

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

**Klaus Schmid**

[schmid@sse.uni-hildesheim.de](mailto:schmid@sse.uni-hildesheim.de)

Software Systems Engineering  
University of Hildesheim

[www.sse.uni-hildesheim.de](http://www.sse.uni-hildesheim.de)

ReVaMP<sup>2</sup>

# 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.*

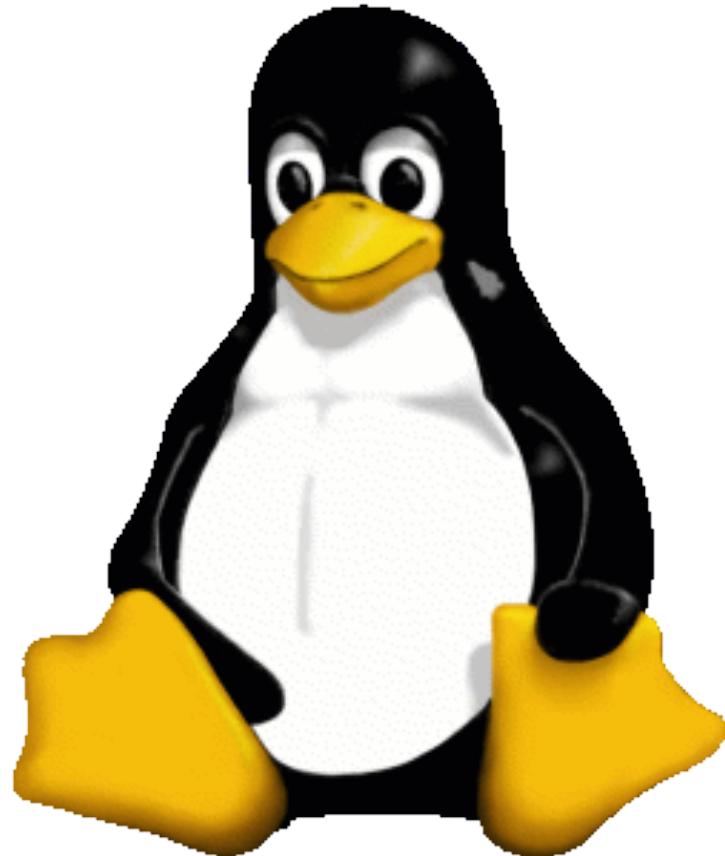
# Why Linux

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

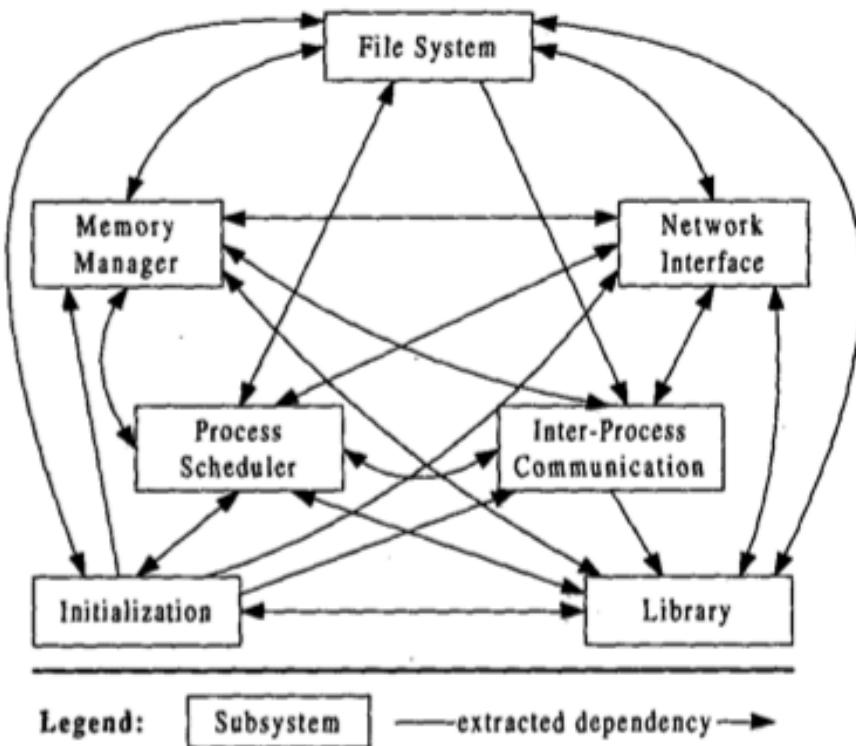
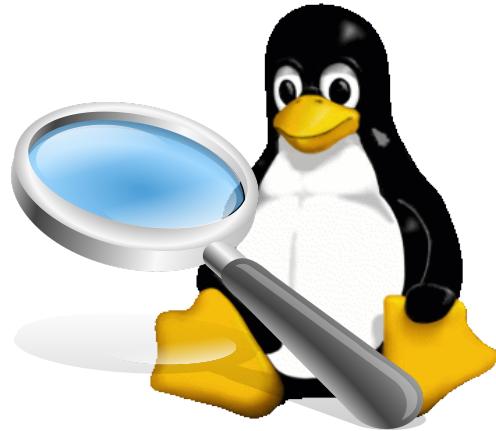
# The scientific process



