Presentation for the Workshop on Open Source Software
And Intellectual Property in the Software Industry
Held at IDEI, University of Toulouse, 20th January 2005

Free/Libre & Open Source Software Development and 'the Economy of Regard'

By

Jean-Michel Dalle
Jean-Michel.Dalle@upmc.fr

Paul A. David
pad@stanford.edu

Rishab Aiyer Ghosh
rishab@dxm.org

Frank A.Wolak
wolak@stanford.edu

OUTLINE

1. Human motives, incentives and non-market resource allocation mechanisms
   - 1.1 “Rationales” vs. “motivation-at-the-margin”
   - 1.2 The “economy of regard” and *C-mode* development
   - 1.3 Norms and behavioral guidelines in *C-mode* projects
2. F/LOSS participation and code-signing behaviors
   - 2.1 FLOSS Survey testimony about code-signing
   - 2.2 The Linux kernel and its sub-projects (“modules”)
   - 2.3 The distribution of participation and credited code
3. An econometric model
   - 3.1 Specification and estimation results
   - 3.2 Interpretations
4. Discussion: implications and speculations

What is so very interesting for economists about the F/LOSS development process ?

- Collective, distributed mode of creating (producing) an information-good: software
- Extensive voluntary participation by communities of skilled and neophyte software developers
- Novel use of IPR to distribute/publish software under “public domain-like” conditions
- Essential dependence of the production mode upon the “anti-proprietary” distribution regime
- Critical role of computer-mediated communications (CMC) for this production system
- **Self-documenting nature of the process permits microlevel studies of ‘collective invention’**

Human motives, incentives and non-market resource allocation mechanisms

- The curious obsession among economists: what is motivating the voluntary efforts of F/LOSS developers?
- A multiplicity of candidate "motives" for human behavior
  - Conscious vs unconscious motives
  - Instrumental vs intrinsic satisfactions
  - Pecuniary vs non-pecuniary rewards
- Heterogeneity in the profiles of developers' "reasons" for being involved
- Individuals acquire "reasons" through action; motives may be "learned" in social interactions and so aren't stable
- "Rationales" vs. "motivations-at-the-margin"

Distribution of FLOSS Survey (2002) respondents among the main motivational groups identified by principal components analysis

Initial Motivational Groups	n	%
Enthusiasts	104	3.7
Software improvers	285	10.2
Recognition seekers	308	11.1
Materialists	327	11.7
Ideologists	472	17.0
'Triers' (= diffuse motives)	1288	46.3
Total	2784	100.0

Source: R. Glott et al., Motivations of Free/Libre and Open Source Developers. May 2004.

Motivational profiles of the FLOSS (2002) Survey respondents undergo change: continuation reasons evolve away from initial joining reasons

Flows from initial to continuing motivational groups within the FLOSS community

CONTINUING MOTIVATIONAL GROUPS	INITIAL MOTIVATIONAL GROUPS						
	enthusiasts	software improvers	recognition seekers	materialists	ideologists	'triers'	total
recognition seekers	33.7	8.0	41.6	14.1	7.2	6.8	12.0
skills improvers	32.7	22.8	37.0	26.3	43.6	31.1	32.5
software improvers	30.8	37.5	13.0	39.4	14.4	23.2	24.2
ideologists	2.9	33.7	8.4	20.2	34.7	39.9	31.2
total	100	100	100	100	100	100	100

Figures are percentages of the initial motivational groups.

Source: R. Glott, R. A. Ghosh, B.L. Krieger, Motivations of Free/Libere and Open Source Software Developers , May 2004

FLOSS-US

The Free/Libre/ Open Source Software Survey for 2003

To go immediately to the questionnaire, click here:

A Web Survey of Software Developers

conducted by the Stanford University (SIEPR) research project on

Economic Organization and Viability of Open Source Software

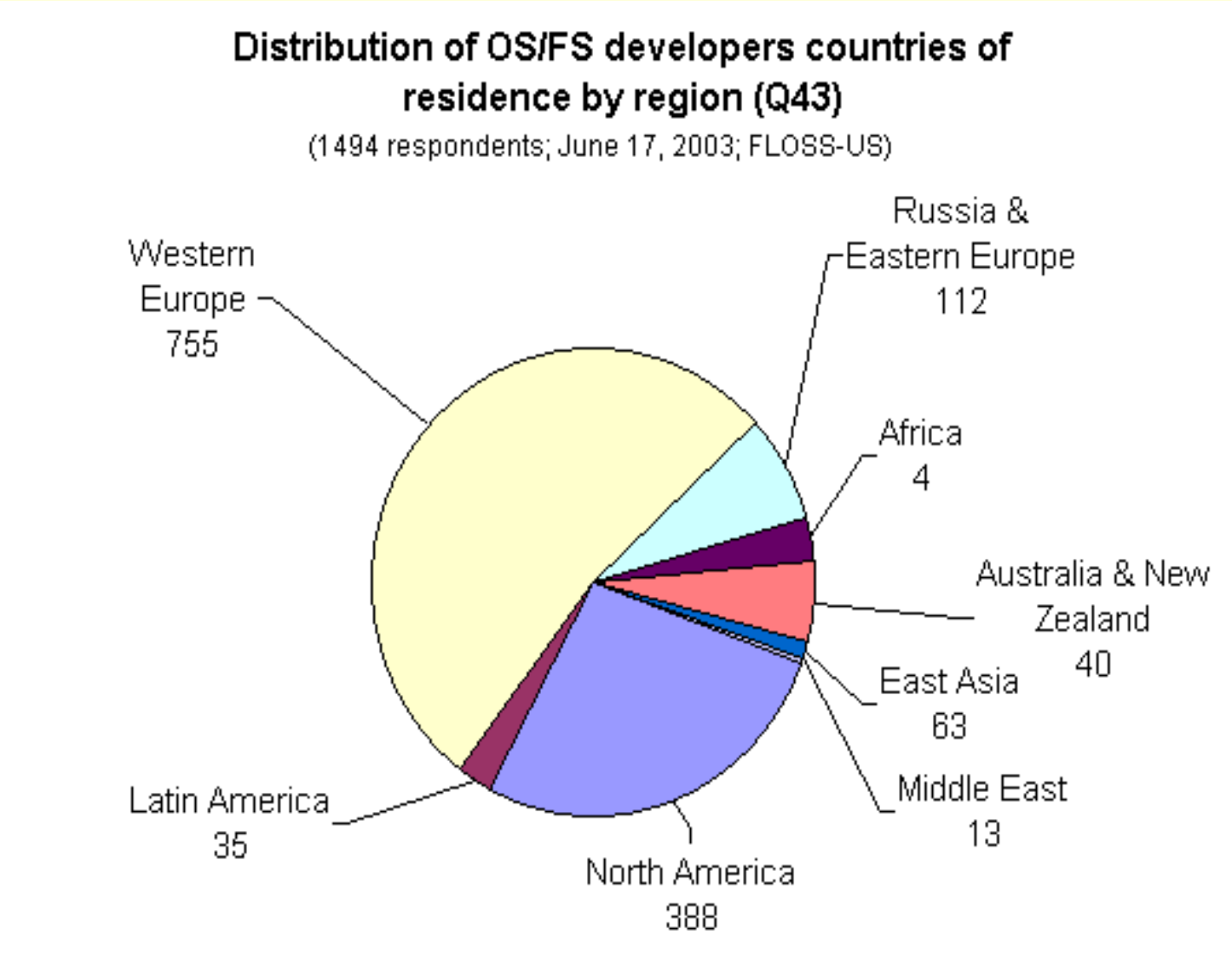
With funding support from the National Science Foundation.

