



Free/Libre and Open Source Software Metrics

Sponsored through Framework Programme Sixth (Call 5) by



The FLOSSMetrics Consortium consists of: Universidad Rey Juan Carlos, University of Maastrich, Wirtschaftsuniversitaet Wien, Aristotle University of Thessaloniki, Conecta s.r.l., Zea Partners and Philips Medical Systems PMS Nederland B.V.

Document Information

Version: 1.0
Date : Oct 31, 09
 revision: 0

Owning Partner: UM

Author(s):
 Kirsten Haaland
 Sulayman Sowe

Reviewer(s):
 Rishab Ghosh
 Stefan Koch

To:
 Public

Purpose of distribution:
 Final version

Printed on at

Status:

- Draft
- To be reviewed
- Proposal
- Final/Released

Confidentiality:

- Public - Intended for public use
- Restricted - Intended for FLOSSMETRICS consortium only
- Confidential - Intended for individual partner only

Deliverable ID: D11.2

Title:
 Productivity study report

License for distribution:

This work is licensed under a [Creative Commons Attribution-Share Alike 3.0 License](http://creativecommons.org/licenses/by-sa/3.0/)
 (The license can be found in <http://creativecommons.org/licenses/by-sa/3.0/>)



Productivity study report

Deliverable ID: D11.2

Page : 2 of 30

Version: 1.0

Date: Oct 31, 09

Status : Final

Confid : Public


Deliverable: D11.2

Title: Productivity study report

Executive Summary:

This deliverable is the formal deliverable of the high level study investigating the productivity of FLOSS programmers in economic terms; it is also included as a paper in the overall high level study overview in WP5.

We have investigated the productivity of open source software developers in a sample of FLOSS project from the FLOSSMETRICS database. Combining data from both CVS logs and a survey we previously have conducted, with a cohort of 159 developers respondents. We focus on the primary production of FLOSS code, and the actual costs (in terms of actual coding effort/time). We find that the primary production, i.e. the direct relationship between time input into coding and the code output explains 14,8% of the relationship, and that being employed by a firm to do development is the single most important aspect for predicting increased productivity.

	<p>Productivity study report</p> <p>Deliverable ID: D11.2</p>	Page : 3 of 30
		Version: 1.0
		Date: Oct 31, 09
		Status : Final Confid : Public

CHANGE LOG

Ver.	Date	Author	Description
0.1	22/05/2009	Kirsten Haaland	Initial proposal for structure
0.2	25/06/2009	Kirsten Haaland	Introduction and preliminary analysis
0.3	17/07/2009	Sulayman Sowe	In depth analysis
0.4	09/10/2009	Kirsten Haaland	Write-up
0.5	15/10/2009	Rishab Ghosh	Minor final changes
1.0	29/10/2009	Stefan Koch	Review and release

APPLICABLE DOCUMENT LIST

Ref.	Title, author, source, date, status	Deliverable Identification



	<p>Productivity study report</p> <p>Deliverable ID: D11.2</p>	Page : 4 of 30
		Version: 1.0
		Date: Oct 31, 09
		Status : Final Confid : Public

TABLE OF CONTENTS

1. Introduction.....	5
2. Background and related work.....	6
2.1 Software cost estimation models.....	6
2.2 Increasing or decreasing returns to scale.....	7
2.3 Other output.....	8
2.4 Estimating Proprietary project versus open source.....	8
3. Research methodology.....	10
3.1 CVS data.....	11
3.2 Survey data.....	11
4. Analysis and Discussions.....	13
4.1 Nonparametric correlations between coding output and time spent.....	16
4.2 Modeling developers productivity.....	17
4.3 Trends in weekly gross output and coding weekly hours.....	20
4.4 Other factors to consider	22
5. Conclusion.....	28
6. References.....	29


	<p>Productivity study report</p> <p>Deliverable ID: D11.2</p>	Page : 5 of 30
		Version: 1.0 Date: Oct 31, 09
		Status : Final Confid : Public

1. INTRODUCTION

Software productivity has been one of the most studied aspects of software engineering (Scacchi 2005). Productivity is generally defined as a ratio of inputs to outputs. The reason for software productivity attracting so much attention, is the obvious economic importance, where productivity is intrinsically linked with the subject areas cost and effort estimation; in essence it presents the same phenomena in multiple ways. Overall, software productivity finds itself at the intersection between software engineering and economics.

Software production is a complex process, and software productivity covers a wide subject area, including the software product, the software production process and the production setting. This further includes but is not limited to programmer/developer productivity, streamlining of the production process such as identifying bottlenecks, studying and accounting for the quality and complexity of code and process. In any productivity study, the challenge is to clearly define, identify and measure the inputs and the outputs. Some aspects of productivity are easier to quantify than others, however it does not follow that it necessarily is the best indicator. This study is specifically investigating the productivity of the *primary* production of software; that is, the actual time that specifically was spent coding is separated from the time spent on other activities. This is an attempt to counter a trend where input in terms of time could be related to output in so many domains relevant to participating in a community.

This deliverable is structured as follows: section 2 gives an overview over the background and related work, followed by section 3 describing the methodology for this study and the data sets, section 4 presents the analysis and discussion, and section 5 concludes.

	<p>Productivity study report</p> <p>Deliverable ID: D11.2</p>	Page : 6 of 30
		Version: 1.0 Date: Oct 31, 09
		Status : Final Confid : Public

2. BACKGROUND AND RELATED WORK


2.1 SOFTWARE COST ESTIMATION MODELS

Following the classification of software cost estimation techniques as outlined by Shepperd (2007), three main categories of estimation techniques are identified, namely:

1. Algorithmic or parametric models
2. Machine learning methods (i.e. induced prediction)
3. Expert judgement

In another paper (Jørgensen and Shepperd 2007), a even more detailed breakdown of estimation approaches are identified, namely: regression, analogy, expert judgement, work break-down, function point, classification and regression trees, simulation, neural network, theory, Bayesian and combination of estimates. They show that while regression still is most popular, machine learning methods are on the rise. Also, in general the diversity of estimation techniques is increasing over time.

Turning to the algorithmic and parametric models; COCOMO as developed by Boehm (1981) is probably the most well known method, later an improved version called COCOMO-II was developed that requires the setting of parameters. For an overview of loglinear models including the number of projects (ranging from 15 to 63), the size of the projects measured in size by SLOC and Function Points, as well as the degree of economies or diseconomies of scale, see Banker and Kemerer (1989). Generally software cost estimation models have investigated the productivity of software for a relatively low number of projects, all developed in a firm setting, and actually most cost and effort estimation models are designed for in-house development. This makes the application of these estimation methods problematic for FLOSS projects, as the setting and scope are typically not comparable. Thus when for example applying COCOMO to FLOSS projects, one calculates the substitution costs of what it would have cost to produce the same code in a proprietary setting, such as estimating the substitution cost of Debian using COCOMO in


	<p>Productivity study report</p> <p>Deliverable ID: D11.2</p>	Page : 7 of 30
		Version: 1.0 Date: Oct 31, 09
		Status : Final Confid : Public

FLOSSImpact (2006); which is not what it actually did cost to produce the code in a distributed collaborative way. The same report also shows the distribution of code output by individuals, firms and universities (p. 50), and affiliation in terms of motivation and productivity could make a difference. Mockus et.al (2003) develops a parametric model for predicting effort changes and how it is distributed over time. Further, within the mainstream of software cost estimation models, Function Points (FP) are becoming more widely used, often replacing LOC or SLOC in effort estimations. For example see Premraj et. al. (2004) for a discussion on productivity over time using FP on a Finnish dataset on proprietary projects. Function Points weights function types in the measure. It in principle addresses some of the issues associated with using SLOC, such as programming language, functionality, complexity and quality. However, obviously using Function Point analysis for FLOSS software is not feasible, and SLOC is still a viable alternative for sizing open source. Investigating the relationship between FP and SLOC in the ISBSG database, a database that reports FP and SLOC for about 400 proprietary projects, we have previously found that there is a strong relationship, which increases the confidence in the measure.

In later years, there has been a shift from algorithmic and parametric models to machine learning methods, such as implemented by Bibi et.al. (2008). Using machine learning methods is promising since it does not necessitate the assumption of a functional form or structure of the relationship in the same way as parametric models do. Finally, surveys show that human centric prediction techniques, also known as expert judgement, is among the most wide spread prediction technique, however this is an under-researched area, even though it is on the rise. (Shepperd 2007, Passing and Shepperd 2003). For a very extensive overview of software development cost estimation studies, see Jørgensen and Shepperd (2007).

