Understanding Linux – pictures from a journey in product line analysis and evolution

Klaus Schmid

schmid@sse.uni-hildesheim.de

Software Systems Engineering
University of Hildesheim

www.sse.uni-hildesheim.de

About this talk

- There are tons of studies, tools, research results,…
- Only high-level overview, no claim for completeness
- Only some is our work, mostly others

.. interspersed with personal views.

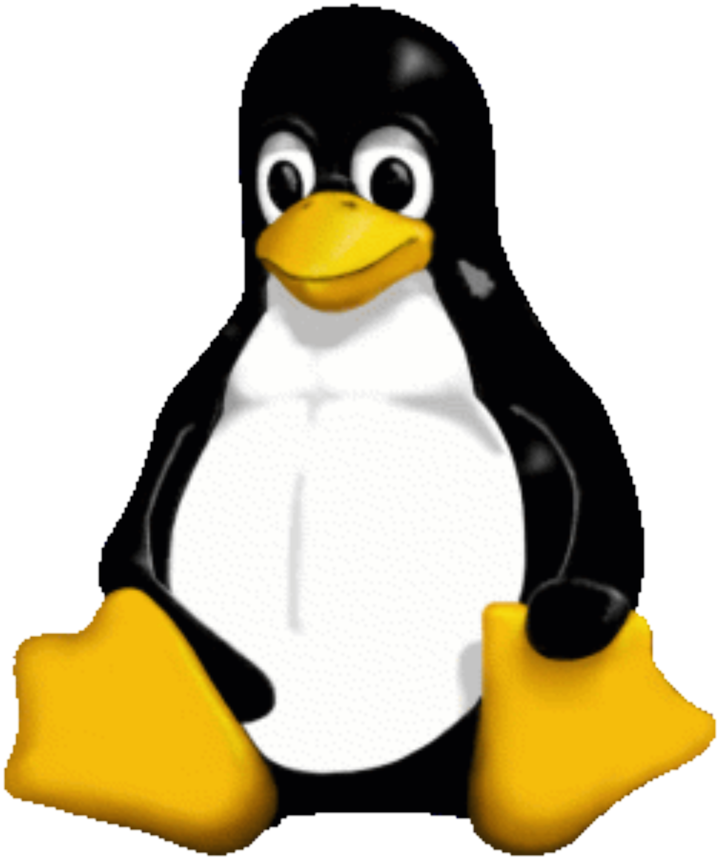
.. and general comments on the scientific process.

Why Linux

Product Line Analysis and Reengineering =

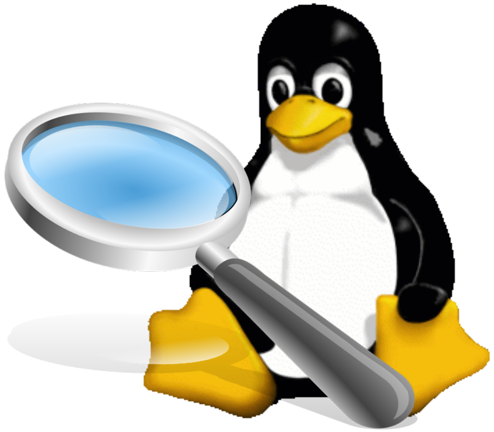
- *Extract information => Tool building*
- *Make sense of it*

Linux as „model organism“



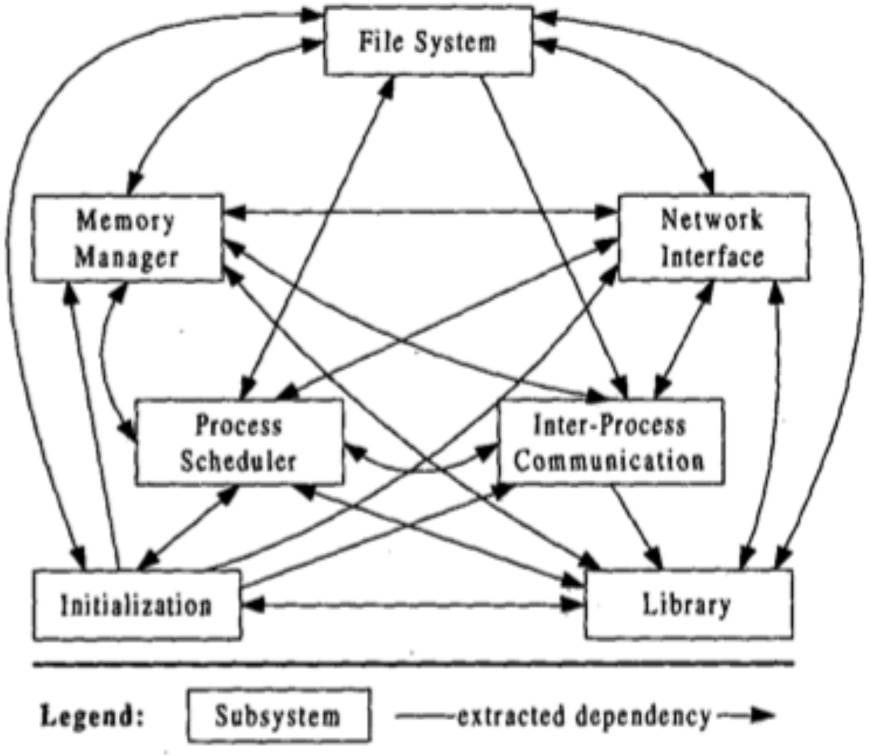
- Provides reference point
- Makes research comparable
- Allows (ideally) to build on each other
- Improves the scientific process