Visit: **http://siepr.stanford.edu/programs/OpenSoftware_David/OS_Project_Funded_Announcmt.htm**

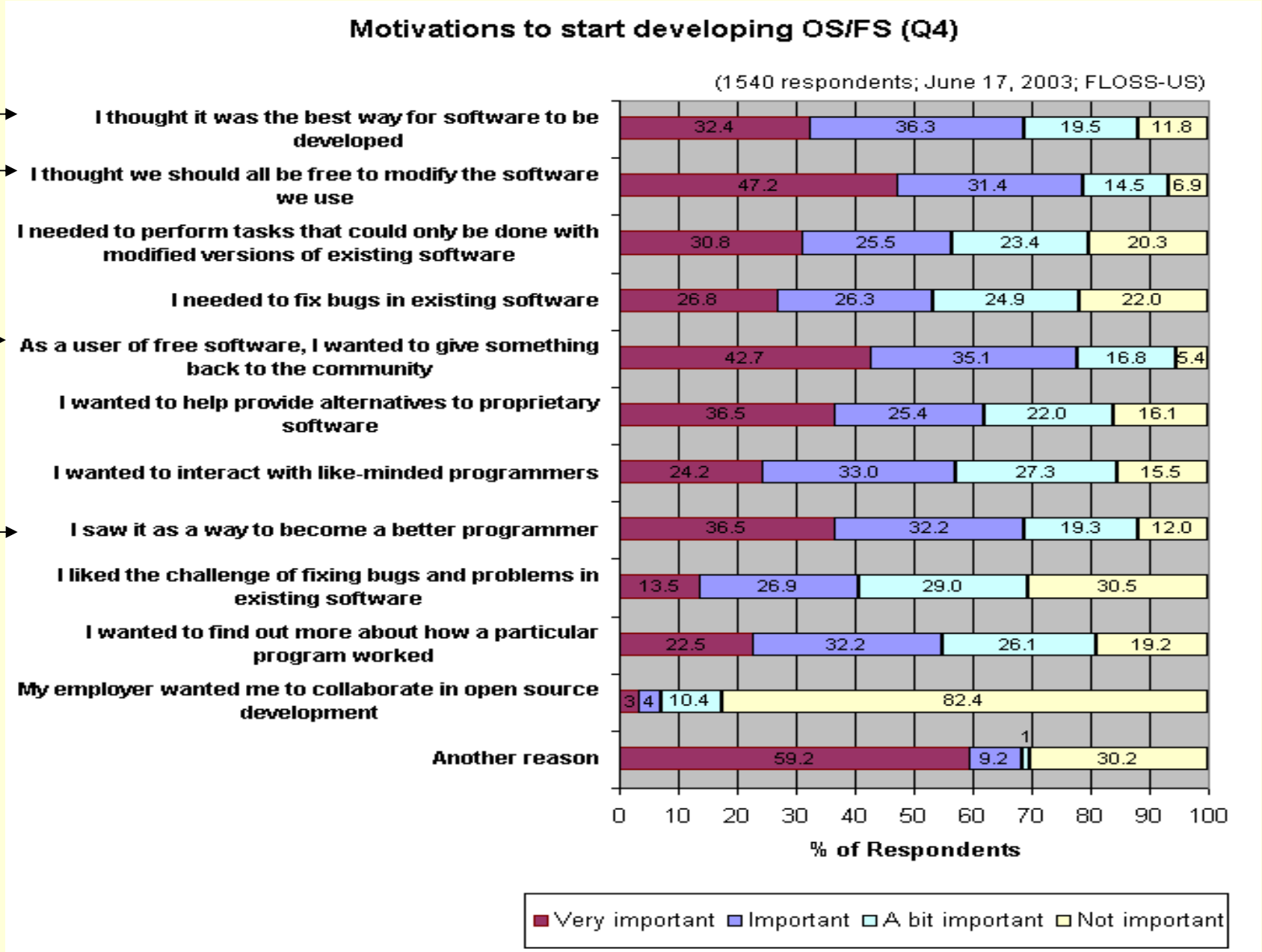
The FLOSS-US Survey: First Report (September 2003) is available at:

http://www.stanford.edu/group/floss-us/report/FLOSS-US-Report.pdf

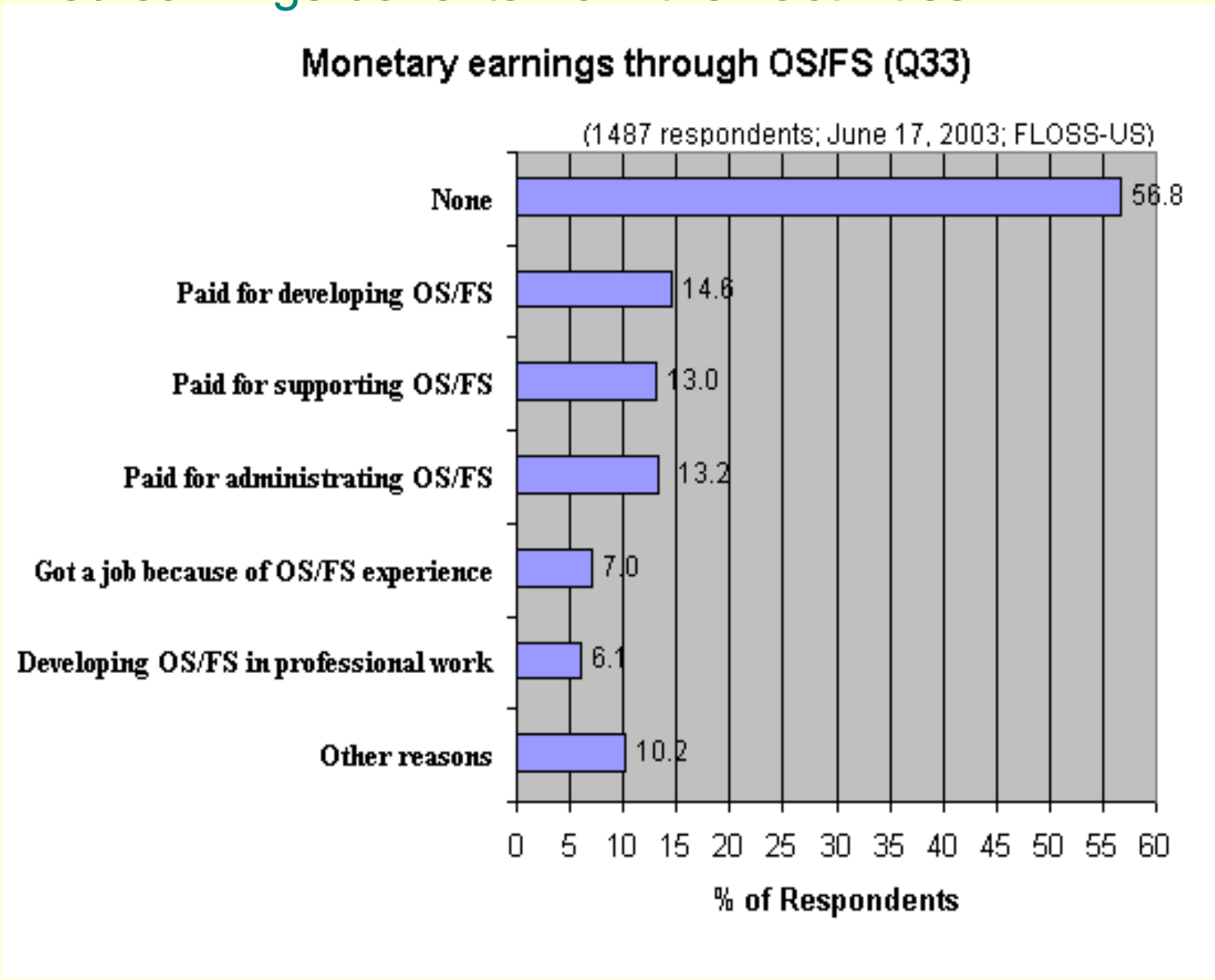
The relative representation of regions outside Western Europe in the SIEPR/NSF FLOSS-US (2003) Survey is c. 49%, compared with 30% in the EC FLOSS (2002) Survey



