ABSTRACT

Most empirical studies about Open Source (OS) projects or products are vertical and usually deal with the flagship, successful projects. There is a substantial lack of horizontal studies to shed light on the whole population of projects, including failures. This paper presents a horizontal study aimed at characterizing OS projects.

We analyze a sample of around 400 projects from a popular OS project repository. Each project is characterized by a number of attributes. We analyze these attributes statistically and over time.

The main results show that few projects are capable of attracting a meaningful community of developers. The majority of projects is made by few (in many cases one) person with a very slow pace of evolution.

1. INTRODUCTION

The Open Source (OS) model of software development has gained the attention of both the business, the practitioners' and the research communities. The OS process has been described by the seminal paper by Eric Raymond [4] and [5]. However, sound empirical studies are still very limited in number and mostly vertical, i.e. they deal with a single, high impact project [3], [6], [10], [16].

On the other hand, few are the preliminary horizontal studies that have been performed on major OS repositories, like [11], [12], but still they remain on the surface of the data calculated by the administrators of the site themselves. In these cases, those data are parsed from the HTML pages and used to perform descriptive statistics. Our study uses a similar horizontal approach, but goes deeper into data collection: besides readily available metrics that can be computed automatically, other project attributes are extrapolated or computed by hand. Further, we consider evolution of the projects. The measures are computed at a point in time, then repeated some time later. Our long term objective is to gain further understanding about OS project dynamics, and also to draw useful lessons for software development in general. It should be noted that the OS process provides open process and product data, and therefore is a rare opportunity for empirical research.

As an example of an open process-oriented issue, the literature studies the evolution of traditional (non OS) projects. As a result, evolution is organized in a significant number of releases in a short time, and this is usually considered an instability factor [7], [8]. On the contrary, in the OS community this type of evolution is an evidence of vitality showing the commitment of the authors, and the level of appreciation from users [9].

The paper is organized as follows. Section 2 presents the research methodology, Section 3 illustrates the research questions that we formulated, Section 4 the project attributes defined to answer the questions, Section 5 analyzes the data collected. In Section 6 we discuss the validity threats to this study.
6) Analysis. Descriptive statistics and simple charts are used to analyze collected data, simple correlations are sought for.

Concerning step 5), sampling, we carried it out three times, on the same subset of projects, on February 2001, January 1st, 2002, and July 1st, 2002. Since the FreshMeat site ran out of synchronization with the effective status of evolution of several projects, we had to manually inspect their home sites to obtain the latest version.

3. RESEARCH QUESTIONS
Our starting point is a set of questions about OS development, inspired by the literature and by our experience. These questions drive the analysis of the sample. We present here these questions organized by homogeneous groups.

3.1 Generic characterization
First of all we need to describe projects in term of generic characteristics, and possibly to cluster them.

Q1: What are the generic characteristics of projects developed as OS? As characteristics we intend application domain, programming language, age, size and so on. Most of these attributes are collected by the FreshMeat portal.

3.2 The community of developers
Successful OS projects have hundreds and possibly thousands of developers. However, how often an OS project can count on such a large pool of developers? How large is the community of developers of average projects? Is this community subject to changes?

Q2: How many people are involved in development?
The number of people involved in development is not available on the portal, but is computed by the authors through indirect means.

The OS process has been initially associated to chaotic development (the bazaar [4]). Instead, [3] demonstrates that, in the Apache project, the process is not chaotic at all, but neatly organized around three groups: a core team (around 10 people) that initiates the project, controls the architecture and steers the development, a larger group that uses the deliverables and proposes patches and enhancements, an even larger group that simply uses the deliverables. To understand if this organization is common to other OS projects we define Q3 and Q4.

Q3: What roles do developers have?
Q4: Is there a core team? How many people are in the core team?

3.3 The community of users
The success of Linux or Apache is impressive and the number of OS projects in portals is also impressive, but how many do really succeed, in terms of both delivery of functionality, users, and capability of attracting a large community of developers? From this general question we derive a more narrow question, based on the size of the community of users. Remark that Q2 and Q5 together identify a community of users and developers that are indicators of the success of a project.

Q5: How many users have the projects?

3.4 Architecture and documentation
The functionality offered by a project and its match with an unsatisfied need from a pool of users is probably one of the main drivers for the success (in term of users and developers) of an OS project. However, we have no means to evaluate this match. Another factor is probably the level of documentation (a better level of documentation should enlarge the number of both users and developers) and the level of modularity (a better level of modularity should ease the work of developers and therefore augment their number).