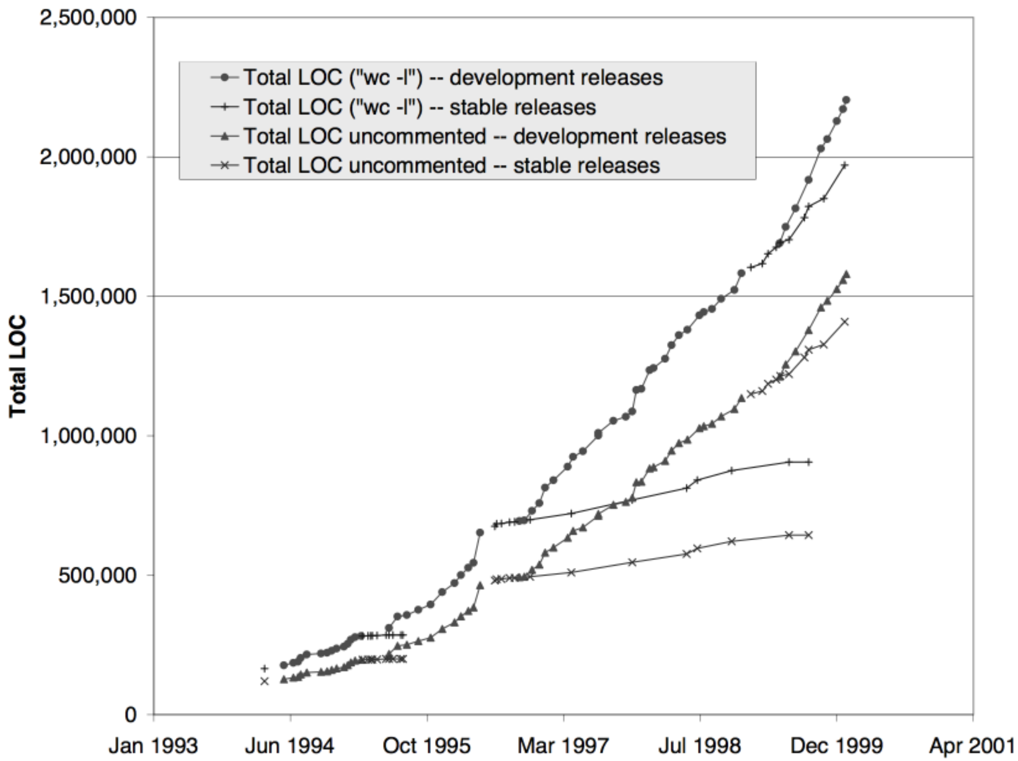
Linux as a case study



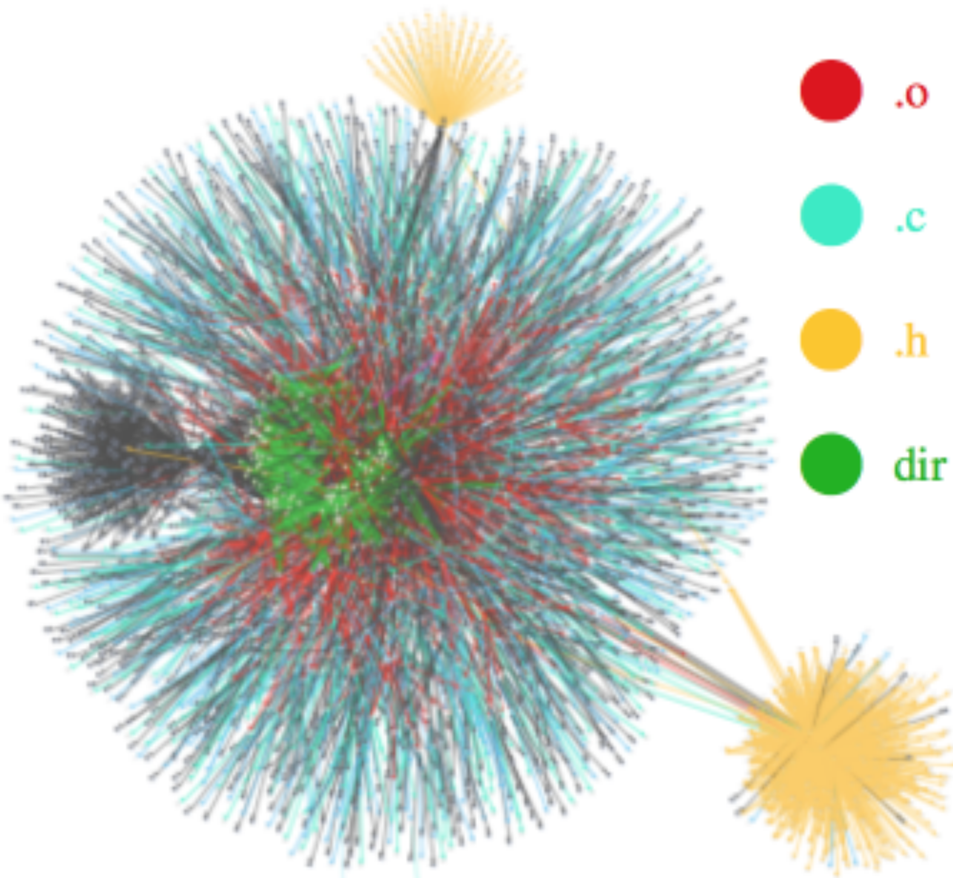
History

- Reflections on development process „Cathedral and Bazaar", 1997/1999
- Architecture Analysis [BHB99]
- Evolution[GT00]





Linux as a case study



Linux 2.6.0 build process
(phase vmlinux)

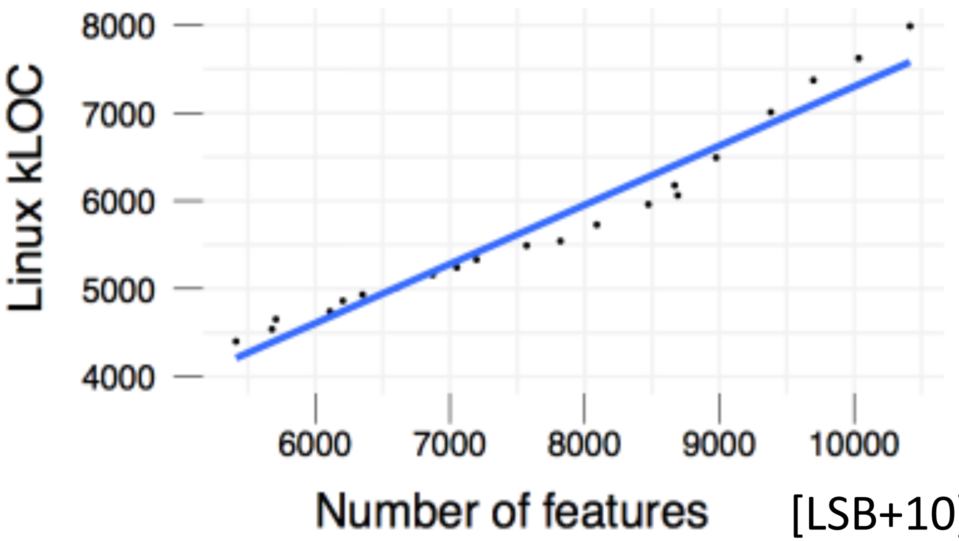
First analysis explicitly addressing variability (afaik):

[AST+07] B. Adams, K. de Schutter, H. Tromp, and W. de Meuter, *“The Evolution of the Linux Build System”*, Electronic Communication of the European Association of Software Science and Technology, vol. 8, pp. 1–16, 2007.

