Discussion of a Large-Scale Open Source Data Collection Methodology

Michael Hahsler and Stefan Koch
Department of Information Business, Vienna University of Economics and BA
{michael.hahsler|stefan.koch}@wu-wien.ac.at

Abstract

This paper discusses in detail a possible methodology for collecting repository data on a large number of open source software projects from a single project hosting and community site. The process of data retrieval is described along with the possible metrics that can be computed and which can be used for further analyses. Example research areas to be addressed with the available data and first results are given. Then, both advantages and disadvantages of the proposed methodology are discussed together with implications for future approaches.

1. Introduction

Open source software has been a hotly debated issue in the last years, especially following the success of several well-known projects like Linux or the Apache Web Server. While open source software entails several interesting questions, including legal ones, one particular interest lies in the associated development model. The main ideas of this development model are described in the seminal work of Raymond [29], 'The Cathedral and the Bazaar', in which he contrasts the traditional type of software development of a few people planning a cathedral in splendid isolation with the new collaborative bazaar form of open source software development. In this collaborative form of software development, a large number of developer-turned users come together without monetary compensation to 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.

In order to allow for this to happen and to minimize duplicated work, a current version of the source code of the software needs to be accessible. To this end, frequent releases together with software licenses that grant the necessary rights to the users, like free redistribution, inclusion of the source code, the possibility for modifications and derived works are necessary. The Open Source Definition lists a number of requirements for specific licenses, with the most prominent example, which is even more stringent, being the GNU General Public License (GPL), developed by the GNU project and advocated by the Free Software Foundation.

Increasingly, empirical studies of open source software development and projects have been performed in the last time. This trend is very encouraging, as it might serve to lead any discussion of this new development paradigm away from purely ideological debates onto a higher level of argumentation. Several differing approaches to collecting quantitative empirical data on open source projects are employed. While the literature yields several in-depth studies of a small number of projects [10], mostly large, well-known and successful ones, large-scale quantitative investigations going into software development issues are scarce. Current quantitative analyses use information provided by version-control-systems [24,25,18], the meta-information included in Linux Software Map entries [8], or data retrieved directly from the source code itself [11], but the number of projects studied remain small.

On the other hand, project hosting sites like Sourceforge.net have been discovered as a source of data. SourceForge.net is owned by Open Source Development Network, Inc. which is a wholly owned subsidiary of VA Software Corporation. The mission of SourceForge.net is 'to enrich the open source community by providing a centralized place for open source developers to control and manage open source software development'. To fulfill this mission goal, a variety of services is offered to hosted projects, including tools for managing support, mailing lists and discussion forums, web server space, shell services and compile farm, and source code control. Thus, this site aims at enabling virtual communities to form around projects, and, by easing cross-participation between projects, creating a single community out of these. From this source, mostly the statistics published by Sourceforge.net itself are currently used. For example, Crowston and Scozzi [7] used the available data for validating a theory for competency rallying, which suggests factors important for the success of a project. Hunt and Johnson [17] have analyzed the number of downloads of projects occurring. Krishnamurthy [20] used the available data of the 100 most active mature projects.

In this paper we propose a methodology that automates retrieval of public data [6] from a project hosting site spanning a large number of open source projects both large and small, using information stored by the available software development and communication tools. Studying software systems and development processes using data from these repositories offers several advantages [6]: This approach is very cost-effective, as no additional instrumentation is necessary, and it does not influence the software process under consideration. In addition, longitudinal data is available, allowing for analyses considering the whole project history.

2. Methodology

2.1. Data retrieval

For applying the proposed methodology, SourceForge.net was chosen as the site to be considered. Especially the source code control system offered, in the form of CVS (Concurrent Versions System), a free system which is being used extensively in the free software community [9], was used as the main source of information. Several works have already demonstrated that important information about software development can be retrieved from repositories storing information from such systems [1,24,25,18]. For example, data concerning the participants' contributions to projects, their cooperation, and the progression of projects in size and participants over time could be analyzed.