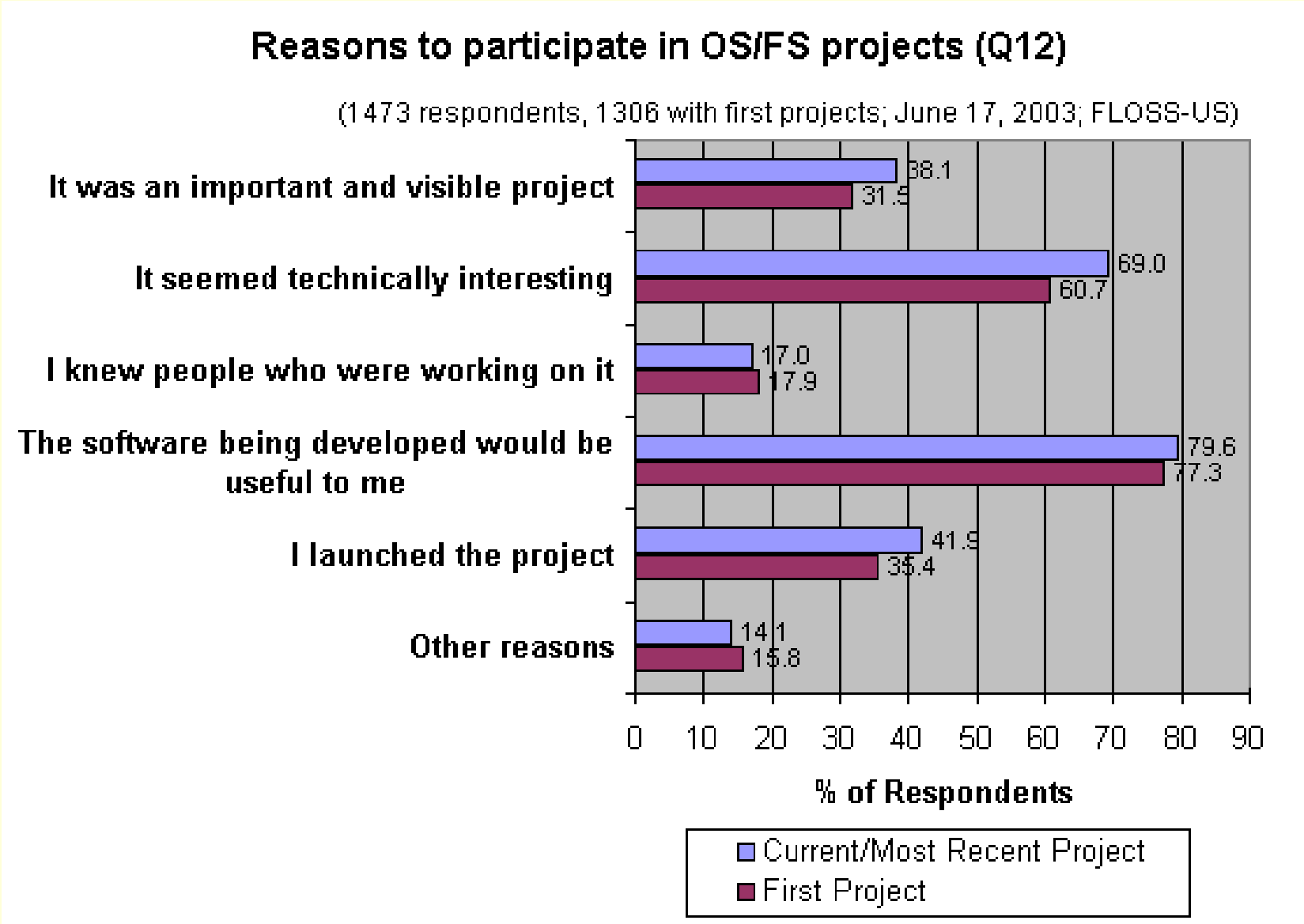
Still, ideological and self-improvement motives are salient among initial motivations of FLOSS-US (2003) developers



…and 56.8% of FLOSS-US respondents cite not having direct or derived earnings benefits from their activities.



But, FLOSS-US developers' explain their project *choices* in terms different from the reasons given for contributing...



The importance of "the personal utility of the software" among the reasons given by FLOSS-US (2003) respondents for their current project choices reflects the predominance of small, *I-mode* projects.

- Of 1473 respondents listing a "current project", 64.8% described it as "unknown" or "slightly known': 33.0% launched it alone;
  : 46.8% launched it with others.
- Of 1306 respondents listing their "first projects", 61.7% described it as "unknown" or "slightly known"; 35.4% launched it with others.
- Of 238 "newbies" (those starting a "first &current" project in 2001-03), 87.9% described it as unknown or slightly known; 42.4% launched it alone; 51.3% launched it with others.
- For respondents reporting the proportion of code they contributed to their "current project," the upper-tail of the distribution is:

Proportions of code	***All 1055 Respondents***	***238 "Newbies***
≥ 0.75	**44 %**	**54%**
≥ 0.95	**31 %**	**44%**

- Of 1451 respondents reporting code contributed to current projects, 58.9 % said ≥ 0.75 of their submitted code was included in the project's release version.

Note that 72% of SourceForge groups in 2003 had only 1 participant.

Shifting the focus from "motives" to "motivations-at-the-margin"

Conventional economic analysis is far more usefully engaged where, instead of providing an answer to the question "Why is this done?" the subject of the conversation is changed to "In what circumstances is this done?" and "When is rather more (rather than less) of this done?"

This finesse, substituting analysis of what might be referred to as "motivation at the margin," makes better use of the insights that the economist's métier can provide about the way specific incentives and constraint affect the incremental allocation of resources.