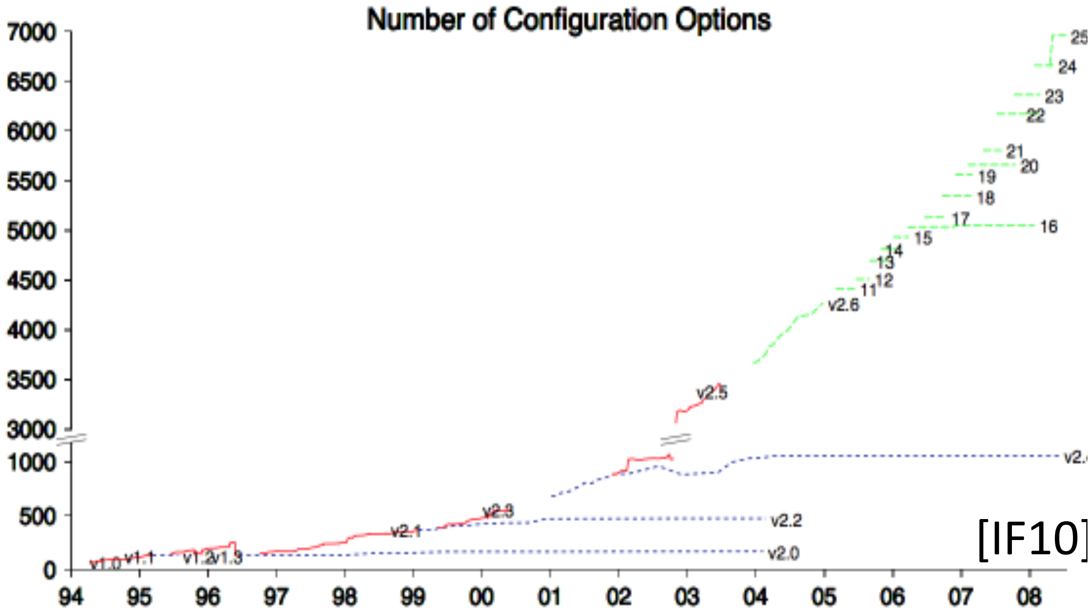
Linux as a Product Line Case Study

Characteristics of Linux:

- Large: Linux 4.10 (Commit: d528ae0 vom 16.03.2017)
  - Code size: 57.985 files (VM: 1382, Build: 2554, Source: 57985)
  - # features (variabilities): ~17.591
  - Average depth of feature tree (2012): 3.7
  - Median of branching (2012): 85
- Size: similar to large industrial product lines



[LSB+10]



[IF10]

Linux as a Product Line Case Study

Actually ~20 PLs
(1 per arch)

As a PL:

- Kconfig-based variability description:
  - 3-valued logic
  - Also: strings, integer, hexadecimals (rarely used)
  - Invisible features
- C-preprocessor-based (including partially runtime)
- Make (kbuild) resolves about 60% of variability

Is it realistic?

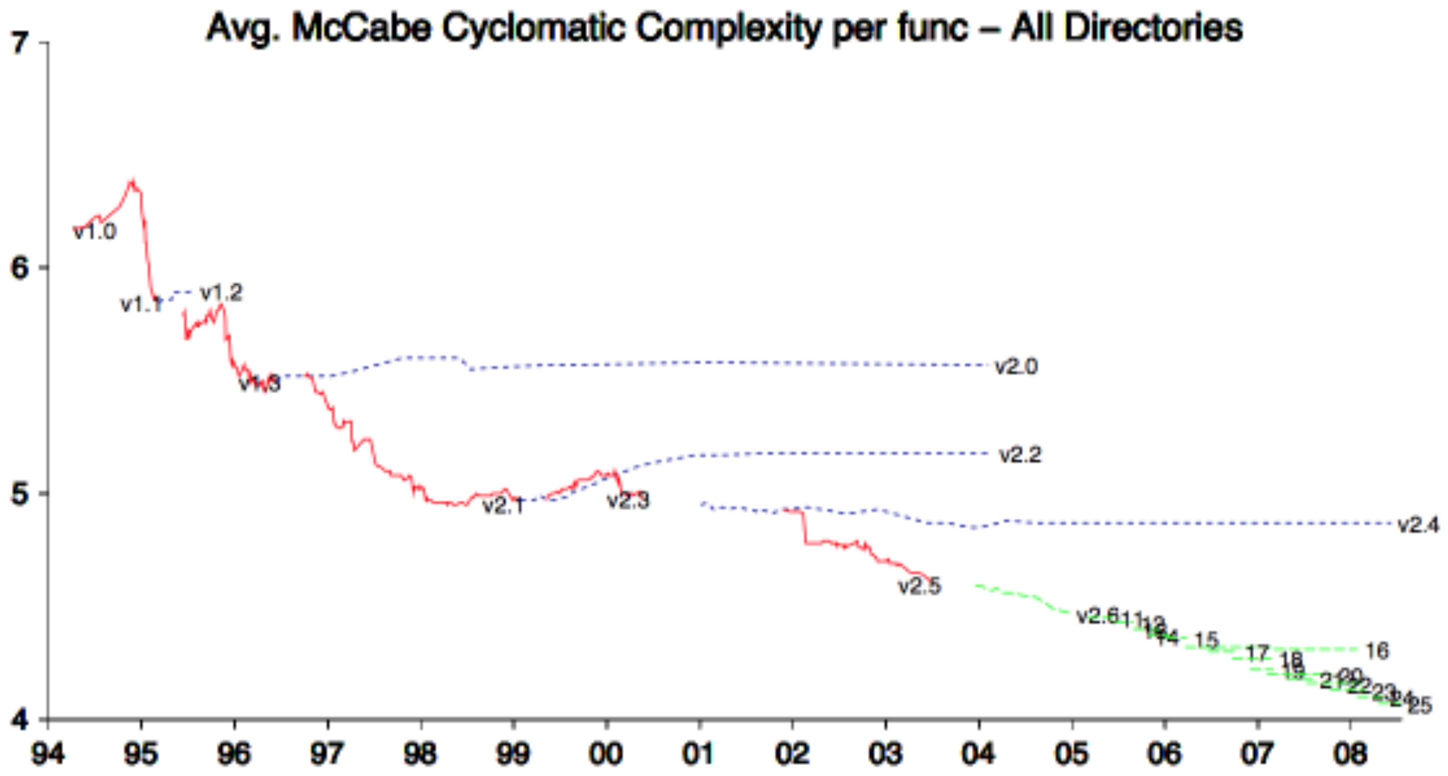
Linux as a proxy for industrial research

- Compared with other possible case study repositories (e.g., SPLOT)
  - SPLOT et al. do not originate from realistic processes [SL+10]
  - Linux (and other OS-examples) are broader, much higher branching, fewer feature groups [BS+13]
  - ..