Q6: What is the level of modularity of projects?
Q7: What is the level of documentation of projects?

3.5 Evolution of OS projects
We want to study the OS projects not only statically, but also their evolution over time. To do this we perform several samplings, on the same subset of projects. Three samplings have been done till now, and more are planned in the future, at regular intervals.

Q8: What’s the variation of project attributes over time?

4. PROJECT ATTRIBUTES
Starting from the research questions listed above we identified relevant project attributes. Some attributes are defined by the portal owners and readily available (e.g. application domain, programming language, type of license). In other cases we define and compute the attributes by indirect means (e.g. through the analysis of change logs and CVS). In some cases, we had to compromise between available resources and accuracy of measures and use proxies.

Table 1 shows all project attributes. Each attribute is described by a name, a short definition, the possible values, how it is computed, and which research question it answers to.
<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
<th>Values</th>
<th>How computed</th>
<th>Used in</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>Age of the project</td>
<td>Days</td>
<td>As a proxy we use the date of first posting of the project on the portal, defined by portal owners. We calculated the time buckets based on that</td>
<td>Q1</td>
</tr>
<tr>
<td>Application domain</td>
<td>Main domain covered by the application</td>
<td>Scientific, Security, Database, etc.</td>
<td>Defined by portal owners.</td>
<td>Q1</td>
</tr>
<tr>
<td>Programming language</td>
<td>Programming language used to develop the application.</td>
<td>C, C++, Java etc.</td>
<td>Selected by the project and reported by the portal owners</td>
<td>Q1</td>
</tr>
<tr>
<td>Size</td>
<td>Size of the source code.</td>
<td>Kbytes</td>
<td>Computed by our study. Source code posted on the portal is measured, excluding documentation and unessential code, such as HTML, GIF, JPG.</td>
<td>Q1</td>
</tr>
<tr>
<td>Number of developers, stable developers, transient developers</td>
<td>A developer is a person who contributes isolated code patches, as well as a continued contribution of code. Bug reporting or contribution of ideas are not considered as development. Developers are further divided into transient and stable, see section 5.2</td>
<td>Integer values</td>
<td>Computed by our study through manual analysis of change-logs, AUTHORS files, THANKS files and others.</td>
<td>Q2, Q3, Q4</td>
</tr>
<tr>
<td>Number of users</td>
<td>A user uses the application developed by the project.</td>
<td>Integer values</td>
<td>The number of subscribers to a project is used as a proxy of number of users.</td>
<td>Q5</td>
</tr>
<tr>
<td>Modularity level</td>
<td>Degree of modularity of the source code</td>
<td>Coded in three values, as described in section 5.6</td>
<td>Computed by our study. As a proxy we use the number of directories in which the source code is organized.</td>
<td>Q6</td>
</tr>
<tr>
<td>Documentation level</td>
<td>Level of documentation of a project (source code, APIs)</td>
<td>Coded in three values, as described in section 5.6</td>
<td>Computed by our study. As a proxy we use the number of files dedicated to documentation</td>
<td>Q7</td>
</tr>
<tr>
<td>Popularity</td>
<td>Defined by portal owners as follows:</td>
<td>0% - 100%</td>
<td>Defined and computed by portal owners</td>
<td>Q7</td>
</tr>
</tbody>
</table>
- U stands for the count of visits to the project home page
- R is the number of visits to the project on FreshMeat pages
- S is the subscribers number

<table>
<thead>
<tr>
<th>Status</th>
<th>Stage of development of a project</th>
<th>Coded in six different values (Planning, Pre-alpha, Alpha, Beta, Stable, Mature)</th>
<th>Defined and computed by portal owners</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success of project</td>
<td>Number of users</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Vitality, Popularity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vitality</td>
<td>Defined by portal owners</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>as follows:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( V = \frac{R \times A}{L} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- R is the number of releases in a certain period (t)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- A is the age of project (in days)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- L stands for the number of releases in t</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5. ANALYSIS
We now analyze the data collected in the samples. Where available, we display some graphics and we try to draw some conclusions or answers to the questions we defined in Section 3.

5.1 Generic characterization (Q1)

5.1.1 Application domain:
The variation in application domain is very high, with all kinds of applications developed. Errore. L'origine riferimento non è stata trovata. shows the frequency distribution of projects by application domain: when a project covers more than one domain (e.g. Internet and Security), each of them is counted. The most common domains are ‘Internet’, ‘System’, ‘Software Development’, ‘Communication’ and ‘Multimedia’.

