GNU/Linux Qualification - Kernel DLC Metrics

Nicholas Mc Guire <safety@osadl.org>

February 3, 2017

Outline

- Context
- Qualification
  - Identifying issues
  - Mitigation
  - Prediction
- Conclusions

SIL2LinuxMP context

- Assessment of non-compliant development
- Claim: propertiee are comparable to compliant development
- Argument: it is a manged process
- Evidence:
  - Basis: treat (Design—Implement—Integrate) as blackbox and see how many fault manage to get through all of the checks.
  - Probability: estimate how many faults will be found -> residual faults
  - Severity: assess the severity of findings by analyzing a sufficiently large **random** sample
  - $Risk = Probabilty * Severity$

Even though this seems to be quantitative - read it as a qualitative statement of "as good as a compliant development" (or maybe not...)

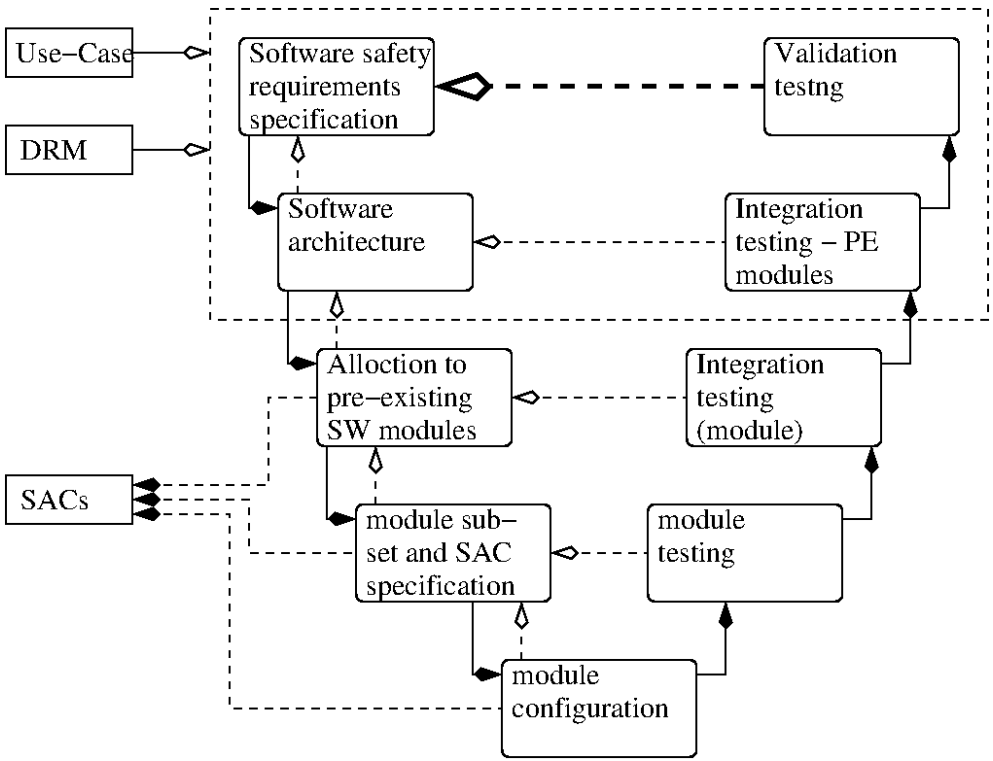
Systematic Faults

Software faults are (generally) considered systematic faults - if you present the input that triggers the fault it will **always** trigger.
Thus systematic software faults:

- Have **no failure rate** at code level
- Are mitigated by processes executed by humans
- Have a failure rate at the human/process level
  - Requirements
  - Design
  - Implementation
  - Test and integration
  - Deployment and maintenance

We are interested in assessing the process level "failure rate" to infere the expected probability of a yet undiscovered systematic fault being present.

SIL2LinuxMP DLC/SLC overall flow



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

The top of the V-model is more or less unchanged - the bottom is **select and constrain** replacing design-implement-integrate at the software modul level.

Linux kernel Procedures

- CodingStyle - simple and relatively short (40+ rules)
- checkpatch.pl - exhaustive and fussy (400+ rules)
- Amendment by tooling (sparse/coccinelle/checkpatch –strict) to cover some aspects that are not sufficiently addressable by coding style
- Amendment by procedures (SubmittingPatches,SubmitChecklist)
- Patch review procedure
- Multi-layer integration process
- Systematic compile/boot testing (build-bots/kernelCI)

So how good do we do in the kernel ?

Following rules ?

The distribution of fixes tags hash length for v4.4...v4.4.13 for all those who love statistical evidence 17.6% non-conformance ...bad ?

```
count  hash-len
    7  xxxxxxx
   11  xxxxxxxx
    8  xxxxxxxxx
   14  xxxxxxxxxx
    6  xxxxxxxxxxx
  484  xxxxxxxxxxxx    <--- 12 the "proper" value
   31  xxxxxxxxxxxxx
    4  xxxxxxxxxxxxxx
    4  xxxxxxxxxxxxxxx
    5  xxxxxxxxxxxxxxxx
    1  xxxxxxxxxxxxxxxxxxxx
   19  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

reasonable conditions

drivers/media/dvb-frontends/dib7000m.c:926 bad conditional

```
/* P_dintl_native, P_dintlv_inv,
   P_hrch, P_code_rate, P_select_hp */
value = 0;
if (1 != 0)
        value |= (1 << 6);
if (ch->hierarchy == 1)
        value |= (1 << 4);
if (1 == 1)
        value |= 1;
switch ((ch->hierarchy == 0 || 1 == 1) ?
        ch->code_rate_HP : ch->code_rate_LP) {
```

...and reasonable control flow

drivers/staging/rtl8723au/hal/rtl8723a_bt-coexist.c:7264 else duplicates if