Consideration of "motivations at the margin" is more germane to understanding the coordination of F/LOSS development work performed in C-mode, i.e., by communities engaged on large and complex projects, rather than small, I-mode projects.

Some basic microeconomic questions about the F/LOSS production mode :

- **How are the human resource inputs mobilized in *C-mode*?**
- **What kinds of inputs are supplied by participants in *C-Mode*?**
- **How are these inputs allocated and coordinated within projects? (I.e., among tasks of a particular kind ,esp. coding, bug-fixing).**
- **What factors motivate participants to devote effort to particular sub-projects within a large and complex software system, e.g. Linux?**

QQ: Can surveys of developer motives help us answer these questions? What can be learned by analysis of the code structure and authorship at the project level?

Allocation mechanisms between the market and the gift: the "economy of regard" (Offer, 1997) --

- distinct from the classical conceptualization of the "gift economy"
- and positioned the intermediate space of non-market social systems
- involves voluntary, partially personalized (quasi-anonymous) transactions that

  are indirectly reciprocated

  subject to individual discretion in timing and magnitude.

Remark: Characterizing F/LOSS production in *C(ommunity)-mode* -- as contrasted with *I(ndependent)-mode* (a la Dalle and David 2003) situates these social organizations within the broader array of epistemic communities and institutionalized communities of practice that belong to "the economy of regard".

OUTLINE

2. F/LOSS within-project participation decisions and developers' self-identification (code-signing)

   2.1 Behavioral foundations for participants in *C-mode* projects

   2.2 FLOSS Survey responses re: code-signing

   2.3 The Linux kernel and its sub-projects ("modules"): developer contributions & technical dependencies

   2.4 The distribution of participation and credited code

Behavioral foundations for *C-mode* software development

Hypothesized *value norms* in the "economy of regard" governing resource allocation in a large project – Dalle & David's (2003) caricature of Raymond's *'Homesteading the noosphere'* (1999):

(a) Launching a new project is usually more valued than contributing to an existing project to which some contributions already have been made.

(b) Contribution to early releases typically are more valued than later versions.

(c) There is a hierarchy of "peer regard" attaching to the originally, and technical significance of elements in the code of a complex project:

i.e., contributing to the Linux kernel is (potentially) valued more highly than Linux implementation of an existing and widely used applications program; and the latter dominates writing an obscure driver for a new printer.

Behavioral foundations for *C-mode* software development

Hypothesized *value norms* in the "economy of regard" governing resource allocation in a large project -- continued:

(d) The hierarchy of peer-regard corresponds with (and possibly reflects) differences in the tree-like structure of meso-level *technical* dependences among the "modules"of a large project:

i.e., there is a lexicographic ordering of valuations for contributions, such that work on modules on which many other modules "call" are more highly rewarded than work on modules that "call" many others.

(e) New sub-projects are created in relation to existing ones, adding a new functionality, with corresponding diminution of peer-regard:

e.g., initiating a new module located one level higher in the 'tree'gains less peer esteem than does starting new nodes on the "lower branches".

Developers who are active (reporting participation in) many projects appear to attach greater importance to marking source code as theirs—in projects where that is permitted

Marking sourcecode as yours? v.degree of activity in OS/FS community Crosstabulation

% of responses within each degree of activity* group

		degree of activity in OS/FS community			
		low activity	medium activity	high activity	Total
Marking sourcecode as yours?	Yes, I consider this as very important	54.4%	61.0%	71.1%	57.8%
	Yes, but it is not important to me	37.8%	34.9%	25.3%	35.9%
	No	7.8%	4.1%	3.6%	6.3%
Total		100.0%	100.0%	100.0%	100.0%

Chi-Square Tests

	Value	df	Asymp. Sig. (2-sided)
Pearson Chi-Square	27.539[a]	4	.000
Likelihood Ratio	28.587	4	.000
Linear-by-Linear Association	24.998	1	.000
N of Valid Cases	2156		

a. 0 cells (.0%) have expected count less than 5. The minimum expected count is 10.47.

Source: Cross-tabulation from FLOSS Survey (2002) data, prepared by R. Glott. June 2004.

Developers who report larger weekly inputs of time on F/LOSS projects also tend to attach greater importance to marking source code as theirs

Marking sourcecode as yours? * Hours per week spent in FLOSS development Crosstabulation

% within Hours per week spent in FLOSS development

		Hours per week spent in FLOSS development						
		Less than 2 hours	2 - 5 hours	6 - 10 hours	11 - 20 hours	21 - 40 hours	More than 40 hours	Total
Marking sourcecode as yours?	Yes, I consider this as very important	52.4%	56.6%	59.2%	61.7%	63.8%	60.9%	58.0%
	Yes, but it is not important to me	38.5%	37.2%	36.7%	32.4%	28.1%	34.0%	35.6%
	No	9.1%	6.2%	4.0%	5.9%	8.0%	5.1%	6.4%
Total		100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%

Chi-Square Tests

	Value	df	Asymp. Sig. (2-sided)
Pearson Chi-Square	21.588[a]	10	.017
Likelihood Ratio	21.840	10	.016
Linear-by-Linear Association	9.620	1	.002
N of Valid Cases	2189		

