Diskussionspapiere zum Tätigkeitsfeld Informationsverarbeitung und Informationswirtschaft, Hans R. Hansen und Wolfgang H. Janko (Hrsg.), Nr. 22, Wirtschaftsuniversität Wien, 2000.

Results from Software Engineering Research into Open Source Development Projects Using Public Data

Stefan Koch and Georg Schneider

Department of Information Business

Vienna University of Economics and BA, Augasse 2-6, A-1090 Vienna, Austria

{stefan.koch|georg.schneider}@wu-wien.ac.at

***Abstract*-This paper presents first results from research into open source projects from a software engineering perspective. The research methodology employed relies on public data retrieved from the CVS-repository of the GNOME project and relevant discussion groups. This methodology is described in detail and some of the results concerning the special characteristics of open source software development are given.**

I. Introduction

Open source software development [21], [6] has generated much interest in the last years, especially following the rise of Linux. As several similar projects like GNU project's utilities and libraries including the gcc compiler and Emacs editor, the Perl and Tcl languages, the Apache WWW server, and the FreeBSD operating system exist, research from a software engineering perspective on this decentralized form of software development should be intensified. As a first step, it seems necessary to assess the differences and characteristics of such projects and their special organisational form [23]. Therefore quantitative research into this form of collaborative development is necessary, which today is very scarce [8], [10].

Open source software is characterized by several differences to traditional software development and distribution, including the free redistribution, the inclusion of the source code, the possibility for modifications and derived works, which must be allowed to be distributed under the same terms as the original software, and some others [18]. One example for a licence that fits these criteria is the well-known GNU General Public Licence (GPL). The guiding principle for open source software development is that by sharing source code, developers cooperate under a model of rigorous peer-review and take advantage of "parallel debugging" that leads to innovation and rapid advancement in developing and evolving software products [8], [19].

In this paper we present a methodology for software engineering research into open source projects using data retrieved from a publicly available CVS-repository and discussion lists. This methodology is then applied to the GNOME project and results concerning the programmers and files constituting this development effort together with the progression of the project over time are described.

II. Research Methodology

The main idea for this research into open source software development was to use existing data on the projects available to the public [7]. Therefore the CVS-repository of the GNOME project was used for data collection. It was assumed that several important aspects of a large scale software project in this special form of organisation could be checked using this source [2].

CVS (Concurrent Versions System) is a version control system which is being used extensively in the free software community. Access is accomplished via a client which requires a password authentification. In order to access CVS-archives in a more convenient way the Mozilla project developed Bonsai which allows to connect to a particular archive via a web-based interface.

GNOME, the GNU Network Object Model Environment, is an open source software project building a desktop environment for users and an application framework for software developers. This vendor neutral project includes a set of standard desktop tools and applications, e.g. the well-known GNU Image Manipulation Program (GIMP), and uses the Common Object Request Broker Architecture (CORBA).

To provide for further refinement, additional sources of data, e.g. discusssion lists and bug-tracking archives, were also identified. As all data retrieved needed to be managed, storage in a database was chosen. Therefore, a data model of an open source software project was developed to include all publicly available data (see Fig. 1).

The following notes seem to be necessary for understanding the entity-relationship model presented: There exist both coders (or programmers) that actually do work on the project by submitting (“checking in”) files. On the other hand, there are posters that participate in discussions pertaining to the software. One real-world person can fulfill both roles, as will be most common, but the possibility exists for people to only post messages in discussion lists or programmers who do not participate in discussions. A file, as identified by a filename and a directory path (which is necessary as some filenames are duplicates, e.g. a file named “makefile” exists in several directories) can be checked in to the CVS-system by one and only one programmer. The CVS-repository then stores this checkin with the changes in the lines-of-code (LOC) and further data. A posting is a separate message to a discussion list pertaining to the GNOME project, maybe in reply to a prior posting. A module consists of several files and constitutes a self-contained part of the GNOME project (e.g. gnome-core, gnumeric, gimp). A module can therefore also be viewed as a separate sub-project. A release is a complete and public version of a predefined set of modules of the GNOME project.

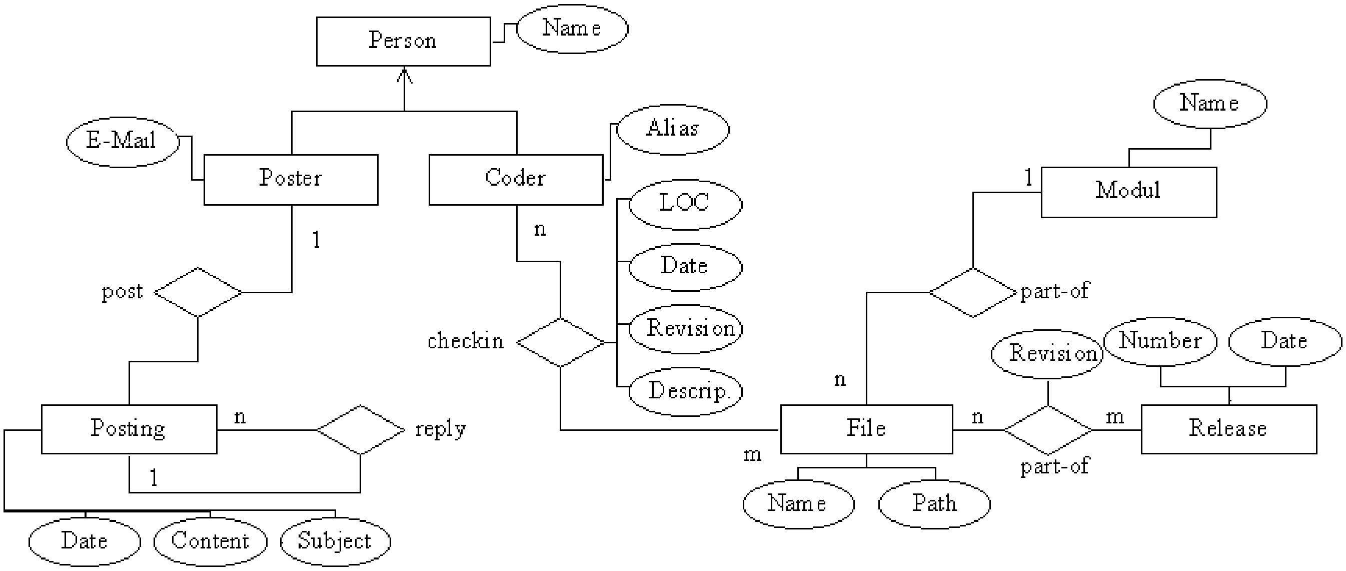

Fig. 1. Entity-relationship model of data available concerning the GNOME project.

As a first step, the web interface of the CVS-repository was used to retrieve the necessary data concerning checkins. This data included programmer, file, date, lines-of-code added and deleted, revision number and some comment given by the programmer for every checkin. This was done with a Perl-script which generated successive queries simulating a browser-based input form. Each query concerned a particular day in the history of the CVS archive. The requests were distributed over a four day period in order to distribute the load. The result of each individual query was a HTML page which was subsequently parsed extracting the necessary attributes conforming to the data model exposed above and fed into a database (Postgresql under Linux). The necessary queries were then performed and the output analysed using a statistical package. Of course, the data concerning programmers was strictly anonymized.

Also retrieved by a Perl-script were the postings to the relevant discussion lists including the sender, the subject, time and complete text. For the analysis of the posting behaviour of the programmers, the short name each programmer uses for checkins had to be matched to the full name or e-mail adress used for posting. For 175 persons this has been possible using several regular expressions with human check-up.

This approach of using existing information publicly available eliminated one of the most pressing problems in software engineering research, the lack of data, especially concerning the past history of projects. In addition, this approach does not intrude on the software project under consideration and is inexpensive [7], [2].

III. Results

This section details the results from analysis of the CVS-repository data and the discussion lists. For the most part, these results concern the programmers, e.g. their participation, and the files composing the software development effort. In addition, the changes in the project over time are explored. Before these results are presented, the metrics used in the analysis are described.

A. Metrics used

The metrics described here were either derived directly from the CVS-repository or have been computed from this data. Some of these can also be directly seen on the data model presented above (see Fig. 1).

The first metric to be used is the number of lines-of-code (LOC) added to a file. The definition of this often disputed metric LOC [11], [16] is taken from the CVS-repository and therefore includes all types of lines-of-code, e.g. also commentaries. In addition, any LOC changed is counted as one line-of-code added and one line-of-code deleted. The grand total of LOC added was 6 300 000 for the whole project and all programmers. The next metric is defined analogous and pertains to the LOC deleted. For the whole project, 4 500 000 LOC have been deleted.

The difference between the LOC added and the LOC deleted therefore gives the change in size of the software artefact under consideration in the corresponding time period. These changes can be cumulated over time to give the size at any moment.

The metric of checkin refers to the submission of a single file by a single programmer. Overall, 220 000 checkins have been made for the project.

The time spent on the project is defined for every programmer as the difference between his first and his last checkin. As this therefore includes all time elapsed, not necessarily only time spent actually working on the project, this measure can only give an upper bound for actual time spent working (and no time sheet data as in [2] are available).

In addition, a programmer is defined as being active in a given period of time if he performed at least one checkin during this interval. The total time spent by all programmers on this project equals 74 000 days, roughly 200 years. When the results from a questionnaire to Linux developers [10] are taken, which give a mean of 13.9 hours per week spent on development, this translates to about 145 000 person-hours of effort (or 954 person-months). As the following comparisons of results will show, this measure can indeed be taken. For each file, the time worked on is also defined as the time elapsed between the first and the last checkin. The same considerations of course apply as for programmers, i.e. this measure can also only be taken as an upper bound.