```
...
} else if (maxInterval == 2) {
        btdm_2AntPsTdma(padapter, true, 15);
        pBtdm8723->psTdmaDuAdjType = 15;
} else if (maxInterval == 3) {
        btdm_2AntPsTdma(padapter, true, 15);
        pBtdm8723->psTdmaDuAdjType = 15;
} else {
        btdm_2AntPsTdma(padapter, true, 15);
        pBtdm8723->psTdmaDuAdjType = 15;
}
```

...no conditions with side-effects

drivers/ide/cmd640.c:680 redundant logic expression with side-effect

```
if (inb(0xCF8) == 0x00 && inb(0xCF8) == 0x00) {
     spin_unlock_irqrestore(&cmd640_lock, flags);
     return 1;
}
```

This has been in here since kernel 2.3.X (pre-dates git) The earlier 2.2.X kernels do not have this construct
How did this get into the kernel ?

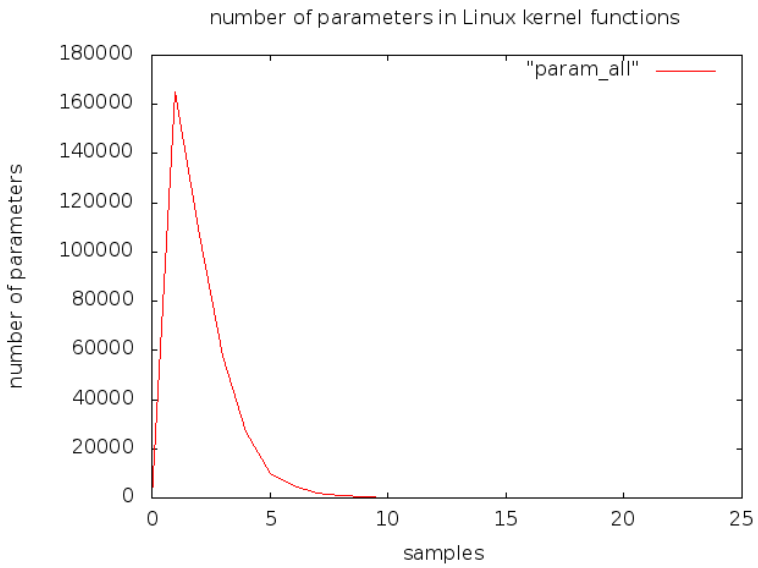
..and reasonable number of parameters

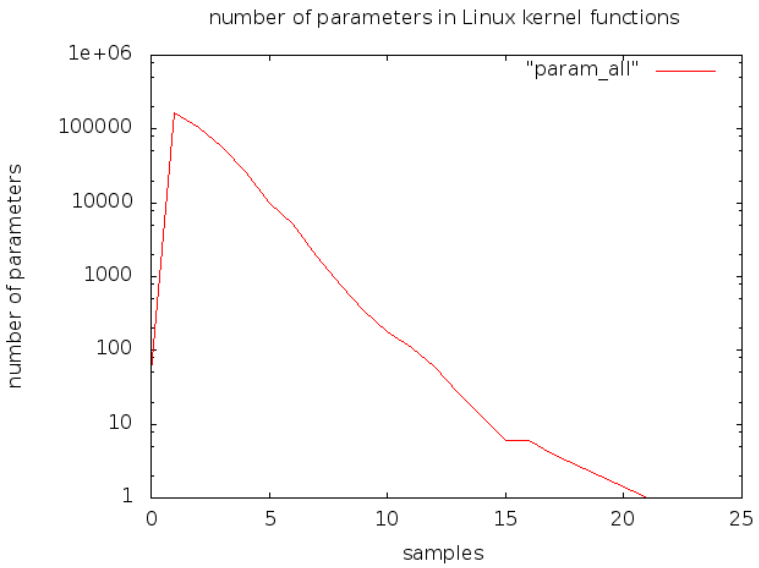
fs/ceph/caps.c:send_cap_msg,line 968 out of control parameter list

```
static int send_cap_msg(struct ceph_mds_session *session,
              u64 ino, u64 cid, int op,
              int caps, int wanted, int dirty,
              u32 seq, u64 flush_tid, u32 issue_seq, u32 mseq,
              u64 size, u64 max_size,
              struct timespec *mtime, struct timespec *atime,
              u64 time_warp_seq,
              kuid_t uid, kgid_t gid, umode_t mode,
              u64 xattr_version,
              struct ceph_buffer *xattrs_buf,
              u64 follows, bool inline_data)
{
```

Plain ugly - no excuse for this one - simply exclude ceph from the list of suitable fs.

Linux total parameter distribution





There is a few hundred functions that are over the reasonable limit of 7-8 parameters.

identifying problem cases

In our selected minimum config there are two "bad" functions - both are in lockdep:

```
<function(name='__lock_acquire',
source_file='kernel/locking/lockdep.c',
line='3068',
column='12',
parameter_number='9')>
<function(name='print_bad_irq_dependency',
source_file='kernel/locking/lockdep.c',
line='1492',
column='1',
parameter_number='10')>
```

Type consistency - system components

Component	Nr Functions	Inconsistent	%
kernel	374600	10727	2.85
glibc	9184	268	2.92
busybox	3645	43	1.18

versions: kernel 4.1-rc2, glibc-2.9, busybox-1.2.2.1

Type consistency - kernel core

	kern	mm	ipc	init	net	lib	total	%
wrong	1	1	0	0	1	1	4	0.5
sign	**97**	**65**	**4**	**1**	**218**	**21**	**406**	**47.4**
down sized	**4**	**5**	0	0	**21**	**5**	**35**	**4.0**
up sized	66	34	8	0	123	3	234	27.3
declar ation	8	0	0	0	15	2	25	2.9
false pos	31	17	4	0	89	12	153	17.9
	207	122	16	1	467	44	857	

API compliance - completion

semantic patch	findings	files	confirmed
duplicate_init_completion.cocci	2	2	2
check_for_signal_ignored.cocci	6	4	6
false_declare_completion.cocci	6	5	6
false_init_compltion.cocci	9	6	9
check_unhandled_return.cocci	10	8	4
check_for_negativ_ret.cocci	11	9	3