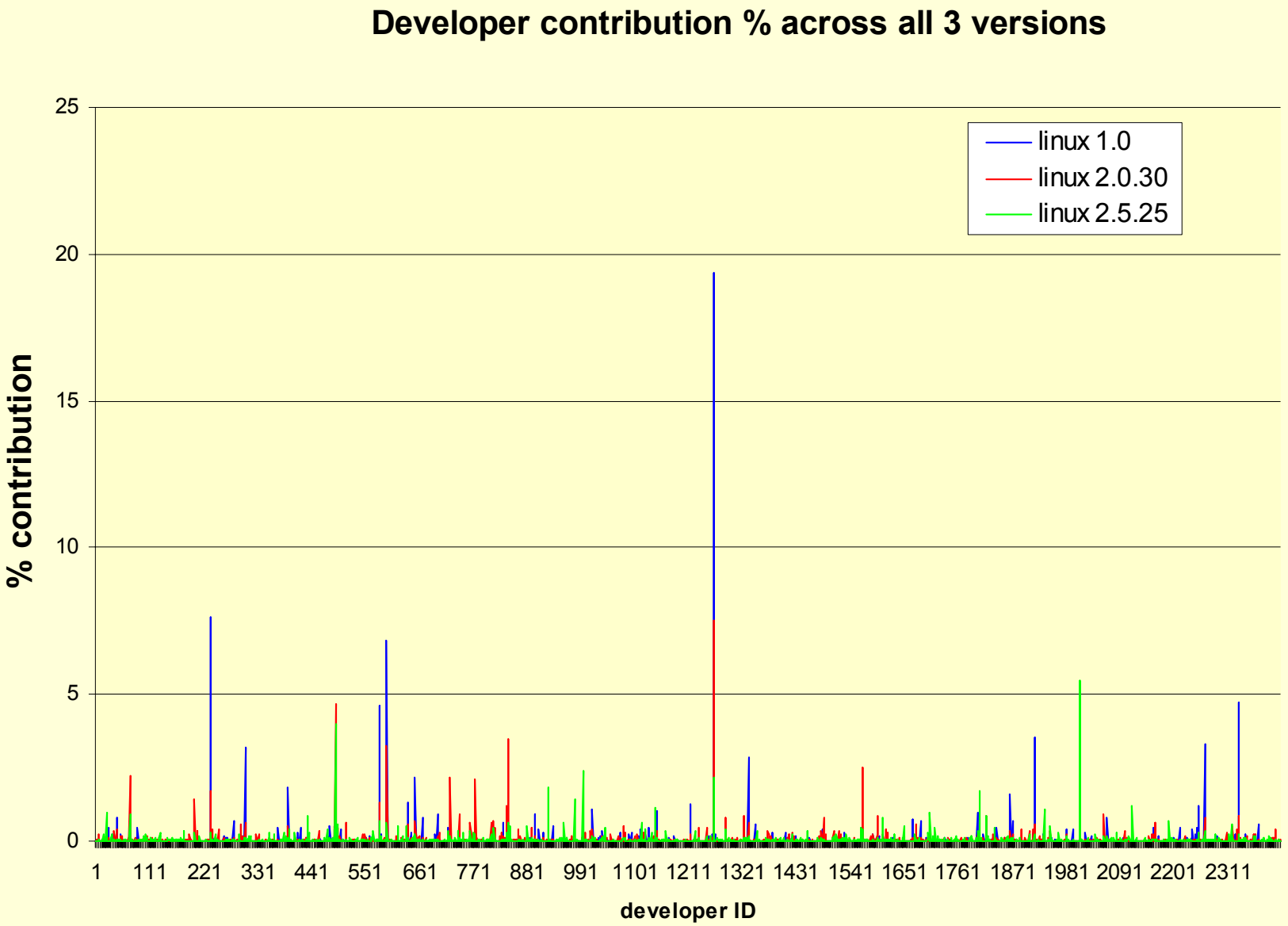
a. 0 cells (.0%) have expected count less than 5. The minimum expected count is 9.98.

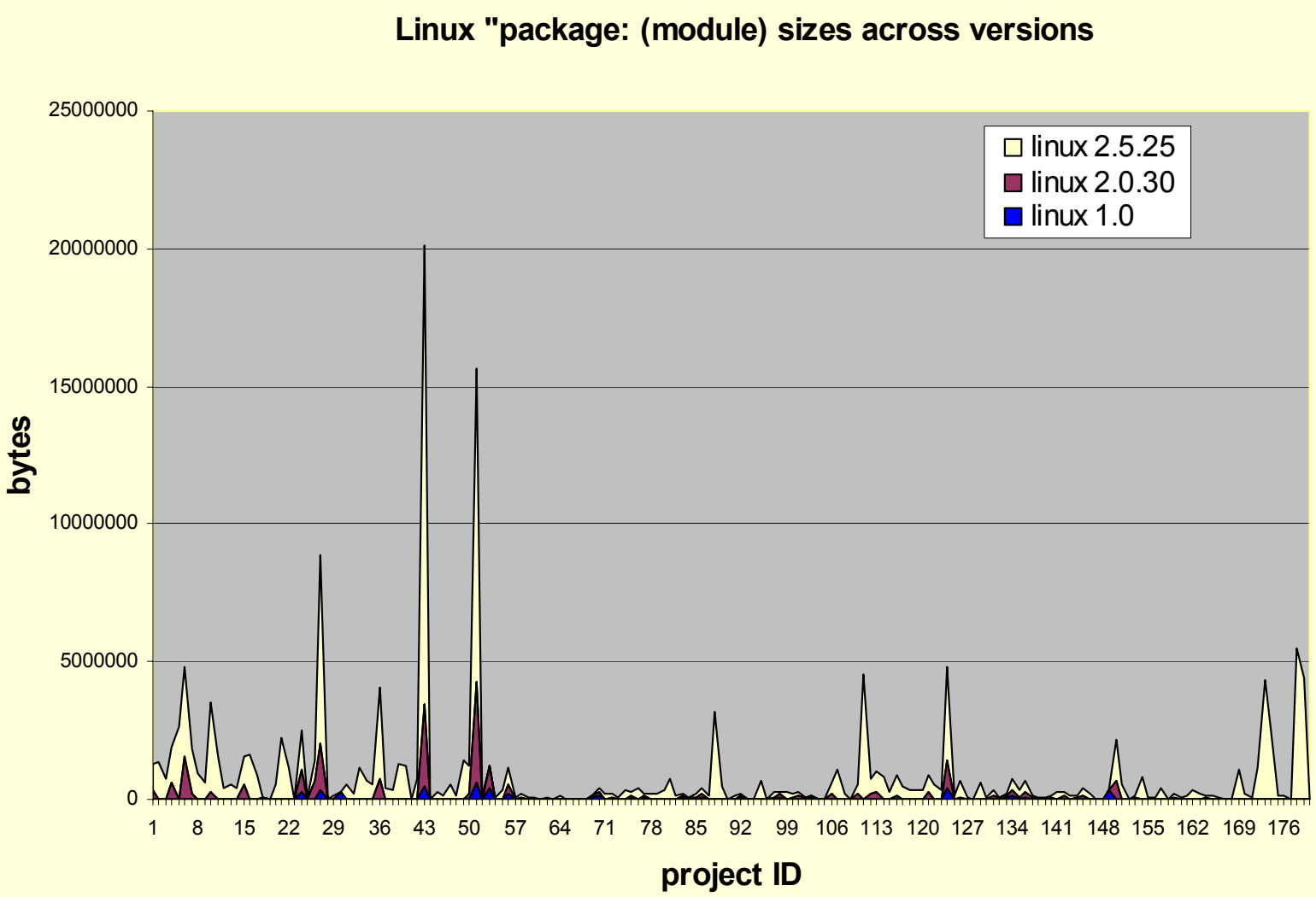
Source: Cross-tabulation from FLOSS Survey (2002) data, prepared by R. Glott. June 2004.

LICKS Project: Overview of Linux kernel code-base

Linux kernel	**Ver. 1.0**	**Ver.2.0.30**	**Ver.2.5.25**
Approximate release date	Mar-94	Apr-97	Jul-02
Number of “packages”*	30	60	168
Number of files	593	2,155	12,451
Number of defined functions*	1,748	7,808	48,006
Physical lines of code*	121,987	527,773	3,157,543
Bytes of code (millions)	4.54	21.05	133.85
Number of *identified* authors*	158	616	2,263
Percent of code “un-credited”*	18.8	12.2	14.9

*See Ghosh and David (2003): “packages” defined for LICKS;”authors” identified by CODD algorithm from email signature; “un-credited” bytes (KBOC): CODD found no signature.