Further metrics computed from those described above are the LOC added per checkin and the LOC added per hour. Both are derived from division of the corresponding values.

In addition, the postings made to mailing lists pertaining to the GNOME project were analysed. The most important metric derived is the number of postings. The grand total of postings made during the observed time period was 19 909. A posting concerning another posting made earlier in time is defined as a reply.

B. *Data on persons involved in the project*

Open source software development is assumed to be performed by more people than traditional development, as these do not spend all their time working on the project. In the GNOME project, 301 programmers were identified that currently work, or have worked upon, this software. As will be shown, these programmers differ significantly in their effort for this software project. For additional data on contributors to open source development, including their country of origin, see [8] and [10].

The mean LOC added in total by a given programmer was found to be 21 000 with a standard deviation of 67 000. The corresponding values for LOC deleted were 15 000 and 48 000. The maxima were 931 000 for LOC added and 621 000 for LOC deleted. Fig. 2 shows the values of LOC added for all programmers sorted by this value and Fig. 3 the corresponding histogram. As one can see from these results, there are indeed significant differences between the programmers, with a majority contributing a quite small amount, a result also found in [8]. In [10] the Linux kernel developers reported a mean contribution of 2 648 LOC with a standard deviation of 9 268 (see Fig. 4). Again, the distribution within the population is similar to the results from other sources. Also the number of patches submitted to the kernel reinforces this finding [10].

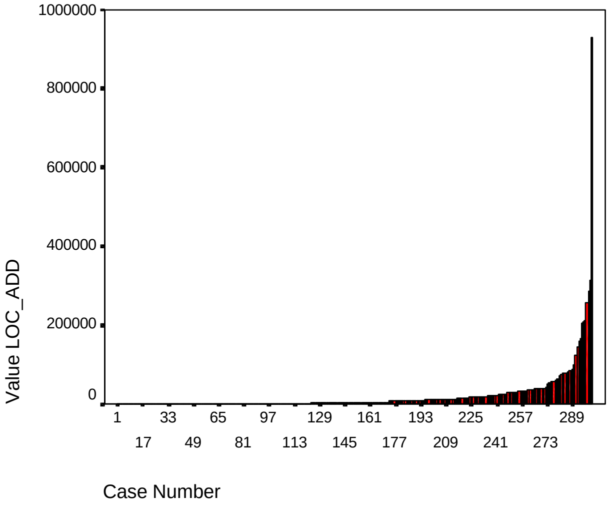

Fig. 2. LOC added for each programmer (sorted by LOC added).

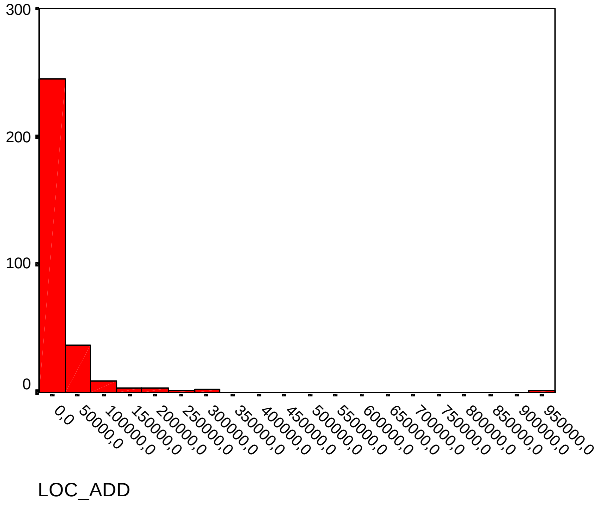

Fig. 3. Histogram of LOC added per programmer.

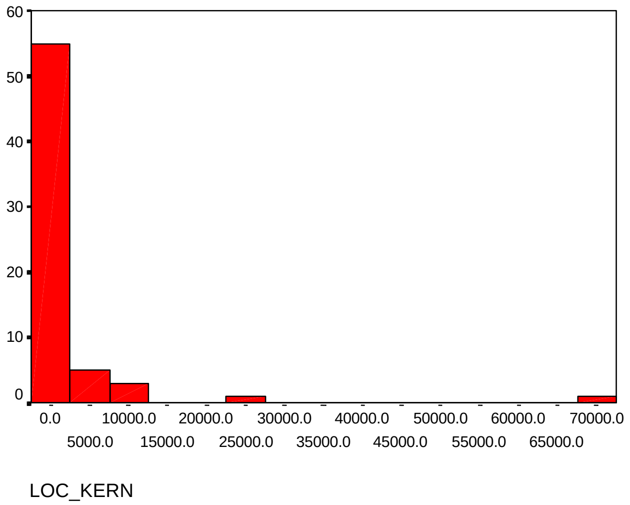

Fig. 4. Histogram of LOC per Linux kernel developer [10].

The number of checkins performed by a programmer was in the mean 731 with a standard deviation of 1 857 and a maximum of 23 000 (see Fig. 5 for the number of checkins for each programmer, Fig. 6 for the corresponding histogram).

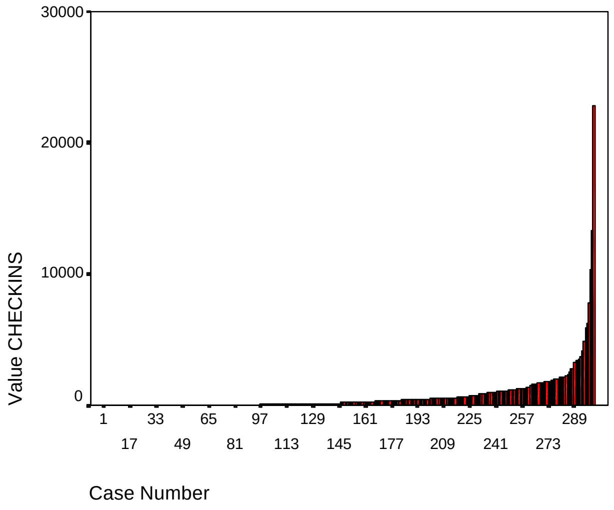

Fig. 5. Number of checkins for each programmer (sorted by number of checkins).

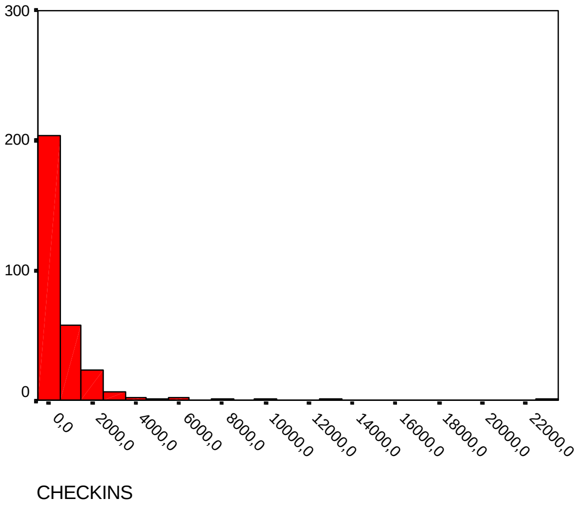

Fig. 6. Histogram of number of checkins per programmer.

The time spent on the project had a mean of 246 days, a standard deviation 213 and a maximum of 993. In [10], Linux kernel developers reported a mean time of 17.2 months (standard deviation of 22.3) of involvement (but of course this project has been in existence for a longer period of time). The LOC added per single checkin had a mean of 28, the LOC deleted of 20. The standard deviations were 38 and 35 respectively, and the maxima 287 and 238. The productivity as measured in LOC added per hour (again taking the mean value of working hours per week from [10]) had a mean of 53 with a standard deviation of 166. Fig. 7 shows the total time spent on the project for each programmer, Fig. 8 gives the corresponding histogram. Compared to the data concerning the number of LOC added and deleted, the differences between the programmers seem much smaller. This finding will be explored further later on.

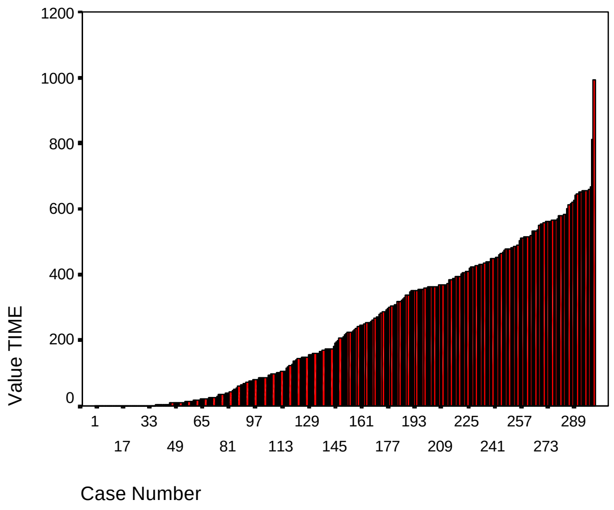

Fig. 7. Time spent on the project for each programmer (sorted by time spent on the project).

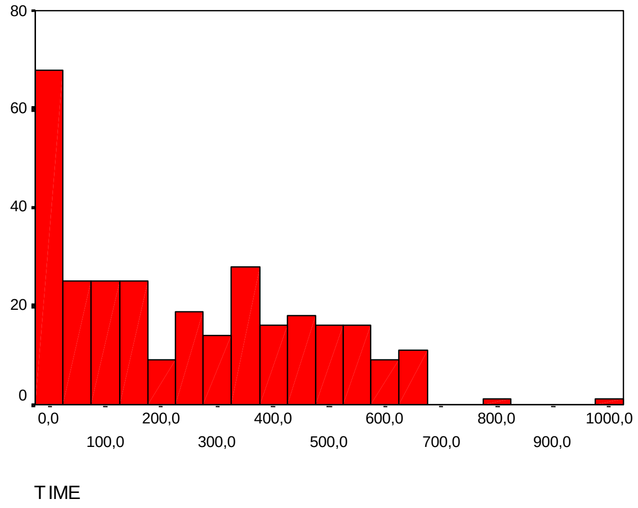

Fig. 8. Histogram of time spent on the project for each programmer.