To gather additional and more detailed information, data from the web pages and CVS servers of the projects hosted was retrieved. The retrieval process is depicted in Figure 1.

As the amount of data to be retrieved was estimated to be very high, a relational database was employed for storage, and later analysis. The data model was based on Koch and Schneider [18].

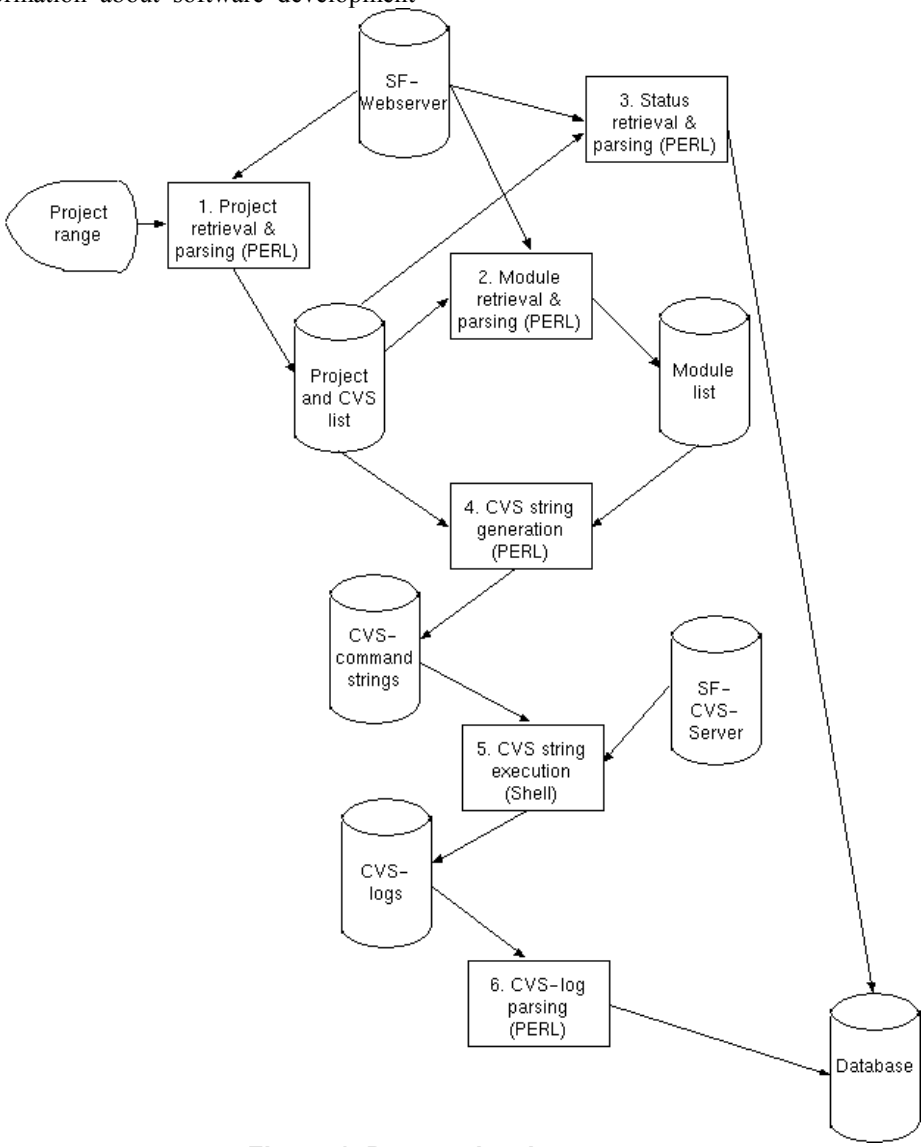

Figure 1. Data retrieval process

The first step was to consult the SourceForge.net homepage that displays the number of currently hosted projects (at the relevant date 23,000). All possible project numbers starting by one and up to this number were selected. The first step was to get a list of projects which are both still hosted and have the CVS service enabled. This was done by querying for each project (as identified by its number) its CVS information web page

hosted at SourceForge.net, containing information regarding CVS server name and password, if enabled. This resulted in one HTML page retrieved per project which was then parsed for the necessary information. Both tasks were performed using Perl scripts (step 1), resulting in 21,355 candidate projects with enabled CVS service. The project titles, CVS server names and passwords were extracted and stored. As SourceForge.net also has a development status indicator assigned to each project, this information was retrieved for the projects using again Perl scripts for downloading the relevant web pages (the summary page for each project) and parsing them (step 3). The resulting status was stored in the database for each project to be later used for analysis.