- But is this similar to industrial?
  - On the code level similar to industrial product lines, but more intensive use of preprocessor [HZ+16]
  - Tangling, scattering, nesting similar
  - Variability model ? Build system ?
  - Usage of non-boolean variability?

Linux: Evolution Characteristics

- There seems to be continuous refactoring
  - E.g., cyclomatic complexity is even reducing



Linux: Analysis

- Verification (global)
  - Dead Feature
  - Undead Feature
  - Dead Code
  - Undead Code
  - Misconfiguration
  - …
- Type-checking (global)
- Static checking (local, based on covering)
- Metrics-based analysis
  - Characterizations
  - Security defects
  - ..

Some results of ours

© Google Maps + own graphics

Understanding Source Code

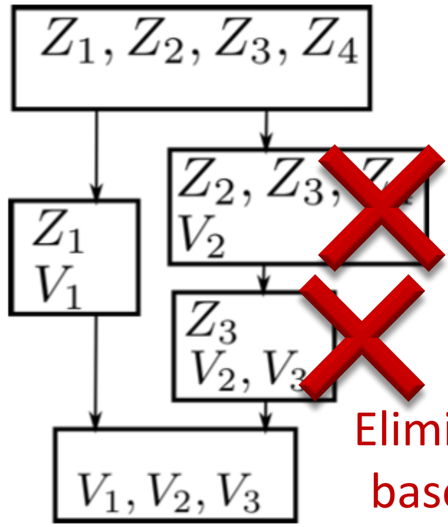
- Lattice for representing the variability

```
#if RUN
int i=y+x;
int z=0;
#endif
#if WAIT
while(x<z+1){
x=x*z++;
}
#if LOAD
load(x,z,i);
#endif
read(i);
#endif
```

Source code

	V_1	V_2	V_3
Z_1	x		
Z_2		x	
Z_3		x	x
Z_4		x	

Relation



Elimination based on Variability model

Hasse Diagram

[LAS+16] D. Lüdemann, N. Asad, K. Schmid, and C. Voges, *Understanding variable code: Reducing the complexity by integrating variability information*, International Conference on Software Maintenance and Evolution (ICSME), pp. 312–322, 2016.

Understanding Source Code

- Observation:
  - Lattices can be reused based on inconsistency
  - However:
    - Most reductions are trivial reductions (X, ¬ X)
    - Total number of reductions is not high, but smaller files are reduced more

V. pF	ØV. pF	Files	Conc.	Conc. pF	Red.	Red. pF	Cont.	Cont. pF	Øper.	Time pF
0	0	13180	-	-	-	-	-	-	-	-
1-10	2.1	7863	34152	4.3	3525	0.5	3507	0.5	8.41%	21s
11-20	14.0	129	2699	20.9	116	0.9	109	0.8	4.59%	4m
21-30	24.7	30	983	32.8	30	1.0	24	0.8	3.27%	9m
31-40	33.9	9	336	37.3	10	1.1	4	0.4	3.45%	6m
41-50	44.3	7	276	39.4	7	1.0	3	0.4	2.57%	4m
51-60	55.7	3	145	48.3	3	1.0	3	1.0	2.11%	8m
61-70	63.4	5	314	62.8	5	1.0	4	0.8	1.76%	15m
71-80	-	-	-	-	-	-	-	-	-	-
81-90	85.5	2	134	67.0	2	1.0	2	1.0	1.50%	12m
91-100	93.0	2	183	91.5	2	1.0	2	1.0	1.11%	33m
>100	233.0	1	141	141.0	1	1.0	1	1.0	0.71%	15m

[LAS+16]

TABLE VI
RESULTS OF THE EXTENDED FCA FOR LINUX KERNEL 4.2.3.
(V=VARIABLES, CONC=CONCEPTS, RED=REDUCTIONS, CONT=CONTRADICTIONS, ØPER= AVERAGE PERCENTAGE OF RED. CONCEPTS, PF=PER FILE)

Understanding Source Code

- Observation:
  - Much higher percentage of reduction, if architecture-specific

		x86		IA64		ARM		blackfin		SPARC		PowerPC	
Variables	**Files**	Red	Øper	Red	Øper	Red	Øper	Red	Øper	Red	Øper	Red	Øper
1-10	7863	3796	8.94%	4215	9.80%	3823	8.97%	4283	9.81%	4001	9.27%	3816	8.99%
11-20	129	178	7.26%	219	8.46%	164	6.44%	238	8.36%	190	7.71%	176	7.26%
21-30	30	61	7.55%	71	8.60%	56	6.90%	67	7.03%	78	9.13%	63	7.69%
31-40	9	59	21.55%	64	23.09%	60	21.80%	13	4.23%	59	21.55%	59	21.55%
41-50	7	90	33.24%	90	33.24%	89	32.93%	7	2.57%	90	33.24%	90	33.24%
51-60	3	53	35.46%	53	35.46%	53	35.46%	3	2.11%	53	35.46%	53	35.46%
61-70	5	40	16.15%	55	19.64%	44	17.08%	36	8.82%	49	18.19%	46	17.49%
71-80	-	-	-		-	-	-	-	-	-	-	-	-
81-90	2	31	22.90%	31	22.90%	31	22.90%	2	1.50%	31	22.90%	31	22.90%
91-100	2	29	18.20%	29	18.20%	29	18.20%	2	1.11%	29	18.20%	29	18.20%
>100	1	16	11.35%	16	11.35%	16	11.35%	1	0.71%	16	11.35%	16	11.35%

[LAS+16]

Evolution Analysis

To what extend to commits really impact variability

- Not every change of Kconfig is modifying variability
- Not every change of code in an #ifdef is impacting variability
- …

./arch/x86/events/perf_event.h

```
#ifdef CONFIG_X86_32
  return ip > PAGE_OFFSET;
#else
  return (long)ip < 0;
#endif
```