Next, possible relations between these variables were explored. Fig. 9 shows a scatterplot for total of LOC added plotted against LOC deleted, Fig. 10 for LOC added against checkins and Fig. 11 for LOC added against the time spent on the project.

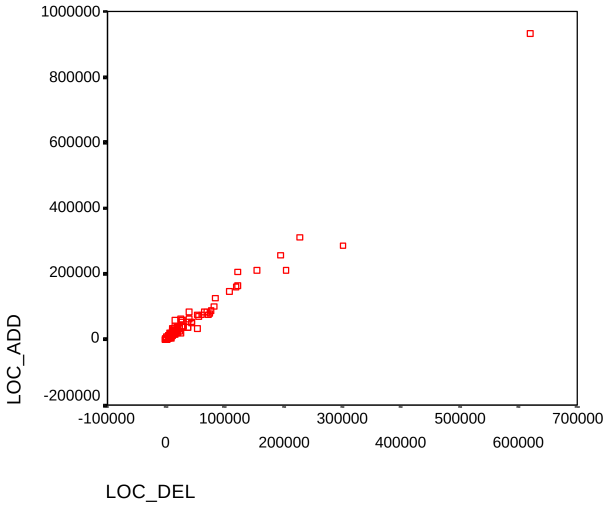

Fig. 9. LOC added against LOC deleted for each programmer.

As one can easily see, these variables seem to correlate strongly. In fact, a Pearson correlation of 0.985 was found in further analysis. As these measures include any single LOC changed as one LOC added and one LOC deleted, this correlation does not seem to be surprising, and might hint at the existence of a high number of changes in the source code.

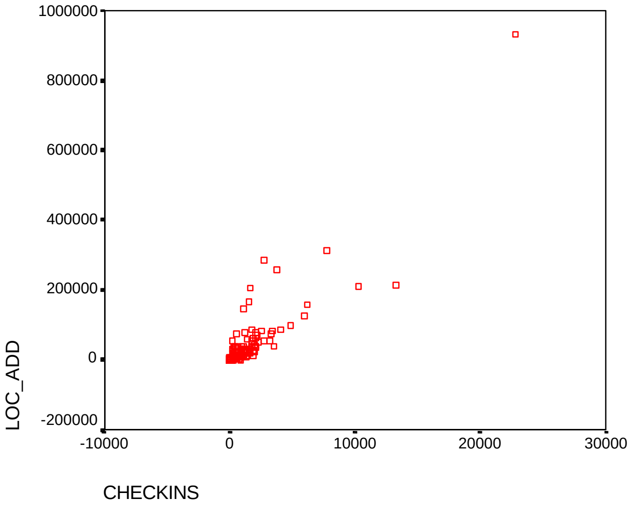

Fig. 10. LOC added against number of checkins for each programmer.

There is also a high degree of correlation to be found between the LOC added and the number of checkins. The Pearson correlation is 0.894, so it can be said that programmers that add more LOC also use more checkins. As the correlation between this total LOC added and the LOC added per checkin is not high (Pearson correlation of 0.166) there is no relation stating that very active programmers use bigger checkins, i.e. those containing more LOC. This could be taken as an indication that there is no significant difference in programming style between programmers.

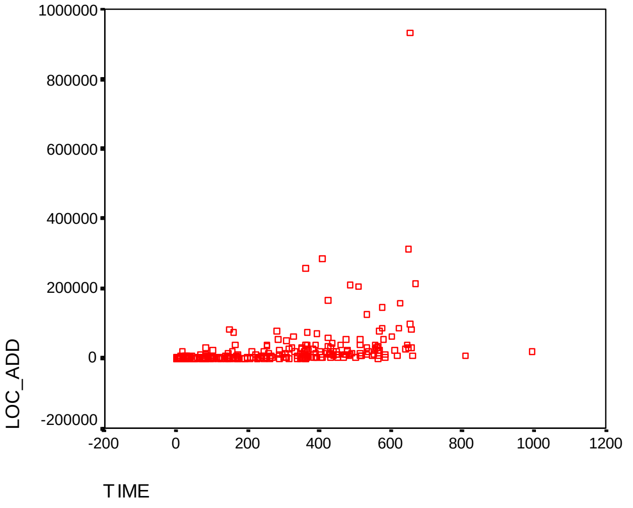

Fig. 11. LOC added against time spent on project for each programmer.

As the scatterplot suggests (see Fig. 11), there is no strong relation between time spent on the project and the total number of LOC added, used as a measure of output. The Pearson correlation between these variables is 0.349, so it can not be said that programmers who stay on the project longer also contribute significantly more. In [10], a similar correlation coefficient of 0.405 is found. It is to be assumed that this constitutes a difference to commercial software development, where people spend all their (working) time on the project, thereby constantly generating output, while participants in open source development only donate some of their spare time, most often irregularly, to this effort.

Data on the postings made to discussion lists pertaining to the GNOME project shows 19 909 postings made to several discussion lists out of which 6 903 had been replies to other postings. 1 881 different posters have been identified. The mean number of postings per person therefore is 11.

Data on the postings made by programmers was also explored (see Fig. 12). Each programmer did post a mean of 43 messages with a standard deviation of 116. These programmers in sum contributed 7 455 messages out of the total of 19 909 messages retrieved from the discussion lists. The mean of programmers with 43 messages per person is therefore higher than the mean of all different 1 881 posters identified. For the group of programmers, as the scatterplot indicates (see Fig. 13), a correlation of 0.691 could be found between number of postings and sum of LOC added, showing that more productive programmers are also more active participants in the discussion lists.

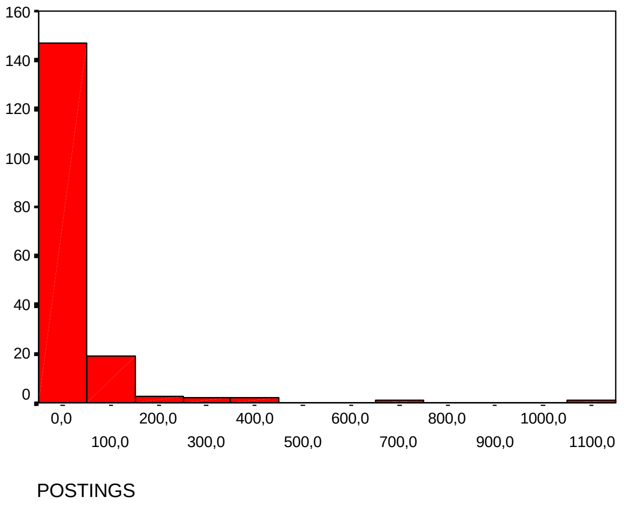

Fig. 12. Histogram of postings made by each programmer.

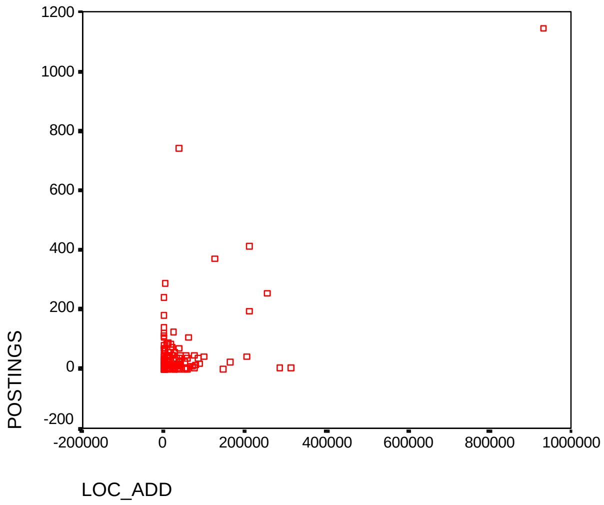

Fig. 13. LOC added against postings made for each programmer.

A cluster analysis was performed to discover groups of programmers with similar characteristics and observe the associated distribution in the population. The analysis was based in a first step on the total of LOC added and set to produce five clusters. The first cluster had a center of 931 000 LOC and contained only one programmer. The second cluster had a center of 255 000 LOC and contained five programmers, an equal amount to the third cluster with a center 159 000 LOC. The fourth cluster cointained 41 programmers and had a center of 50 000 LOC. The fifth and last cluster contained 249 programmers with a center of 5 000 LOC. A reduction to 3 clusters did not change this result significantly, as the clusters two and three and four and five were pooled, therefore resulting in a distribution of 1, 10 and 290 programmers. A clustering using both LOC added and time spent on the project produced the same clusters with time spent on the project decreasing between the groups in accordance with the total of LOC added, but not as sharply. These results clearly show that there is a minority of programmers that produce most of the output, as has also been visible from Fig. 2 and Fig. 3. The existence of such an "inner circle" can be assumed to be another difference to traditional forms of software development, where the effort will be in most cases split more evenly between a smaller group of people.

C. *Data on files*

Since the beginning of the GNOME project 38 634 files have been worked on. Of course, these files differ significantly in size and complexity. The total of LOC added to any file was taken as a metric for it's size, the number of checkins was used to gain an understanding for complexity, as more changes and therefore checkins were postulated to be necessary for more complex files.

The mean for LOC added in total for a given file was 163 with a standard deviation of 1 136 and a maximum of 60 000 (see Fig. 14 for the histogram). The LOC deleted were found to have a mean of 117, standard deviation of 984 and a maximum of 60 000.