*Standing on  
the shoulders  
of “giants”*

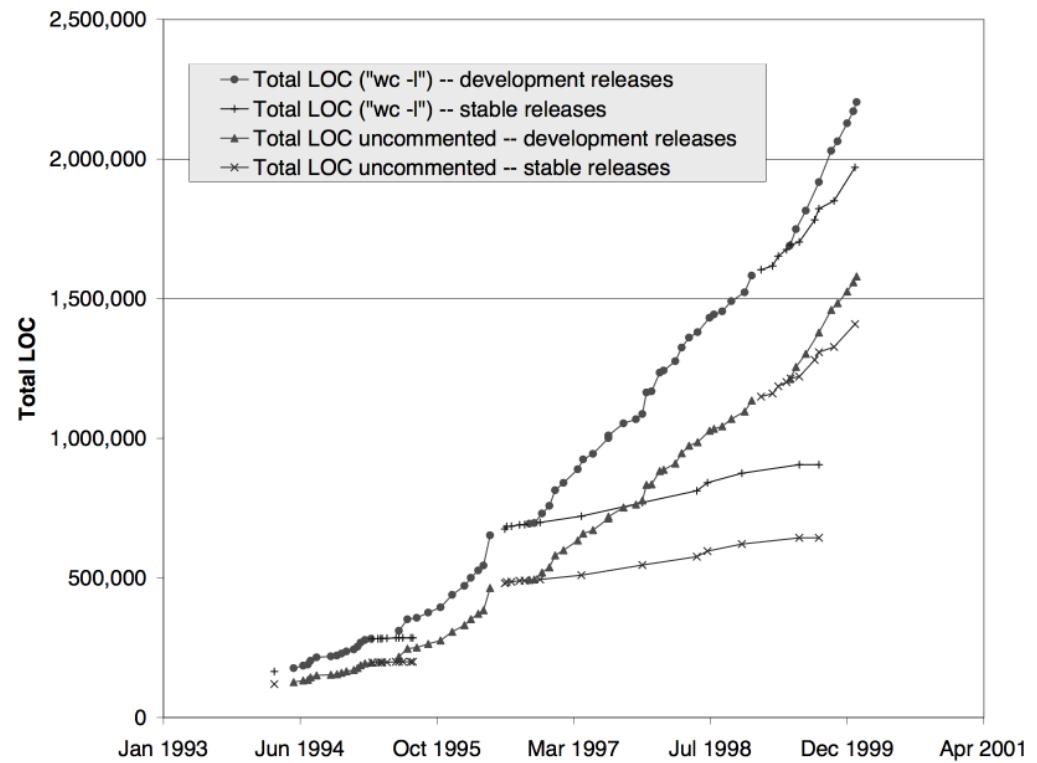
© youtube.com

# Linux as a case study

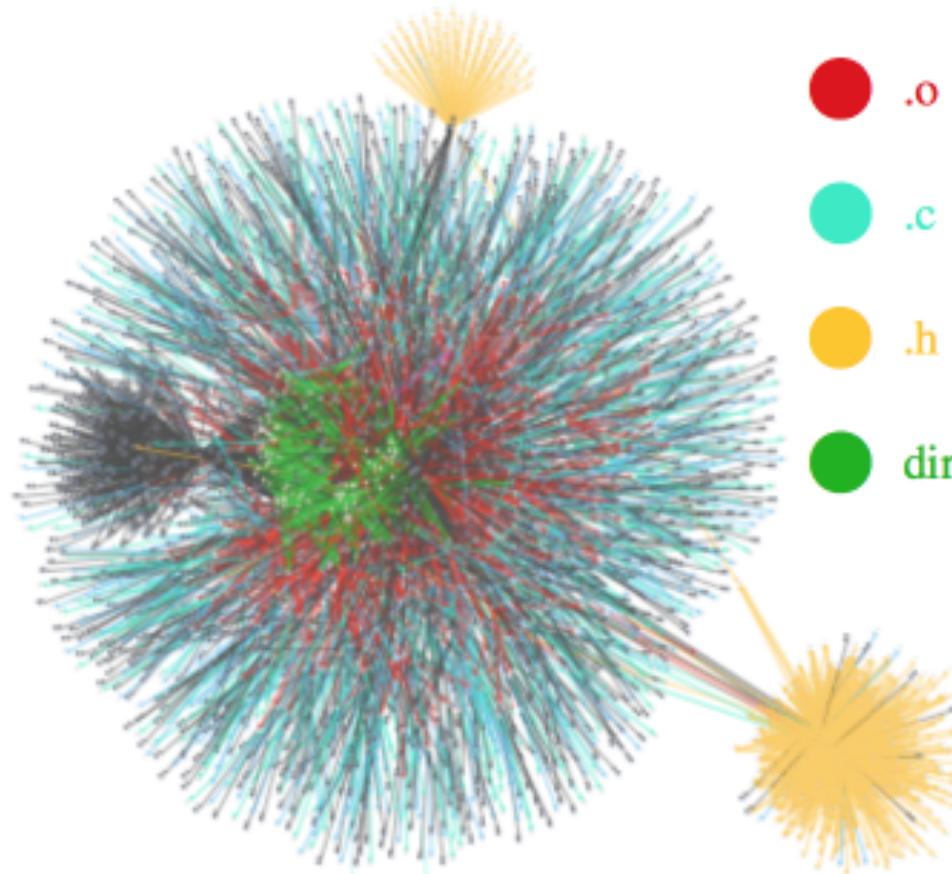


## History

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



# Linux as a case study



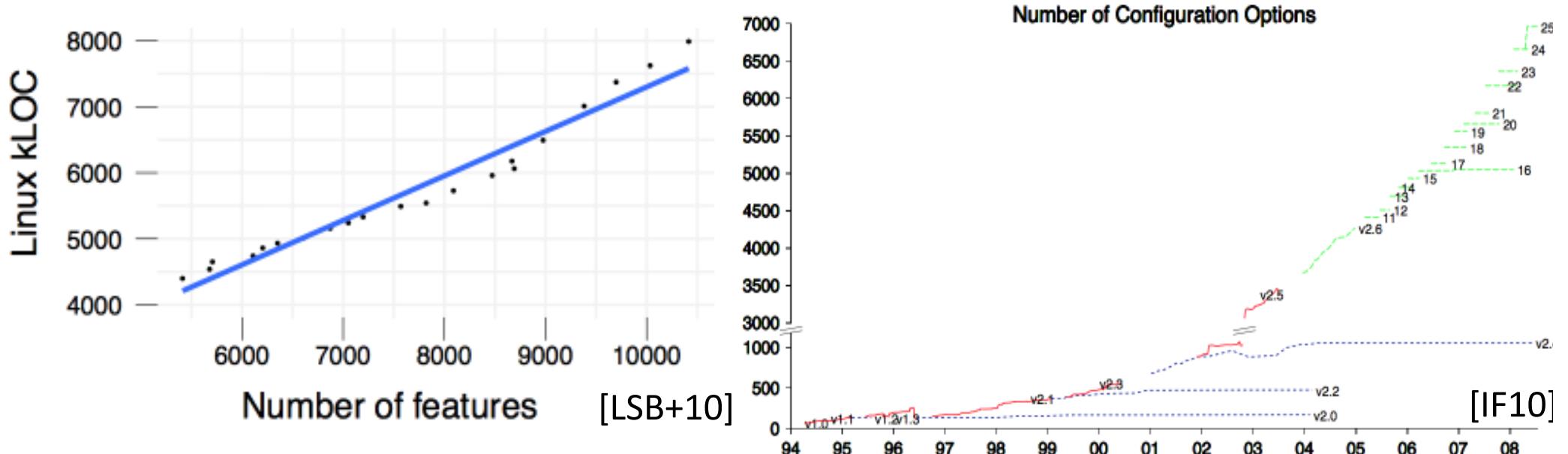
Linux 2.6.0 build process  
(phase vmlinux)

