git data -> approximation of failure records
  - Uncertainty: trending over versions
  - Data-size issues: borrowing strenght.

Lets look at some examples

Dirty approach:

- hypothesis -> plausible assumptions
- analyze available indirect data: e.g. bugs -> bug-fixes
- analyze available metrics -> data-base
- approximate/biased sample -> assess assumptions
- for relevant effects -> clean approach based on SRS
- verify process/methods/metrics by independent CA
- continuous monitoring to ensure validity of claims

Dirty approach:

- hypothesis -> plausible assumptions
- analyze available indirect data: e.g. bugs -> bug-fixes
- analyze available metrics -> data-base
- approximate/biased sample -> assess assumptions
- for relevant effects -> clean approach based on SRS
- verify process/methods/metrics by independent CA
- continuous monitoring to ensure validity of claims
    - Risk: you might have to pull the plug because you violate your metrics

Dirty approach:

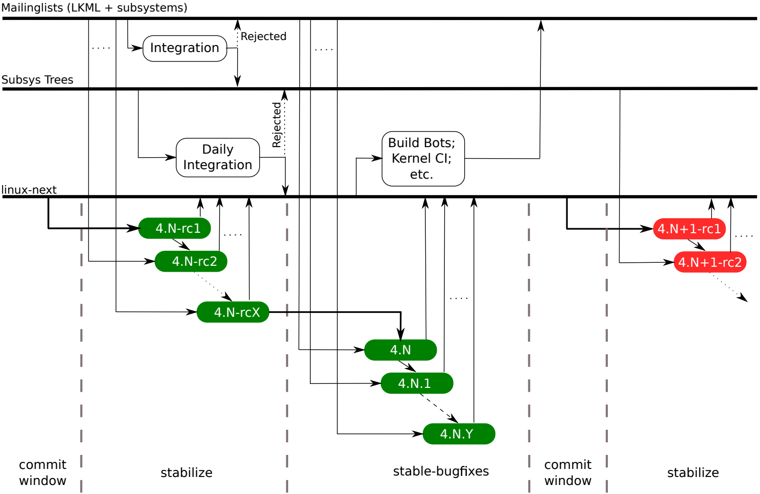
- hypothesis -> plausible assumptions
- analyze available indirect data: e.g. bugs -> bug-fixes
- analyze available metrics -> data-base
- approximate/biased sample -> assess assumptions
- for relevant effects -> clean approach based on SRS
- verify process/methods/metrics by independent CA
- continuous monitoring to ensure validity of claims
  - Risk: you might have to pull the plug because you violate your metrics ... no need to wait until you kill someone

Limitations

- Independence violations: clustered findings/fixes
- Temporal uncertainties: time is **NOT** continuous
- zero-truncation and zero-inflation
- evolution of underlying process
- indirect data -> increased error terms
- small data sets -> borrowing strength
- Nonhomogenous population -> heterosedasticity issues
- Data uncertainty: timestamp 1.1.1970, 14.2.2038 ?
- ...