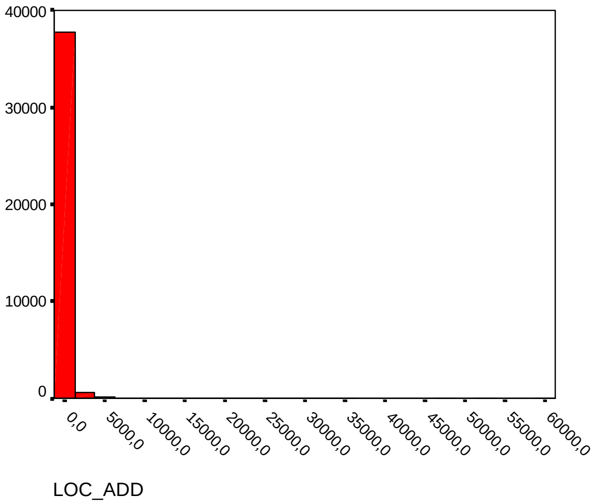

Fig. 14. Histogram of LOC added per file.

The number of checkins for a given file had a mean of 6 with a standard deviation of 18 and a maximum of 1 583. Fig. 15 shows the corresponding histogram.

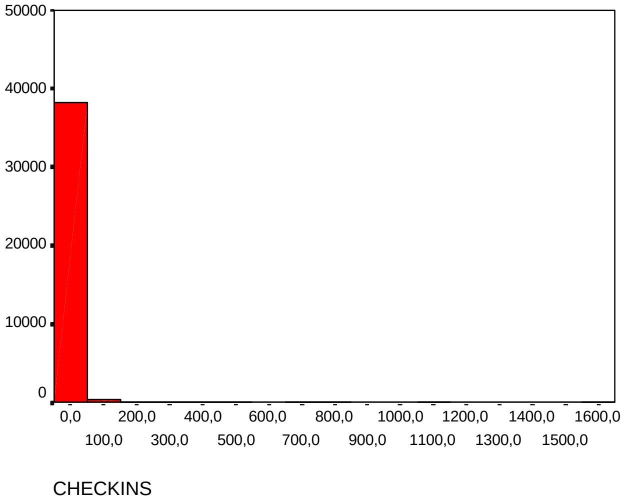

Fig. 15. Histogram of checkins per file.

The time worked on a given file had a mean of 95 days (standard deviation 152, maximum 971, see Fig. 16). The number of LOC added with a single checkin for a file had a mean of 15 with standard deviation of 152 and maximum of 20 000.

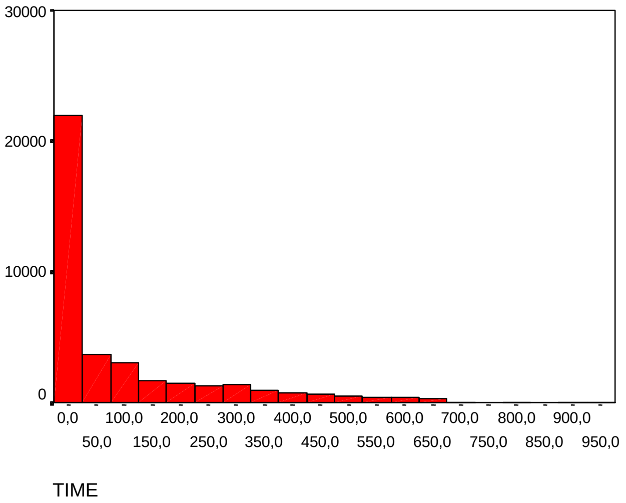

Fig. 16. Histogram of time worked on each file.

Again, the possible relations between these variables described above were explored using scatterplots (see Fig. 17, 18 and 19).

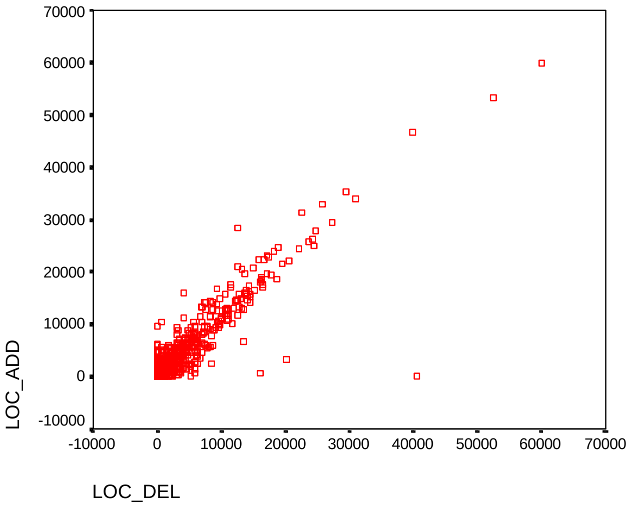

Fig. 17. LOC added against LOC deleted for each file.

The correlation between the number of LOC added to and deleted from a given file is again very high with a Pearson correlation of 0.933. Of course, the same comments as for programmers apply.

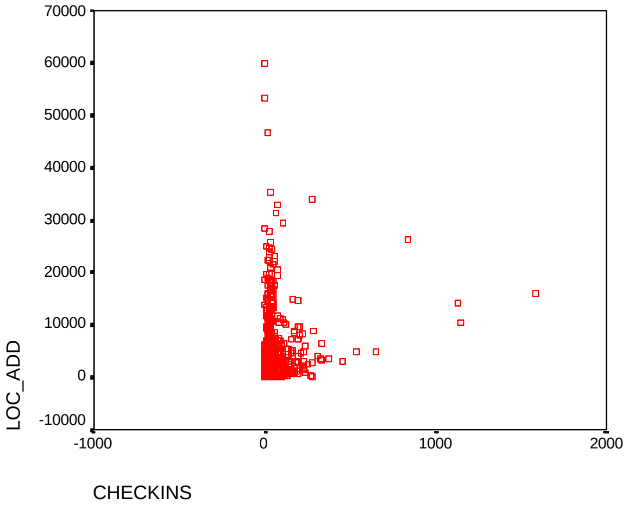

Fig. 18. LOC added against number of checkins for each file.

A correlation between the LOC added and the number of checkins exists but is not very strong (Pearson correlation of 0.341). As a stronger correlation of 0.602 can be found between the total LOC added to a file and the LOC added per single checkin, this seems to indicate that larger files are checked in using greater chunks. In addition, the measure of total number of checkins taken as indicator for complexity does not correlate strongly with size as taken from the total of LOC added. This is in contrast to the findings of [13], who have found a high correlation between size and complexity measures, and [3], whose results have shown the size as having a significant impact on maintenance costs.

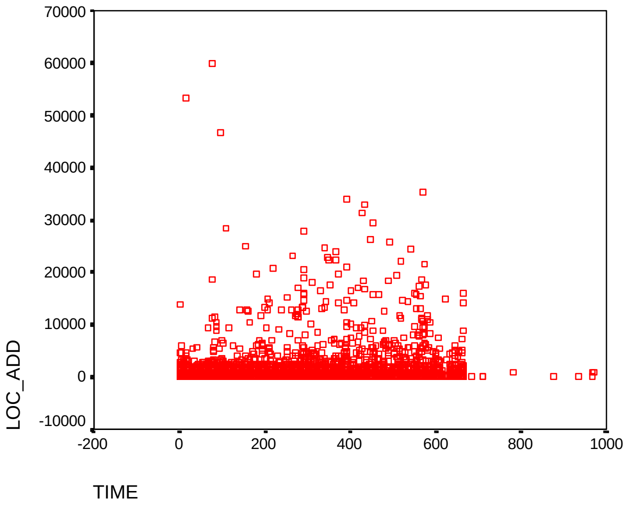

Fig. 19. LOC added against time worked on each file.

As for programmers, the relation between time and total effort (as measured in the sum of LOC added) is not very strong. This Pearson correlation of 0.196 indicates that the time spent between first and last checkin of a single file is not a good predictor for the total effort spent on this file. This also corresponds to the relation that larger files are checked in using larger checkins, as therefore the time worked on large files need not be longer than for smaller ones (assuming constant time intervals between checkins).

A cluster analysis was also performed on the files to gain an understanding for groups with common features and their distribution in the total population. The first cluster analysis was based on the total of LOC added and set to produce five clusters. The first cluster contained two files and had a center of 57 000 LOC. The second cluster had a center of 34 000 LOC and contained seven files, the third cluster 18 000 LOC with 65 files. The fourth cluster contained 226 files and had a center of 6 600 LOC. The fifth and last cluster contained the vast majority of files (38 000) and had a center of 86 LOC. A change to 3 clusters did not produce any significant changes. Using both the total number of LOC added and the number of checkins for production of five clusters showed some differences. The first cluster contained three files with a center of 53 000 LOC and 9 checkins. The second cluster contained nine files with 30 000 LOC and 161 checkins, the third cluster 77 files with 16 000 LOC and 71 checkins. The fourth cluster had a center of 5 800 LOC and 66 checkins and contained 246 files. The last cluster again consisted of the majority of 38 000 files and had a center of 83 LOC and 5 checkins. These results show that there is a vast differnce between the files in both size as given by total of LOC added and number of checkins. The vast majority of files is quite small and is not changed very often.

D. Data on programmers' work on single files

Preliminary analysis was also performed for the relations of programmers with files. In the mean, 1.8 distinct programmers work together on a single file. A given programmer added in the mean 90 LOC to a given file he worked on (standard deviation of 658, maximum of 60 000). He deleted a mean of 64 LOC with standard deviation of 595 and maximum of 60 000. The number of checkins a single programmer performed for a given file was 3 (standard deviation 7 and maximum 492). The size of a single checkin for a given file was in the mean 22 with a standard deviation of 208 and a maximum of 20 000. Further analysis is prepared to explore the association between programmers and files, e.g. if there is a chief programmer for each file adding most of the LOC, if more complex files are worked on by more programmers and so on.