As the CVS interface can only handle statements concerning the modules of which a project is composed, the names of the modules of each project were also necessary. In addition, this would yield the information which projects actively use the CVS service. Therefore, the web page for browsing the CVS repository was retrieved for each project in the list and parsed (step 2). This showed that only 8,791 projects were actively using the CVS service and thus were usable for further analysis. Using this information together with the CVS server name information, a Perl program was used to generate a shell script for querying the CVS servers and retrieving the necessary data (step 4). This was done by first checking out the source code for each module and then issuing the "log" command for it (which is only possible for checked out items).

Executing this large shell script (about 110,000 statements) resulted both in the downloaded source code and an output log file for each project (step 5). The CVS log command produces the whole history of all files in the module. This shows the work of the programmers on the project by submitting ("checking in", "committing") files. 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 CVS by a programmer. The CVS-repository then records this commit with the changes in the lines-of-code (LOC) and further information. This information, now contained in the log files, was then extracted by yet another Perl script and stored in the database (step 6). The number of lines-of-code checked in with the first commit for each file („initial") was computed from the source code itself, as it is not recorded automatically in CVS.

Subsequent analyses were performed using queries to the database and processing with R, a free statistics package. Overall, information was retrieved for 8,621 projects. The download took more than one month, and the downloaded files use about 33 GB of disk space. All downloads and queries to the SourceForge.net servers were supplied with ample sleeping periods so as to not delay services for other users due to overloads.

2.2. Metrics

The first metric used is the number of lines-of-code (LOC) added to a file. The definition of this often disputed metric LOC [16,27] is taken from the CVS-repository and therefore includes all types of lines-of-code, e.g. also commentaries [9]. In addition, any LOC changed is counted as one line-of-code added and one line-of-code deleted. The LOC deleted are defined analogous. The difference between the LOC added and deleted therefore gives the change in size of a software artifact under consideration in the corresponding time period. These changes can be cumulated to give the size at any moment.

The metric of commit refers to the submission of a single file by a single programmer.

The total time spent on the project can be defined for every programmer as the difference between the date of his first and last commit, but as this therefore includes all time elapsed, this measure can only give an upper bound for actual time spent working. Therefore we will adopt the metric of a programmer as being active in a given period of time if he performed at least one commit during this interval, which has been shown to be better suited [18].

The next metric was directly taken from the SourceForge.net repository, which has a development indicator assigned to each project. This indicator has seven possible values, reaching from planning, pre-alpha, alpha, beta to production/stable and mature, and to inactive. This indicator is assigned by the project administrator, and need therefore not necessarily be a correct description of the current status.

Furthermore, although quite obvious, the participation of a given programmer in a given project, easily extracted from the data, gives additional information for repository level analyses.

3. Possible analyses

Several different analyses are possible using the data retrieved. Figure 2 gives an overview of the issues that can be addressed. We describe these issues in detail in this section, together with some first results and related research from literature. Three major levels of analysis are distinguished, starting from participant level, going up to the level of a project and its characteristics and lastly the hosting and community site overall.