Real data is not clean -> RIDM: manage/quantify uncertainty

Why we can do this - Linux DLC Stability

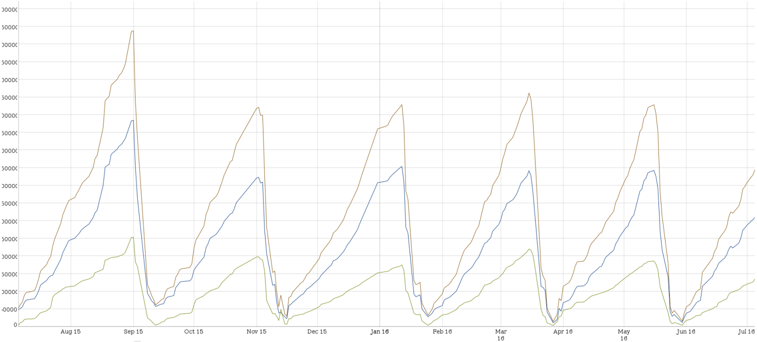


relating DLC to changes

4.4 life-cycle sequence

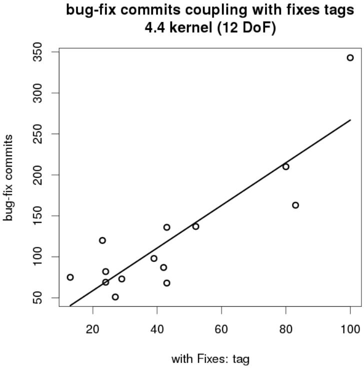
	files	lines		commits	lines added
	changed	add	del		per commit
rc1	9981	697599	460116	12226	57.0
rc2	334	4149	5497	386	10.7
rc3	245	3346	2342	277	12.0
rc4	363	4968	1672	331	15.0
rc5	256	1766	1304	260	6.7
rc6	263	2272	1236	309	7.3
rc7	91	977	429	109	8.9
rc8	73	709	448	82	8.6
4.4	88	518	280	102	5.0
4.4.1	80	644	280	120	5.3
4.4.2	112	1680	568	136	13.3
4.4.3	140	1307	585	343	3.8

DLC process stability over Versions

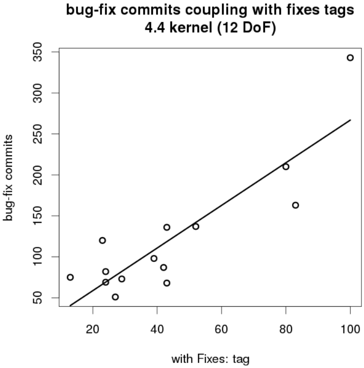


Source: http://neuling.org/linux-next

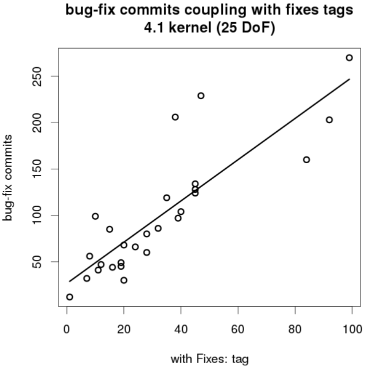
Asessing suitability of Fixes: tag



Asessing suitability of Fixes: tag



bug-fix commits coupling with fixes tags
4.4 kernel (12 DoF)



bug-fix commits coupling with fixes tags
4.1 kernel (25 DoF)

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

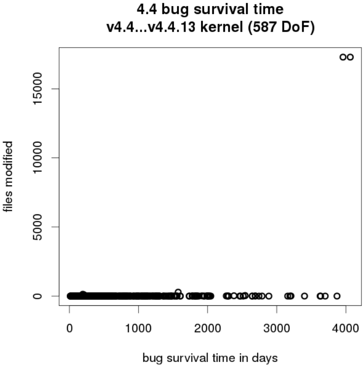
Analyze strength of coupling

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.

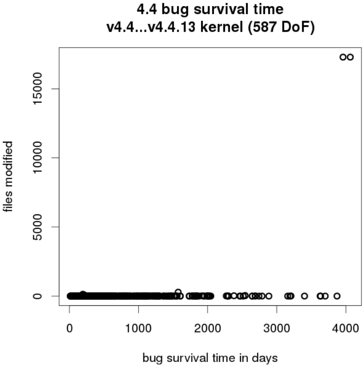
```
Call: lm(formula = bugs ~ fixes, data = d)

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
```

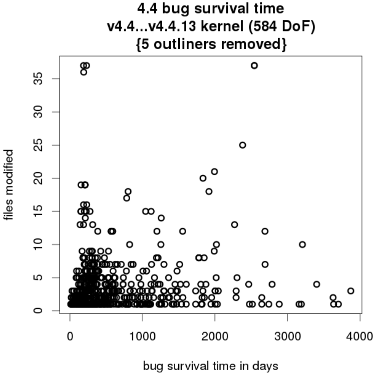
Conclusion: using the Fixes: tags should be a fairly robust representative of the full population (of bugs)

Bug introduction time (Fixes: tag)