E. Data on progression of the project over time

The progression of the GNOME project over time was also explored to gain an insight into the evolution of this software (see Fig. 20 and 21). The term software evolution refers to the dynamic behaviour of software systems as they are maintained and enhanced over their lifetimes [12]. A paucity of empirical studies in this area to this date has been reported, while most of the existing studies suffered from small sample size due to focusing on system releases as unit of analysis [12].

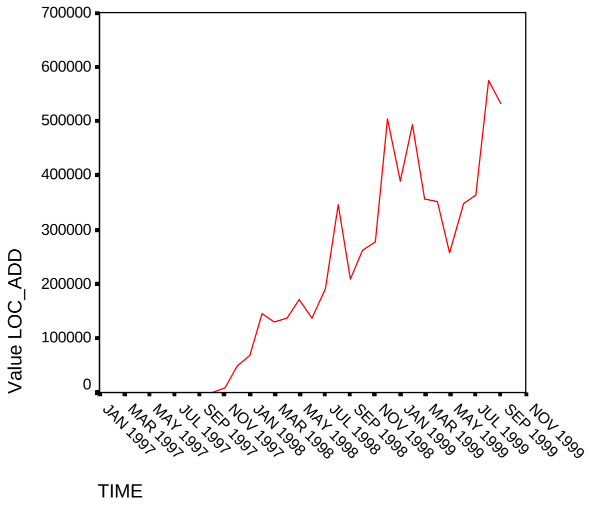

Fig. 20. Total of LOC added in each month.

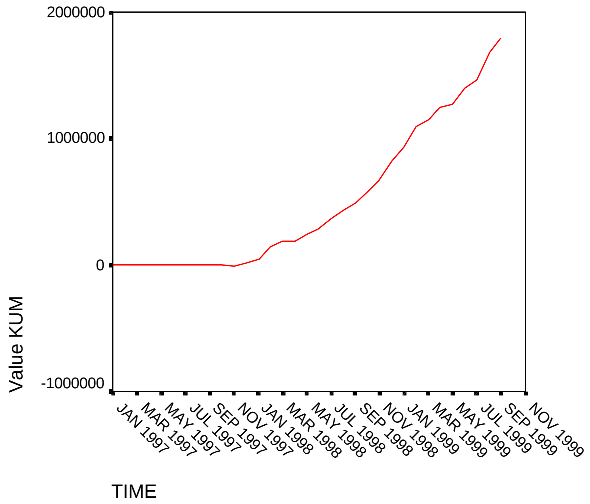

Fig. 21. LOC difference in each month cumulated.

As can be seen, the total size of the GNOME project has experienced a steady increase over time, with the cumulated LOC difference over time taken to gain an understanding of this progression. In accordance with [15], [20] and [17] the development is assumed to become more slow towards the end of the life cycle. A flattened end of the plotted curve is therefore a hint that the project is nearing completion. This point does not seem to have been yet reached for the GNOME project.

For each module, the progression over time was also analysed, i.e. viewing each one as a separate project (see Fig. 22 - 28). This analysis was undertaken to uncover if there were differences in the projects in their progression in the life cycle. It is possible that the total growth of the GNOME project is in different time periods supported by different projects, i.e. that the life cycle of the GNOME project is composed of several sub-cycles of projects whose starting points are shifted in time. The cumulated LOC difference over time was again taken to gain an understanding of the growth of the size of any module with flattened end of the plotted curve as a hint for completion.

As can be seen, there are indications that several of these modules has already progressed to a later stage in the life cycle. For example, the size of the module gnome-core (see Fig. 24) seems to have stabilized at this time. The same progression can be seen for gtk (see Fig. 25) and ORBit (see Fig. 27). As these modules constitute a basis for several others, this shift in time can be explained by the resulting dependencies. These results provide support for the notion presented above that the GNOME project is composed of several sub-projects who start at different points in time and proceed at different speed.

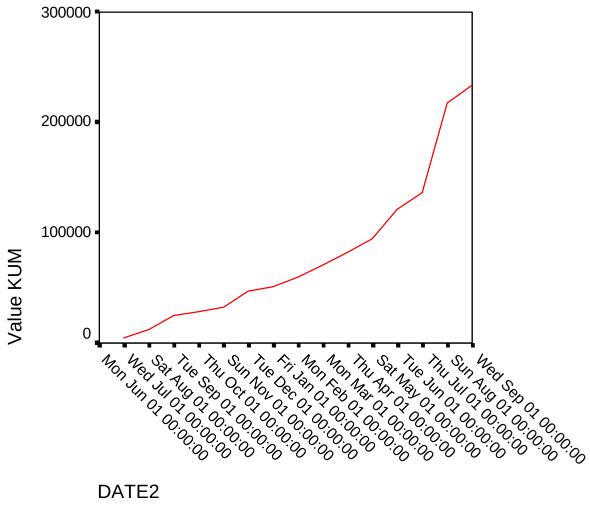

Fig. 22. LOC difference in each month for gnumeric cumulated.

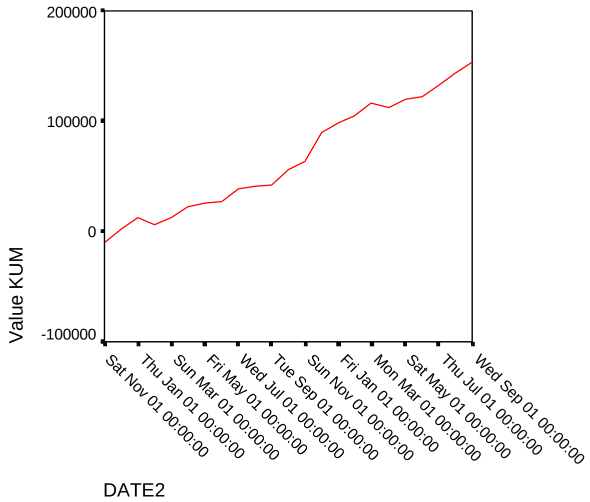

Fig. 23. LOC difference in each month for gimp cumulated.

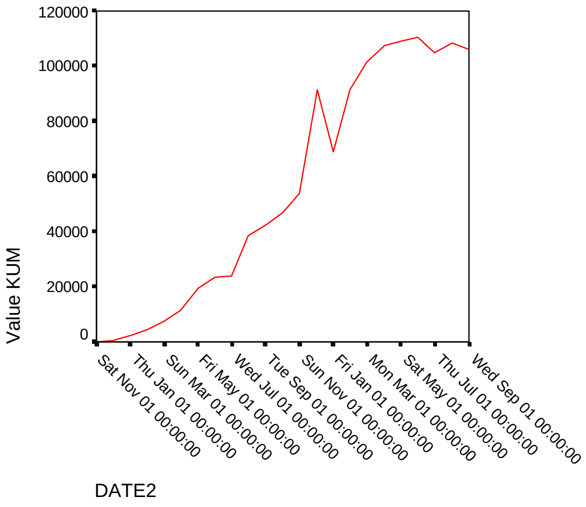

Fig. 24. LOC difference in each month for gnome-core cumulated.

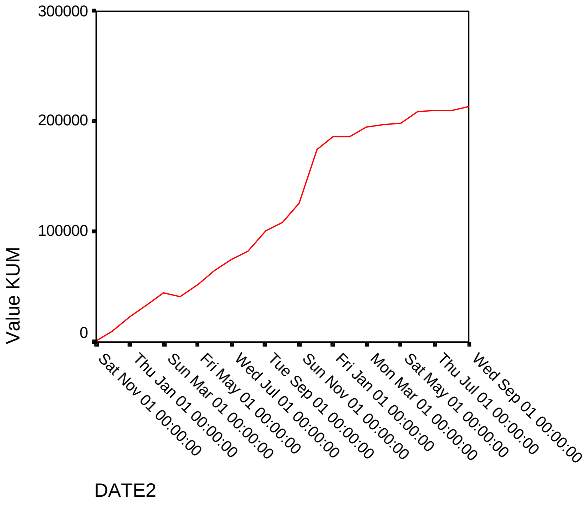

Fig. 25. LOC difference in each month for gtk cumulated.

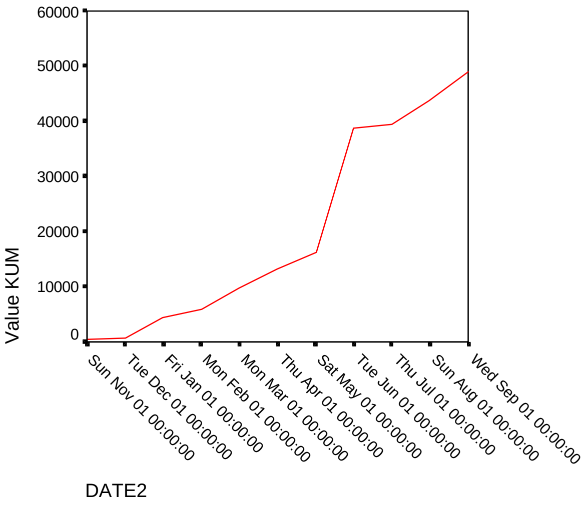

Fig. 26. LOC difference in each month for glade cumulated.