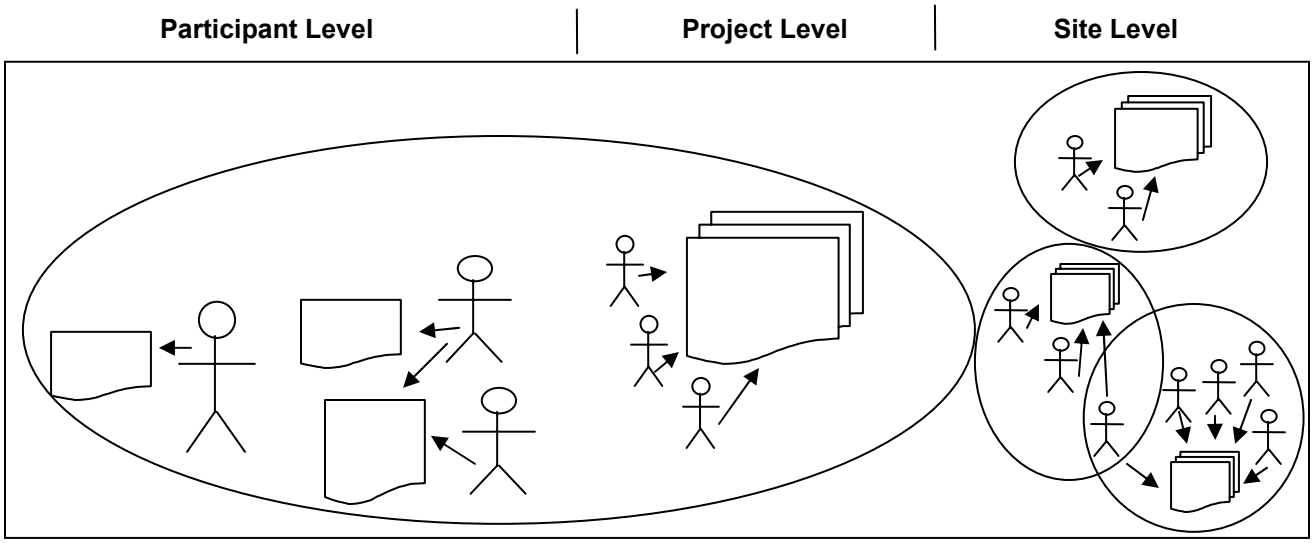

Figure 2. Overview of possible analyses

3.1. Site level

The first important characteristic of the project hosting site overall that can be checked using the available data is the distribution of both the assets available (i.e., the programmers) and the resulting outcome (i.e., commits, size and project status). Afterwards, possible relationships between these variables can be explored.

Analyzing the 100 most active mature projects on Sourceforge.net, Krishnamurthy [20] showed that most of the projects had only a small number of participants (median of 4). Only 19 per cent had more than 10 developers and 22 per cent only had one developer. Hunt and Johnson [17] analyzed the number of downloads of projects. They show that the distribution of projects according to this number is also heavily skewed and follows a power law (or Pareto or Zipf) distribution. This form of distribution has been recognized in a number of fields including distribution of incomes, word usage and web site popularities. A power-law implies that only few instances are extremely common, whereas most instances are extremely rare. While there are several explanations for the occurrence of this sort of distribution, in the case of open source software development some communities' increased success, attractiveness and popularity leads to more programmers participating, which in turn might make the community even more successful and thus constitute a positive feedback loop for these communities.

Regarding the output of the projects, a similar situation could be ascertained analyzing the respective metrics like number of commits or size in lines-of-code. Figure 2 shows a histogram of project sizes, clearly indicating a very skewed distribution within the site.

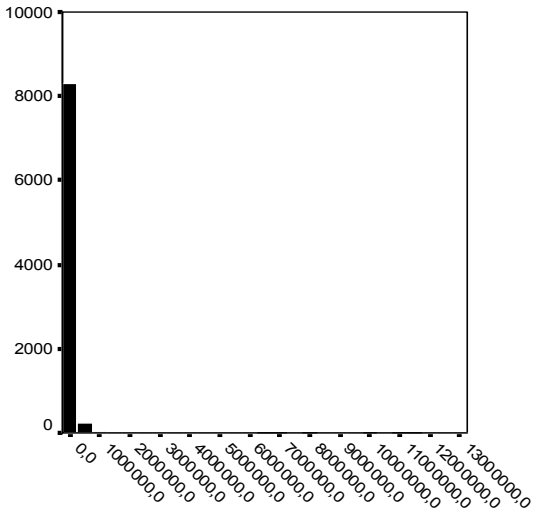

Figure 3. Histogram of project size

More possible analyses on the repository level pertain to the collaboration of programmers on several projects. This includes the simple number of projects