2.2 INCREASING OR DECREASING RETURNS TO SCALE

Increasing, decreasing or constant returns to scale tend to cause discussion, and the empirical evidence is inconclusive (Shepperd 2007, Kitchenham 2002). However, overall it is commonly assumed that software exhibits diseconomies of scale for proprietary project, on the other hand Dolado (2000) has analyzed existing production functions by use of genetic programming algorithm, and finds no support for complex production functions. On the other hand, empirical

	<p>Productivity study report</p> <p>Deliverable ID: D11.2</p>	Page : 8 of 30
		Version: 1.0
		Date: Oct 31, 09
		Status : Final Confid : Public

findings on open source has found evidence for linear or super-linear growth (Godfrey et. al., 2000, Succi et al 2001, Robles et. al. 2005; Koch 2005). For our study we make no assumptions to the functional form a priori.


2.3 OTHER OUTPUT

Output of participation in a FLOSS community does not necessarily involve the production of code. Community members can for example identify bugs or document software, among many other things. See FLOSSImpact (2006), Lakhani and Wolf (2005) and David and Shaprio (2008) for overviews of motivations in FLOSS projects.

2.4 ESTIMATING PROPRIETARY PROJECT VERSUS OPEN SOURCE


As open source software is growing in size and importance, it has not been easy for researchers to replicate or conduct comparable productivity studies in the open source domain. The reason for this is related to the difficulty of obtaining data, specifically measures of input entering into the production process. This is easier in a traditional company setting, because the company has (some) control over the employees, where the software development happens in-house. In a traditional software production setting, i.e. in a firm, the management or researchers can get programmers to self-report, or a team manager can measure it, or an outside analyst or observer can be hired, or automated performance monitors can be installed to monitor employees, just to mention some alternatives (Scacchi 1995). However, the primary output, being the software product and its corresponding code, is available in both cases.

In other words, for studies focusing on a proprietary setting it has been more feasible to measure both input and output, while for open source software it is generally only possible to observe the output of the production process, the code itself. Further, and ironically, due to the nature of proprietary software licensing, the software output is not generally available to researchers, and this also makes it difficult to compare studies done on proprietary software productivity vs. open source software productivity. Further, the cost estimation studies and databases for proprietary

	<p>Productivity study report</p> <p>Deliverable ID: D11.2</p>	Page : 9 of 30
		Version: 1.0 Date: Oct 31, 09
		Status : Final Confid : Public

projects are based on completed software development projects, not entirely appropriate for FLOSS as it often includes incomplete projects, developed outside firms.

The output of programming activities is mostly readily available through public repositories and software version control data (“CVS”). However, all these available data sources are output measures, and to calculate the productivity we need to relate the output to the input. Further, one needs to take account of the amount of time and effort the programmer spends on various other activities, not all directly related to coding, such as communicating on mailing lists and writing documentation.


	<p>Productivity study report</p> <p>Deliverable ID: D11.2</p>	Page : 10 of 30
		Version: 1.0 Date: Oct 31, 09
		Status : Final Confid : Public

3. RESEARCH METHODOLOGY

Our research methodology aims to overcome the problems inherent in investigating productivity in the FLOSS context, namely that only output is generally available. FLOSSMETRICS itself is a database of output data, and productivity is linking input to output for a given period time. In other words, one needs to have the actual time a developer spend on a given task, to know how much he produced for a given time. The volunteering nature in FLOSS development makes it difficult for developers to clock, or remember, the exact time they spend on a given task. The problem becomes even more acute if a researcher wants to obtain times which goes a bit further from the current activity. For example, a developer may find it difficult to remember exactly how many hours he/she spend coding 2-3 weeks ago (at the time when question is being asked), let alone a year ago. It is not possible to use CVS check-ins, because artifacts like this are not accurate for input estimation, as they say nothing about the time actually spent coding and performing other activities.

Programmer input and effort can range from anything from primary SLOC production, documentation, testing, participating in mailing lists and forums, packaging software, etc. Cyber archeology enables FLOSS researchers to use the wealth of information described as software trail to study who produces which artefact, and how much of it. The critical factor is 'when' the artifact was produced. In software repositories, we can record and datamine times when a certain software artifact was produce but this only tells us when the artifact first appeared or was checked-in into the repository and NOT how much time the individual spent producing that artifact. Although, the wide availability of FLOSS data enabled by FLOSSMETRICS leverages some of the empirical challenges researchers faced, it is now apparent that much effort is needed in terms of data cleaning and aggregation. Our research is no exception to this challenge.

In an attempt to overcome these challenges in our productivity study, we adopted what we call "data convergence research methodology" or DCRM. This simply means combining two or more data sources which appear to offer a means to investigate a common research theme. We are currently strongly arguing for DCRM to investigate many of software's complex issues. For

	<p>Productivity study report</p> <p>Deliverable ID: D11.2</p>	Page : 11 of 30
		Version: 1.0
		Date: Oct 31, 09
		Status : Final Confid : Public

example, to deeply understand the dynamics and evolution of FLOSS communities or 'bug communities' it is important that researchers consider and attempt to integrate data from more than one repository (mailing lists, CVS, BTS) alone.

Our DCRM utilises two data sources; projects CVS repositories and a survey with developers.

3.1 CVS DATA

We downloaded CVS/SVN data dumps of 323 projects from the FLOSSMETRICS database. Historically, in 2006 was just when the FLOSSMETRICS started harvesting data from SourceForge.net. The data was processed to obtained primary and derived (computed from the primary) metrics which we assume to be vital measures in determining productivity. It contains the lines added and deleted per person and per project for the last year (at the point the survey was performed), and most importantly the exact lines of code added and deleted for the week coinciding with the programmers answering the questionnaire.

3.2 SURVEY DATA

For the FLOSS productivity survey we contacted, via email, 8370 committers to the hundred largest FLOSS projects in our data set soliciting their support to participate on an online survey. We received 460 (approximately 6%) 'willing to participate' emails from these developers, saying that they would be willing to take part in our survey. Subsequently, email was sent to each developer with an URL pointer to the online questionnaire. The questionnaire (available on request) contained 10 questions, covering topics such as the activities of the respondents in the FLOSS community, the time they belong to the project community, the time they spend in producing FLOSS code for the project, trends in time spending in the project, monetary rewards, and demographics. The questions were worded in a way that it was absolutely clear to the respondent to which particular FLOSS project his answers had to relate. To participate the respondents had to provide their CVS id, ensuring they are real developers, as well as being able to match the reported time input reported in the survey with the CVS code output. Out of the 460 developers who were willing to participate on our survey, 159 completed the survey, corresponding



to a response rate of about 35%. Figure 1 shows how the developers responded to the items in our survey.