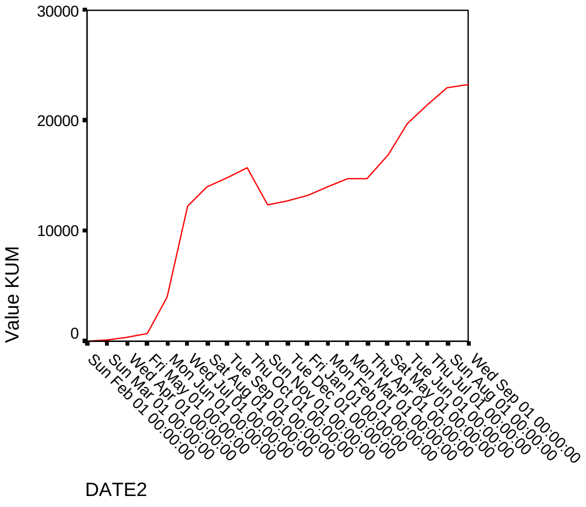

Fig. 27. LOC difference in each month for ORBit cumulated.

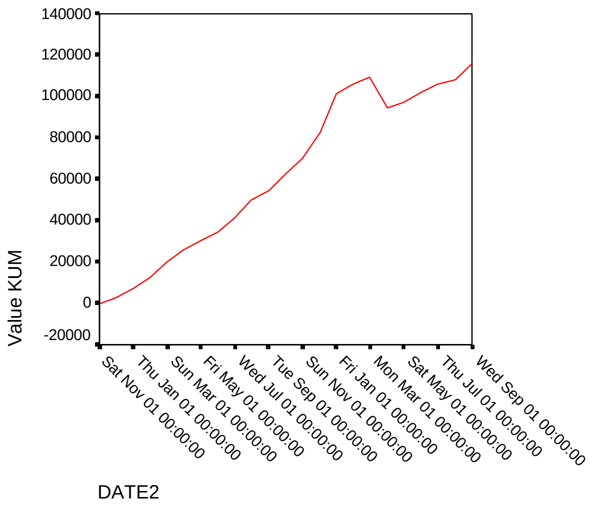

Fig. 28. LOC difference in each month for gnome-libs cumulated.

The most important aspect in open source development projects is the participation of programmers, as a higher number of contributors will also likely lead to more output (this relationship can be confirmed as will be shown in the following). As can be seen (see Fig. 29), the number of active programmers has seen a staggering rise between November 1997 and the end of 1998. During the year 1999 this number has been roughly constant at around 130 persons. The factors resulting in this development can not be seen from the data retrieved for this research, but it can be assumed that marketing-like instruments like press coverage in the open source community (mostly on the WWW) have played an important role. Also many developers will be likely to join an open source project in the beginning (or during take-off) than at the end, as development will be more highly regarded than maintenance and the influence on the whole project could be greater if joined early on. In [10], 43.1 percent of the Linux kernel developers questioned had joined the project during first draft. Another reason for this development could be seen from the research of [15], [20] and [17] who argue that only a given amount of persons can be working on a project in a productive manner at a given time (in relation to the problems ready for solution at this point). In the light of this interpretation, the peak manning of the project has already been reached and will only see a downfall from now on. It is also possible that the organisation of more programmers than the number active in 1999 is not feasible given the structures in place at the moment (e.g. the CVS-system), i.e. that this boundary is not inherent in the problem worked on. This would lead to the conclusion that more effective organisational structures would allow more programmers to work on the project and thus lead to higher output. More work should than be invested in the design of such structures.

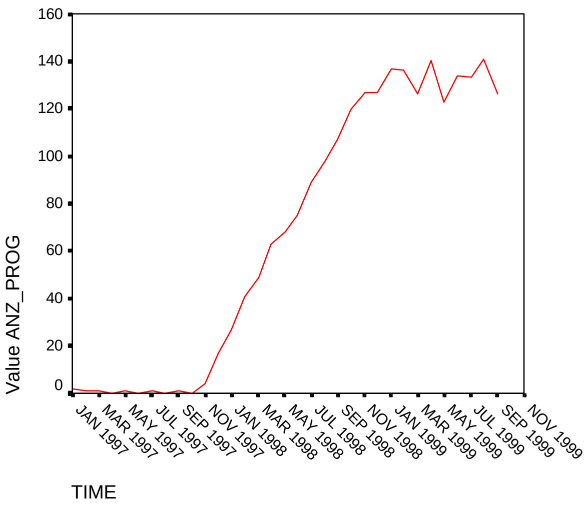

Fig. 29. Number of active programmers each month.

A correlation of 0.932 was found between the total of LOC added and the number of active programmers each month (see Fig. 30). This confirms the intuitive relationship between these two variables and therefore leads to the conclusion, that the aim of each open source project needs to be to attract as many contributors as possible. In spite of this, the striking differences in productivity between the programmers that have been detailled above need to be considered, so pure mass of participants is not enough to sustain such a project but a certain quality needs also to be present.

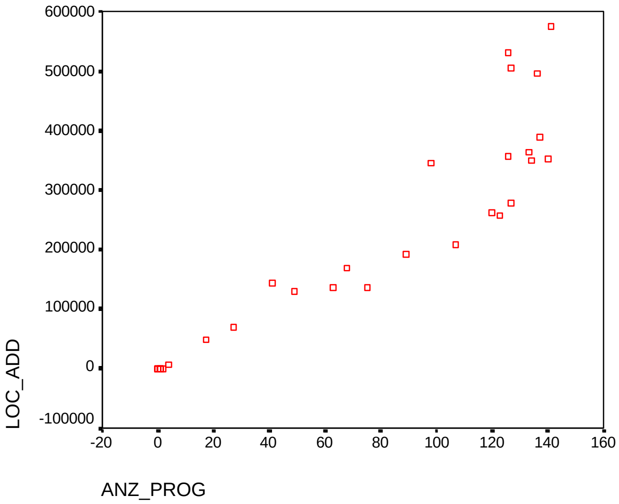

Fig. 30. Total of LOC added against active programmers in each month.

For each programmer, the cumulated LOC difference in each month was analysed to see if any patterns in the contribution to the project exist (see Fig. 31 – 38). As can be seen, a clear pattern does not emerge for the programmers examined.

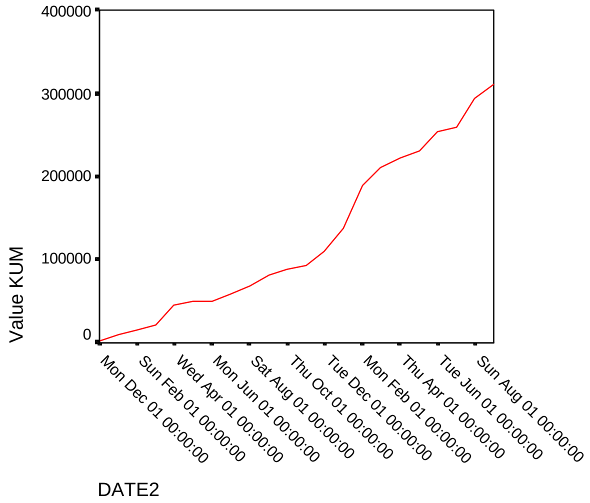

Fig. 31. LOC difference in each month for programmer 1 (ordered by total of LOC added) cumulated.

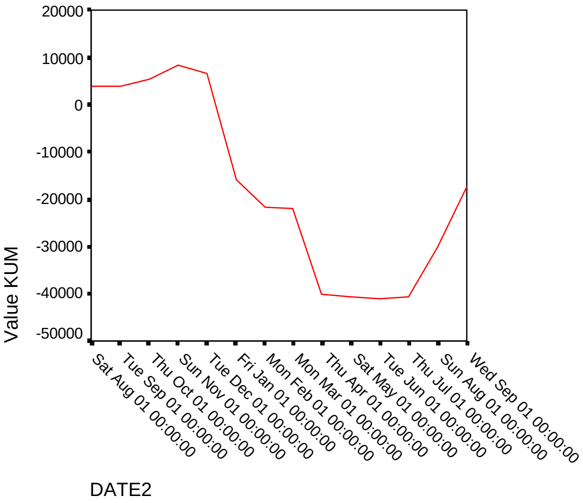

Fig. 33. LOC difference in each month for programmer 3 (ordered by total of LOC added) cumulated.

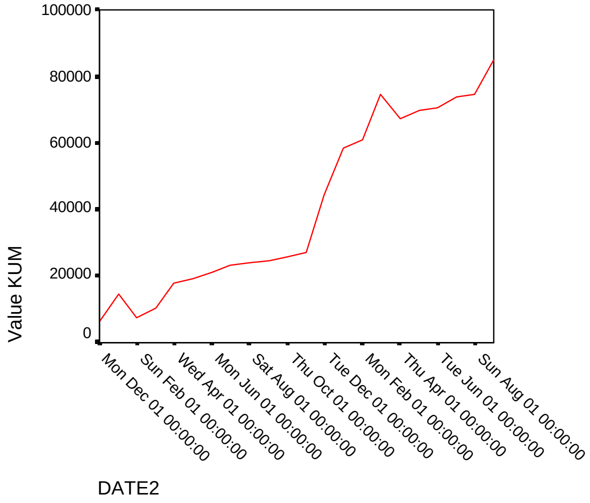

Fig. 32. LOC difference in each month for programmer 2 (ordered by total of LOC added) cumulated.

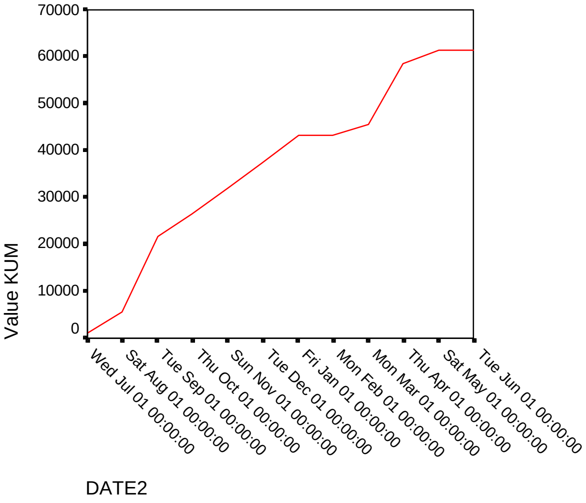

Fig. 34. LOC difference in each month for programmer 4 (ordered by total of LOC added) cumulated.