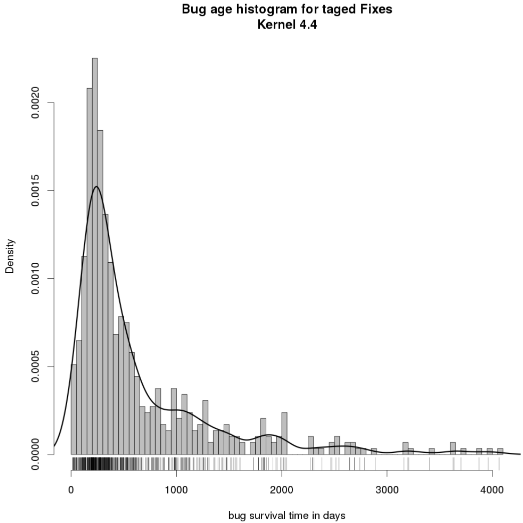


Bug introduction time (Fixes: tag)

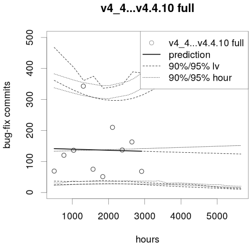


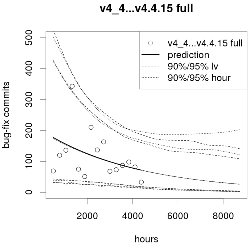


Distribution of bug survival in v4.4...v4.4.13

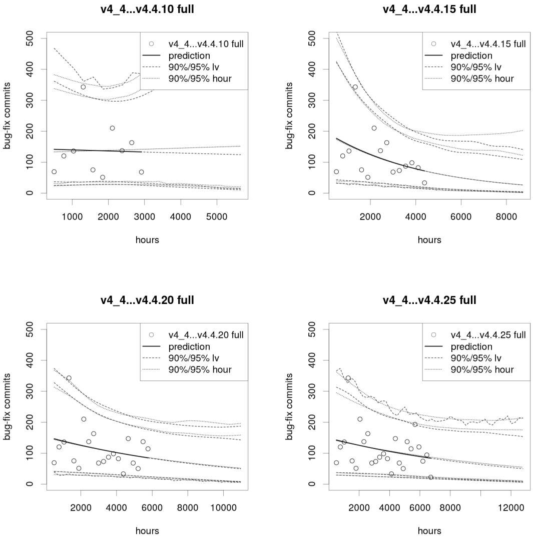


Prediction on kernel v4.4

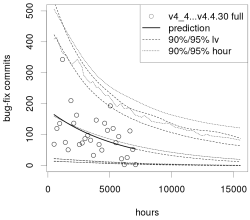


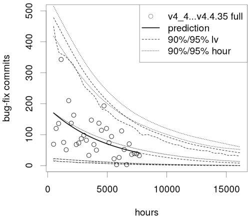


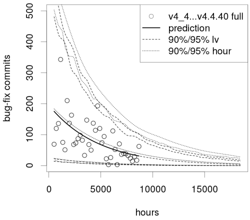
Prediction on kernel v4.4

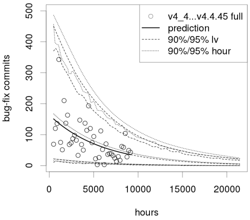


Prediction on kernel v4.4 cont

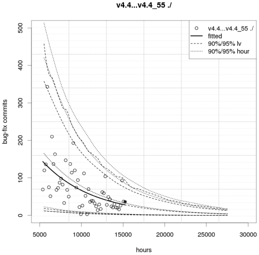

v4_4...v4.4.30 full

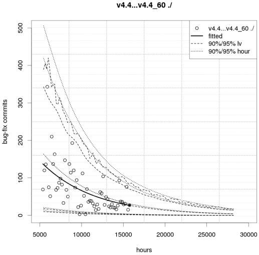

v4_4...v4.4.35 full

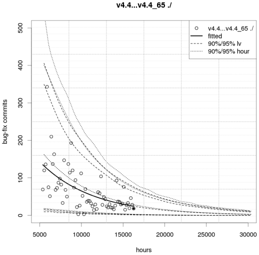

v4_4...v4.4.40 full

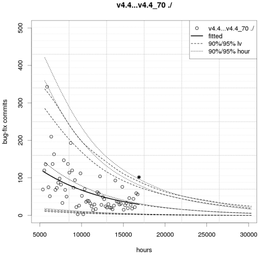

v4_4...v4.4.45 full

Prediction on kernel v4.4 cont

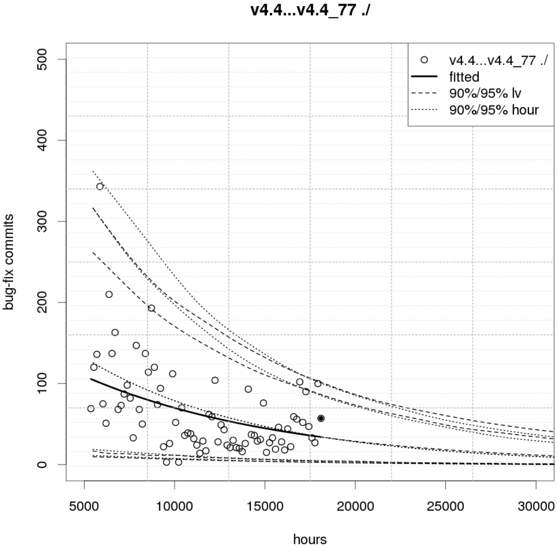

v4.4...v4.4_55 ./


v4.4...v4.4_60 ./

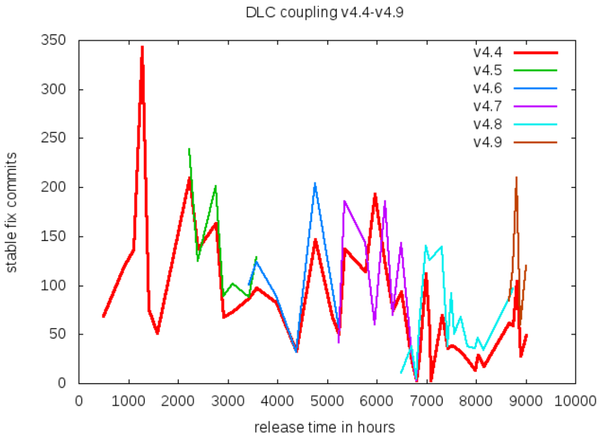

v4.4...v4.4_65 ./


v4.4...v4.4_70 ./