that programmers work on, e.g., Ghosh and Prakash [11] have found that most of the programmers, exactly 11,500 out of the 12,700 analyzed, have only worked on one or two projects. In the ecology considered here, an even greater amount of 94.1 per cent worked on less than 3 projects. The collocation of projects on a single hosting site does therefore not lead to increased participation in other projects on the same site. Of even more interest seems the author clustering as proposed by Ghosh [12], or building a graph consisting of projects as vertices and edges representing common participants [23]. This would allow for identification of "linchpin" developers, sub-groups or similar phenomena using social network analysis [22].

3.2. Project level

The prior results presented above for distribution of inputs, i.e., programmers, and outputs lead naturally to the assumption that both type of measures of projects are correlated, i.e., that projects with a small number of programmers only achieve small numbers of commits and lines-of-code. As all data is available for these analyses, simple correlation coefficients can be computed.

Crowston and Scozzi [7] have found in their analysis of the Sourceforge.net published data, that although the number of programmers was associated with higher levels of activity, it also coincided with less advanced states of development. Projects having well-known developers also show higher levels of activity, but in addition more advanced states of development.

Another interesting aspect to explore is the evolution of open source projects. The study of software evolution was pioneered by the work of Lehman and Belady [2] on the releases of the OS/360 operating system, and led to many other works (e.g., [21]), in which the laws of software evolution were formulated, expanded and revised. These laws entail a continual need for adaptation of a system, followed by increased complexity and therefore, by applying constant incremental effort, a decline in the average incremental growth. Turski has modeled this as an inverse square growth rate [32]. The first study on software evolution in open source systems was performed by Godfrey and Tu [13], who have analyzed the Linux operating system kernel and found a super-linear growth rate, contradicting the prior theory of software evolution. They modeled the growth of lines-of-code best using a quadratic model with number of days since version 1.0 as independent variable, but found that most of the kernel size stemmed from the device drivers which are relatively independent of each other. Nevertheless, if the evolution of open source software systems would prove to be distinctly different from those of commercial systems, it would give an indication of major differences in development modes and their results. The SourceForge.net project repository offers a multitude of projects both large and small to validate the theory of software evolution. To do this, a model taking size in lines-of-code as a function of days has to be computed. For the type of model, naturally several possibilities exist, including a simple linear and a quadratic model (and of course models of higher order). Then the quality of these models can be compared. The most interesting fact to explore is whether or not the growth rate is decreasing over time according to the laws of software evolution. This can be checked by analyzing the second derivate of the quadratic model (or directly using the coefficient of the quadratic term). A negative sign would indicate decreasing growth rate in accordance to the laws of software evolution, but would form a contradiction to the findings of Godfrey and Tu [13] for the Linux operating system. In addition, it can be explored whether there are any characteristics of projects that lead to super-linear growth. To explain presence or absence of super-linear growth, the projects can be divided in two groups according to their growth behavior (super-linear or not) and checked for differences, e.g., in size, number of participants, or similar measures. Preliminary analysis showed that about 39 per cent of the projects exhibit super-linear growth, with projects in this group in general being larger with more participants.

Additional analysis might include verifying the application of modern programming practices. For example, Hahsler [14] has shown that text analysis of commit comments can be used to uncover whether patterns are used in a project, and, using further inspection of other project variables, which project characteristics lead to increased adoption.