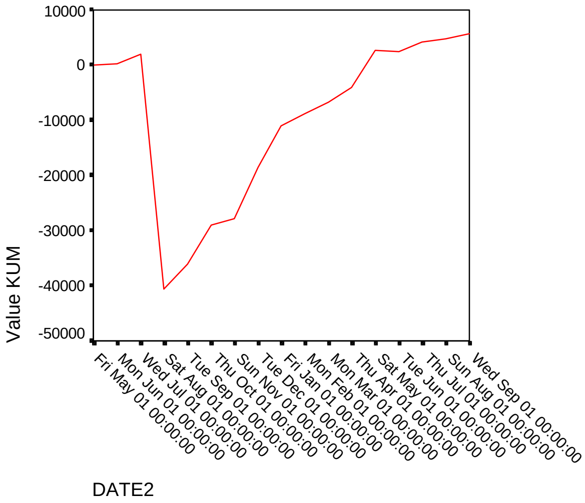

Fig. 35. LOC difference in each month for programmer 6 (ordered by total of LOC added) cumulated.

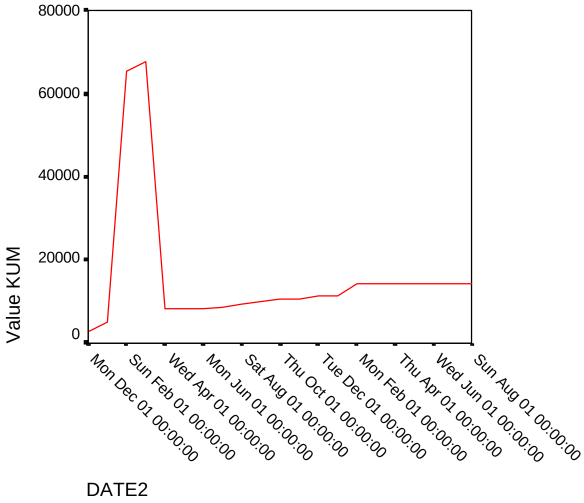

Fig. 36. LOC difference in each month for programmer 14 (ordered by total of LOC added) cumulated.

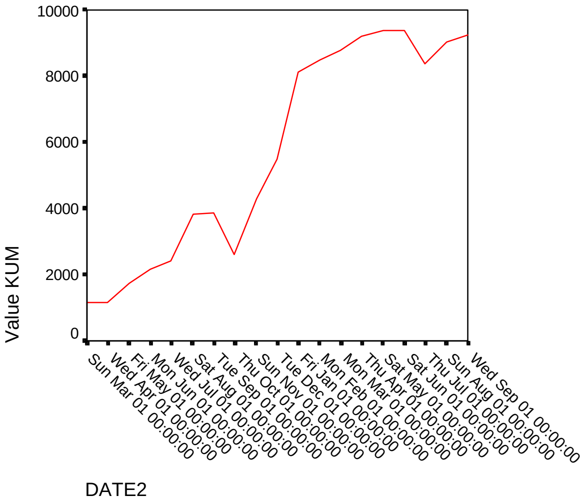

Fig. 37. LOC difference in each month for programmer 43 (ordered by total of LOC added) cumulated.

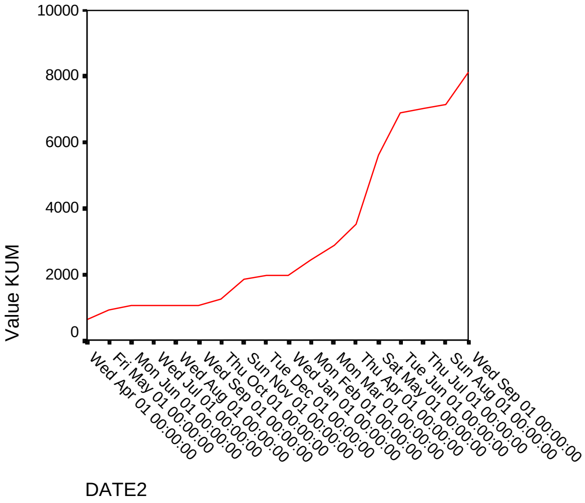

Fig. 38. LOC difference in each month for programmer 85 (ordered by total of LOC added) cumulated.

The number of new postings made each month was also explored to gain an understanding for the development of activity on the discussion lists over time. As can be seen the activity was strongest in the year 1998 which coincides with the build-up in active programmers (see Fig. 39). A possible explanation would be that more communication is necessary to accomodate for new programmers joining the project [5], [1]. After they have joined, less interaction is necessary between them. Therefore the correlation between the total of postings for each month and the difference in active programmers was also explored which yielded a result of 0.366 (see Fig. 40).

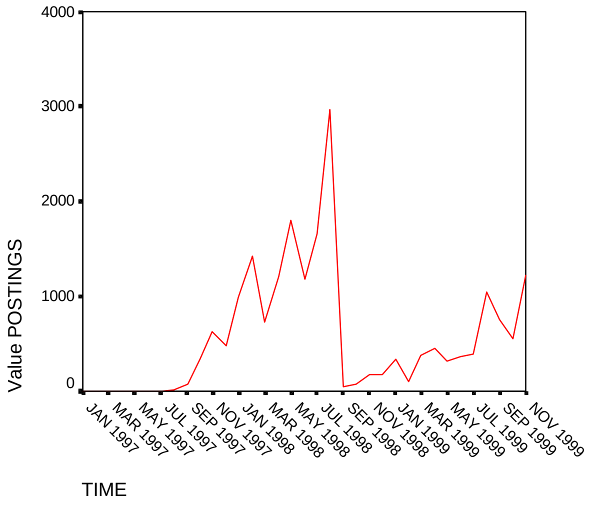

Fig. 39. Total of postings made in each month.

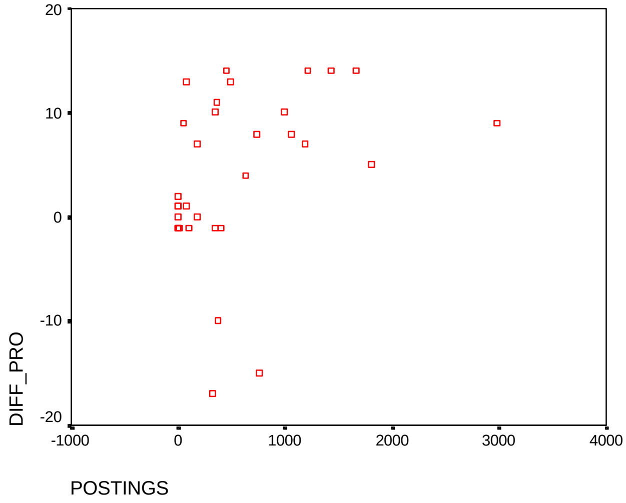

Fig. 40. Total of postings made in each month against difference in active programmers.

The correlation of the number of postings as a measure for activity on the discussion lists with the total of LOC added in each month as a measure of programming effort was also explored (see Fig. 41).

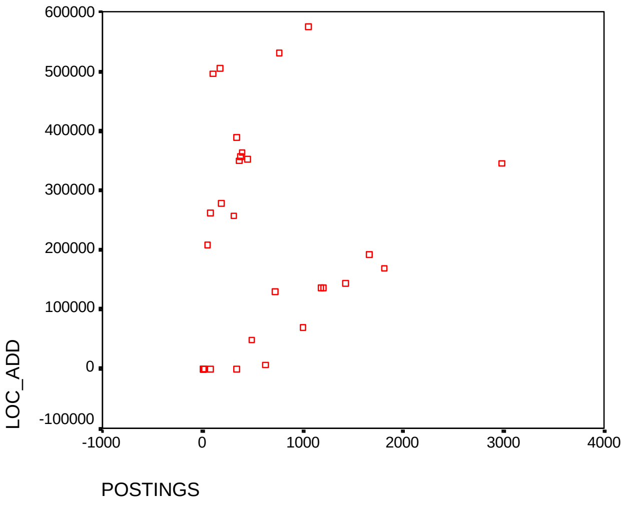

Fig. 41. Total of postings made against total of LOC added in each month.

A correlation of 0.227 was found, not supporting a relationship between activity on the discussion lists and programming effort expanded for the GNOME project overall.

F. Effort Estimation

In this section, a first approach to estimating the effort for the GNOME project is detailed based on [15] and [20]. According to this approach, a development project is modeled as a series of problem-solving efforts by the manpower involved to reach a set of objectives constituting technological progress. The number of problems is assumed to be unknown but finite. Each solving of a problem removes one element from the list of unsolved problems. The occurrence of such an event is random and independent, it is assummed to follow a Poisson distribution. The number of people usefully employed at any given time is assumed to be approximately proportional to the number of problems ready for solution at that time. Therefore, the manpower usefully employed towards the end of a project becomes smaller as the problem space is exhausted. The learning rate of the team is modeled as a linear function of time

$$p(t) = 2\,a\,t$$

which governs the application of effort. Therefore the cumulative manpower effort *C(t)* in person-years is null at the start of project and grows monotonically towards the total effort *K*. The rate of variation of the cumulative manpower involved, *dC/dt* represents the number of persons involved in the development at any time, *m(t)*, and is given by:

$$dC(t)/dt = p(t)\,[K - C(t)]$$

Using the learning rate defined above, the cumulative effort at any given time t becomes

$$C(t) = K\,[1 - \exp(-\,a\,t^2)],$$

and the manning of the project can now be calculated by differentiating the cumulative cost function relative to the time:

$$m(t) = 2\, K\, a\, t \exp(-\, a\, t^2).$$

This function represents a Rayleigh-type curve governed by the parameter *a* which plays an important role in the determination of the peak manpower m_0. By deriving the manpower function relative to the time and finding the zero value, the relationship between time of peak manning t_d and a can be found:

$$t_d^{\,2} = 1 / (2\, a)$$

Furthermore, the value of the peak manning m_0 can be obtained by substituting the value of *a* in the manpower function:

$$m_0\;\; t_d \sqrt{e}\;\; = K$$

Using this relationship, the total manpower required can be determined once peak manning has been reached.