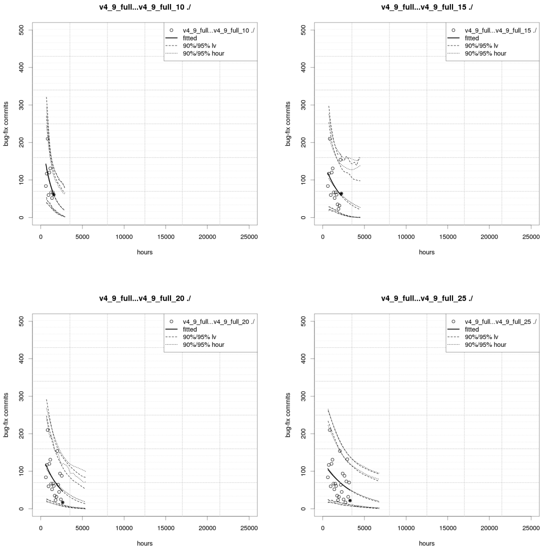
Prediction on kernel v4.4 current



LTS non-LTS coupling



Prediction on kernel v4.9 cont



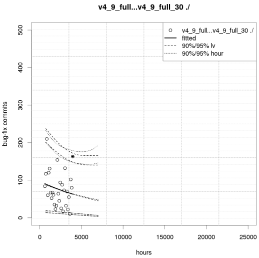
v4_9_full...v4_9_full_10 ./

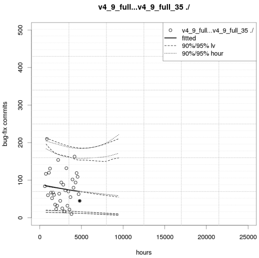
v4_9_full...v4_9_full_15 ./

v4_9_full...v4_9_full_20 ./

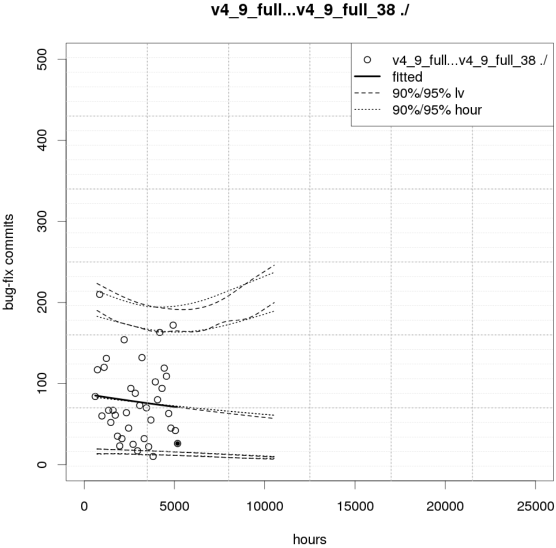
v4_9_full...v4_9_full_25 ./

Prediction on kernel v4.9 cont

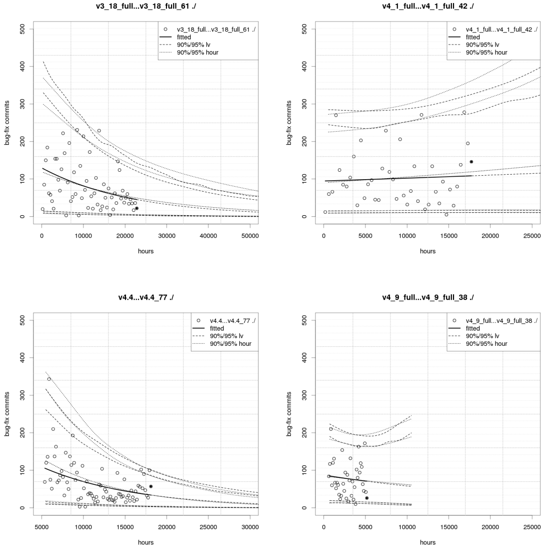




Prediction on kernel v4.9 current



...compare v3.18 ... v4.9



complete kernel - estimated residual fixes

ver	bugs/lv	bugs/hour	time-interval
3.18	1474	2011	3.16a
4.1	5154	6050	2.62a
4.4	1312	1468	1.81a
4.9	2496	2606	0.56a

Note these are simulation rsult snapshots (no error estimation yet)
We only can predict fixes (because that is what we measured) and we can only predict about as far into the future as we have robust data from the past - so widely variing prediction intervals.