Developer contribution % across all 3 versions
% contribution
25
20
15
10
5
0
linux 1.0
linux 2.0.30
linux 2.5.25
1
111
221
331
441
551
661
771
881
991
1101
1211
1321
1431
1541
1651
1761
1871
1981
2091
2201
2311
developer ID


Linux "package: (module) sizes across versions
25000000
20000000
15000000
10000000
5000000
0
bytes
linux 2.5.25
linux 2.0.30
linux 1.0
1 8 15 22 29 36 43 50 57 64 71 78 85 92 99 106 113 120 127 134 141 148 155 162 169 176
project ID

MODULES OF THE LINUX KERNEL (All 3 vers.): 164 out of total of 180

1 arch_alpha
2 arch_arm
3 arch_cris
4 arch_i386
5 arch_ia64
6 arch_m68k
7 arch_mips
8 arch_mips64
9 arch_parisc
10 arch_ppc
11 arch_ppc64
12 arch_s390
13 arch_s390x
14 arch_sh
15 arch_sparc
16 arch_sparc64
17 arch_x86_64
18 boot
19 Documentation
20 drivers_acorn
21 drivers_acpi
22 drivers_atm
23 drivers_base
24 drivers_block
25 drivers_bluetooth
26 drivers_cdrom
27 drivers_char
28 drivers_dio
29 drivers_fc4
30 drivers_FPU-emu
31 drivers_hotplug
32 drivers_i2c
33 drivers_ide
34 drivers_ieee1394
35 drivers_input
36 drivers_isdn
37 drivers_macintosh
38 drivers_md
39 drivers_media
drivers_message
drivers_misc
42 drivers_mtd
43 drivers_net
44 drivers_nubus
45 drivers_parport
46 drivers_pci
47 drivers_pcmcia
48 drivers_pnp
49 drivers_s390
50 drivers_sbus
51 drivers_scsi
52 drivers_sgi
53 drivers_sound
54 drivers_tc
55 drivers_telephony
56 fs
57 fs_adfs
58 fs_affs
59 fs_autofs
60 fs_autofs4
61 fs_bfs
62 fs_coda
63 fs_cramfs
64 fs_devfs
65 fs_devpts
66 fs_driverfs
67 fs_efs
68 fs_exportfs
69 fs_ext
70 fs_ext2
71 fs_ext3
72 fs_fat
73 fs_freexvfs
74 fs_hfs
75 fs_hpfs
76 fs_intermezo
77 fs_isofs
78 fs_jbd
79 fs_jffs
80 fs_jffs2
81 fs_jfs
82 fs_lockd
83 fs_minix
84 fs_msdos
85 fs_ncpfs
86 fs_nfs
87 fs_nfsd
88 fs_nls
89 fs_ntfs
90 fs_openpromfs
91 fs_partitions
92 fs_proc
93 fs_qnx4
94 fs_ramfs
95 fs_reiserfs
96 fs_romfs
97 fs_smbfs
98 fs_sysv
99 fs_udf
100 fs_ufs
101 fs_umsdos
102 fs_vfat
103 fs_xiafs
104 ibcs
105 include_asm
106 include_asmalpa
107 include_asm-ar
108 include_asm-cris
109 include_asm-generic
110 include_asm-i386
111 include_asm-ia64
112 include_asm-m68k
113 include_asm-mips
114 include_asm-mips64
115 include_asm-parisc
116 include_asm-ppc
117 include_asm-ppc64
118 include_asm-s390
119 include_asm-s390x
120 include_asm-sh
121 include_asm-sparc
122 include_asm-sparc64
123 include_asm-x86_64
124 include_linux
125 include_math-emu
126 include_net
127 include_pcmcia
128 include_scsi
129 include_sound
130 include_video
131 Infraestructure
132 init
133 ipc
134 kernel
135 lib
136 mm
137 net
138 net_802
139 net_8021q
140 net_appletalk
141 net_atm
142 net_ax25
143 net_bluetooth
144 net_bridge
145 net_core
146 net_decnet
147 net_econet
148 net_ethernet
149 net_inet
150 net_ipv4
151 net_ipv6
152 net_ipvx
153 net_ipx
154 net_irda
155 net_khttpd
156 net_lapb
157 net_llc
158 net_netlink
159 net_netrom
160 net_packet
161 net_rose
162 net_sched
163 net_sunrpc
164 net_unix

The persistence and generation of un-credited code across versions of the Linux kernel

Modules present in	Version 2.0.3	Version 2.5.25
No. with >20% of bytes uncredited:	11	24
No. with >20% of bytes uncredited in 2.5.25 that also were present in 2.0.3:	11	
Mean % of bytes uncredited in the recurring modules with >20% uncredited	45.6	48.0

OUTLINE

3. An econometric model

   3.1 Specification and estimation results

   3.2 Interpretations

THE MODEL OF CODE-SIGNING

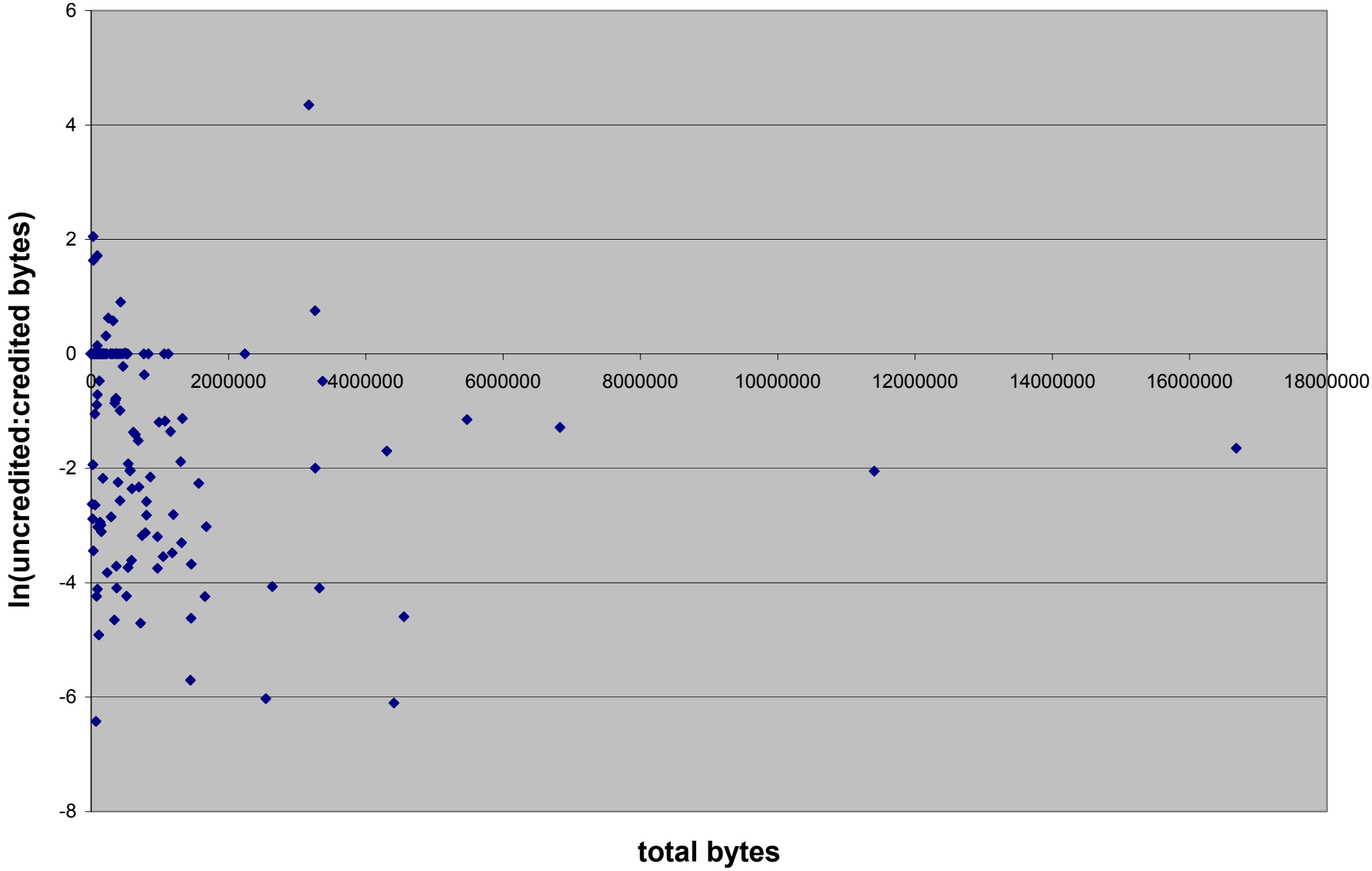
Define the following three **dependent variables:**