check_for_return_unused.cocci	62	42	2
check_for_signed_return.cocci	**126**	81	36
check_wrong_context2.cocci	0 (!)	0	-

Root-cause ?: The completion API was not documented

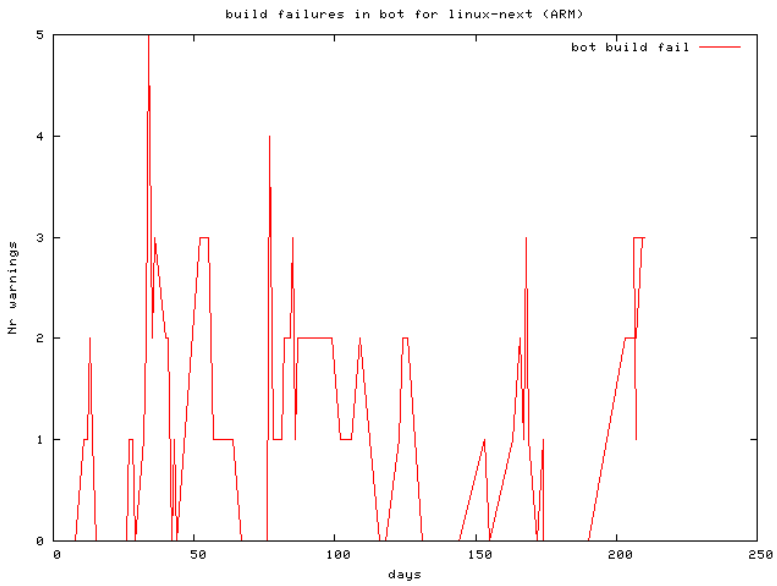
API compliance - useleep_range

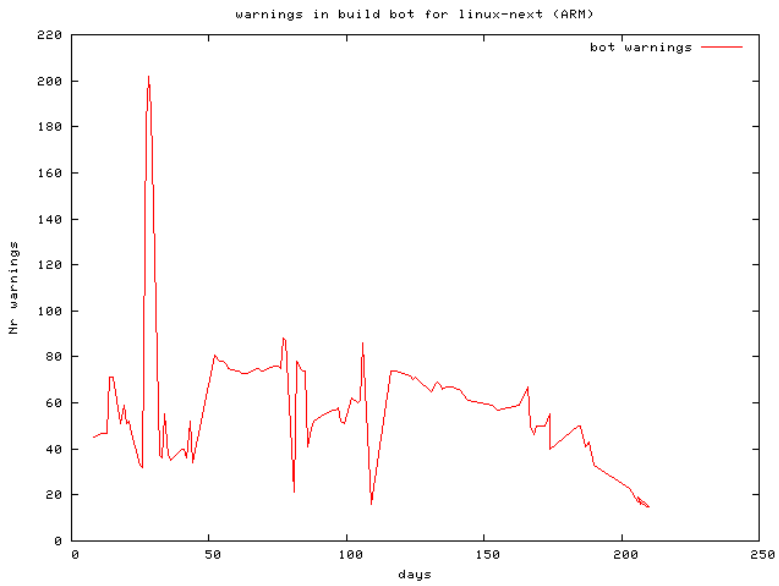
usleep_range(min,max) in linux-stable 4.9.0: 1648 calls total

Calls	Rel.	Issue	%	%
1488		pass numeric values only		90.29
	27	min below 10us	1.81	
	40	min above 10ms	2.68	
		numeric min out of spec		**4.50**
76		preprocessor constants		4.61
	1	min below 10us	1.31	
	8	min above 10ms	10.52	
		preprocess min out of spec		**11.84**
85		expressions		5.15
	1	min below 10us	1.50**x**	
	6(2)	min above 10ms	7.50**x**	
		expression min out of spec		**9.0**

Root-cause: quirky behavior - the timer is set at max not min

Build bot failures/warnings (ARM)





- Trending of linux-next ("input" to linux-stable)
- This covers 4.0,4.1,4,2,4.3 -rc (release candidates)
- Source: Build bot for Mark Brown <broonie@kernel.org>

Record and respond to findings

- The code development largely looks stable and can be mapped to SIL2 requirements