estimated residual fixes densities

ver	files	fix/file	LoC	fix/LoC	density
3.18	46269	0.031 - 0.045	12711638	1.1^{-4} — 1.6^{-4}	1/7400
4.1	47661	0.108 - 0.127	13023108	4.0^{-4} — 4.6^{-4}	1/2300
4.4	50277	0.027 - 0.031	14023274	9.3^{-5} — 1.0^{-4}	1/10100
4.9	54069	0.046 - 0.048	14919875	1.6^{-4} — 1.7^{-4}	1/5600

Note these are simulation rsult snapshots (no error estimation yet) and not normalized ! - sorry did not realize that until I put the slides together

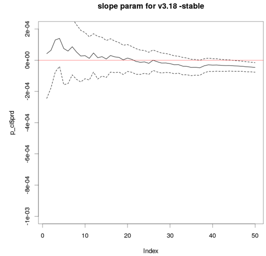
kernel DLC Trend v3.2 - v4.4(9)

Trending bugs-fixed over sublevel for -stable kernels

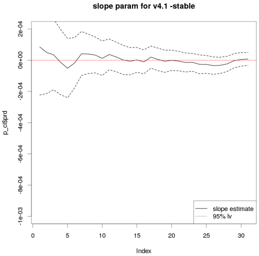
ver	slope	p-value	DoF	AIC
3.2	0.005910	0.06	83	904.76
3.4	0.0001224	**0.958**	112	1117
3.10	-0.004909	0.0166	103	1006.9
3.12	-0.002298	0.451	69	750.78
3.14	-0.014073	$6.26e-07$	78	770.44
3.18	$-4.615e-05$	0.00161	60	647.83
4.1	$7.656e-06$	0.724	41	470.13
4.4	-0.014925	$1.15e-05$	76	767.56
4.9	-0.005012	**0.585**	37	397.98