- .o
  - .c
  - .h
  - dir
- 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



# Linux as a Product Line Case Study

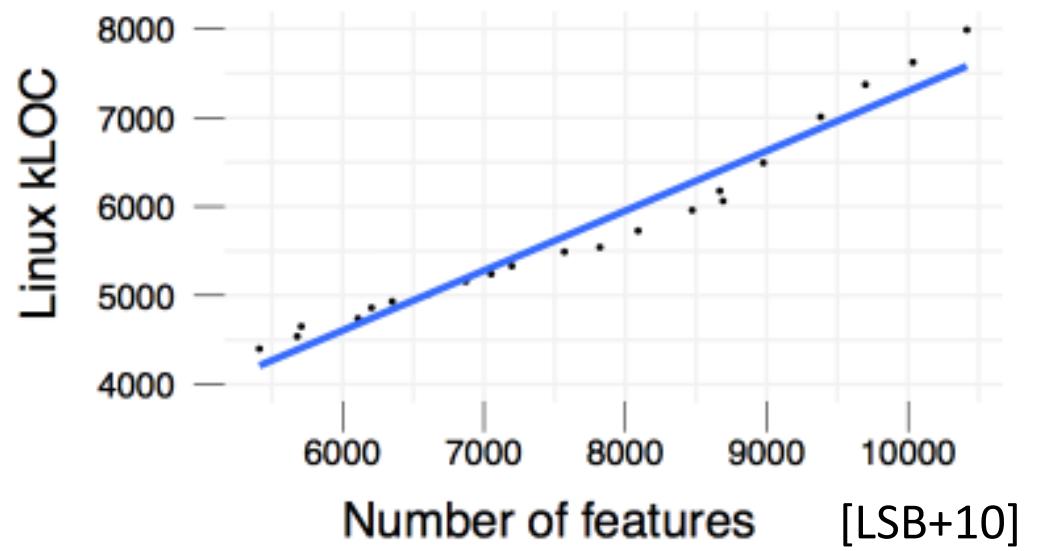
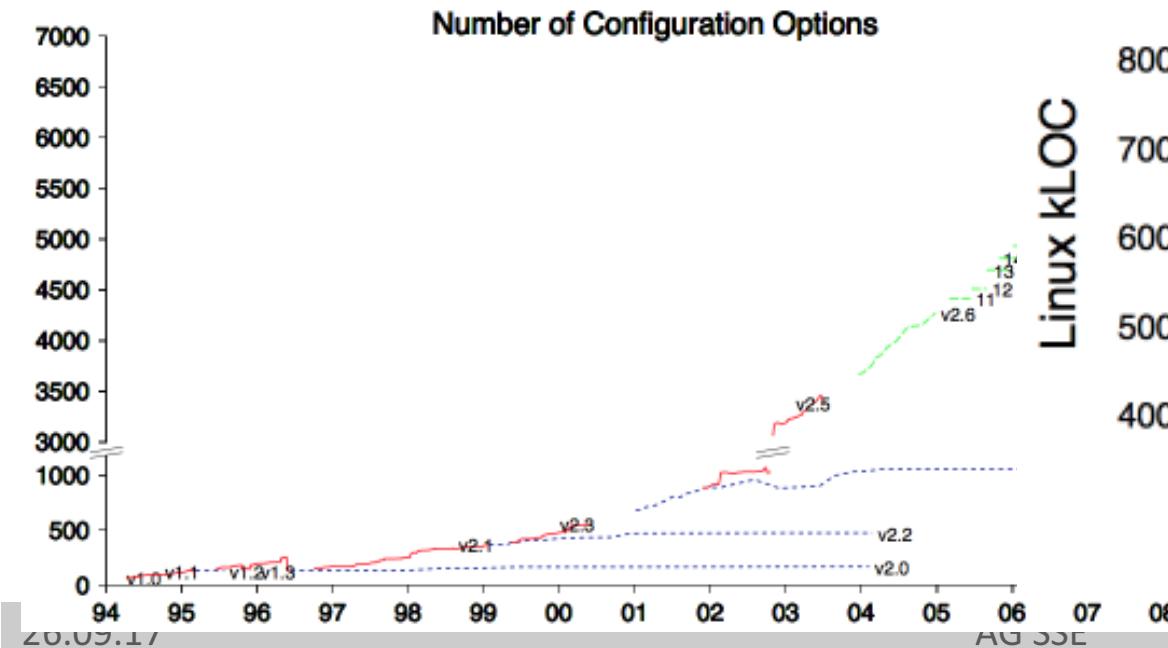
As a PL:

- 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

Actually ~20 PLs  
(1 per arch)

# Linux: Descriptive Statistics

- Size: similar to large industrial product lines
  - 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
  - Constantly growing..