- There are some findings that need to be addressed
  - Most can be handled by proper selection
  - Some - notably types - need to be addressed by analysis and cleanup
- There is quite some work to do - no desaster yet
- The kernel as a whole has some QA issues that need to be communicated to the kernel community - and where possible addressed by automated methods.

SIL2LinuxMP will not solve all (not even find all) kernel problems - but we do think we can **find** -> **analyze** -> **fix** issues for the SIL2LinuxMP core and while at it, contribute to improving the general kernel QA.

Handling of "bad"-code

Can we handle this ?

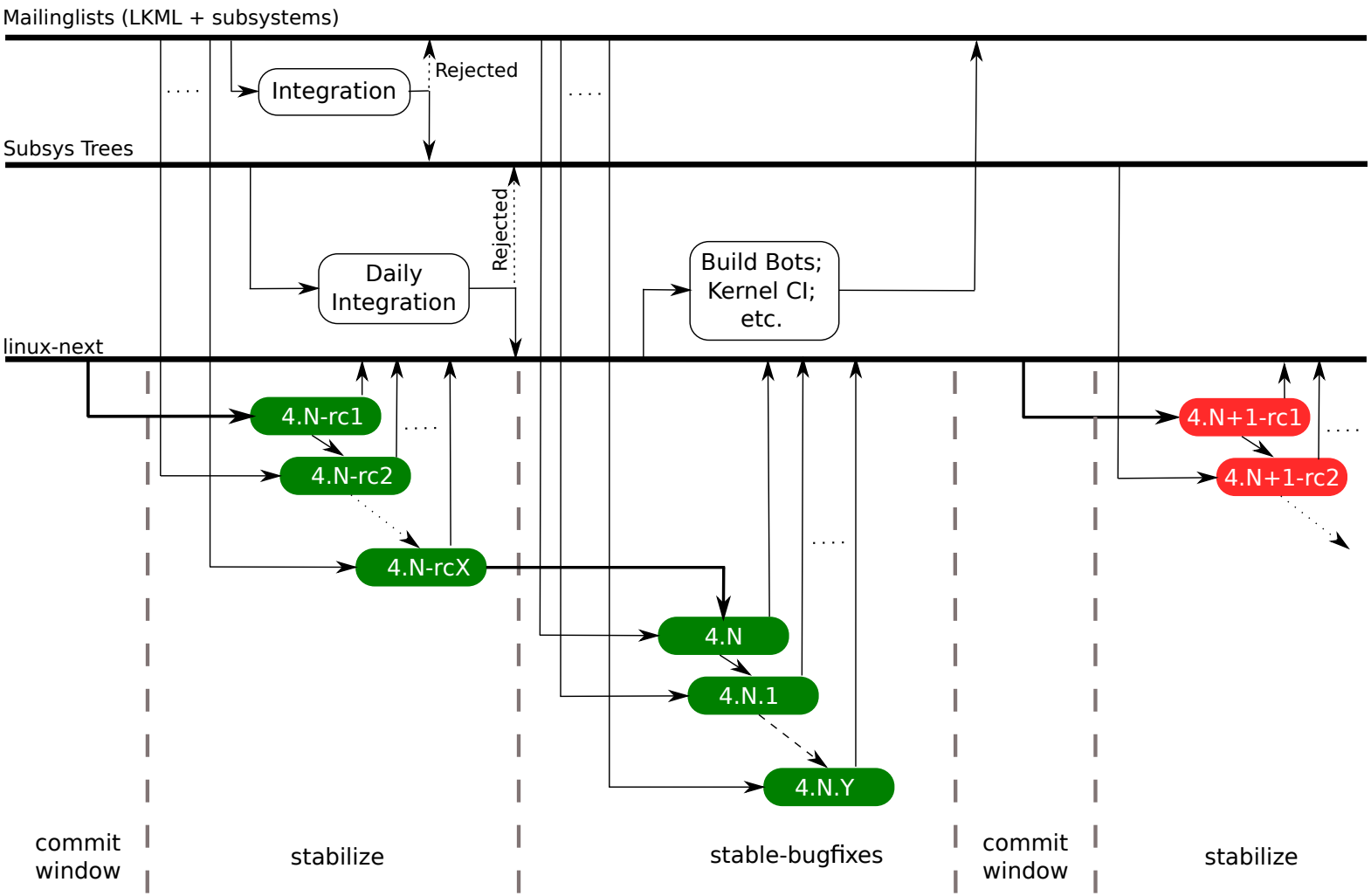
- Careful selection - review based configuration.
- Tools - automate it - formal methods.
- Fix those issues in the core code SIL2LinuxMP needs (aprox. 1k patches)
- Build up interface to the community - "fix once" is the goal
- Push the tools out to the developers (once they are clean)
- Build awareness in the community - notably of types