Introduction of changes

Deletion of #else-block

Commit File

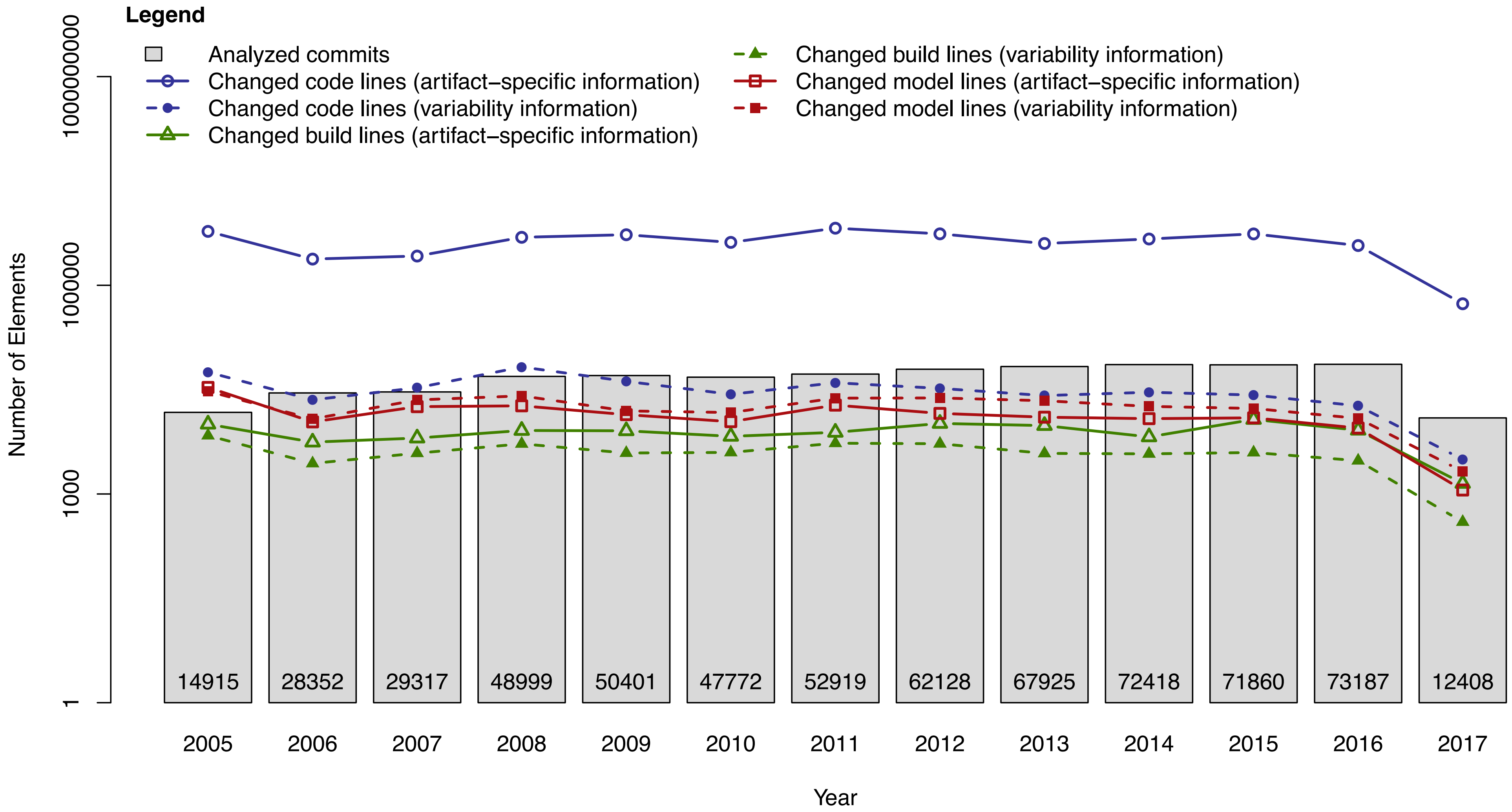
```
#ifdef CONFIG_X86_32
  return ip > PAGE_OFFSET;
-#else
-  return (long)ip < 0;
#endif
```