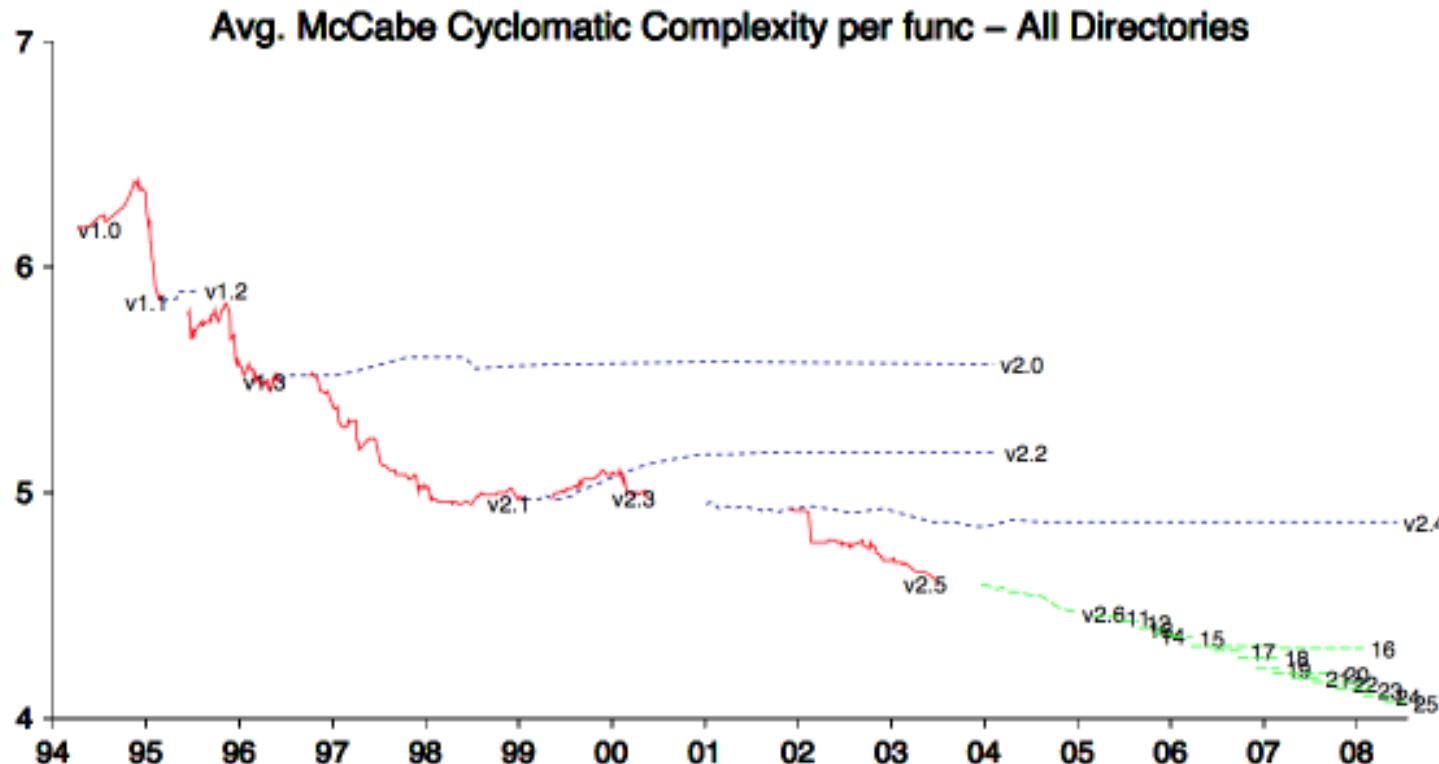
# Is it realistic?

## 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

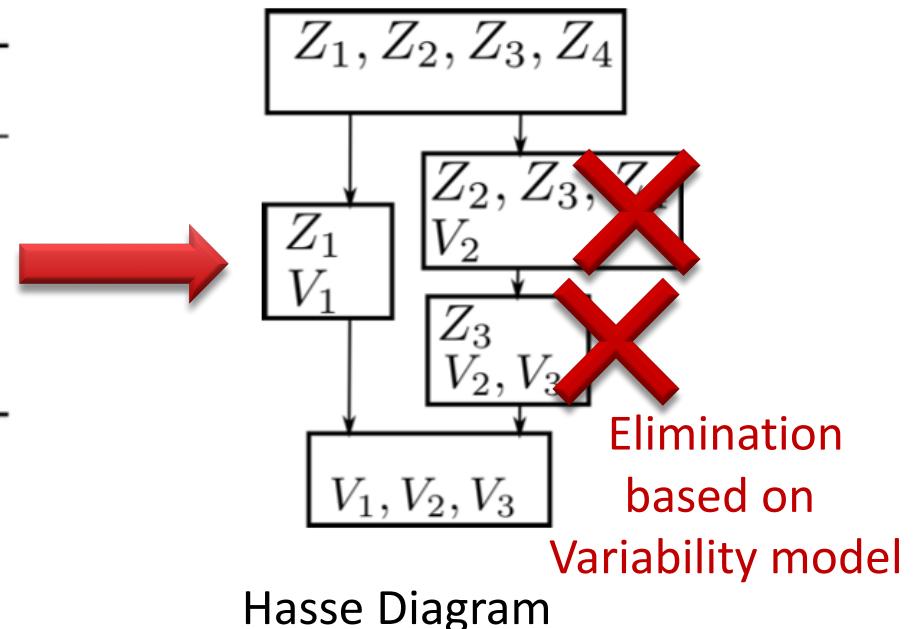
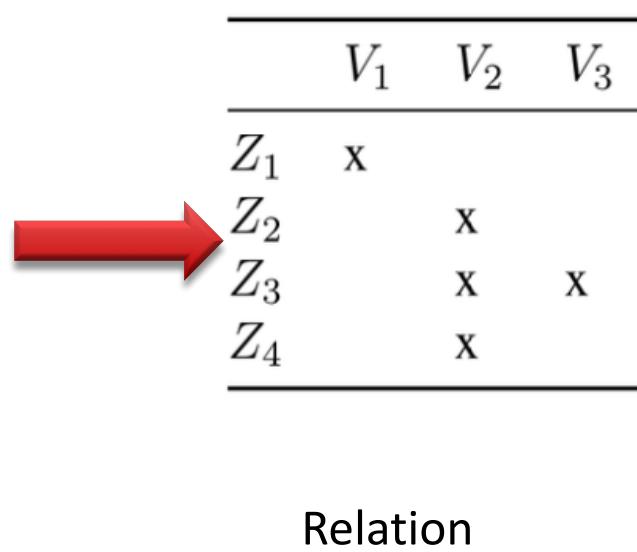


© Google Maps + own graphics

# Understanding Source Code

- Lattice for representing the variability

```
1 #if RUN
2 int i=y+x;
3 int z=0;
4 #endif
5 #if WAIT
6 while(x<z+1){
7 x=x*z++;
8 }
9 #if LOAD
10 load(x,z,i);
11 #endif
12 read(i);
13 #endif
```