It is interesting to compare horizontal applications (applications used to build other software, the end user is required to program and is, likely, a software professional) with vertical ones (applications used by an end user, no programming is required). Horizontal applications (categories Internet, System, Software development, Communications, Database, Security) account for 66%. We interpret this data as evidence that the OS community is largely oriented to produce applications for the same community. This could be a limiting factor in the effect of OS development outside the OS community itself.

5.1.2 Size:
Within a range of values comprised between 0 and 24 MBytes, 40% of projects have size < 100Kbytes, 82% < 1Mbyte. Assuming a line of code has 20 characters on average, 40% of project is < 5KLOC, 82% < 50KLOC. Small projects, suitable to one or two developers, largely prevail. On the other hand it should be considered that an
OS portal like the one considered hosts projects in early stages of development, that are likely to be smaller in size. From this point of view it will be necessary to analyze the growth rate of projects (see section 5.7).

5.1.3 Age
Age is the number of days since a project has been posted on the portal. Therefore this measure ignores all the history a project may have undergone before being posted. At the time of our first sampling (February 2001), we find that 60% of the projects age is less than one year, 22% from one to two years, 15% two to three years, and around 2% more than three years. This distribution confirms the finding from size distribution, most projects are small and young.

5.1.4 License
The GPL license is the most popular (77%), LGPL (6%) and BSD (5%) follow, all other licenses account from 3 to 1%.

5.1.5 Programming language
C is the most used programming language (41.5%), followed by C++ and Perl (around 14% each), then PHP, Java and Python (5 to 8%). It is interesting to note that several projects maintain different modules written with different programming languages.

5.2 The community of developers (Q2)
The distribution of projects by developers is as follows: 194 projects out of 397 (49%) have only one person developing the application, 60 (15%) have 2 to 3 developers, 80 (20%) 4 to 10, 37 (9%) 11 to 20, 26 (6%) have more than 20. Once more this figure confirms the observations from size and age: most projects are small, young and have few developers. However, nearly half the projects have one developer only, this casts doubt on the common perception of the open source as a community: projects with one developer clearly do not form a community. This observation needs to be cross-checked with the community of users attracted by projects (see section 5.5).

The study on the Apache project [3] and [16] identifies in 10 to 20 the best size of the developing community, but this size applies only to 15% of our sample. Any study on process issues related to a distributed community of developers will have to consider only this subset of projects.

Next we compare the number of developers with the size of the project. We expect that larger projects will have more developers.

Instead, we find that there is no meaningful increase of size with developers: average code size of projects with 1 to 3 developers is 733 kB, 4 to 10 developers projects have average size of 605 kB; 10 to 20 developer: 621 kB; more than 20 developers: 656 kB.

From this point of view the first cluster is more productive because less developers produce the same amount of code, and the latter is the less productive. This surprising result may depend on the definition we used for developers (providing one patch is sufficient to be considered a developer).

For this reason we divide developers in stable (they have a prolonged collaboration with the project) and transient. Observing change logs, we decided to consider ‘transient’ those developers providing at most one patch in the development of any section of the project: furthermore, we considered as ‘transient’ also those programmers offering a limited (up to 3) number of patches to the same part of the code. In most cases this is a single logical change, performed in two or three steps.

We define ‘stable’ a developer contributing more than one piece of code, and that is applied for all kind of patches, from trivial to considerable ones.

5.3 Stable and transient developers (Q3)
Figure 2 shows the percentage of projects vs. the number of developers involved, divided in stable and transient. We immediately observe that the vast majority of projects (73%) have only one stable developer. These projects are initiated by one person, which remains, time after time, the only stable contributor. On the other hand, 10% of projects have two stable developers, the rest have ten or more.

If we consider transient developers, the striking fact is that 55% of projects have no transient developers, 9% have one, 8% two, 20% between two and ten.

Repeating the clustering of Section 5.2 (see Figure 3), using stable developers only, the clusters change considerably. Projects with 1 to 3 stable developers are the vast majority (353 out of 406, 87%) and reach an average code size of 438 Kb. When a project is supported by 4 to 10 stable developers, the average code size we found is 1867 Kb: this happens for 31 projects out of 406, or in 7% of the data set. Furthermore, 7 projects out of 406 have between 10 and 20 stable developers, and the average code size is 2543 Kb. Finally, we observe that the 4 cases out of 406 whose stable team is made of more than 20 developers achieve a code size of 3223 Kb. Considering stable developers, the average code size of projects grows con-
d)vertical studies on these kind of projects are useful to shed light on successful processes, but should probably not be considered representative of all OS projects.