3.3. Participant level

Most prior studies have found a distinctly skewed distribution of effort between the participants in open source projects. For example, Mockus et al. [25] have shown that the top 15 of nearly 400 programmers in the Apache project added 88 per cent of the total lines-of-code. In the GNOME project, the top 15 out of 301 programmers were only responsible for 48 percent, while the top 52 persons were necessary to reach 80 per cent [18], with clustering hinting at the existence of a still smaller group of 11 programmers within this larger group. A similar distribution for the lines-of-code contributed to the project was found in a community of Linux kernel developers by Hertel et al. [15]. Also the results of the Orbiten Free Software survey [11] are similar, the developers up to the first decile were responsible for 72 per cent, the second for 9 per cent of the total code. Figure 4 shows for two projects both the lines of perfect equality and below them the respective Lorenz curve, based on number of commits per programmer. The Gini coefficient, a measure of inequality often used in economics, is defined as the area in between the two curves. In most projects, a very

unequal distribution between the participants can be seen (more akin to the project on the left hand side).

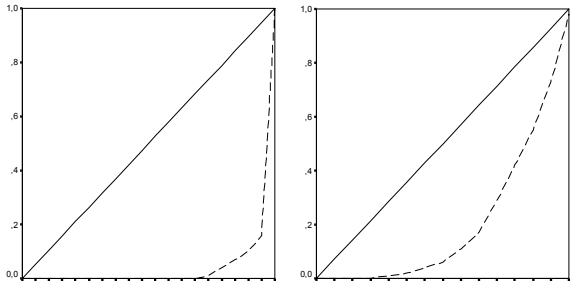

Figure 4. Lorenz curves for distribution of commits within two projects

In addition to the distribution between programmers' commits, also the relationships between other variables of programmers' contributions can be analyzed, which should yield high positive correlations in the case of commits and LOC, but might deviate in the case of time or total number of different projects worked on.

In addition, the number of LOC added per single checkin can be computed to uncover potential differences in working style. Like by Koch and Schneider [18], the number of programmers working together on single files can also be checked to uncover patterns of cooperation. Analyzing the contributions of programmers over time (using several fixed time intervals), it could be checked whether differences in total contributions are due to different intensities of contribution or longer time on the project.

3.4. Productivity

There are several factors in a project that might influence the productivity within the project. As a first idea, the distribution of effort in the development team can be explored. The question to be answered is whether a observed distribution (e.g., a very skewed distribution) is a good way of organizing the work, i.e., if this form of distribution leads to good performance. Therefore, the situation within projects needs to be explored, defining some measure of inequality like, for example, the Gini coefficient described above [31].

The next possible influence on productivity in a project is the number of active programmers. Following the reasoning of Brooks, an increased number of people working together will decrease productivity per person due to exponentially increasing communication costs [5]. Therefore, the number of programmers and the achieved progress in each project over given time intervals need to be analyzed. To uncover any effects an increased number of people working together has on productivity, the relationship to the mean number of commits and lines-of-code added per programmer in a period can be explored.

3.5. Effort

The main indicator for how the open source software development model compares with the traditional, commercial models is the effort expended. As for open source software development not even project leaders know how much time is expended by their participants, as no time sheets or similar mechanisms are employed, this effort for the software development needs to be estimated.

Currently, several estimation models are available, all proposed and calibrated for commercial software development. Several of these models contain restrictions which are not fully compatible with open source software development. Wheeler [33] used the basic model of COCOMO [3] in organic development mode on the Red Hat 7.1 distribution of GNU/Linux, and arrived at an estimation of nearly 8,000 person-years of effort representing a value of over one billion dollar to produce the over 30 million lines-of-code. COCOMO uses simply lines-of-code as input and necessitates choosing one out of three available modes of software development. Naturally, this model can be applied using data collected from SourceForge.net, resulting in a mean effort per project of 18.56 person-years (with median 2.02 person-years), summing up to a total effort for all projects of 160,020 person-years. COCOMO II [4] seems to offer several advantages over its predecessor, as it allows for both increasing and decreasing economies of scale, a prototype-oriented software process and flexibility in the requirements. As it takes about the same input, application is again possible.