Source code

[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 ( $X, \neg X$ )
    - 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
<b>0</b>	0	13180	-	-	-	-	-	-	-	-
<b>1-10</b>	2.1	7863	34152	4.3	3525	0.5	3507	0.5	8.41%	21s
<b>11-20</b>	14.0	129	2699	20.9	116	0.9	109	0.8	4.59%	4m
<b>21-30</b>	24.7	30	983	32.8	30	1.0	24	0.8	3.27%	9m
<b>31-40</b>	33.9	9	336	37.3	10	1.1	4	0.4	3.45%	6m
<b>41-50</b>	44.3	7	276	39.4	7	1.0	3	0.4	2.57%	4m
<b>51-60</b>	55.7	3	145	48.3	3	1.0	3	1.0	2.11%	8m
<b>61-70</b>	63.4	5	314	62.8	5	1.0	4	0.8	1.76%	15m
<b>71-80</b>	-	-	-	-	-	-	-	-	-	-
<b>81-90</b>	85.5	2	134	67.0	2	1.0	2	1.0	1.50%	12m
<b>91-100</b>	93.0	2	183	91.5	2	1.0	2	1.0	1.11%	33m
<b>&gt;100</b>	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

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

[LAS+16]

# Evolution Analysis

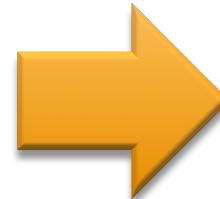
To what extend do commits really impact variability

- 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**

```
780 #ifdef CONFIG_X86_32
781     return ip > PAGE_OFFSET;
782 #else
783     return (long)ip < 0;
784 #endif
```

*Introduction of changes*



*Deletion of #else-block*

**Commit File**

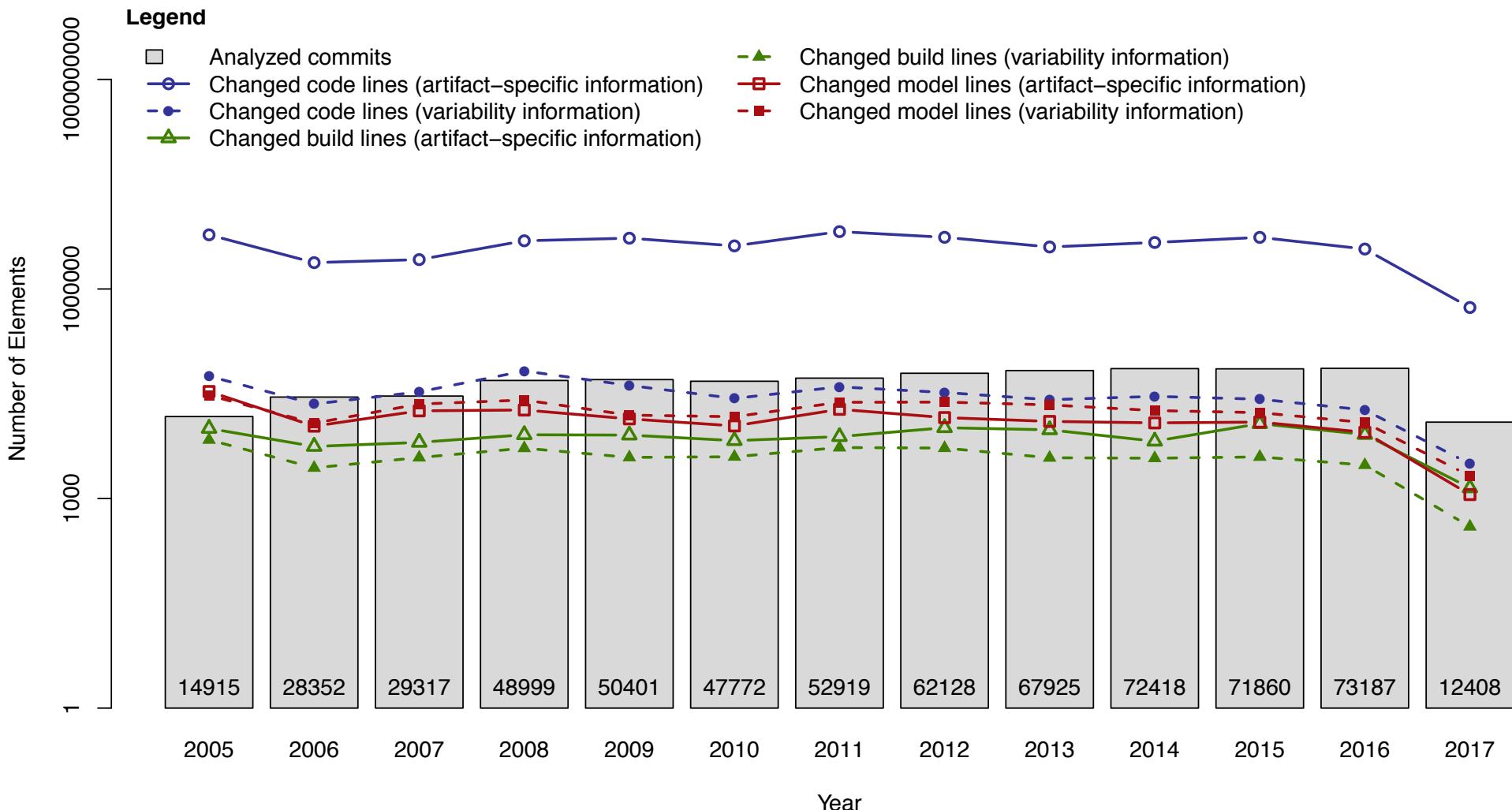
```
786 #ifdef CONFIG_X86_32
787     return ip > PAGE_OFFSET;
788 -#else
789 -    return (long)ip < 0;
790 #endif
```