With those data at hand, it seems evident that the most interesting scenarios in terms of evolution speed and capacity of attracting new developers are Scenario 3 and Scenario 4. Regarding those two scenarios, we observe that the average number of stable developers is 2 in Scenario 3, 6 in Scenario 4. We also observe that the average number of ‘transient developers’ is 3 in Scenario 3, and 19 in Scenario 4. That leads to a ratio coordinators/global developers of 40% in Scenario 3, and a ratio of 24% in Scenario 4, which somehow resembles the conclusion on study of [3] and [16]: the number of coordinators seems to stabilize around 15-20% of the global number of developers when it grows onto a mature stage of its evolution.

5.4 Core team (Q4)

We define core team in term of number of developers. A project has a core team if it features ten or more developers. This filters out most projects without a community of developers, therefore without coordination problems. 64 projects out of 406 (16%) fall under our definition of core team.

Within this set of projects, two patterns appear. Projects with only one stable developer (16 out of 64, we call it set A), projects with more than one stable developer (48, set B). Let's examine first projects in set A, with one stable developer and at least nine other developers. This situation suggests that a single developer is steering the evolution of the project, and recalls the description made in [4] of projects that have succeeded in gathering an initial community. According to [4] the conditions to attain this state are a defined and clear architecture and an adequately appealing function offered; both conditions require a meaningful size of the code. The main developer gathers and assembles contributions at his will, and the issues of coordination and collaboration are reduced by a strictly hierarchical organization. In the 16 projects the average variation in the code size between two following releases is 19%, which is pretty impressive. The average number of transient developers is 15, the average number of users is 9, the average code size is 1700 KLOC (KLOC) and the average age is 525 days. If the parallel with the project described by Raymond [4] holds, these numbers describe projects that have succeeded in attracting a community, but still are in an initial phase with only one stable developer.

Let's examine now projects with more than one stable developer, set B. These projects have an average code variation of +12%, and an average code size of 1900 KLOC, and average age 637 days, values that are not very far from the previous ones. Their core team is composed of an average of 8 stable developers, 21 transient developers, and 18 users. Overall, the number of developers is double than in set A, and the same happens for the number of users. Since the average age is similar but the number of

Figure 3 – Distribution of stable developers over projects

![Code size vs. stable developers](image-url)

If we consider the sum of stable and transient developers, 50% have one (i.e. these are the projects with one stable developer and no transient developers), 7% two (one stable and one transient), 28% between three and ten developers, 15% more than ten developers. Let's call these situations, respectively, scenario 1,2,3 and 4.

We identify scenarios 1 and 2 as those projects that are not capable of attracting a community of developers. The reasons may be several: they have just been posted on the portal, they are not being advertised adequately, they develop an application of limited interest. In the following we will study these scenarios. Scenarios 1 and 2 happen to represent 212 projects out of 406. We go further in our analysis and we define “young” those projects appearing in Scenarios 1 and 2, whose age at the time of sampling is less than one year, “aged” the others. At the time of sampling 120 out of 212 were young, 92 were aged. Young projects are trying to attract a community of developers. Aged projects had time to attract a community, but did not. If we also consider the pace of evolution of the project, under the form of new releases per period, aged projects that do not evolve are likely to have been abandoned.

Scenarios 3 and especially 4 identify projects that have a community of collaborating developers. From a process perspective, only scenario 4 is relevant to study the chaotic vs. organized development issue. In section 5.4 we will study this issue. At this point we can observe that scenario 4 (where projects like Linux and Apache would fall) represents a minority of OS projects. In other words vertical studies on these kind of projects are useful to shed...
developer is double, other causes (such as the type of functionality or the visibility of a project) appear to be the drivers for attracting more developers. In set B the ratio between stable and transient developers is 0.38, and 0.44 between stable developers and users, both higher than what found in [3] (0,10 and 0,01).

5.5 The Community of Users (Q5)

With the number of developers, the number of users is another indicator of the success of a project in attracting a community. We use the number of subscribers to a project as a proxy of number of users. The assumption is that a subscriber, after registering, downloads the application as a proxy of number of users. The assumption is that a subscriber, after registering, downloads the application and uses it.