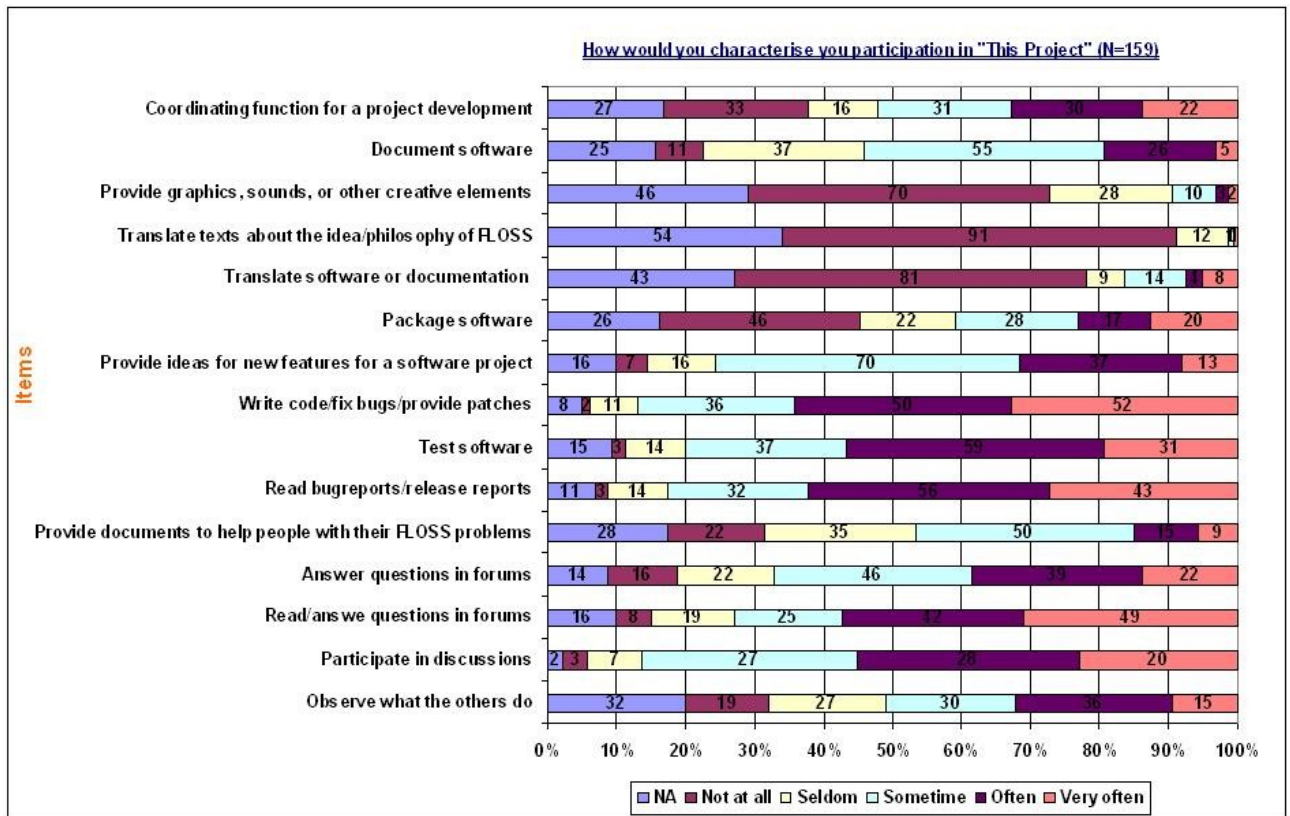



Figure 1 portrays that output is not only code, however for the primary production we are mainly focusing on Source Lines of Code. These other factors can be used to 'rank' or place individuals.

	Productivity study report Deliverable ID: D11.2	Page : 13 of 30
		Version: 1.0 Date: Oct 31, 09
		Status : Final Confid : Public

4. ANALYSIS AND DISCUSSIONS

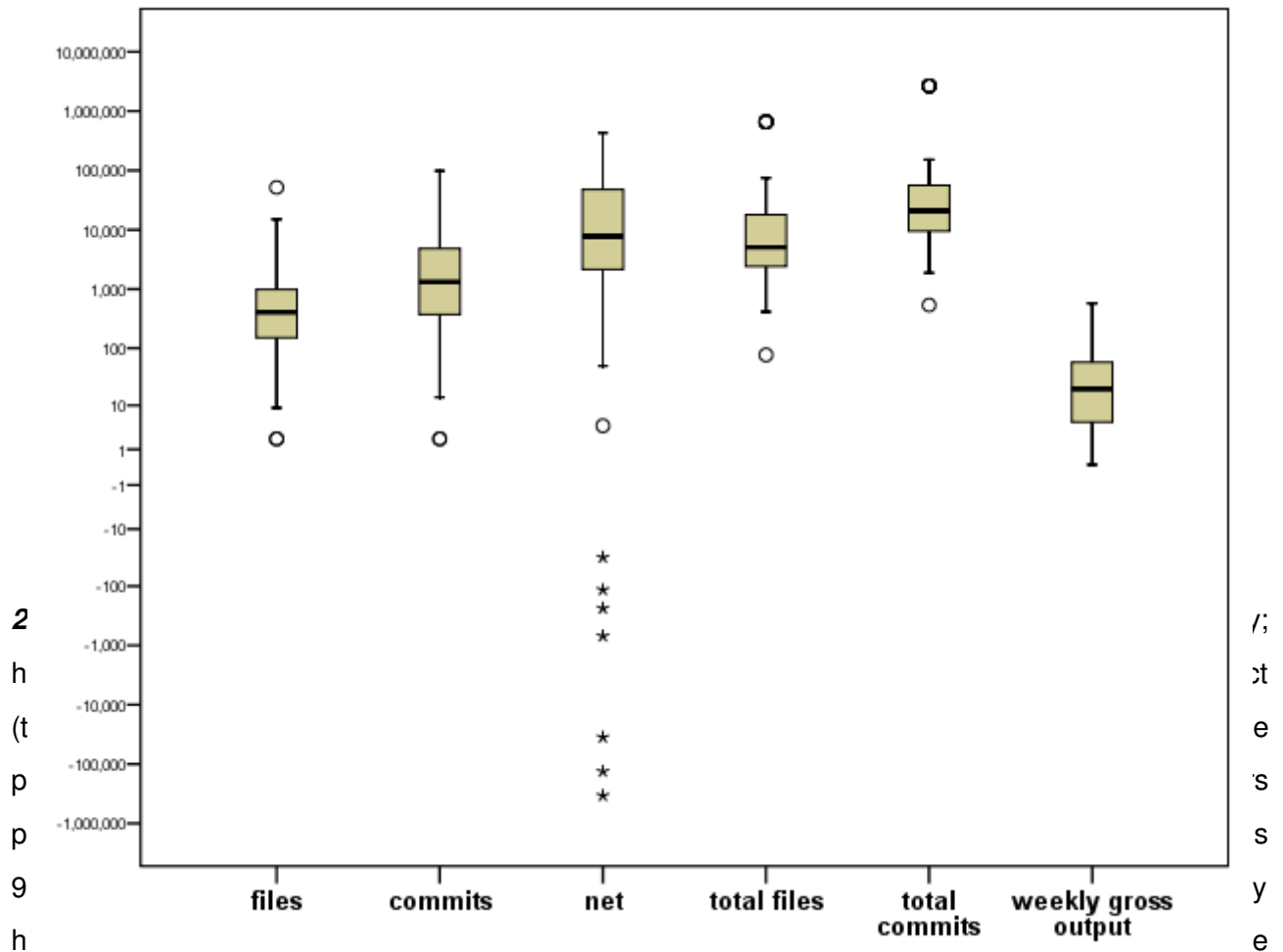
From the 159 developers who completed the survey, we have complete data in the metrics we wanted to measure, for 87 individuals. The metrics are divided into three main categories.

1. Coding dependent metrics obtained from developers activity logs. These are used as inputs to model developers productivity. Table 1 shows the descriptive statistics for these variables;

Table 1: Descriptive statistics of code dependent metrics

		files	commits	net	total files	total commits	weekly gross output
N	Valid	87	87	87	87	87	87
	Missing	0	0	0	0	0	0
Mean		1715.08	5056.15	30525.99	57512.46	215953.09	316,55
Median		406.00	1328.00	7790.00	5056.00	20955.00	96,75
Std. Deviation		5932.031	11833.769	82631.625	165870.605	668345.821	509,45
Skewness		7.343	6.107	1.244	3.390	3.443	2,85
Minimum		2	2	-340758	77	538	1,25
Maximum		51691	98417	430698	659623	2653435	3013,9
Sum		149212	439885	2655761	5003584	18787919	27539,52


The metrics shows that at the individual level; the mean number of files committed by a developer is 1715.08 (Std. Dev. = 5932.31), the mean commits per developer is 5056.15 (Std. Dev. = 11833.769) and the net lines added per developer (mean = 30525.99; Std. Dev. = 82631.625). At the project level, the mean for the total commits (sum = 18787919) per project is 215953.09 (Std. Dev. = 668345.821). The weekly gross output, which is our primary dependent variable for calculating the productivity of the 87 developers, is the number of lines of code a developer produces in the week he/she answered the questions in our survey. The mean weekly gross output per developer is 316.55 (Std. Dev. = 509.45). In total the developers contributed 27,539.52 lines of code. The maximum number of lines contributed by the most prolific developer was 3013.90 and the minimum was 1.25 lines of code. The distribution of these variables are shown in the box plots in figure 2. Note the many outliers in the net lines added. It shows that, although the developers have a positive net lines added to their project's code base, a large number of them are also deleting lines from their code base.



related to their projects. Bivariate correlations shows that the times developers spend on either coding or other activities are highly correlated with Pearson $r = .862$ ($p < 0.000$). However, Kendall's tau_b for the ranked values was a bit lower ($\tau = .743$; $p < 0.000$).