Another approach, the Rayleigh-Norden model [26], starts from the main idea that any development project is composed of a set of problems which need to be solved by the manpower employed. The application of manpower is governed by a learning rate linear in time, 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 function increases until the point of peak manning, which as Putnam [28] has shown is close to the deployment of the software, and then decreases due to the exhaustion of the problem space. The manpower function therefore represents a Rayleigh-type curve governed by a parameter which plays an important role in the determination of the peak manpower. If the point of peak manning has been reached, both the parameter for the Rayleigh-curve and the total manpower to be expended can be computed. Koch and Schneider [18] and Koch [19] have demonstrated the use of this model for the GNOME project, in which the manpower function (the number of active developers) closely follows the model, and also reaches peak manning at the time of the first major release. This in-depth analysis including the consideration of release dates is impracticable for a

whole project ecology with more than 8,000 projects. Therefore, the point in time of peak manning and the respective number of active programmers need to be retrieved for each project. From this, the Rayleigh-curve can be constructed automatically, and the total manpower to be expended can be computed. The Rayleigh-curve was originally developed to compute manpower in person-years for commercial programmers with a 40 hours week. However, since we use the number of active open source programmers for calculation, the result is also in "open source programmer-years" (which use normally less than 40 hours per week). Using the average number of working hours of open source programmers, this measure can be converted [18,19] for comparison into full 40 hours weeks. For example, Hertel et al. [15] report that in the group of Linux kernel developers participating in their survey, about 18.4 hours per week are spent on open source development by each person. The results of the Norden-Rayleigh model are considerably lower than those achieved by COCOMO estimation, and give a mean effort per project of 0.69 person-years (median 0.19 person-years) with a total effort for all hosted projects of 5,965 person-years.

4. Discussion

Compared with other approaches currently discussed, the main advantages of the described methodology are that a large amount of projects can be used as input, without human intervention. Project identification and retrieval is performed automatically. The inclusion of other measures besides data from a source code control repository is possible, as demonstrated by including the status variable retrieved from the Sourceforge.net web pages.

Nevertheless, several problems were encountered using the proposed methodology. First, execution of the generated shell script for querying the different source code control repositories of the different projects was not automatically supervised. If the script terminated due to whatever reason (e.g., a crash of the executing machine) progress had to be checked in the generated log files and the rest of the script had to be restarted manually cutting down the script to those commands not yet executed. A better solution would be to read the relevant access and progress information from a database or a file and generate all necessary shell commands on the fly.

The quality of the data retrieved can not be absolutely ascertained. There might be several effects introducing a bias to some results, for example a high amount of checkins performed by participants for other people might lead to increased concentration and inequality indices. Some of these problems can be detected and alleviated, e.g., by inspecting the checkin comments.

Several analyses especially based on the data from source code repositories might be automated to a much higher degree as, for example, shown by the CVSAnalY tool [31]. This tool also includes matching of single files on certain types using, e.g., inspection of file extensions, or recognizing common filenames like readme. This might allow for identification of contributor groups that work on different parts of the project like documentation or translation [31].

Of special interest seems the recently proposed GlueTheos approach [30], a modular system automating the retrieval and analysis processes from several kinds of repositories including source code control system, mailing lists, etc. What seems to be missing is support for identification and retrieval of data on a large number of projects. Currently, setup needs to determine the starting points manually, like for instance the address of the relevant source code repository. Thus setup needs to be performed for each project individually, which limits its capabilities. Therefore we propose to extend the approach by including a first step before data retrieval, termed identification, to allow for automated identification of a large number of projects and their retrieval starting points.

5. Conclusions

In this paper we proposed a methodology for identifying a large number of open source software projects from a single project hosting and community site and automatically retrieving their data from several public sources provided by the site. We described which metrics can be derived from this data, and discussed what analyses are possible based on the metrics. We demonstrated the applicability of the methodology by giving some examples from SourceForge.net. This has shown that indeed a large number of research areas can be addressed using the methodology. Advantages of applying this approach include the cost-effectiveness of automated data retrieval, absence of influence on the software process studied, and, due to the availability of historical data, the possibility of longitudinal analyses. Therefore, several insights into the inner workings, coordination and evolution of open source software communities can be gained from publicly available data at low cost.