As the manpower distribution for the GNOME project has been retrieved from the data (see Fig. 29) and seems to follow a Rayleigh-type curve, this information can be used for estimating the total effort. The peak manning seems to have been reached between November 1998 and September 1999. Therefore the time elapsed between the beginning of the project (in January 1997) and the peak manning, t_d, is set to 2.25 years, taking the middle of this range. The peak manning m_0 is set to 131.8 persons, the mean of the manning in these months. The next step necessary is to convert the peak manning to full-time employees, as this type is assumed in the model used. For this conversion, some value for the time actually invested in the project is necessary. As has been shown, the study of [10] shows at several points similar characteristics of the programmers questioned to the data retrieved from the GNOME project. Therefore it is possible to use the resulting value of 13.9 hours per week spent on the project. This results in a conversion to a peak manning of 45.8 persons. Using these values in the model of [15] and [20], a total effort *K* of 169.9 person-years is obtained. Again using the figure for time spent on the project from [10], to this date 79.5 person-years have already been invested. A further characteristic of the curve proposed is the manpower build-up which is related to the nature of the software being developed (also called the difficulty gradient) and empirically seems to remain constant around the values of 8 for entirely new software with many interactions and interfaces with other systems, 15 for new stand-alone systems and 27 for software rebuilt from existing software. The manpower build-up for the GNOME project obtained from the previous results is 14.9, a value strikingly close to one of the values proposed. As this project in fact is stand-alone, this result seems reasonable. Several of the sub-projects, if these were to be considered separately, might in contrast have many interactions with other sub-projects and therefore result in a different manpower build-up closer to 8.

IV. Conclusion

The area of open source software development has become very important to the software industry. Therefore research into this field from the perspective of software engineering also gains importance, especially quantitative data on this collaborative form of development is needed [8]. This paper has presented a methodology that can be applied to several other open source software development projects. First results showed that insights into this kind of development can indeed be gained.

In open source development, more people are involved than in traditional organisational forms, but the data shows the existence of a relatively small “inner circle” of programmers responsible for most of the output. Those programmers are also more active participants in the discussions pertaining to the project, although all programmers show a higher than average activity in the mailing lists compared to other participants. There is no relation to be seen between the time between first and last activity for the project and the output produced. This seems another striking difference to traditional software development. There is only a small number of programmers working together on a file, indicating a high degree of division of labour.

It has been shown that the project under consideration has seen a steady increase in size as measured in LOC over the time inspected. There is no indication that the end of the life cycle has yet been reached. Analysis of the sub-projects has given support to the theory that some of these have already progressed to a later stage in their evolution while others lag behind. Therefore the growth of the GNOME project is in different time periods supported by different projects, i.e. the life cycle of the GNOME project is composed of several sub-cycles of projects whose starting points are shifted in time.

The number of active programmers has seen a staggering rise during a prolonged time period, but has for the last year been relatively stable. The reasons for this might be inherent in the problem worked on or may indicate deficiencies in the form of cooperation employed. Also more psychologically motivated factors might have contributed to this effect. It seems interesting that the highest amount of activity on the discussion groups has been seen during the time of the manpower build-up, hinting at a more pressing need for communication and coordination. The intuitive relationship between the number of active programmers and the output produced for the project was confirmed. The attraction of participants is therefore identified as one of the most important aspects of open source development projects.

A first attempt at estimating the effort for this open source project has been made. The results seemed to give realistic numbers for this project and therefore indicated that the theory of [15] and [20] is applicable to this type of software development, although some conversions are necessary. The results can of course only give some indication at the effort necessary, the time to completion depends mostly on programmers participation, as has been shown above, and is not controlled by any central management.

Further research is to be undertaken using the data collected to this point, and further sources of data still need to be explored. For example, content analysis of the postings retrieved can yield further important information concerning interactions between software developers [5], [22], the diffusion of information, different programming styles, the evolution of the software products and several other communication metrics [9]. The bug-tracking archive can supply data on software quality. Following this, the data from these different sources can be integrated. In addition, the source code for each file could be retrieved and analysed using measures like cyclomatic complexity [14]. These results could be correlated with e.g. LOC or number of checkins to gain insights into the relationships between size, complexity and maintenance effort [3]. The conceptual and concrete architecture of the software system can also be extracted [4].

In a next step, the methodology presented needs to be applied to other open source software development efforts to allow for comparison between projects and the discovery of common features. This profile can then be compared to data from commercial software development projects. To ensure the validity of this comparison, the necessary metrics of course need to be available for several of these traditional software projects. As almost certainly some sort of versioning control system will also be used in this organisational form of software development, the same methodology can again be applied to some extent, of course given the consent of the controlling organization.

References

[1] T. Abdel-Hamid and S.E. Madnick, *Software Project Dynamics: An Integrated Approach*, Englewood Cliffs, New Jersey: Prentice-Hall, 1991.

[2] D. Atkins, T. Ball, T. Graves and A. Mockus, “Using Version Control Data to Evaluate the Impact of Software Tools,” *Proceedings of the 21st International Conference on Software Engineering (ICSE 1999)*, pp. 324-333, 1999.

[3] R.D. Banker, S.M. Datar, C.F. Kemerer and D. Zweig, “Software Complexity and Maintenance Costs,” *Communications of the ACM*, vol. 36, pp. 81-94, November 1993.

[4] I.T. Bowman, R.C. Holt and N.V. Brewster, “Linux as a Case Study: Ist Extracted Software Architecture,” *Proceedings of the 21st International Conference on Software Engineering (ICSE 1999)*, pp. 555-563, 1999.

[5] F.P. Brooks jr., *The Mythical Man-Month: Essays on Software Engineering*, Anniversary ed., Reading, Massachusetts: Addison-Wesley, 1995.

[6] C.B. Brown, “Linux and decentralized development,” *first monday*, vol. 3, March 1998.

[7] J.E. Cook, L.G. Votta and A.L. Wolf, “Cost-effective analysis of in-place software processes,” *IEEE Transactions on Software Engineering*, vol. 24, pp. 650-663, August 1998.

[8] B.J. Dempsey, D. Weiss, P. Jones, and J. Greenberg, “A quantitative profile of a community of open source Linux developers,” *Technical Report TR-1999-05*, School of Information and Library Science, University of North Carolina at Chapel Hill, October 1999.

[9] A.H. Dutoit and B. Bruegge, “Communication metrics for software development,” *IEEE Transactions on Software Engineering*, vol. 24, pp. 615-628, August 1998.

[10] S. Hermann, G. Hertel and S. Niedner, “Linux Study Homepage,” avaible online: http://www.psychologie.uni-kiel.de/linux-study/, 2000.

[11] W.S. Humphrey, *A Discipline for Software Engineering*, Reading, Massachusetts: Addison-Wesley, 1995.

[12] C.K. Kemerer and S. Slaughter, “An Empirical Approach to Studying Software Evolution,” *IEEE Transactions on Software Engineering*, vol. 25, pp. 493-509, July/August 1999.

[13] H.F. Li and W.K. Cheung, “An Empirical Study of Software Metrics,” *IEEE Transactions on Software Engineering*, vol. 13, pp. 697-708, June 1987.

[14] T.J. McCabe, “A complexity measure,” *IEEE Transactions on Software Engineering*, vol. 2, pp. 308-320, December 1976.

[15] P.V. Norden, “On the anatomy of development projects,” *IRE Transactions on Engineering Management*, vol. 7, pp. 34-42, March 1960.

[16] R.E. Park, “Software size measurement: A framework for counting source statements,” *Technical Report CMU/SEI-92-TR-20*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 1992.

[17] F.N. Parr, “An alternative to the rayleigh curve model for software development effort,” *IEEE Transactions on Software Engineering*, vol. 6, pp. 291-296, May 1980.

[18] B. Perens, “The Open Source Definition,” C. DiBona et al., eds., *Open Sources: Voices from the Open Source Revolution*, Cambridge: O’Reilly & Associates, 1999.

[19] J.M. Perpich, D.E. Perry, A.A. Porter, L.G. Votta and M.W. Wade, “Anywhere, Anytime Code Inspections: Using the Web to Remove Inspection Bottlenecks in Large-Scale Software Development,” *Proceedings of the 19th International Conference on Software Engineering (ICSE 1997)*, pp. 14-21, 1997.

[20] L.H. Putnam, “A general empirical solution to the macro software sizing and estimating problem,” *IEEE Transactions on Software Engineering*, vol. 4, pp. 345-361, July 1978.

[21] E.S. Raymond, *The Cathedral and the Bazaar*, Cambridge: O’Reilly & Associates, 1999.

[22] C.B. Seaman and V.R. Basili, “An Empirical Study of Communication in Code Inspection,” *Proceedings of the 19th International Conference on Software Engineering (ICSE 1997)*, pp. 96-106, 1997.

[23] P. Vixie, “Software Engineering,” C. DiBona et al., eds., *Open Sources: Voices from the Open Source Revolution*, Cambridge: O’Reilly & Associates, 1999.