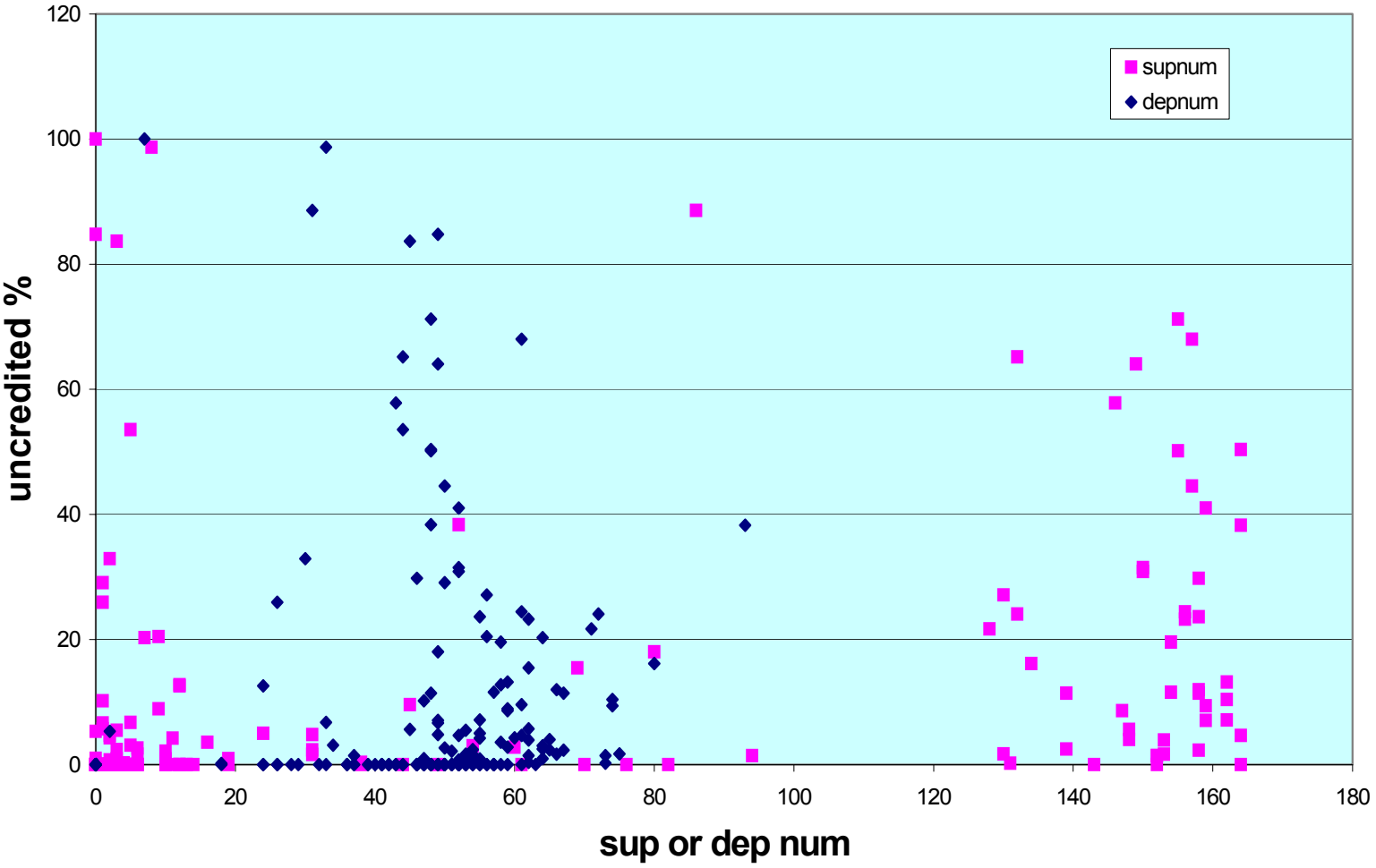
$\mathbf{y_{1t}}$ = ***log(uncredit/(numbytes - uncredit))***
= logarithm of ratio of uncredited to credited bytes in the package assuming both uncredited and credited bytes are positive

$\mathbf{y_{2t}}$ = ***log(ndevelop)***
= logarithm of total number of developers that worked on package (only those cases that signed = 1 is the value of ***totdev*** observed, which corresponds to ***log(ndevelop)***.

$\mathbf{y_{3t}}$ = dummy variable that equals 1 if all of the bytes in package *t* are credited (belong to physical lines of codes that were signed).

Associated with each dependent variable is a set of **regressors** – $\mathbf{X_{1t}}$, $\mathbf{X_{2t}}$, and $\mathbf{X_{3t}}$, respectively.


linux25 - loguncr vs. totalbytes
ln(uncredited:credited bytes)
6
4
2
0
-2
-4
-6
-8
0
2000000
4000000
6000000
8000000
10000000
12000000
14000000
16000000
18000000
total bytes


linux 25 - uncredited % vs. dependency count measures
uncredited %
sup or dep num
supnum
depnum
0
20
40
60
80
100
120
140
160
180

THE MODEL OF CODE-SIGNING: SPECIFICATIONS

Posit the following three structural equations:

$$\mathbf{y}_{1t} = \mathbf{X}_{1t}'\beta_1 + \alpha\, \mathbf{y}_{2t} + \epsilon_{1t} \qquad \textbf{(1)}$$

$$\mathbf{y}_{2t} = \mathbf{X}_{2t}'\beta_2 + \epsilon_{2t} \qquad \textbf{(2)}$$

$$\mathbf{y}_{3t} = \mathbf{X}_{3t}'\beta_3 + \epsilon_{3t} \qquad \textbf{(3)}$$

where

$\mathbf{y}_{3t} = \mathbf{1}$ **if** $\mathbf{y}_{3t}^* > \mathbf{0}$, and $\mathbf{y}_{2t}$ is observed and $\mathbf{y}_{1t}$ is not observed;

$\mathbf{y}_{3t} = \mathbf{0}$ **if** $\mathbf{y}_{3t}^* \leq \mathbf{0}$ and $\mathbf{y}_{2t}$ is only known to exceed

$$\mathbf{y}_{2t}^{\textbf{sign}} = \textit{log(ndevelop)},$$

and $\mathbf{y}_{1t}$ is observed.