*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

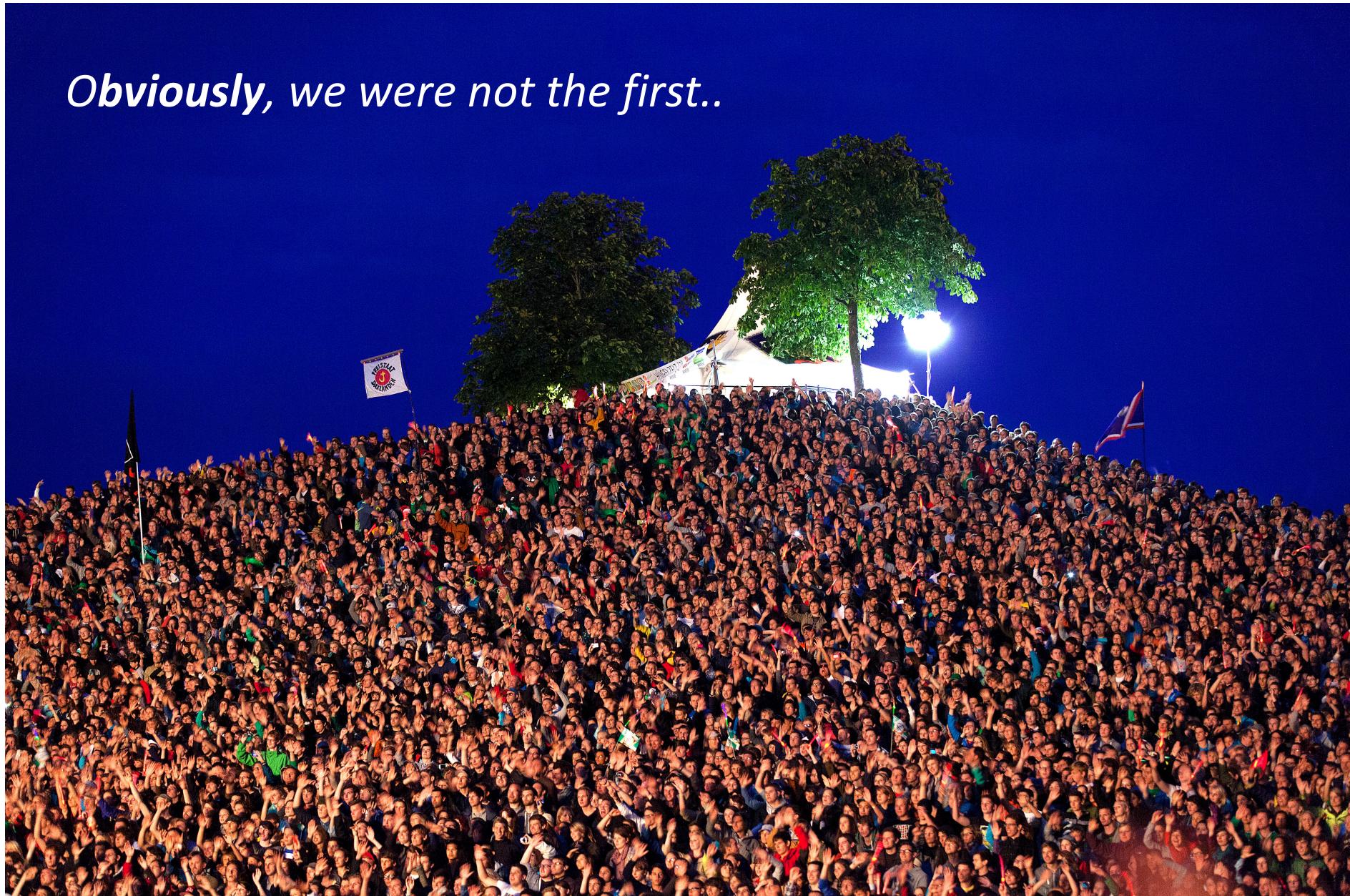
What was it like, when we started?



© Google Maps + own graphics

# Our journey

*Obviously, we were not the first..*



# Opportunities for reuse



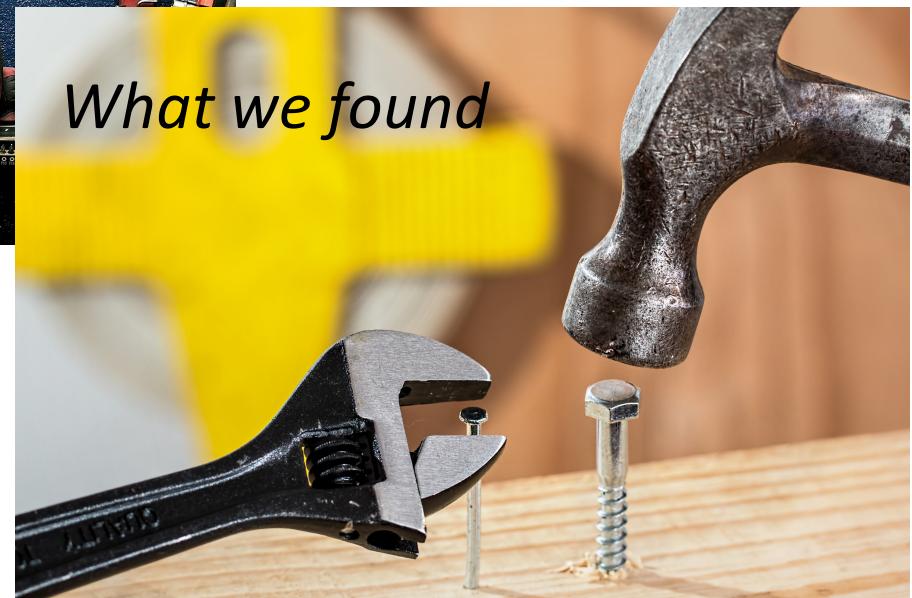
# Reusing existing tools



- Well-documented
- Easy to use and reuse

Considerable effort for:

- Learning
- Applying
- Adapting



# Reusing existing tools

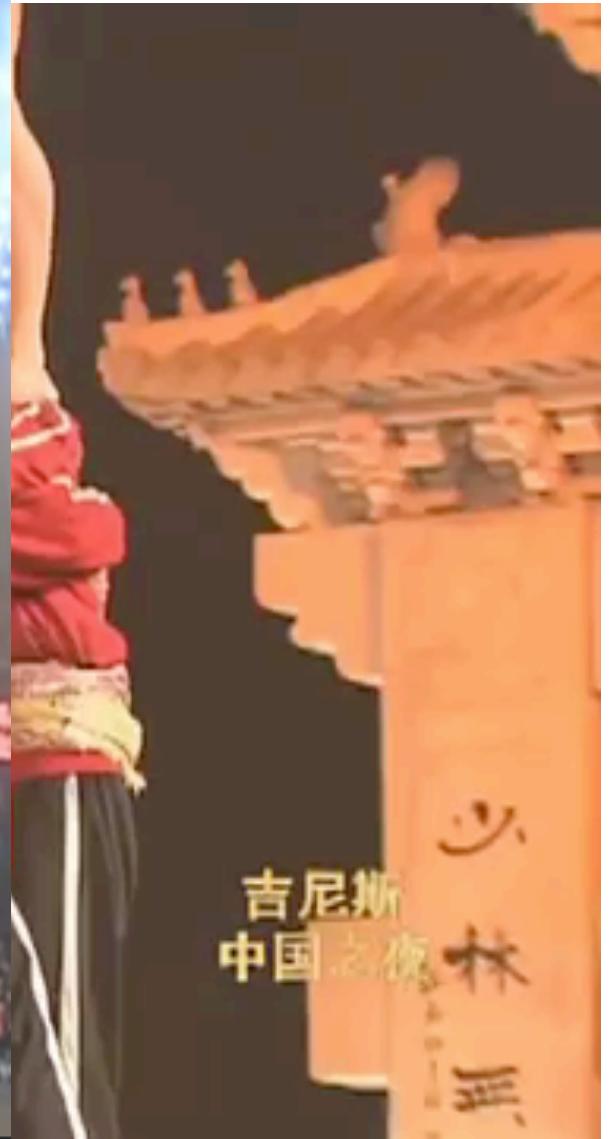
- Numerous tools
  - Extraction
    - Kconfig
    - Kbuild
    - Source-code
  - For analysis
  - Understand Evolution
  - For Visualization
- Issues:
  - What exactly is analyzed?  
(Correctness)
  - Reliability
  - Repeatability
  - Transferability



# Linux and its tools

- Numerous tools
  - Extraction
    - Kconfig
    - Kbuild
    - Source-code
  - For analysis
  - Understand Evolution
  - For Visualization
- Issues:
  - What exactly is analyzed? (Correctness)
  - Reliability
  - Repeatability
  - Transferability

# On the shoulders of giants



1:2

© youtube.com

# Analyzing KConfig

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

```
1 config BOOL_VAR
2   bool "A Boolean Variable"
3   select BOOL_VAL2
4
5 choice
6   bool "A Choice"
7
8   config BOOL_VAL1
9     bool "1st Boolean Value"
10
11  config BOOL_VAL2
12    bool "2nd Boolean Value"
13 endchoice
```

KConfig



```
1 c 1 BOOL_VAR
2 c 3 CHOICE_1
3 c 2 BOOL_VAL2
4 c 4 BOOL_VAL1
5 c 5 MODULES
6 p cnf 5 8
7 -1 2 0
8 -1 2 0
9 3 0
10 2 4 -3 0
11 -2 3 0
12 -4 3 0
13 -2 -4 0
14 -5 0
```

Dimacs

# Analyzing KConfig

		Config options	Choices	Hierarchies	Constraints	Attributes	If									
contains →		bool	tristate	string	numerical	bool	tristate	menu	via Constraints	depends on	select	prompt	default	range	visible if	
Config options	bool	x	x	x	x	x	x	x	✓	✓	✓	✓	✓	✓	x	x
	tristate															
	string															
	numerical															
Choices	bool	✓	⚠	⚡	+/-	+	x	⚠	⚠	⚠	⚠	⚠	✓	✓	⚠	⚠
	tristate	⚠	✓	✓	⚡	+/-	x	⚠	⚠	⚠	⚠	⚠	✓	✓	✓	⚠
Hierarchies	menu	✓		✓	✓	✓	✓	✓	✗	✓	✗	✓	✗	✗	✓	✓
	via Constraints	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	+
Constraints	depends on	✓	—	—	—	—	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗
	select	—	—	—	—	x	+	✓	✓	✓	✓	✓	✓	✓	✓	✓
Attributes	prompt	x		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	default	x		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	range	x		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	visible if	x		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
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 DIMACS	dumpconf RSF	rsf2model Model	modified DIMACS	dumpconf RSF	new dumpconf Model	vm2bool DIMACS	
Handling Attribute option modules	X	X	X	X	X	X	X	X
Constraint Precedence	✓	✓	X	✓	✓	✓	✓	✓
Missing Config Options	✓	✓	✓	✓	✓	✓	✓	✓
Config Options × select	✓	(P)	X	X	(P)	X	X	X
Boolean Config Options × default	X	✓	(P)	X	✓	(P)	✓	✓
Tristate Config Options × default	X	✓	✓	✓	✓	✓	✓	X
Tristate Choices × Boolean Config Options	X	✓	✓	—	✓	—	✓	✓
Boolean Choices × Tristate Config Options	X	✓	✓	X	✓	X	X	X
Choices × Hierarchies	✓	X	X	X	✓	X	✓	✓
Choices × (broken) Hierarchies	✓	X	✓	✓	✓	✓	✓	✓
Choices × if	✓	X	X	X	X	X	✓	✓
Choices × Constraints	X	✓	X	—	✓	—	—	X
Choices × prompt	X	X	X	X	✓	X	✓	—
default × default	✓	X	X	✓	✓	✓	✓	✓

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,
#ifndef CONFIG OMAP2_DSS_DSI
    dsi_init_platform_driver,
#endif
#ifndef CONFIG OMAP2_DSS_VENC
    venc_init_platform_driver,
#endif
#ifndef CONFIG OMAP4_DSS_HDMI
    hdmi4_init_platform_driver,
#endif
#ifndef 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