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

Parameter stability of prediction

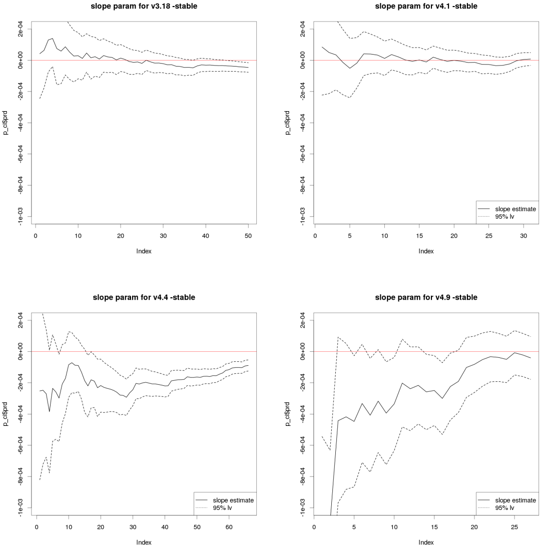


slope param for v3.18 -stable



slope param for v4.1 -stable

Parameter stability of prediction



Stratifying

- identify high-risk subsystems over trending
- identify high-level dependencies/layering
- subset assessment based on specific configuration
- per patch statistics for subset/subsystem
- Testing:
  - temporal capabilities/interference
  - extreme-value distributions for conservative prediction
  - temporal monitoring at runtime -> SAC

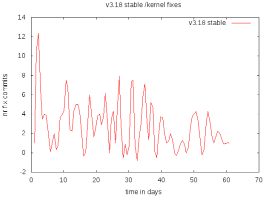
Its not really clean any more as it is based on snooping the data technically a no-go: Mitigation -> assessment by independent authority in our case TueV Rheinland will review the statistical models and if they are valid.

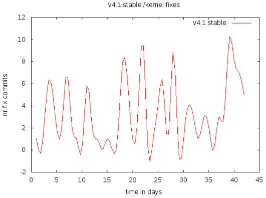
Subsystem Risk example

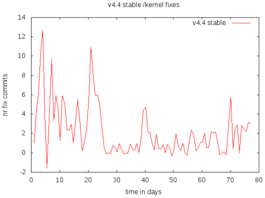
- /kernel vs full kernel
- /include vs /fs

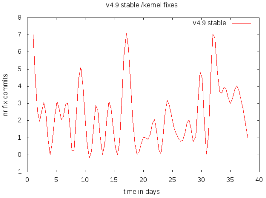
- How fast does a subsystem stabilize -> identify risk
- How much does the full kernel "say" about a subsystem -> borrowing strength