Table 2: Descriptive statistics of code dependent metrics

	total weekly hours on thisproject	coding weekly hours on thisproject
N	87	87
Mean	16.01	9.69
Median	10.00	6.00
Std. Deviation	17.301	9.927

	Productivity study report Deliverable ID: D11.2	Page : 15 of 30
		Version: 1.0 Date: Oct 31, 09
		Status : Final Confid : Public

Minimum	1	1
Maximum	100	50
Sum	1393	843

2. Combined metrics. These are obtained by combing code and time dependent metrics. These variables gives an indication of a developer's activity per unit time. For example, *commitsperhour* is the total number of commits a developer makes in 1 hour. Table 3 shows descriptive statistics of metrics we used in this study.

Table 3 : Descriptive statistics of time dependent metrics

	N	Mean	Median	Mode	Std. Dev.	Maximum	Sum
netperhour	87	0	1.84	1.40	0	248.074	160
plusperhour	87	0	90.59	18.00	0	147.057	7881
commitsperhour	87	0	5.73	1.00	0	10.816	498
weekly_gross_output	87	0	316.55	96.75	5	509.449	27540
weeklygrossperhour	87	0	45.65	19.73	4	76.690	3972
Inweeklygrossperhour	87	0	2.68	2.98	1	1.819	233
Inhoursperline	84	3	-2.70	-2.89	-1	1.781	-227

Where:

- *netperhour*=net lines month 12 per hour spent coding
- *plusperhour*=gross lines month12 per hour spent coding
- *commitsperhour*=commits month 12 per hour spent coding
- *weekly_gross_output*=mean lines per week in month 12 or over past 12 months
- *weeklygrossperhour*=gross lines per hour spent coding - from month 12 or over past 12 months
- *Inweeklygrossperhour*= transformed values of gross lines per hour spent coding - from month 12 or over past 12 months
- *Inhoursperline*= $n(1/\text{weeklygrossperhour})$

4.1 NONPARAMETRIC CORRELATIONS BETWEEN CODING OUTPUT AND TIME SPENT

The time dependent metrics records the total weekly hours developers spent working on all sorts of activities on their projects and the total coding weekly hours they spent just coding on their project. The combined metrics records times developers spend on a particular activity as their work output. Nonparametric correlations was used to test the relationship between the two metrics. In order to compute the correlations, we ranked each variable with the smallest value taking a rank of 1, etc. Table 4 shows the correlation results with Spearman's rho - which are measures of the linear relationship between two variables - with the corresponding statistical significance values. It can be seen that developers weekly gross output is significantly ($\rho = 0.401$; $p < 0.000$) related to their coding weekly hours. In comparison, their relationship between developers weekly gross output and their total weekly hours is slightly lower ($\rho = 0.360$), but this is also statistically significant ($p < 0.001$).


Table 4: Nonparametric correlations between combined and time dependent metrics

		weekly_gross_out ut	total weekly hours	coding weekly hours
Spearman's rho	weekly_gross_output	Correlation	1.000	.360(**)
		Coefficient		.401(**)
		Sig. (2-tailed)	.	.001
total weekly hours		Correlation	.360(**)	1.000
		Coefficient		.861(**)
		Sig. (2-tailed)	.001	.
coding weekly hours		N	87	87
		Correlation	.401(**)	.861(**)
		Coefficient		1.000
		Sig. (2-tailed)	.000	.
		N	87	87

** Correlation is significant at the 0.01 level (2-tailed).

4.2 MODELING DEVELOPERS PRODUCTIVITY

In modelling developers productivity, we used regression analysis to help us determine the 'best' parameters for a productivity function. Based on our metrics, this section provides functions that could best fit a set of observations from our CVS and survey data.

	<p>Productivity study report</p> <p>Deliverable ID: D11.2</p>	Page : 17 of 30
		Version: 1.0 Date: Oct 31, 09
		Status : Final Confid : Public

The linear regression model assumes that there is a linear, or "straight line," relationship between the dependent variable (weekly gross output) and each predictor, for example, coding weekly hours. This relationship is described in the following formula (Draper and Smith, 1981).

$$Y_i = b_0 + b_1 X_{i1} + \dots + b_p X_{ip} + e_i$$

where

- y_i is the value of the i^{th} case of the dependent scale variable
- p is the number of predictors
- b_j is the value of the j^{th} coefficient, $j=0, \dots, p$
- X_{ij} is the value of the i^{th} case of the j^{th} predictor
- e_i is the error in the observed value for the i^{th} case

The model is linear because increasing the value of the j^{th} predictor by 1 unit increases the value of the dependent by b_j units. Note that b_0 is the intercept, the model-predicted value of the dependent variable when the value of every predictor is equal to 0.

Before running the regression, we used a scatter plot of the transformed values of weekly gross output as a dependent variable versus coding weekly hours as independent variable. The resulting scatter plot in figure 3 appears to be suitable for linear regression, with some possible causes for concern. The variability of weekly gross output appears to increase with coding weekly hours. The points outside the dotted box on the far right and left of the graph may exert an undue amount of influence on the lay of the regression line. According to the plot, the linear regression plot explains 14.6% ($R^2=14.6$) in the variability of weekly gross output

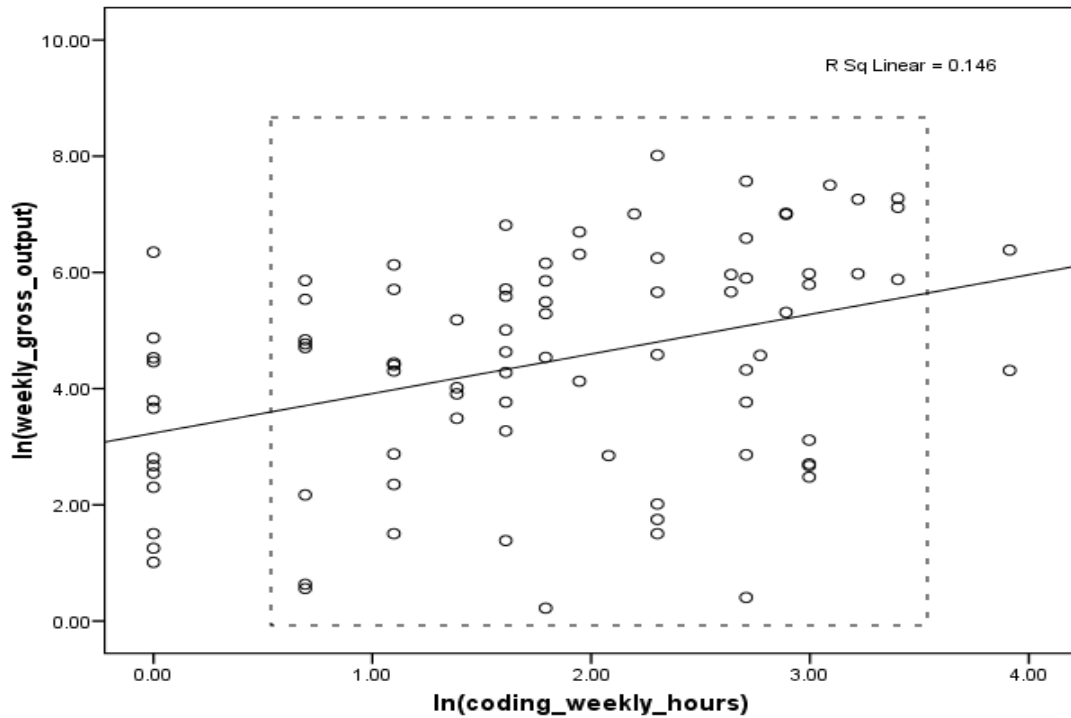


Figure 3: Linear regression showing of weekly gross output as an independent variable by coding weekly hours as a predictor.:

We further investigated the scatterplot by running a diagnostic checking of the regression model.

Table 5 shows the coefficients of the regression line resulting from our diagnostic test of the linear regression model.

Table 5: Coefficients of linear regression model with weekly gross output as dependent variable

Model	Unstandardized Coefficients		Standardized Coefficients	t	Sig.
	B	Std. Error	Beta		
1 (Constant)	.802	.272		2.953	.004
ln(weekly gross output)	.215	.056	.383	3.818	.000

a Dependent Variable: ln(coding weekly hours)

The table shows that weekly gross output (transformed) is equal to 0.215 * coding weekly hours (transformed) – 0.802. This simply means that, for example, if a developer X spend 2.30 Incodinghours on a project, his lnweeklygrossoutput will be 8.01 lines of code.