Figure 4 shows the percentage of projects per population of subscribers in the sampling of January 2002. Considering all projects, 80% of projects have less than 11 subscribers. 42% of projects have 0 to 2 subscribers. 10% have more than 20, 1% more than 100.

Very recent projects are not likely to have many subscribers. Considering only projects older than one year the overall numbers drop considerably (157 instead of 406), but the result does not change much, 79% of projects have less than 11 subscribers. One interpretation is that age of the project is not a driver for subscribers, other factors could be more meaningful. Another possibility is that number of subscribers is not an indicator of success. For the first of these two interpretations, analyzing only projects older than one year we find a quite total absence of correlation between the project’s age and its number of subscribers (-0.08); the maximum number of subscribers (270) is found for a project younger than one year at the time of the first observation, and it’s a multimedia application. We also notice that 144 aged projects out of 339 (42%) could gather at most one subscriber. Among those we observe 101 projects out of 194 (52%) from Scenario 1, 15 from Scenario 2, 32 from Scenario 3, and 20 from Scenario 4. That could mean that a project that can’t attract new developers is not able to attract new users as well.

One could expect to see a lot of users where the application fills some unfulfilled need: a first proxy to check this point is the application domain. We select projects with a good number of users (more than 10). We find that 72% of application domains in that pool of projects consists of horizontal applications, as to confirm that OS software is developed by experts and it is used by experts.

After performing these few basic and static analyses, we are interested in the evolution of subscriptions over time. Over two samplings the subscriptions’ measure is not monotone, in the sense that it can increase as well as decrease (one can easily unsubscribe from a mailing list). We notice that 67 projects out of 194 (35%) could gather new subscribers in Scenario 1, 15 out of 30 (50%) in Scenario 2, 61 out of 110 (55%) in Scenario 3, and 50 out of 63 (80%) in Scenario 4. A driver for the presence of users and their growth seems to be the presence of a core group driving steadily the evolution of the project. Globally, 125 projects out of 406 had a growth both on code size and on subscribers, 168 projects had a growth on code size, and 197 projects had a positive evolution on subscribers: on 37% of cases, an evolution of subscribers didn’t lead to an evolution of code. If we compare the different scenarios, we find also that 17% of projects on Scenario 1 had a growth in both the attributes, Scenario 2 and 3 had a similar percentage (33% and 35%), while 62% of projects in Scenario 4 show an absolute variation on code size and a positive variation on subscribers. We can say that the more the presence of users, the more the code will change.

5.6 Modularity and documentation (Q6, Q7)

As a proxy of modularity we use the number of directories the source code is divided into. The possible values of this attribute are one directory (dirLev 1), two (dirLev 2), more than two (dirLev 3). 163 projects (41%) have one directory, 123 projects (31%) have two, while 108 projects (27%) have more than two.

For documentation we define three documentation levels: comments in the source code (docLev 1), a README file or a Unix-like MAN page (docLev 2), availability of a user manual or API specification (docLev 3). 60 projects out of 398 (the difference with the data set of 406 being the commercial projects hosted by FreshMeat, for which source code is not available) have documentation level 1, 176 projects (44%) have level 2, while 159 projects have level 3.

We then perform some basic analysis to understand what is the grade of relationship among these two attributes and the code size. As one could expect, modularity is related to size. In the global sample, we observe that, in the presence of few directories (dirLev 1 and dirLev 2) the code size of the project remains quite low: 81% of the projects having dirLev 1, and 88% of the projects having dirLev 2 maintain a code size between 10 and 1,000 Kb. With dirLev 3 48% of the projects have a code size between 10 and 1,000 Kb, while another 48% of the projects have a code size between 1,000 and 10,000 Kb. We interpret this
as evidence that the developers reorganize code and add modules as far as the project grows.

If we analyze documentation level and code size, we observe again that poor documentation levels correspond in most of the cases to small projects (97% of the projects having docLev 1, and 95% of the projects having docLev 2 never exceed 1000 Kb). On the contrary, 38% of the projects with documentation level 3 reach larger dimension (between 1,000 and 1,0000 and more). Again, this suggests that the developers add documentation as the project grows.