Known problems can be addressed - open-source/open-processes and a responsive **stable** community is why we **can** address these issues in GNU/Linux

Predicting residual bugs - top down model

- evolution over releases - big picture
  - Trending over PATCHLEVELs
  - Trending within PATCHLEVEL
  - Assessment of process stability
  - Assessment of data uncertainty
- trees: -stable and -next
  - Develop model from literature
  - Assess model against actual data
  - Mitigate findings in data/model
  - Automation issues (not quite done yet)
- Extracting a baseline/core -> allnoconfig
- Extracting a minimal config for target system

Linux DLC Qualities

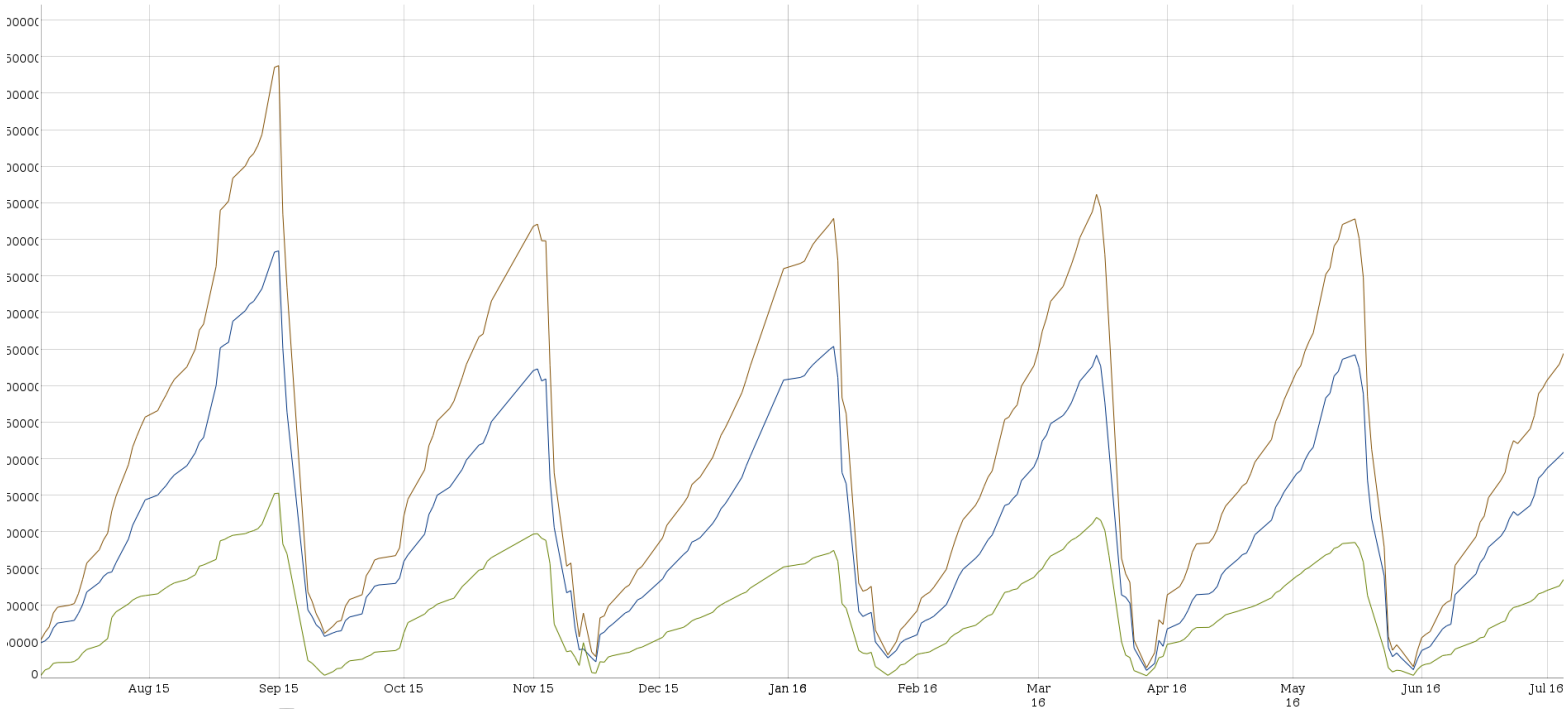


relating DLC to changes

4.4 life-cycle sequence

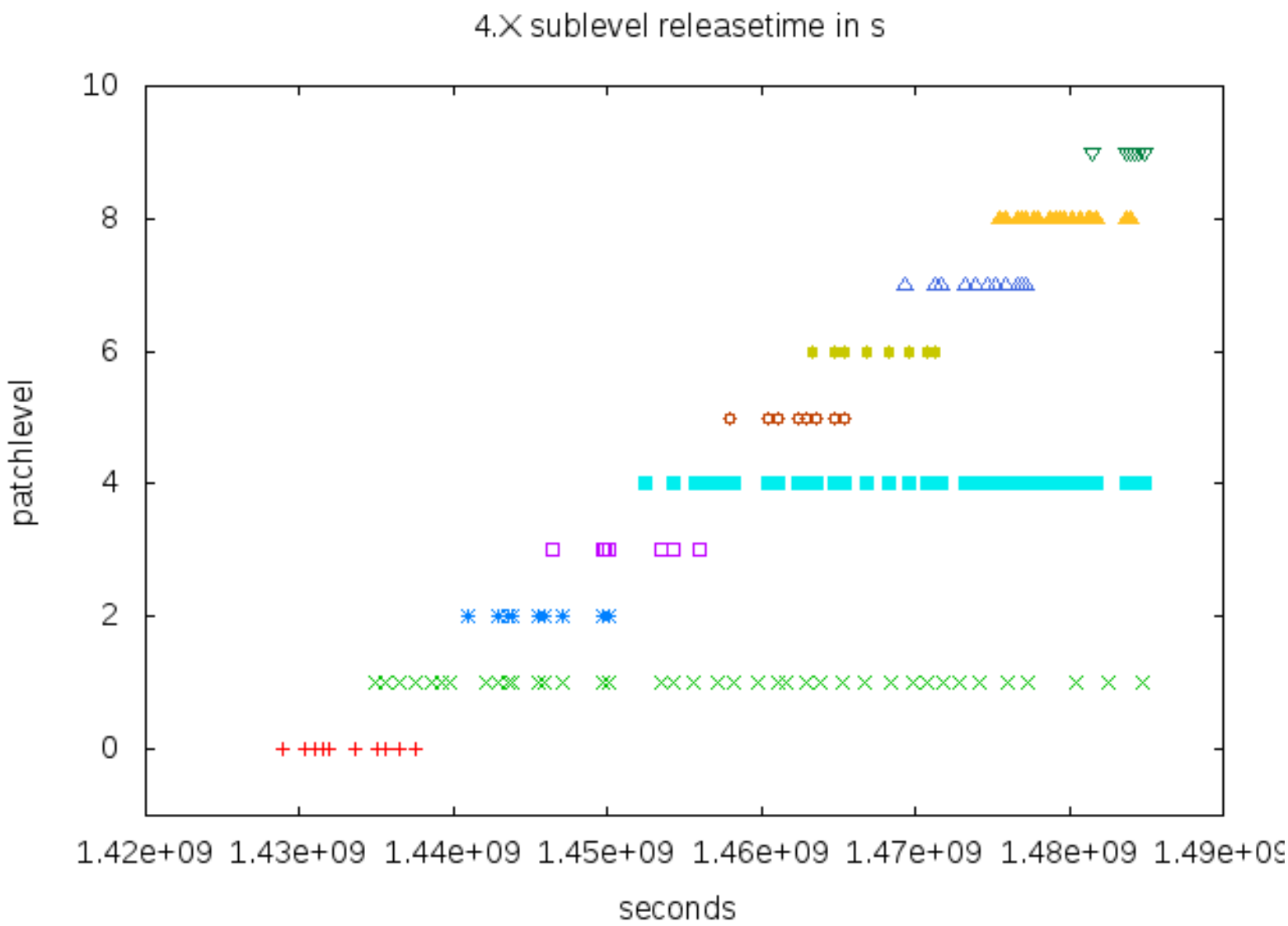
	files	lines		commits	lines added
	changed	add	del		per commit
rc1	9981	697599	460116	12226	57.0
rc2	334	4149	5497	386	10.7
rc3	245	3346	2342	277	12.0
rc4	363	4968	1672	331	15.0
rc5	256	1766	1304	260	6.7
rc6	263	2272	1236	309	7.3
rc7	91	977	429	109	8.9
rc8	73	709	448	82	8.6
4.4	88	518	280	102	5.0