The ANOVA statistics in table 6 tests the acceptability of the model from a statistical perspective. The Regression row displays information about the variation accounted for by your model. The Residual row displays information about the variation that is not accounted for by your model. The regression and residual sums of squares, 14.836 and 86.490 respectively, are far apart, which indicates that much of the variation in weekly gross output is not explained by the regression model. However, the significance value of the F statistic is less than 0.05, which means that the variation not explained by the model is not due to chance.

Table 6: ANOVA statistics showing the effect of residual, unexplained aspect of the linear regression

Model		Sum of Squares	df	Mean Square	F	Sig.
1	Regression	14.836	1	14.836	14.580	.000(a)
	Residual	86.490	85	1.018		
	Total	101.326	86			

a Predictors: (Constant), ln(weekly gross output)

b Dependent Variable: ln(coding weekly hours)

The model summary presented in table 7 reports the strength of the relationship between the model and the dependent variable. R, the multiple correlation coefficient, is the linear correlation between the observed and model-predicted values of the dependent variable. Its large value indicates a strong relationship. R Square, the coefficient of determination, is the squared value of the multiple correlation coefficient. It shows that about less than half the variation in ln(weekly gross output) is explained by the model.

Table 7: Linear regression model using coding weekly hours to predict weekly gross output

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	.383(a)	.146	.136	1.00872

a Predictors: (Constant), ln(weekly gross output)

b Dependent Variable: ln(coding weekly hours)

We increased the predictability of our regression model by adding the influence of paid developers as a predictor in our linear equation. The resulting model summary in table 8 shows an improvement in the predictability of the variability in weekly gross output.

Table 8: Improved linear model summary with the inclusion of paid developers effect

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate		Change Statistics				
	R Square Change	F Change	df1	df2	Sig. Change	F Change	R Square Change	F Change	df1	df2
1	.428(a)	.183	.163	1.768	.183	9.399	2	84	.000	

a Predictors: (Constant), paid employee, ln(weekly gross output)

b Dependent Variable: ln(coding weekly hours)

By incorporating paid developers as a predictors, the linear regression model could explain 18.3% ($R^2=0.183$) in the variability of weekly gross output. This is a slight improvement over the 14.6%.

This represents a significant but small variability in weekly gross output, hence the productivity of the developers in this respect. Thus, we predict that there are other underlying factors which comes into play when explaining the productivity of FLOSS developers.

4.3 TRENDS IN WEEKLY GROSS OUTPUT AND CODING WEEKLY HOURS

We further investigated weekly gross output by ranking the hours each developer spent coding in his project. Figure 4 shows this increasing trend. Note that, even though some of have high weekly gross output, there are some fluctuations in their weekly output. This can be observed between the 71th -82nd developers. The line graph in figure 5 shows a similar trend in coding weekly hours, but with the top 1% of the developers spending more time in coding that the rest of the group.