ESTIMATING THE CODE-SIGNING MODEL

Assume that $\epsilon_t = (\epsilon_{1t}, \epsilon_{2t}, \epsilon_{3t})'$ is a mean zero normally distributed random vector with covariance matrix Ω,

$$\Omega = \begin{bmatrix} \omega_{11} & \omega_{12} & \omega_{13} \\ \omega_{12} & \omega_{22} & \omega_{23} \\ \omega_{13} & \omega_{23} & 1 \end{bmatrix} \tag{4}$$

Define $\beta = (\beta_1, \beta_2, \beta_3)$

The log-likelihood for this model can be written as:

$$L(\beta,\alpha,\Omega) = \sum_{t=1}^{T} y_{3t} \ln\left\{ \int_{-\infty}^{\infty} \int_{-\infty}^{X_{3t}'\beta_c} \phi(z,(y_{2t} - X_{2t}'\beta_2),x \mid \Omega)\,dx\,dz \right\}$$

$$+(1-y_{3t}) \ln\left\{ \int_{y_{3t}^{sign}}^{\infty} \int_{X_{3t}'\beta_3}^{\infty_c} \phi((y_{1t} - (X_{1t}'\beta_2 + (X_{2t}'\beta_2)\alpha),(z - X_{2t}'\beta_2),x \mid \Omega^*)\,dx\,dz \right\} \tag{5}$$

where $\phi(x,y,z \mid \Omega)$ is density of a multivariate $N(0, \Omega)$ random variable,

and Ω^* is the covariance matrix of the multivariate normal random variable.

Maximum Likelihood Estimates for the Model of Code-signing

Variable	Parameter Estimate	Standard Error	t-statistic
	Equation 3		
Constant	6.466	1.004	6.437
Log(numbytes)	-0.527	0.080	-6.582
	Equation 2		
Constant	-6.012	0.866	-6.946
supnum	0.00197	0.01234	0.159
depnum	0.01824	0.00720	2.533
supnum*depnum	6.79E-03	2.172E-02	0.312
Log(numbytes)	0.6352	0.0769	8.260
Linux_2.0	0.7295	0.1888	3.863
	Equation 1		
Constant	-2.416	1.456	-1.659
supnum	0.00913	0.00385	2.373
depnum	-0.04526	0.02423	-1.868
Log(total_developers)	0.31669	0.31597	1.002
	Covariance Parameters		
Ω_{11}*	3.6323	0.8276	4.388
Ω_{22}*	1.1386	0.2101	5.421
corr(ϵ_{1t}*,ϵ_{2t}*)	0.4442	0.2384	1.863
corr(ϵ_{1t}*,ϵ_{3t}*)	-0.4931	0.2635	-1.871
corr(ϵ_{2t}*,ϵ_{3t}*)	-0.8552	0.0507	-16.884

Linux_2.0 = 1 if module appeared in Linux version 2.0; = 0 if module did not appear in Linux 2.0

Interpreting the behavioral evidence from the Linux kernel

General remarks:

- Developers are heterogeneous with regard to their capability and willingness to contribute: there are a group of *major, core developers (MCDs)* whose large code contributions are especially salient during the early life of the project, but whose relative contribution to the code declines over the life of the project.

- Code-signing as a means of gaining recognition and ‘peer regard’ is likely to be less instrumentally important for individuals who already have attained salience and high reputational status within the developer community. Recognized expertise, as well as the desire for ‘peer regard’ may play a role in the allocation of developers’ efforts among the various modules in a project.

- The technical characteristics of the modules, particularly their dependence and supportive position vis-à-vis other packages of code within the project, are found to exert significant systematic effects upon both the extent of developer participation in the module, and the propensity for contributions to be signed.

Empirical results on participation in project-modules

The number of developers contributing to a module is an *increasing* function of:

(a) the size of the package (in bytes);

(b) the number of other modules that depend upon ("call") the package, that being a measure of its technical importance.

The average amount of code contributed (per developer) increases with the size of the package (in bytes). This can be interpreted as reflecting either or both of the following motivational conditions:

(i) Modules that are more complex and whose architecture requires more code (because of their technical functionality) tend to be particularly attractive for the *MCDs* –i.e., those who contribute above average amounts of code.

(ii) Gaining peer attention requires disproportionately greater average efforts from individual developers (gauged by the volume of code contributed) when the package grows larger.

Empirical results on participation in project-modules -- continued

Holding constant size and technical characteristics, modules in Vers. 2.5 that were of Vers. 2.0 “vintage” attract a larger number of contributing developers.

Holding constant the size and vintage of the module, a higher absolute dependency value (*depnum)* positively affects the number of developers that contribute to it.

Remark:

The latter result is consistent with the view that the entry standards (in terms of expertise and the magnitude of the effort required for “commits”) tend to be set lower when *depnum* is larger, permitting a larger number of participants to contribute to the technically less critical modules.

Results on the probability of code being signed

From the Equation 1 estimates it is found the “log odds” – the natural logarithm of the ratio between uncredited and credited bytes in a package—

- varies positively with the support value (*supnum)* and negatively with the dependency value (*depnum)* of the module;

- is unaffected by the number of developers contributing to a module.

Remark:

These findings may be read as consistent with the interpretation advanced for the estimation results on the effects of the technical dependency characteristics of the modules upon the numbers of developers contributing in a module of given (kilobyte) size.

Interpreting the results on the probability of code being signed

The significant "effects" of the modules' technical features on the proportion of uncredited code may be interpreted as reflecting unobserved heterogeneity in the participating developers, under the following suppositions:

- *MCDs* are more concentrated among the contributors to the technically more critical (high support value) packages. But as they are more likely already to have gained the recognition of their peers (and the admiration of neophytes and journeymen programmers), they are less strongly motivated to sign all the code they contribute.

- The ritual of code-signing is followed more assiduously by those who have yet to attain peer recognition and high status in the community. Such individuals form the mass of participants, and they find it easier to make contributions to modules that have higher dependency values – given the (less exacting) standards for "commits" to those modules.

Remark: Supposing two forms of unobserved heterogeneity – i.e., in the motivations of core and peripheral developers, and in the programming standards for modules with different *supp/dep* values – leaves this interpretation less solidly grounded than one would wish.

Discussion

Broad observations and implications

- Heterogeneity of F/LOSS projects may extend to the particular nature of the ‘value norms’ attaching to tasks, making it difficult to generalize from broad survey data (e.g., about importance of code-signing to developers).
- Heterogeneity among the developers associated with a large project at different points in its history, may limit the applicability of the conceptualization of general and static norms characterizing the relevant ‘economy of regard’.

Speculations on future work in this line

- Can contributions of CMDs be identified in earlier releases?
- Can one date the origins of extensive un-credited code?
- Can authorship distributions help explain the distribution of unsigned code among modules?