# 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*

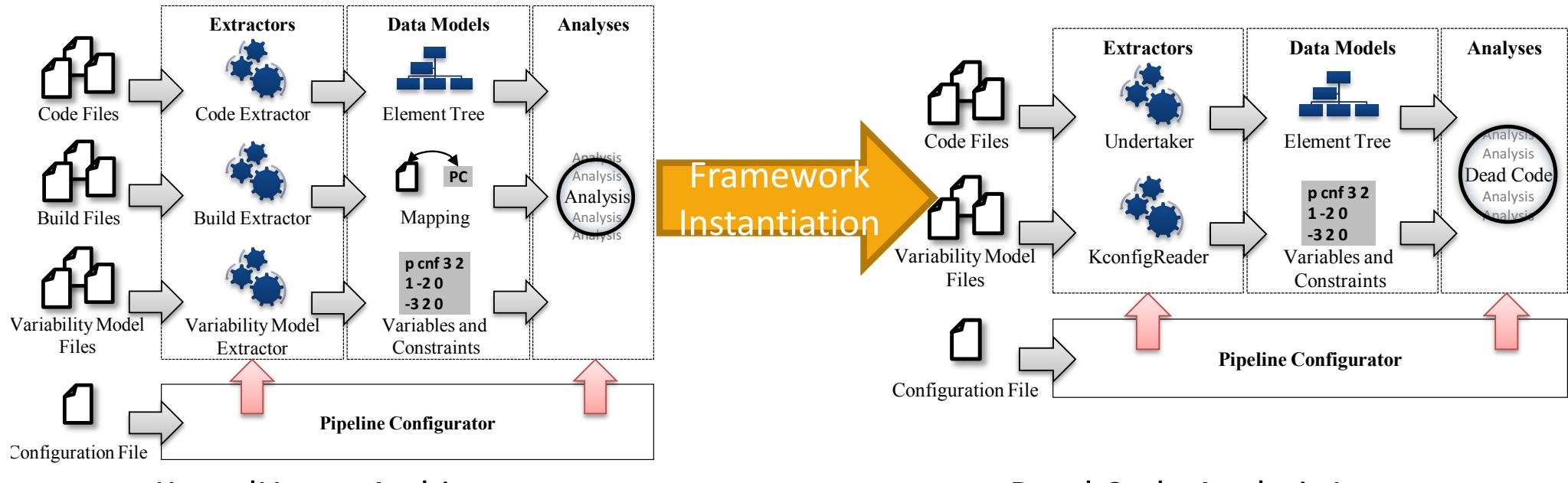
*Experimentation  
Workbench*

# 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>



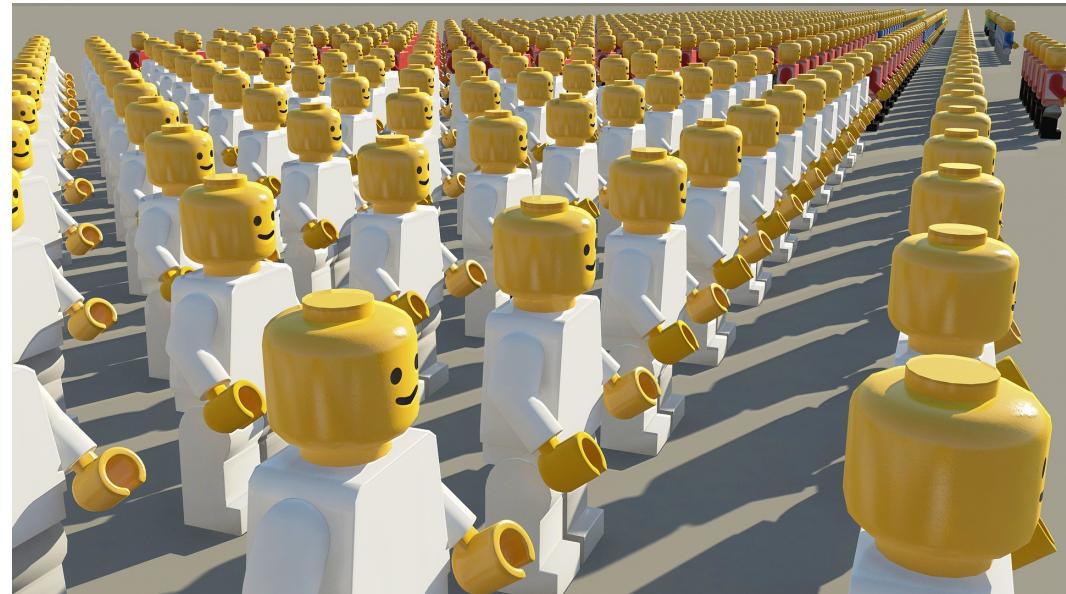
KernelHaven Architecture

Dead Code Analysis Instance

Auto-Archiving covers:  
Inputs, outputs, all code

# Kernelhaven

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



# Summary

## The scientific process



22.09.17

AG SSE

## Misconfigurations

1f

- you have 4 possible configurations
  - only three different products

Is this a problem?

Wednesday: 10:45

# An Empirical Study of Configuration Mismatches in Linux

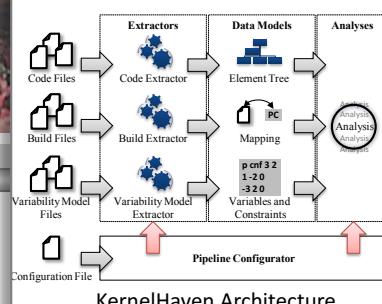
*Sascha El-Sharkawy, Adam Krafczyk and Klaus Schmid*

22.09.17

AG SSE

17

KernelHaven



## Auto-Archiving covers: Inputs, outputs, all code

22.09.17

AG SSE

Dead Code Analysis Instance

				X		✓		✓		✓	
				✓		✓		✓		✓	
				X		(P)		X		X	
				X		(P)		✓		✓	
34				✓		✓		✓		✓	
<b>default</b>											
Tristate Choices × Boolean Config Options		X		✓		✓		—		✓	
Boolean Choices × Tristate Config Options		X		✓		✓		X		X	
Choices × Hierarchies		✓		X		X		X		✓	
Choices × (broken) Hierarchies		✓		X		✓		✓		✓	
Choices × if		✓		X		X		X		✓	
Choices × Constraints		X		✓		X		—		✓	
Choices × prompt		X		X		X		✓		X	
default × default		✓		X		X		✓		✓	

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.

22.09.17

AG SSE

26

# 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 REVaMP<sup>2</sup>, 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.*