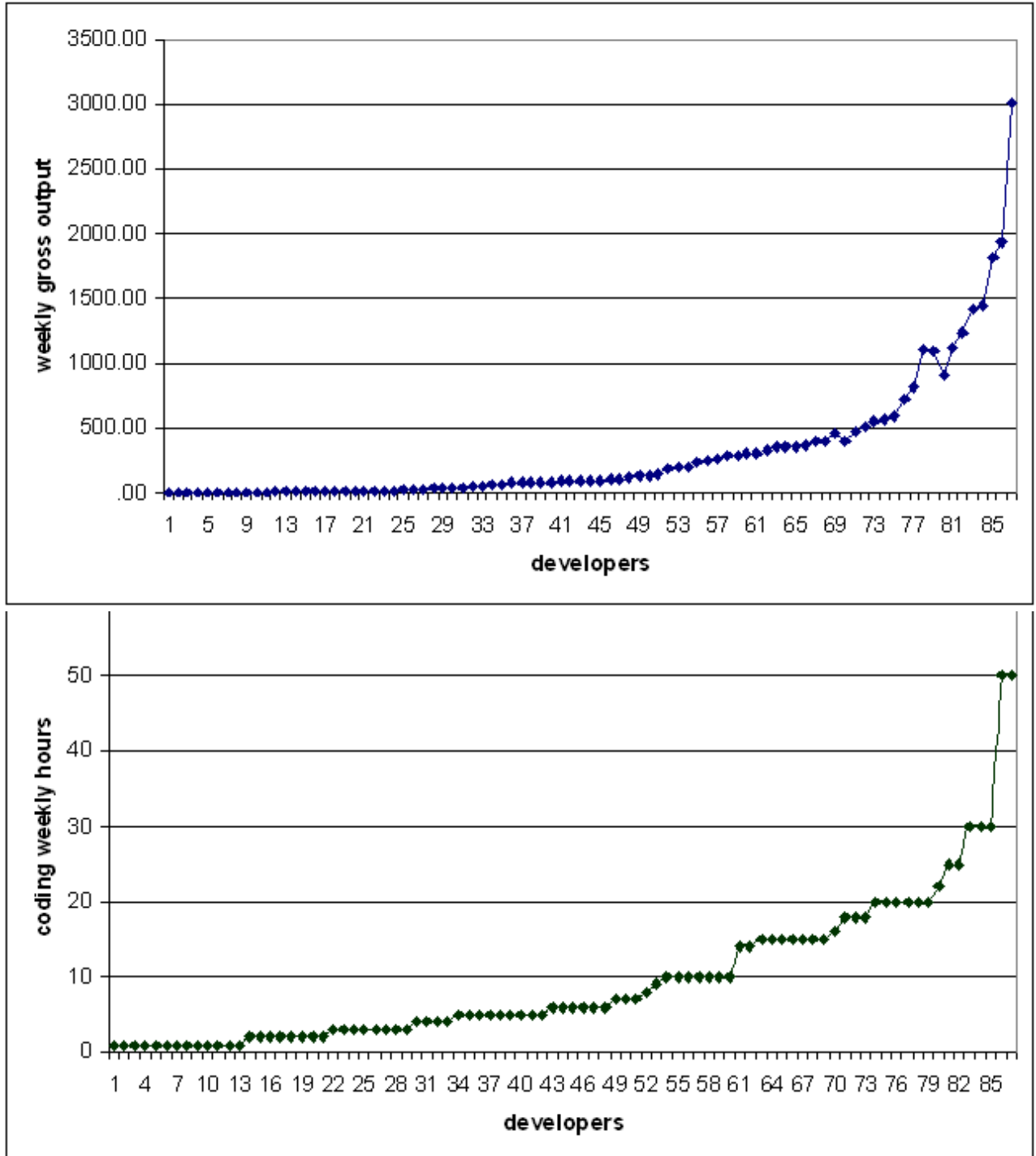


Figure 5: Trend in coding weekly hours

4.4 OTHER FACTORS TO CONSIDER

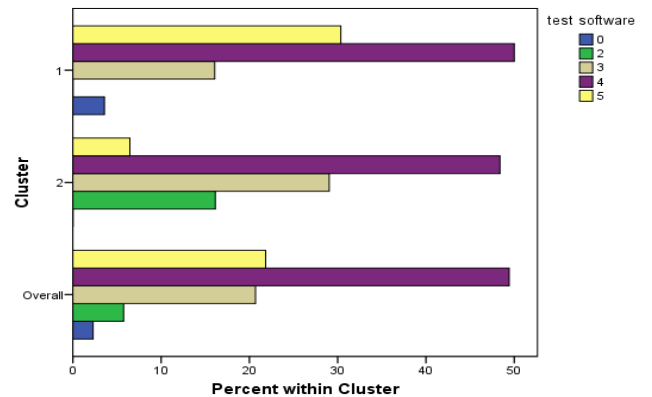
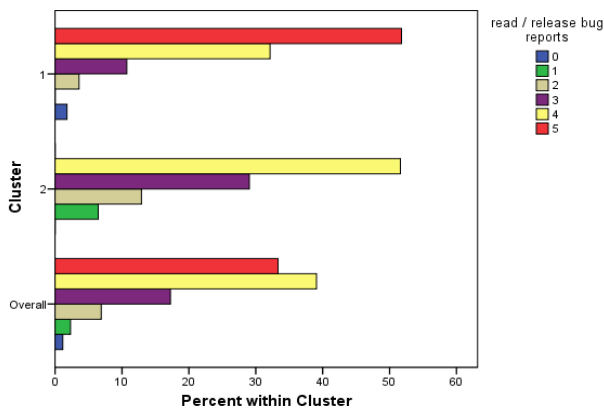
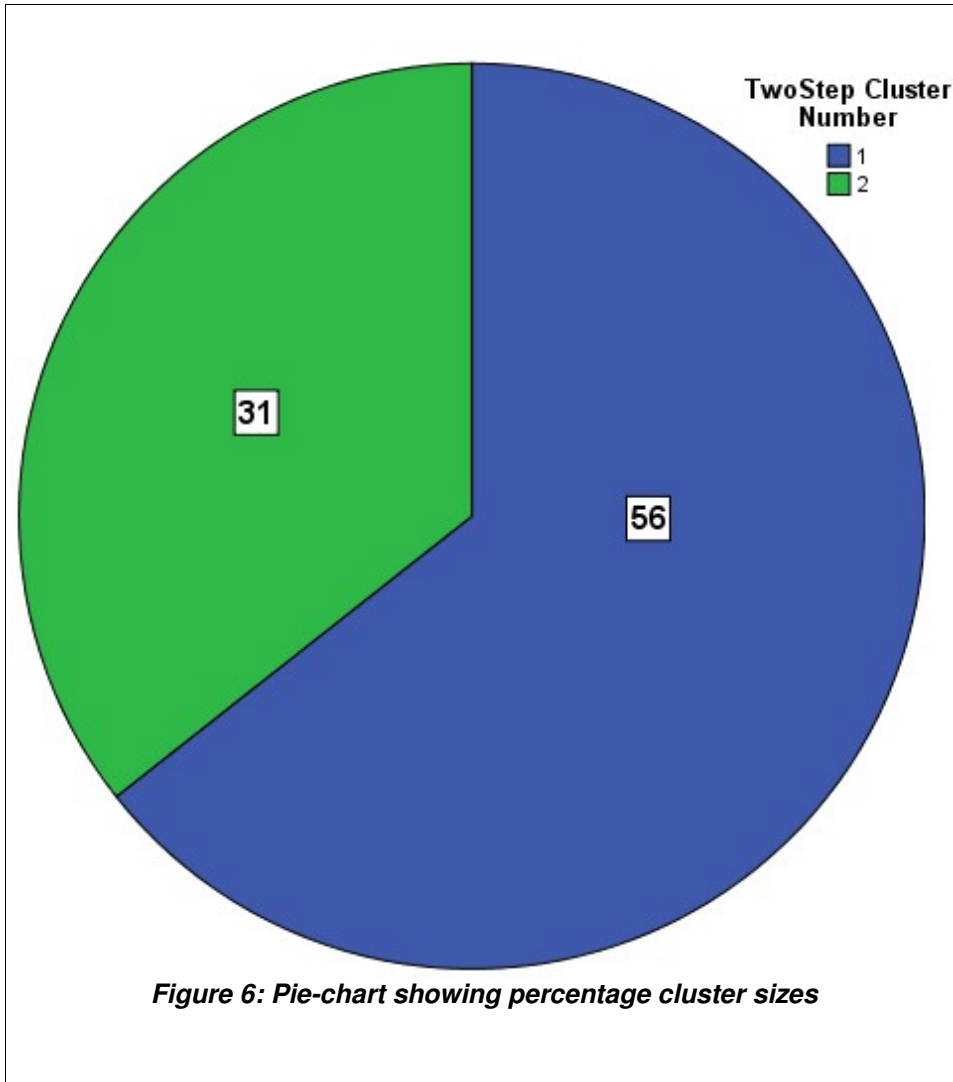
In the questionnaire, developers were asked to rank their participation in various activities as “very often”, “often”, “sometimes”, “seldom”, “not at all”. The activities are coded on a 5 point Likert scale with “very often” coded as 1 to “not at all” coded as 5. No response was coded as zero. The activities in the questionnaire are

1. “I read / release bug reports” with variable *read / release bug reports*
2. “I test software” with the variable *test software*
3. “I write code / fix bugs / provide patches” with the variable *write code/ patches*
4. “I package software” with variable *package software*
5. “I document software with variable *document software*
6. “I have a coordinating function for the project (development)” with variable *coordinate project development*

The TwoStep cluster analysis was used as an exploratory tool to reveal natural groupings (or clusters) within the activities that would otherwise not be apparent from our survey results. Auto-clustering reveal 2 distinct clusters. The percentage size of each cluster is shown in the pie-chart in figure 6. The distribution of the clusters, shown in Table 9, shows that the largest number of developers are in the first cluster (64.4 % ; N=56). The rest of the developers, 35.9% (N=31), are in the second cluster.

Table 9: Cluster distribution of developers activities

		N	% of Combined	% of Total
Cluster	1	56	64.4%	64.4%
	2	31	35.6%	35.6%
	Combined	87	100.0%	100.0%
Total		87		100.0%



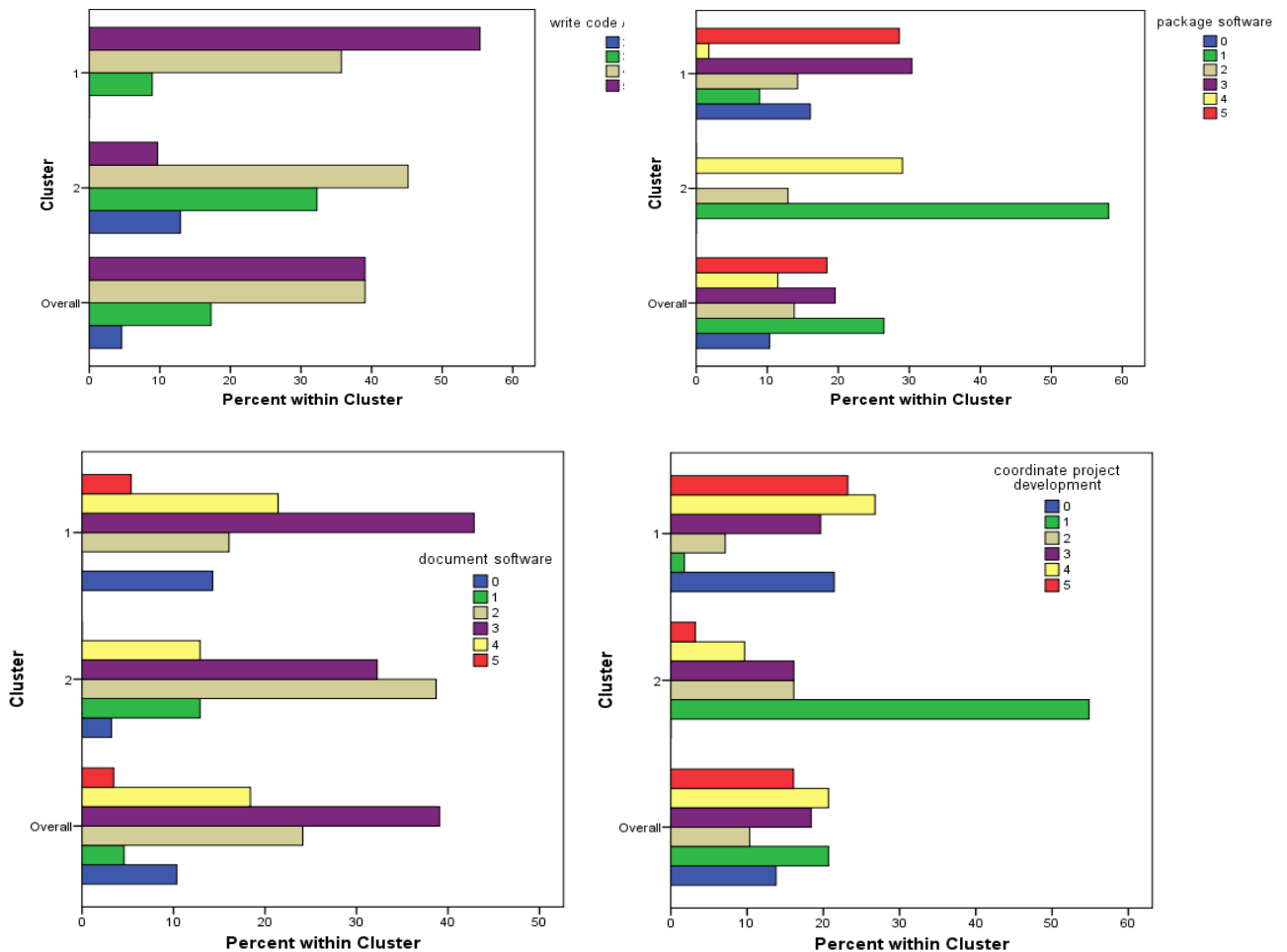


Figure 7: Within cluster percentage showing the percentage of each activity within each cluster and overall.