4.4.1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

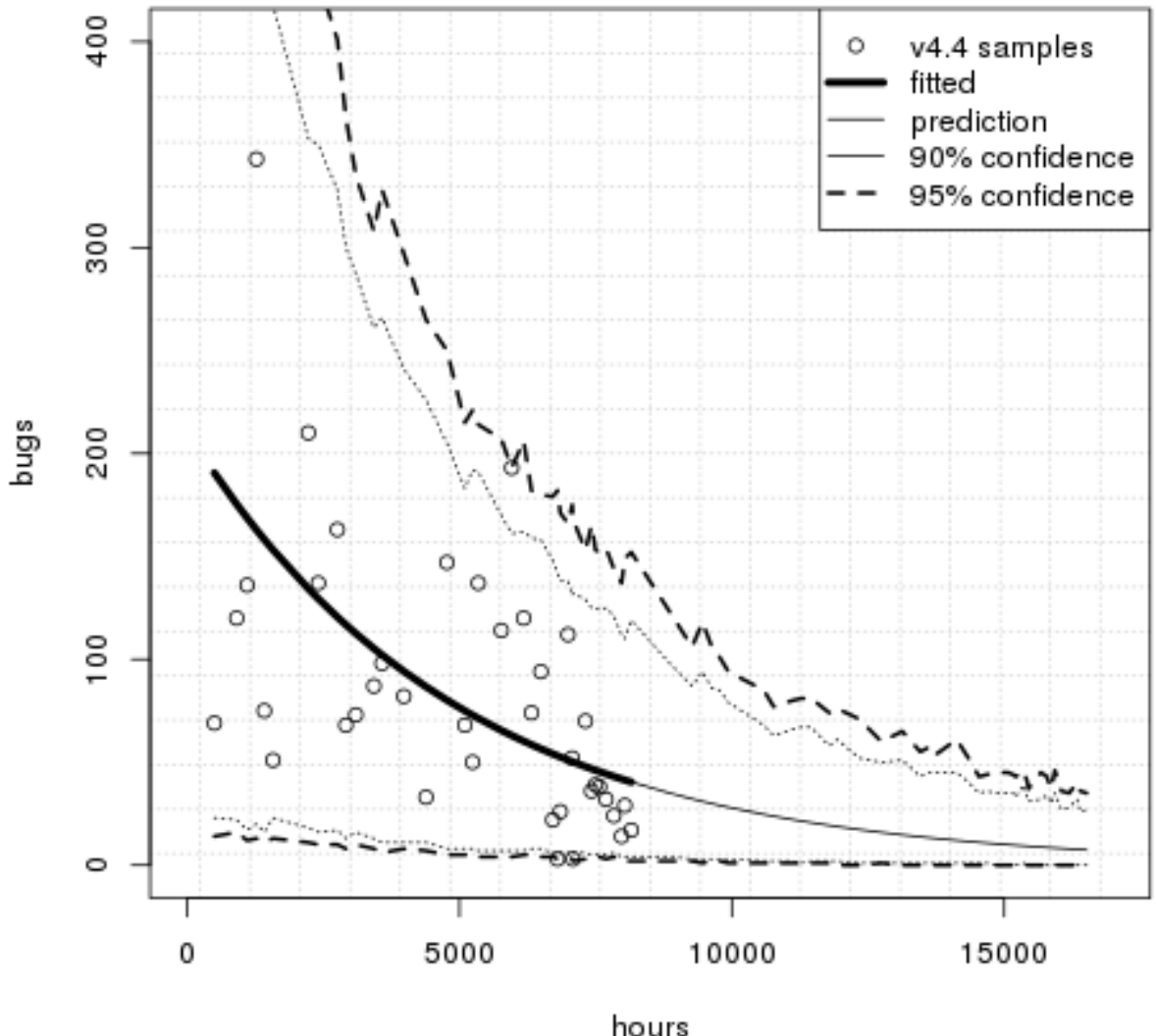


Source: http://neuling.org/linux-next

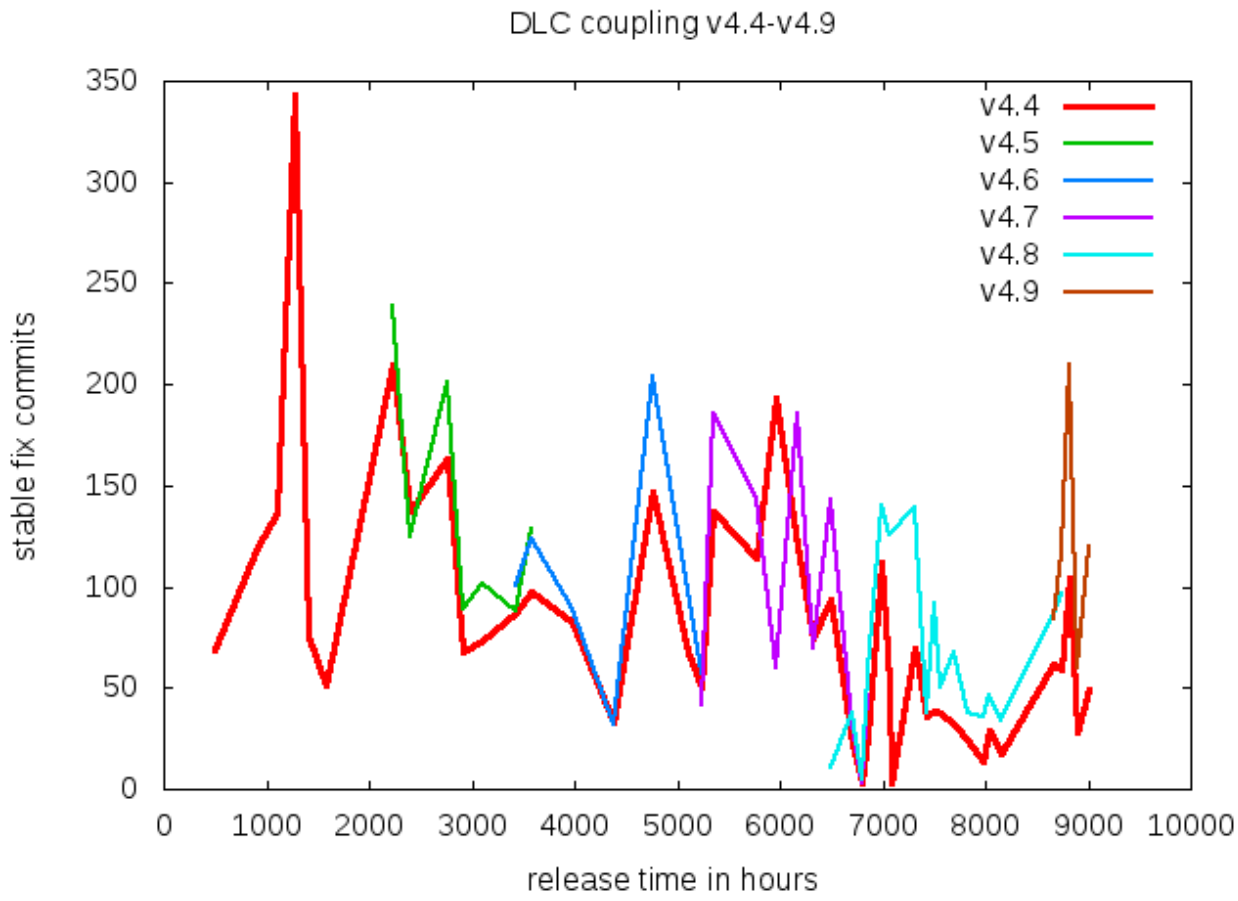
4.X DLC timeline



Statistic argument: Stability of overal DLC



kernel DLC PATCHLEVELS coupling



Strong coupling allows to re-enforce data sets by borrowing strength

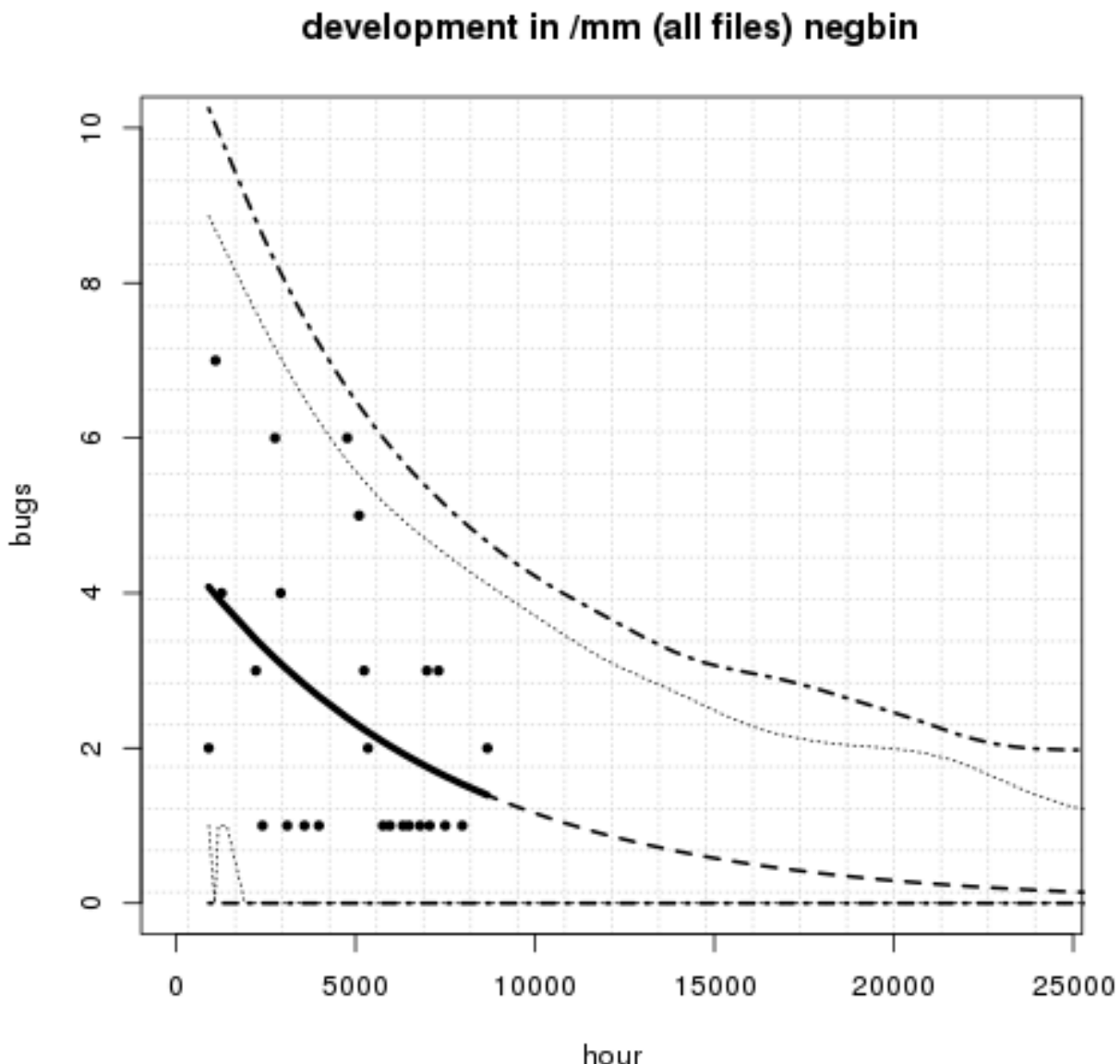
kernel DLC Trend v3.2 - v4.4(9)

Trending bugs-fixed over sublevel for -stable kernels

ver	intercept	slope	p-value	DoF	AIC
3.2	4.207356	0.005910	0.06	83	904.76
3.4	3.953236	0.0001224	**0.958**	112	1117
3.10	4.254555	-0.004909	0.0166	103	1006.9
3.12	4.733430	-0.002298	0.451	69	750.78
3.14	4.656610	-0.014073	$6.26e-07$	78	770.44
3.18	4.853280	-0.017135	0.0497	46	513.64
4.1	4.78926	-0.01404	0.184	36	417.13
4.4	5.060971	-0.033806	$2.71e-06$	46	475.75
4.9	4.88905	-0.04180	**0.573**	**5**	78.148

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.

Statistic argument: Stability of subsystems



i7_min_config_4.4.42 (linux-stable)

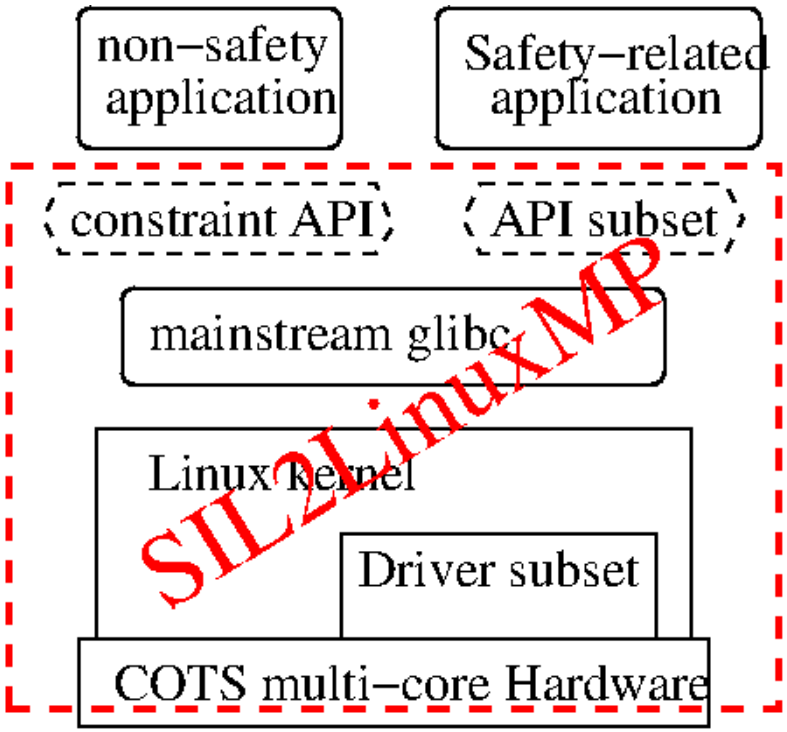
Subsys	files	blank	comment	code	full	%used
arch	480	16725	20728	76208	2040204	3.74
block	52	3973	5619	16614	24753	67.12
crypto	32	1950	1618	8626	76265	11.31
drivers	357	32181	53023	128168	8587655	1.49
fs	138	14157	23640	80227	827737	9.69
include	1196	35986	64271	163952	449811	36.45
init	8	354	391	1846	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

Conclusion

- Most elements are there - selection and integration needed
- Key elements look resonably stable - conditioned on careful selection
- Linux for Safety related systems at SC2/SIL2 looks doable
- Data availability and community response allows statistica approaches.
- The DLC allows a global assessment of processes, methods and tools
- Prediction is possible and allows initiating a quantitative monitoring program.
- TODO: bottom up model starting at properties of individual patches.

Nobody claims this will be simple - and it turned out to be harder to organize than anticipated - technically I think we are doing quite well though.

The Goal



Project launched by **OSADL**

Project hosted at **http://www.osadl.org/SIL2**

Thanks !