/kernel 3.18, 4,1, 4.4, 4.9 raw data

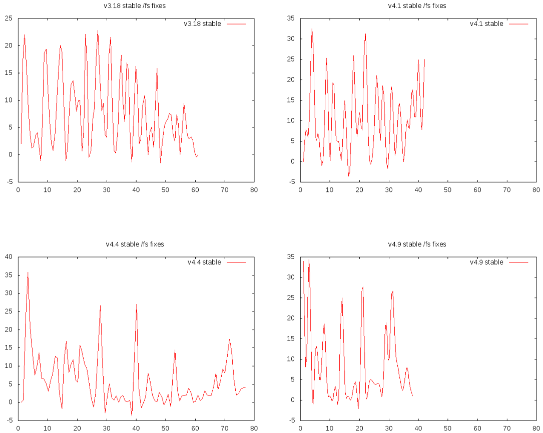




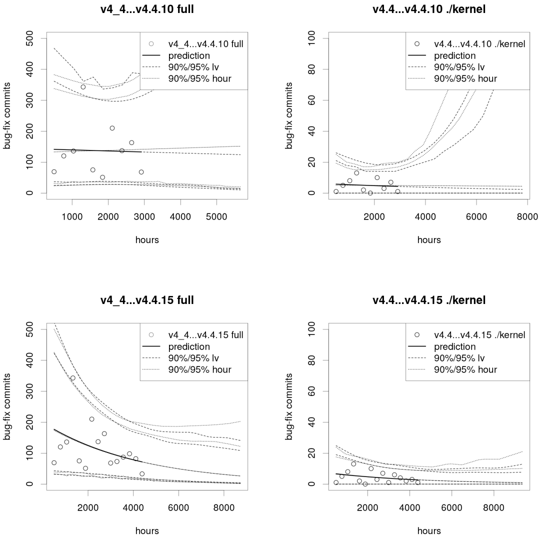




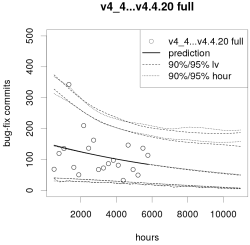
/fs 3.18, 4,1, 4.4, 4.9 raw data

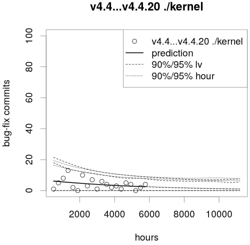


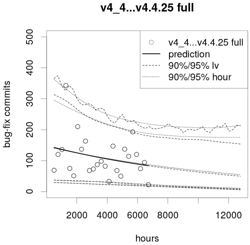
full kernel vs /kernel 1 modeling

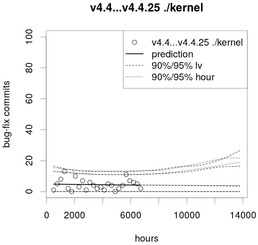


full kernel vs /kernel cont.

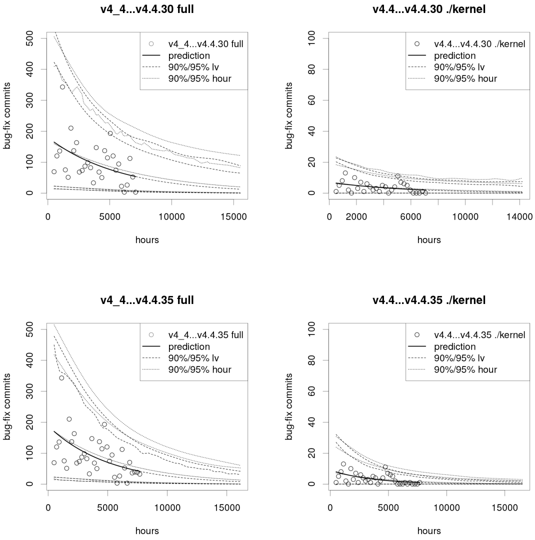




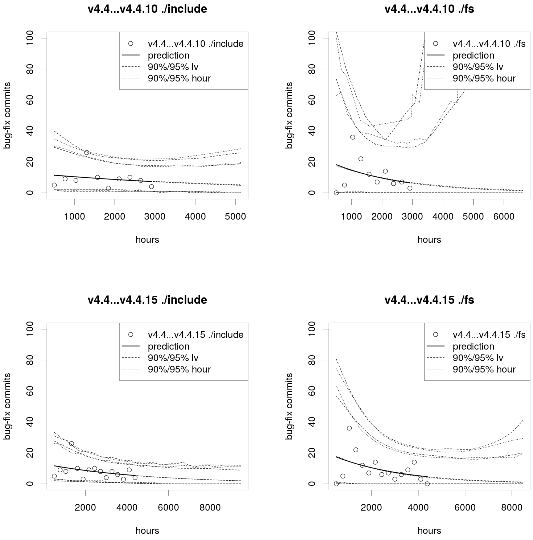




full kernel vs /kernel cont.