Furthermore, Bonferroni Adjustment was applied to the activities in each cluster to show their importance. For the 56 developers (64.4%) in the first cluster (figure 7), the only significant activity is packaging software. All other activities are non-significant. However, for the 31 developers (35.6%), packaging software is also the most significant but other activities (coordinate project development, write code / patches, and read/ release bug reports) are also significant. For this cohort, testing and documenting software are non-significant activities.

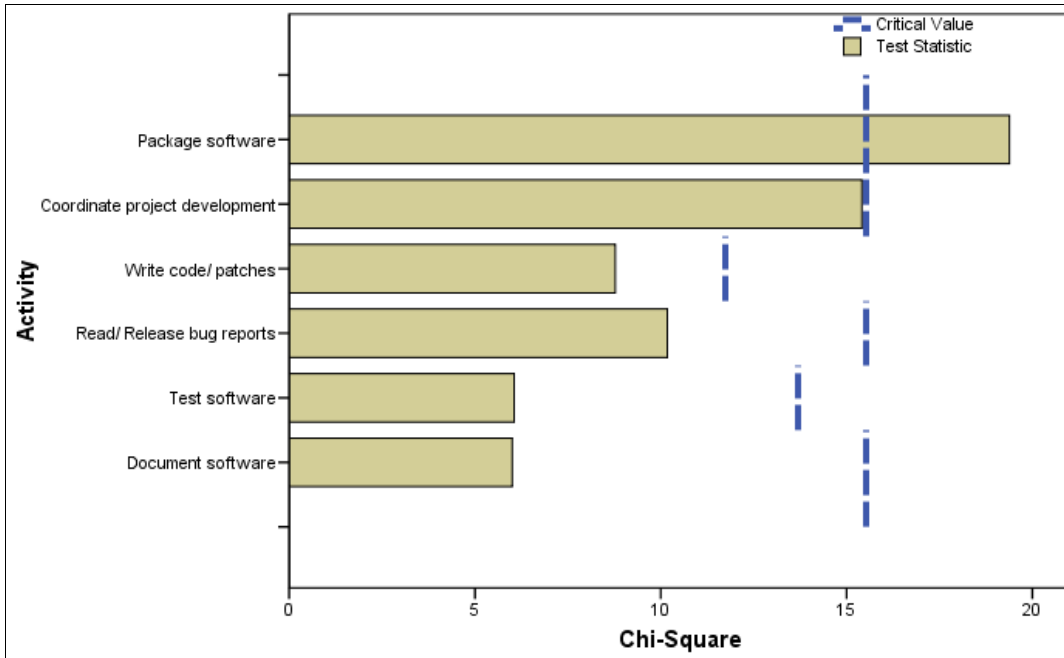
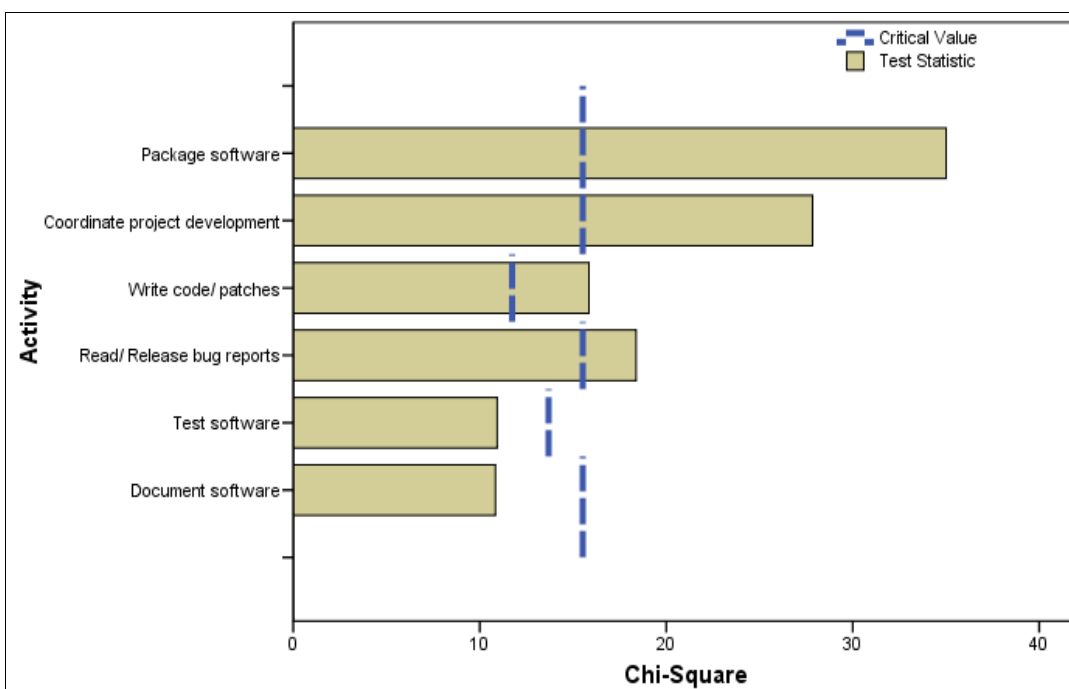


Figure 8a: Bonferroni adjustment applied to cluster 1 showing categorical importance of activities.



For each of these clusters we used regression analysis to see if there is any variation in the predictability of the developers' productivity. Figure 8 shows a small percentage increase in the variability in weekly gross outputs for developers in cluster 1, where packaging software is the one and only one significant activity. For this group of developers a linear model with cubic fit explains 13.9% in the variability of weekly gross output as a function of the total weekly hours developers spend on their project. This is an increase of about 1.6%. Weekly gross output as a function of coding weekly hours registered an increase of 8.2%.

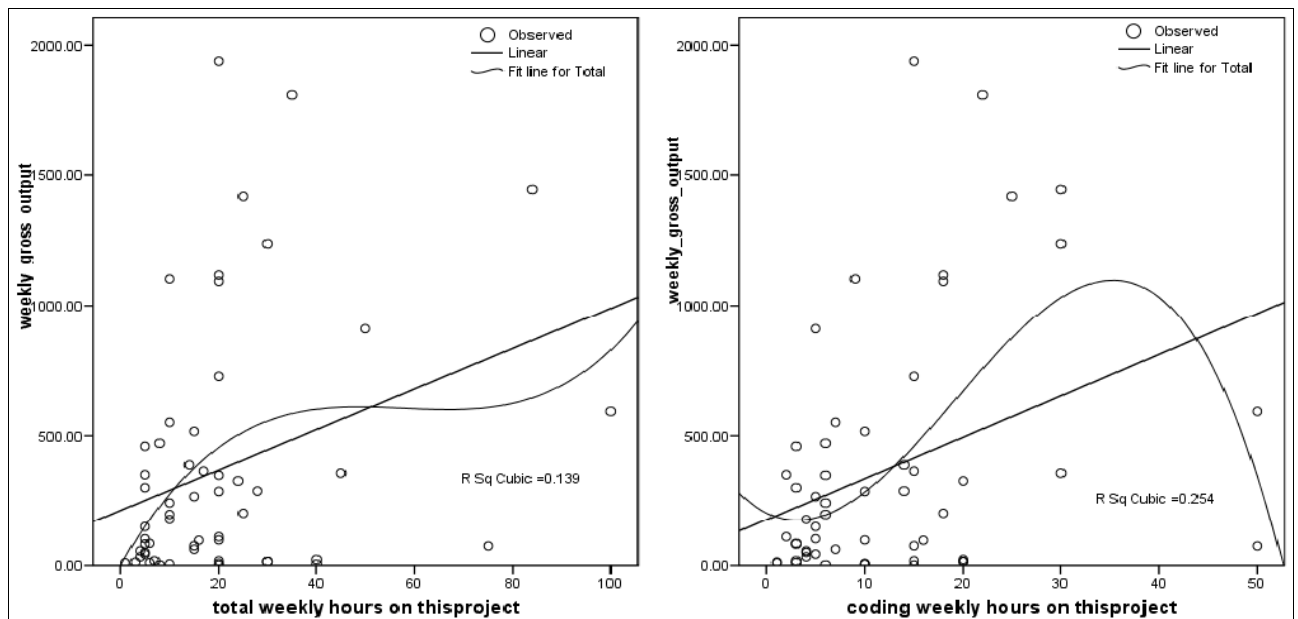


Figure 9a: Linear regression models showing improved variations in weekly gross output for developers in cluster 1