Change to variability information

Change to non-variable information

Evolution Analysis

Changes in Variability vs. other parts



Misconfigurations

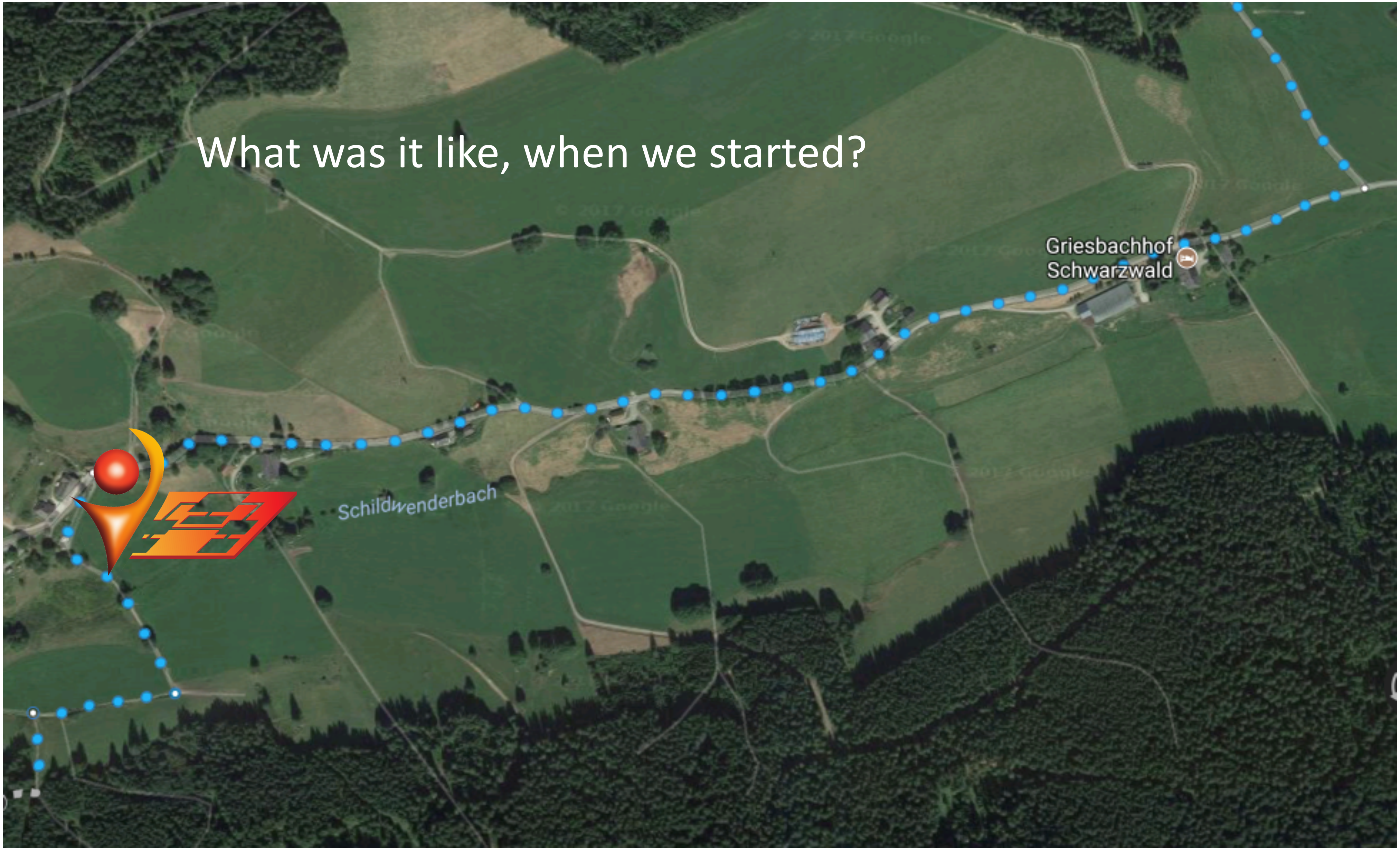
If

- you have 4 possible configurations
- only three different products

Is this a problem?

Wednesday: 10:45
An Empirical Study of Configuration Mismatches in Linux
Sascha El-Sharkawy, Adam Krafczyk and Klaus Schmid

Our journey



© Google Maps + own graphics

Our journey

Opportunities for reuse

Reusing existing tools

- Well-documented
- Easy to use and reuse

Considerable effort for:

- Learning
- Applying
- Adapting

Reusing existing tools

- Numerous tools
  - Extraction
    - Kconfig
    - Kbuild
    - Source-code
  - For analysis
  - Understand Evolution
  - For Visualization

- Issues:
  - What exactly is analyzed? (Correctness)
  - Reliability
  - Repeatability
  - Transferability

Analyzing KConfig

- Kconfig-translation is the way to include variability information in analysis and other processing

```
config BOOL_VAR
    bool "A Boolean Variable"
    select BOOL_VAL2

choice
    bool "A Choice"

    config BOOL_VAL1
        bool "1st Boolean Value"

    config BOOL_VAL2
        bool "2nd Boolean Value"
 endchoice
```

KConfig

```
c 1 BOOL_VAR
c 3 CHOICE_1
c 2 BOOL_VAL2
c 4 BOOL_VAL1
c 5 MODULES
p cnf 5 8
−1 2 0
−1 2 0
3 0
2 4 −3 0
−2 3 0
−4 3 0
−2 −4 0
−5 0
```

Dimacs

Analyzing KConfig

contains → ↑		Config options				Choices		Hierarchies		Constraints		Attributes				If
		bool	tristate	string	numerical	bool	tristate	menu	via Constraints	depends on	select	prompt	default	range	visible if	
Config options	bool	×				×		×	✓	✓	⚠	✓	⚠	—	×	×
	tristate								✓		⚠		⚠	—		
	string								+		—		✓	—		
	numerical								+		—		ϟ	ϟ		
Choices	bool	✓	⚠	ϟ		+		×	⚠	⚠	Ⓢ	⚠	✓	Ⓢ	×	⚠
	tristate	⚠	✓													
Hierarchies	menu	✓				✓		✓	×	✓	×	✓	×	×	✓	✓
	via Constraints								✓		✓		✓	✓	×	+
Constraints	depends on	✓		—		—		—	✓	×		×				×
	select							×	+							✓
Attributes	prompt	×				×		×		×		⚠				✓
	default															
	range															
	visible if															
If		✓				✓		✓	+	×		×				✓

Table 1: Systematic analysis (– = no change, x = not able to model this, ✓ = clearly specified, = inconsistent configurations, = inconsistent models, + = only graphical representation changed, = against specification)

S. El-Sharkawy, A. Krafczyk, K. Schmid. *Analysing the Kconfig Semantics and Its Analysis Tools.* International Conference on Generative Programming: Concepts and Experiences (GPCE), pp. 45-54, 2015.

Analyzing Kconfig-Tools

Case	Undertaker			Kconfig Reader					LVAT
	satyr	dumpconf	rsf2model	modified dumpconf			new dumpconf		vm2bool
	DIMACS	RSF	Model	DIMACS	RSF	Model	DIMACS	Model	DIMACS
Handling Attribute `option modules`	✗	✗	✗	✗	✗	✗	✗	✗	✗
Constraint Precedence	✓	✓	✗	✓	✓	✓	✓	✓	✓
Missing Config Options	✓	✓	✓	✓	✓	✓	✓	✓	✓
Config Options × `select`	✓	Ⓟ	✗	✗	Ⓟ	✗	✗	✗	✗
Boolean Config Options × `default`	✓	Ⓟ	✗	✓	Ⓟ	✓	✓	✓	✗
Tristate Config Options × `default`	✗	✓	✓	✓	✓	✓	✓	✓	✗
Tristate Choices × Boolean Config Options	✗	✓	✓	—	✓	—	✓	✓	✗
Boolean Choices × Tristate Config Options	✗	✓	✓	✗	✓	✗	✗	✗	✗
Choices × Hierarchies	✓	✗	✗	✗	✓	✗	✓	✓	✓
Choices × (broken) Hierarchies	✓	✗	✓	✓	✓	✓	✓	✓	✓
Choices × `if`	✓	✗	✗	✗	✗	✗	✓	✓	✗
Choices × Constraints	✗	✓	✗	—	✓	—	—	—	✗
Choices × `prompt`	✗	✗	✗	✗	✓	✗	✓	✓	—
`default` × `default`	✓	✗	✗	✓	✓	✓	✓	✓	✓

S. El-Sharkawy, A. Krafczyk, K. Schmid. *Analysing the Kconfig Semantics and Its Analysis Tools.* International Conference on Generative Programming: Concepts and Experiences (GPCE), pp. 45-54, 2015.