Going further in our analysis, we look for some relations between modularity, documentation and the number of developers, being stable or transient. What we find is shown in Error. L’origine riferimento non è stata trovata.. When the source code is little modularized (DirLev = 1), we observe more than one stable developer in 23 cases out of 167 (around 14%); we observe 5 cases out of 167 (3%) if we raise the level of stable developers to three. When the source code is more modularized (DirLev = 2), we observe more than one stable developer in 35 cases out of 123 (28%), while we observe more than three stable developers in 13 cases out of 123 (10%). Finally, if the directory level is higher (DirLev = 3), we observe 53 cases out of 108 (49%) where there are more than one stable developer, while we observe 26 cases out of 108 (24%) where the parallel effort is made by more than three stable developers. We conclude that the average number of projects counting on a team (two persons or more) of stable developers doubles itself when modularity increases. The explanation can be either that the participation of more developers obliges to more modularization. Or that more developers produce more code and this obliges to more modularity.

We expect also that the number of transient developers raises as well: that is showed in Figure 6. We observe that when DirLev is 1, there is more than one transient developer in 36 cases out of 167 (21%), and there are more than three transient developers in 17 cases out of 167 (10%); when the DirLev is 2, 52 out of 123 (42%) and 34 out of 123 (27%) are respectively the cases when we see more than one and more than three ‘transient’ developers. Finally, when observing cases with directory level 3, there are 63 projects out of 108 having DirLev=3 with more than one transient developer (58%), while there are 50 projects out of 108 having more than three transient developers.

Then we analyze documentation level vs. stable developers. In Figure 7 we observe the results of this second analysis. We observe that the ratio of projects having more than one stable developer changes from 13% (DocLev = 1) to 19% (DocLev = 2) and to 43% (DocLev = 3). If we raise the threshold we find the values 1.67% (DocLev = 1), 5% (DocLev = 2), and 21% (DocLev = 3). If we analyze the number of transient developers: 18% is the value of projects having more than one transient developer and docLevel 1, 32% with docLevel 2, and 52% with docLevel 3.
Our interpretation is that a rising number of developers is the cause of the increase in documentation and modularization levels.

Figure 8 – DocLevel and Transient Developers

5.7 Evolution (Q7)
The portal owners define for each project two indicators of health, vitality and popularity (see Table 1), the former linked to the evolution of the project, the latter to its success with users. We found that 23% of the projects increased their vitality in the period, while 36% of the projects increased their popularity, in other words only a minority of projects evolves and improves. From our point of view we consider a simpler indicator of change, the variation of size and the variation of subscribers. We found that around 48% of the projects increased their subscriptions. Analyzing further, we formulated the term ‘active’ for those projects increasing each of those four attributes (vitality, popularity and subscribers and developers), giving an even narrower sample satisfying the conditions: 60 projects out of 400 (15%) fall in that category. 85% are considered ‘lethargic’.

Over six months, 90% of projects did not change status. Status is defined by the portal owners (see Table 1) as an indicator of the development stage of a project. That is not surprising, since usually this attribute characterizes a project for a long time (as observed in a further, but limited, sampling made in 2000), and even more since over 60% of the projects lay between a ‘stable’ status and a ‘mature’ status, which can be considered as final for a project. The difference among the two lays in the ability to avoid any crashes on the platform supported (stability), and in the further satisfaction of all the requisites (maturity).

Regarding the number of version released in a period, we divided the projects in the categories ‘static’ (no new versions released) and ‘upgraded’: 57% of them fell in the first category, while 43% had at least one release in the period studied. The maximum number of releases in the analyzed period was 21, while the average global number of releases is 1.3: if we take into account the sole upgraded projects, we find that the average number of releases in-between was 3.

Finally we considered the evolution of code size. At first we considered the distribution of code sizes: we gathered the projects having 0 to 10 Kbytes of code, 10 to 100, 1000 to 10000, and over 10000. Over 40% of projects have less than 100 Kbytes, while only 1% has more than 10000 Kbytes. Figure 9 shows this distribution. Few projects could evolve into any subsequent categories: for example, projects laying between 1000 and 10000 Kbytes passed from 160 to 163.

Figure 9 – Distribution of Code Sizes

We then considered four categories of code evolution: a code variation of more than 50%, between 10% and 50%, between 0% and 10%, and no code variation. We considered all the evolutions as absolute: so negative evolutions were also taken into account since they imply some rework over the project. Figure 9 shows this dynamic distribution. An overview of these results indicates 63% of projects not changing size over six months, while 34% of projects changing less than 1%.