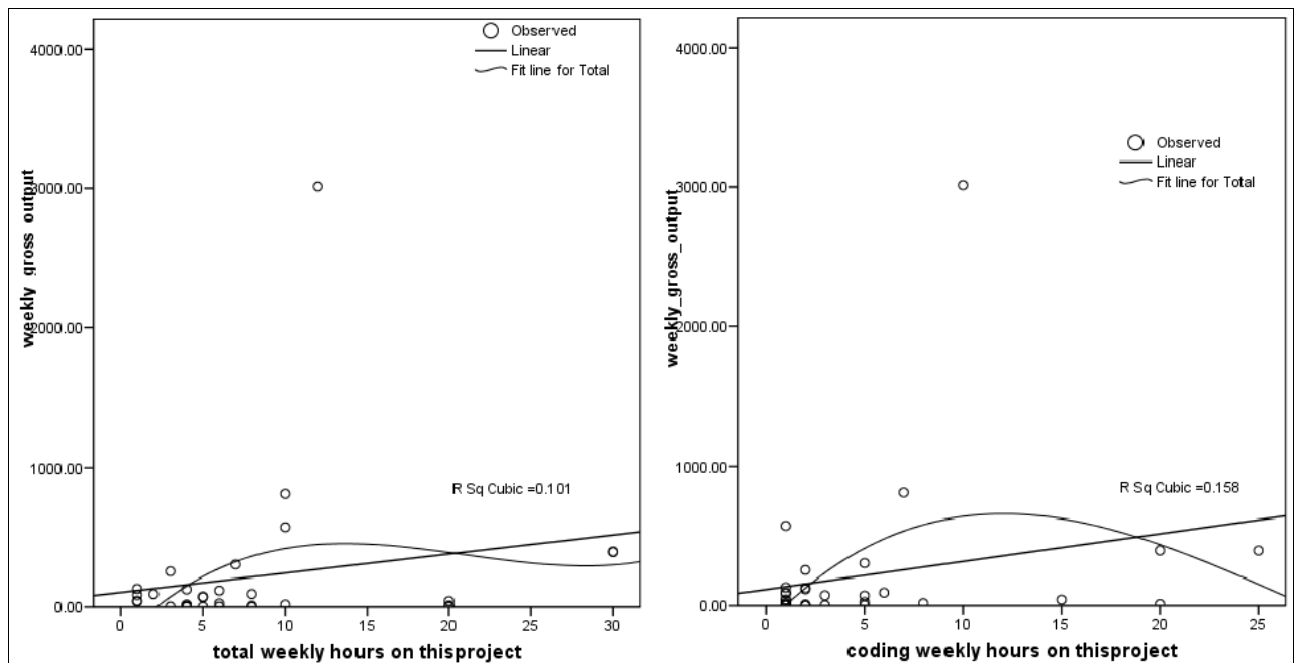




Figure 9b: Linear regression models showing improved variations in weekly gross output for developers in cluster 2

Figure 9 shows that variability of weekly gross output as a function of the total weekly hours developers spend on their project decreased for developers in cluster 2 where four of the activities are significant.

	<p>Productivity study report</p> <p>Deliverable ID: D11.2</p>	Page : 28 of 30
		<p>Version: 1.0</p> <p>Date: Oct 31, 09</p>
		<p>Status : Final</p> <p>Confid : Public</p>


5. CONCLUSION

We see a small but significantly improvement R^2 value when we introduced the element of paid contribution to FLOSS development. This is the case for those developers who were asked, in the survey, whether they are directly or indirectly paid for their contribution. From the cluster analysis we find that time spent on other activities, such as software testing, bug fixing, forum activity and coordination do help categorize individuals to some extent, where packaging software is the most significant factor for developers in cluster 1. All these activities received high percentage scores from the developers in our survey. Developers ranking in terms of long term contribution and activity does surprisingly not seem to play a role, but this could be due to the heterogeneous and small dataset. Similarly, project characteristics such as overall size and age of the project does not appear to be the main determinant of output and productivity. This analysis is the best we can do with the given data sets, and it points to promising future research by researchers combining FLOSSMETRICS data with other sources, surveys in particular. Overall this study demonstrates that FLOSSMETRICS which is a database of output data can successfully and fruitfully be linked to other data sources to investigate interesting and complex phenomena such as productivity.

	<p>Productivity study report</p> <p>Deliverable ID: D11.2</p>	Page : 29 of 30
		Version: 1.0
		Date: Oct 31, 09
		Status : Final Confid : Public

6. REFERENCES

- Banker, R. D., Chang, H., and C. F. Kemerer, 1994, "Evidence on economies of scale in software development," *Info. & Softw. Technol.*, 36, pp 275-282.
- Banker, R. D. and Kemerer, C. F. 1989, "Scale Economies in New Software Development", *IEEE Transactions on Software Engineering*, vol. 15 (10), p.1199-1205.
- Bibi, S., Stamelos, I., and Angelis, L. 2008, "Combining Probabilistic Models for Explanatory Productivity Estimation", *Information and Software Technology*, Elsevier, 50(7-8), pp. 656-669.
- Boehm, B. 1981, *Software Engineering Economics*, Prentice Hall PTR, Upper Saddle River, NJ.
- David and Shaprio, 2008, "Community-Based Production of Open Source Software: What do we know about the developers who participate?" Available at SSRN: <http://ssrn.com/abstract=1286273>
- Dolado, J. 2001, "On the problem of the software cost function". *Information & Software Technology*, 43(1) pp 61-72.
- Draper, N. R., and H. Smith. 1981. *Applied regression analysis*, 2nd ed. New York: John Wiley and Sons.
- Ghosh et. al. 2006, FLOSSIMPACT: The impact of Free/Libre/Open Source Software on innovation and competitiveness of the European Union, available online at: <http://www.flossimpact.eu/>
- Godfrey, M. and Tu, Q., 2000, Evolution in Open Source Software: A Case Study, *Proceedings of the International Conference on Software Maintenance*, San Jose, California, pp. 131-142.
- Kitchenham, B. 2002 "The question of scale economies in software - why cannot researchers agree?" *Info. & Softw. Technol.*, 44, pp 13-24.
- Koch, S, 2005, Evolution of Open Source Software Systems – A Large-Scale Investigation, *Proceedings of the 1st International Conference on Open Source Systems*, Genova, Italy, July.
- Lakhani, Karim R. and Wolf, Robert G., Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects(September 2003). MIT Sloan Working Paper No. 4425-03.

	<p>Productivity study report</p> <p>Deliverable ID: D11.2</p>	Page : 30 of 30
		Version: 1.0
		Date: Oct 31, 09
		Status : Final Confid : Public

- Mockus, A., Weiss, D. M., and Zhang, P. 2003. Understanding and predicting effort in software projects. In Proceedings of the 25th international Conference on Software Engineering (Portland, Oregon, May 03 - 10, 2003). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 274-284.
- Jørgensen, M., and Shepperd, M., 2007 A Systematic Review of Software Development Cost Estimation Studies, IEEE Transactions on Software Engineering, v.33 n.1, p.33-53.
- Passing, U. and Shepperd M., 2003: An experiment on software project size and effort estimation. ISESE 2003: 120-131
- Premraj, R., Shepperd, M., Kitchenham, B. and Fordselius, P. 2005. An empirical analysis of software productivity over time. In METRICS '05: Procs. of the 11th International Software Metrics Symposium, page 37, Como, Italy, IEEE.
- Robles, G., Amor, J, Gonzales-Barahona, J. and Herraiz, I. 2005, Evolution and Growth in Large Libre Software Projects, Proceedings fo the International Workshop on Principles in Software Envolution, Lisbon, Portugal, September, 165-174
- Scacchi, W. "Understanding software productivity," in Advances in. Software Engineering and Knowledge Engineering, vol. 4, 1995, pp. 37–. 70
- Scacchi, W. 1995. Understanding and Improving Software Productivity, Institute of Software Research, presentation online available: www.ics.uci.edu/~wscacchi
- Shepperd, M. 2007. Software project economics: a roadmap. In 2007 Future of Software Engineering (May 23 - 25, 2007). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 304-315.
- Succi, G, Paulson, J. W. and Eberlein, A. 2001. Preliminary Results from an Empirical Study on the Growth of Open Source and Commercial Software Products, EDSE-3 Workshop, May, Toronto, Canada.