Analyzing Kbuild

- Various tools
  - Static (parsing-based)
    - KbuildMiner
    - MakeX
  - (partially) dynamic
    - Golem

- *Determines lower bound*
- *Fast*

- *Identifies more constraints*
- *More robust*

Degree of correctness?

Analyzing Source Code

- Variability Extraction

```
/* INIT */
static int (*dss_output_drv_reg_funcs[])(void) __initdata = {
    dss_init_platform_driver,
    dispc_init_platform_driver,
#ifdef CONFIG_OMAP2_DSS_DSI
    dsi_init_platform_driver,
#endif
#ifdef CONFIG_OMAP2_DSS_VENC
    venc_init_platform_driver,
#endif
#ifdef CONFIG_OMAP4_DSS_HDMI
    hdmi4_init_platform_driver,
#endif
#ifdef CONFIG_OMAP5_DSS_HDMI
    hdmi5_init_platform_driver,
#endif
};
```

DIMACS

Possible tools: e.g., Undertaker, TypeChef

Analyzing Source Code

- Undertaker (Block-extraction)
  - Pros: speed, simplicity
  - Cons:
    - no interpretation of header files
    - no dependency resolution of #define -> ifdef
- Typechef (parse-extraction)
  - Much slower (and needs tons of memory)
  - Full AST with var. annotation
  - Able to identify indirect variability (e.g., variable parameter in array)
  - Can identify that *#if 0* does not contain code

Where are we?

- Many tools available, but
  - Difficult to use
  - Comparisons hard to make
- Many analysis performed (more to do)
- Studies often hard (if not impossible) to replicate

What we want

- Simple / Systematic tool reuse
- Documentation & Replication built in
- Easy to set up and run alternatives

We call it

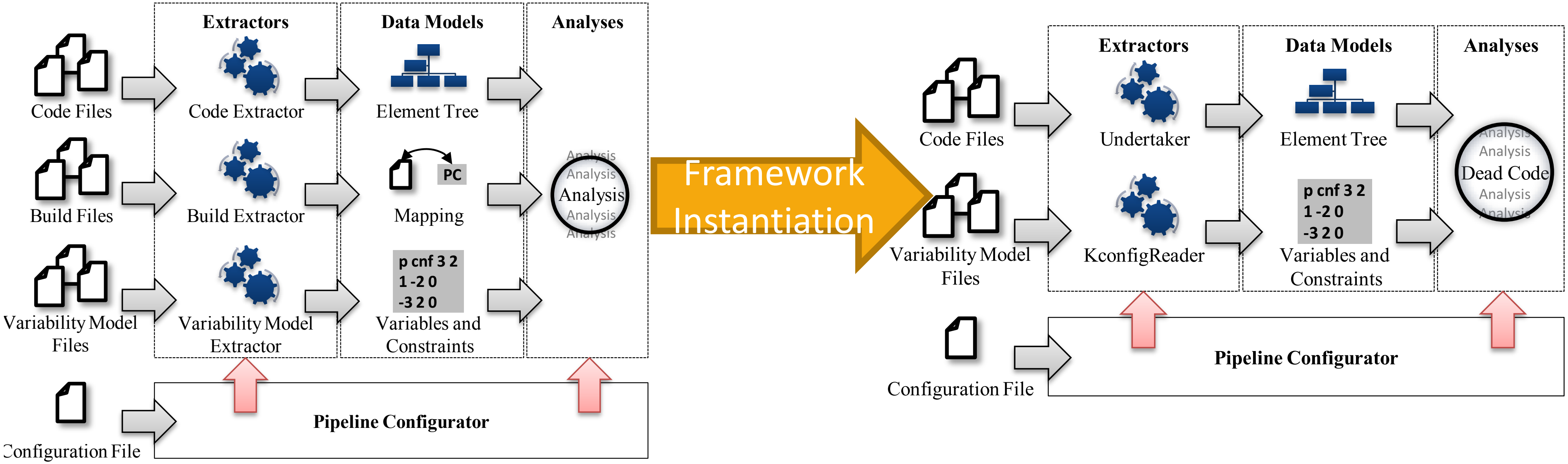
Experimentation Workbench

KernelHaven

- Scope: Product Line Analysis (not only Linux)
- Capabilities:
  - Documentation
  - Takes care of technical aspects (e.g., parallelization)
  - Highly configurable
  - Portable
- Existing plugins (public)
  - KConfigReader
  - Kbuildminer
  - Typechef, Undertaker
  - Feature-Effect-Analysis,
    Metrics support, Undead Code