/include vs /fs 1 modeling



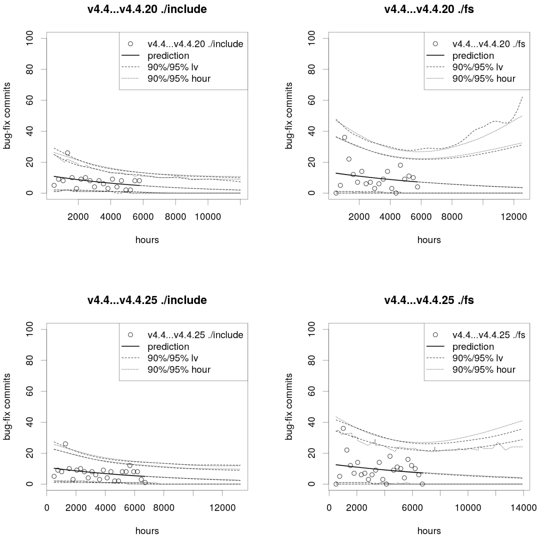
v4.4...v4.4.10 ./include

v4.4...v4.4.10 ./fs

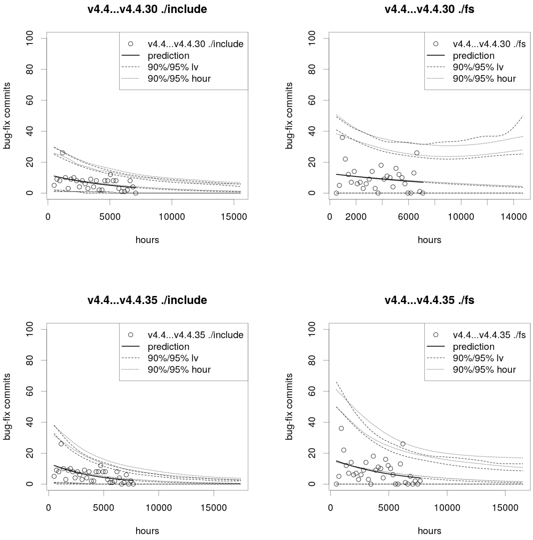
v4.4...v4.4.15 ./include

v4.4...v4.4.15 ./fs

/include vs /fs cont.



/include vs /fs cont.



v4.4...v4.4.30 ./include

v4.4...v4.4.30 ./fs

v4.4...v4.4.35 ./include

v4.4...v4.4.35 ./fs

Conclusion

Minimize the config i7_min_config_4.4.42 -stable

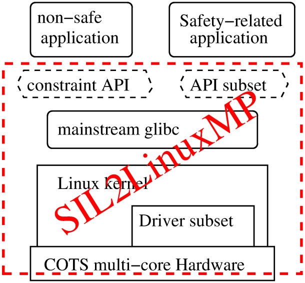
Subsys	files	blank	comment	code	full	%used
arch	480	16725	20728	76208	2040204	3.74
block	52	3973	5619	16614	24753	67.12
crypto	32	1950	1618	8626	76265	11.31
drivers	357	32181	53023	128168	8587655	1.49
fs	138	14157	23640	80227	827737	9.69
include	1196	35986	64271	163952	449811	36.45
init	8	354	391	1846	2712	68.07
kernel	140	15662	28181	64968	161178	40.31
lib	97	2932	6816	16522	81891	20.18
mm	55	7521	15428	34409	70830	48.58
net	151	21472	17714	103505	650973	15.90
security	3	230	612	1127	50929	2.21

Conclusions

- Relatively simple models allow identifying risks
- Trends of models **and** parameters are needed
- Subsysem models allow identifying less stable subsystems
- Trending over development allows detecting things going bad
- Comparison of kernel versions by objective data helps select the most reliable kernel - it is **not** though a single criteria that can give you an answer
- Minimizing your kernel is one simple elimination of residual bugs
- You need something stable ?: v3.18 or v4.4 would be my best guess at this point.

Statistic models may be a suitable extension for selecting kernels and other FLOSS elements to minimize risk. And cleary - **The latest need not be the greatest**

The Goal



non–safe application

Safety–related application

constraint API

API subset

mainstream glibc

Linux kernel

Driver subset

COTS multi–core Hardware

SIL2LinuxMP

Thanks !

Open Source Automation Development Lab eG