Figure 10 shows the variation of code size in the time interval between the two observations. More than half of the projects do not change code size, meaning that no work is made on them, or at most small corrections are made. Around 5% more than double size.

6. VALIDITY THREATS
Some validity threats were acknowledged during the course of our study, we discuss them here.

6.1 The portal syndrome
We are aware that FreshMeat, as well as other portals dedicated to OSS, are often used by developers as an ‘advertising’ place where showing their private efforts: some of those efforts are worth adding external work, while others aren’t. We are aware also of the fact that some major OS successes boost their developers pool through the recognition and the relevance of other developers: Fresh-
Meat can’t resolve the problem of visibility of projects as long as it is driven by ego-boosting ([7] and [8]). These facts for sure have an effect on the presence of many, small and lethargic projects in the data set considered.

6.2 Pseudo-sampling
FreshMeat doesn’t list its projects in an unstructured set where choosing from, but rather filtered through some categories (programming language, application domain, etc.). In our sampling we chose the ‘status’ attribute as filter to select the projects: this created six subset of projects, one per status. Then from each subset we selected randomly half of the projects. This resulted in the data set of 406 projects, corresponding roughly to half of the active projects hosted in 1999. (The total number of projects hosted raised considerably in subsequent years, up to several ten thousands). While the constraint of sampling from subsets of projects instead of from the global set may not be pure random sampling, we believe the problem is fixed by sampling a large (50%) part of the project set.

6.3 Proxies
We used different proxies through this study, whenever the relevant attribute was not measurable.

We assumed that the number of subscribers to a formal news reporting (“mailing list”) could be an acceptable proxy for the number of users.

The number of directories is used as a proxy of modularity of a project. The date of first posting of the project on the portal is considered as age of the project. And the number and type of documentation files is considered as a proxy of documentation level.

When talking about a project’s evolution, we assumed that a project didn’t evolve in time if there weren’t any releases (i.e. variations in the version number) in the time interval we chose: this can be false when the developers don’t change the version number due to early stages of evolution, or minor fixes in the code.

6.4 Change logs
Many assumptions made in our study were based on the presence of a change log documenting the phases which the project passed through during its life. Based on the OS culture, we assumed that all the changes made by persons other than the core developers were documented in the change log and then further repeated in a THANKS file, or in a CONTRIBUTORS file. This assumption may not be true for all projects.

A related problem involves our definition of stable developers, based on analysis of change logs: we define stable developer a person contributing more than one piece of code, and that is applied for all kind of patches, from trivial to considerable ones. In this sense our approximation can be severely biased, since we could have considered as ‘stable’ some developers submitting more than once the same original piece of code.

7. CONCLUSIONS AND FUTURE WORK
We have presented the initial results from a horizontal study on a set of around 406 projects hosted on the FreshMeat portal.

Size of projects is typically small, GPL the most popular license. Horizontal application domains prevail (66%).

Most used languages are C, C++, PERL. Surprisingly, Java comes after those.

Number of developers is typically low: 57% have one or two developers. Only 15% of them have more than 10 developers. We believe for the latter category only the issue of chaotic vs. organized development becomes meaningful. For this category, a core team of coordinators exists, and the ratio of coordinators to developers is, on average, 1 to 4.

Number of subscribers (a proxy of users of the application developed by a project) is also low. 80% of projects have less than 11 subscribers, 1% has more than 100. Surprisingly, the number of subscribers does not appear to grow with the age of a project, nor with its size. 72% of projects with more than 10 subscribers belong to a horizontal domain. To us, this confirms that successful OS software is developed by experts for experts.

As for evolution of projects, another striking fact is that, over six months, 97% of projects did not change size or changed less than 1%.

These data suggest that, despite the huge number of OS projects listed on OS portals, the overall effort put in OS projects (and the pool of developers) is a scarce resource that concentrates on very few projects. Very successful OS projects such as Linux and Apache are probably not the ‘average’ OS project.

While some validity threats must be considered to interpret these results correctly (especially the use of portals as an advertising means that inflates the number of single developer projects with no chance of success), we believe
that this analysis can bring useful insight in the debate about the Open Source movement.
Future work will be dedicated to further analyze the data collected, and to filter out noise from the data set. Besides, new samples will be periodically collected to deepen the analysis on evolution of projects.

8. ACKNOWLEDGEMENTS
We would like to thank Audris Mockus for his precious suggestions and Marco Cantamessa for supporting the initial stages of this work.

9. REFERENCES