https://github.com/KernelHaven

KernelHaven



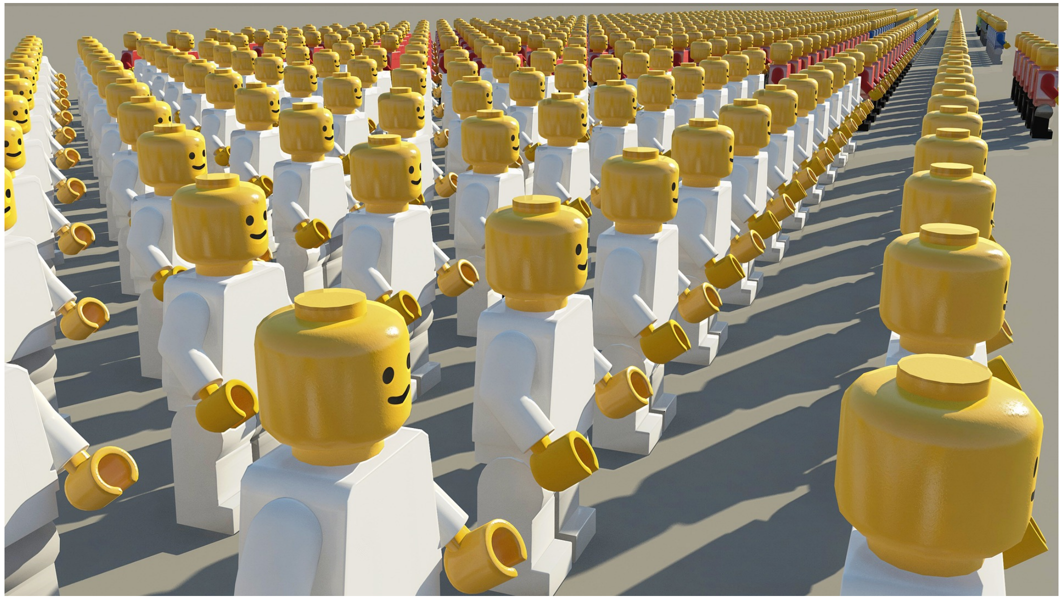
KernelHaven Architecture

Dead Code Analysis Instance

Auto-Archiving covers:
Inputs, outputs, all code

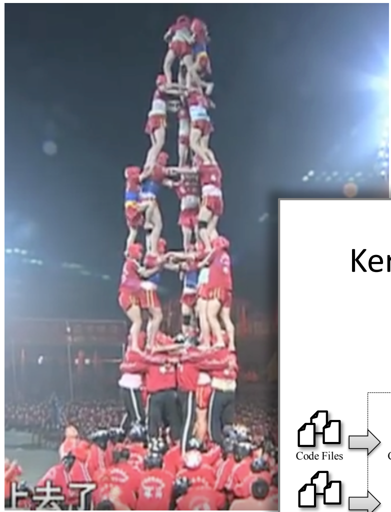
Kernelhaven

- Goal = Facilitate
  - Experimentation in Product Line Analysis
  - Replication (3rd party and others)
  - Inspection / analysis of results & approaches
  - Jump-start for others
  - Focus on the technical / scientific core



Summary

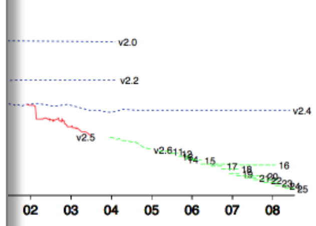
The scientific process



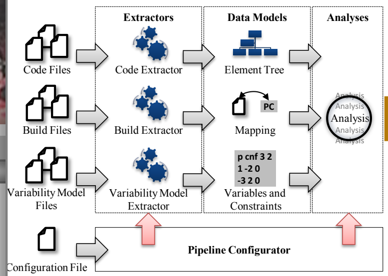
Linux: Evolution Characteristics

- There seems to be continuous refactoring
  - E.g., cyclomatic complexity is even reducing

Avg. McCabe Cyclomatic Complexity per func – All Directories

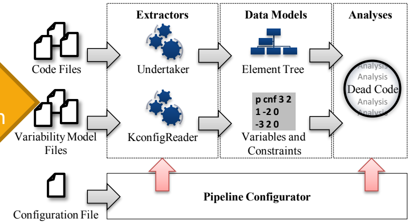


KernelHaven



KernelHaven Architecture

Framework Instantiation



Dead Code Analysis Instance

Auto-Archiving covers:
Inputs, outputs, all code

Misconfigurations

If
- you have 4 possible configurations
- only three different products

Is this a problem?

Wednesday: 10:45
An Empirical Study of Configuration Mismatches in Linux
Sascha El-Sharkawy, Adam Krafczyk and Klaus Schmid

		...er		Kconfig Reader					LVAT
			...f rsf2model	modified dumpconf			new dumpconf		vm2bool
			Model	DIMACS	RSF	Model	DIMACS	Model	DIMACS
			✗	✗	✗	✗	✗	✗	✗
			✗	✓	✓	✓	✓	✓	✓
			✓	✓	✓	✓	✓	✓	✓
			✗	✗	Ⓟ	✗	✗	✗	✗
			✗	✓	Ⓟ	✓	✓	✓	✗
... `default`			✓	✓	✓	✓	✓	✓	✗
Tristate Choices × Boolean Config Options	✗	✓	✓	—	✓	—	✓	✓	✗
Boolean Choices × Tristate Config Options	✗	✓	✓	✗	✓	✗	✗	✗	✗
Choices × Hierarchies	✓	✗	✗	✗	✓	✗	✓	✓	✓
Choices × (broken) Hierarchies	✓	✗	✓	✓	✓	✓	✓	✓	✓
Choices × `if`	✓	✗	✗	✗	✗	✗	✓	✓	✗
Choices × Constraints	✗	✓	✗	—	✓	—	—	—	✗
Choices × `prompt`	✗	✗	✗	✗	✓	✗	✓	✓	—
`default` × `default`	✓	✗	✗	✓	✓	✓	✓	✓	✓

S. El-Sharkawy, A. Krafczyk, K. Schmid. *Analysing the Kconfig Semantics and Its Analysis Tools.* International Conference on Generative Programming: Concepts and Experiences (GPCE), pp. 45-54, 2015.

More information

- Availability of KernelHaven infrastructure: open source at https://github.com/KernelHaven
- There you also find a number of plugins already (more to come):
  - Extractors: TypeChef, KconfigReader, KbuildMiner, Undertaker
  - Analysis: Metrics, UnDead, FeatureEffect,…

Acknowledgement: *The research leading to these results has received funding from the ITEA3 project 15010 REVaMP², which is co-funded in part by the national funding agencies in various countries, including BMBF (German Ministry of Research and Education) under grant 01IS16042H in Germany.*

References

[IF10] A. Israeli, D. Feitelson. The Linux kernel as a case study in software evolution. Journal of Systems and Software, Elsevier, Vol. 83, 485-501, 2010.

[LSB+10] R. Lotufo, S. She, T. Berger, K. Czarnecki, A. Wasowski. Evolution of the Linux Kernel Variability Model, nternational Conference on Software Product Lines, Springer, 136-150, 2010.

[Ray] E. Raymond. The Cathedral and The Bazaar, http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/index.html, Last Retrieved 26.9.2017.

[AST+07] B. Adams, K. de Schutter, H. Tromp, and W. de Meuter, "The Evolution of the Linux Build System", Electronic Communication of the European Association of Software Science and Technology, vol. 8, pp. 1–16, 2007.

[BHB99] I. Bowman, R. Holt, N. Brewster. Linux as a case study: Its extracted software architecture Proceedings of the 21st international conference on Software engineering, 555-563, 1999.

[GT00] M. Godfrey, Q. Tu. Evolution in Open Source Software: A Case Study Proceedings of the International Conference on Software Maintenance, IEEE Computer Society, 131-142, 2000.

[LAS+16] D. Lüdemann, N. Asad, K. Schmid, and C. Voges, Understanding variable code: Reducing the complexity by integrating variability information, International Conference on Software Maintenance and Evolution (ICSME), pp. 312–322, 2016.

[EKS17] Sascha El-Sharkawy, Adam Krafczyk and Klaus Schmid, "An Empirical Study of Configuration Mismatches in Linux", International Software Product Line Conference (SPLC) 2017, pp. 19-28, ACM, 2017.