Both, advantages and disadvantages, of the proposed methodology have been discussed. Other recently proposed approaches offer distinct advantages in several areas, but seem to lack support for automated project identification. Future work should be done on combining the advantages of several of these approaches, maybe by using the flexible architecture provided by GlueTheos.

6. References

[1] Atkins, D., Ball, T., Graves, T. and Mockus, A., 'Using Version Control Data to Evaluate the Impact of Software Tools', in *Proceedings of the 21st International Conference on Software Engineering*, Los Angeles, 1999, pp. 324-333.

[2] Belady, L.A. and Lehman, M.M., 'A model of large program development', *IBM Systems Journal* 15(3): pp. 225-252, 1976.

[3] Boehm, B.W., *Software Engineering Economics*. Englewood Cliffs, New Jersey: Prentice-Hall, 1981.

[4] Boehm, B.W., Abts, C., Brown, A.W., Chulani, S., Clark, B.K., Horowitz, E., Madachy, R., Reifer, D.J. and Steece, B., *Software Cost Estimation with COCOMO II*. Upper Saddle River, New Jersey: Prentice Hall, 2000.

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

[6] Cook, J.E., Votta, L.G. and Wolf A.L., 'Cost-effective analysis of in-place software processes', *IEEE Transactions on Software Engineering* 24(8): pp. 650-663, 1998.

[7] Crowston, K. and Scozzi B., 'Open source software projects as virtual organizations: Competency rallying for software development', *IEE Proceedings - Software Engineering* 149(1): pp. 3-17, 2002.

[8] Dempsey, Bert J., Weiss, Debra, Jones, Paul, and Greenberg, Jane, 'Who is an open source software developer?', *Communications of the ACM* 45(2): pp. 67-72, 2002.

[9] Fogel, K., *Open Source Development with CVS*. Scottsdale, Arizona: CoriolisOpen Press, 1999.

[10] Gallivan, Michael J., 'Striking a balance between trust and control in a virtual organization: A content analysis of open source software case studies', *Information Systems Journal* 11(4): pp. 277-304, 2002.

[11] Ghosh, R. and Prakash, V.V., 'The Orbiten Free Software Survey', *First Monday* 5(7), 2000.

[12] Ghosh, R., 'Clustering and dependencies in free/open source software development: Methodology and tools', *First Monday* 8(4), 2003.

[13] Godfrey, M.W. and Tu, Q. 'Evolution in Open Source software: A case study', in *Proceedings of the International Conference on Software Maintenance (ICSM 2000)*, San Jose, California, 2000, pp. 131-142.

[14] Hahsler, Michael "A Quantitative Study of the Adoption of Design Patterns by Open Source Software Developers", in: Koch, Stefan (ed.) *Free/Open Source Software Development*, IGP, Hershey, PA, 2004.

[15] Hertel, Guido, Niedner, Sven, and Hermann, Stefanie, 'Motivation of software developers in open source projects: An internet-based survey of contributors to the Linux kernel', *Research Policy* 32(7): pp. 1159-1177, 2003.

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

[17] Hunt, Francis and Johnson, Paul, 'On the pareto distribution of sourceforge projects', in *Proceedings of the Open Source Software Development Workshop*, Newcastle, UK, 2002, pp. 122-129.

[18] Koch, Stefan and Schneider, Georg, 'Effort, cooperation and coordination in an open source software project: Gnome', *Information Systems Journal* 12(1): pp. 27-42, 2002.

[19] Koch, Stefan, 'Effort Estimation in Open Source Software Development: A Case Study', in *Proceedings of the 2003 IRMA International Conference*, Philadelphia, PA, 2003, pp. 859-861.

[20] Krishnamurthy, Sandeep, 'Cave or community? an empirical investigation of 100 mature open source projects', *First Monday* 7(6), 2002.

[21] Lehman, M.M. and Ramil, J.F., 'Rules and Tools for Software Evolution Planning and Management', *Annals of Software Engineering* 11: pp. 15-44, 2001.

[22] Luis López, Jesús M. González-Barahona and Gregorio Robles, "Applying Social Network Analysis to the Information in CVS Repositories", in *Proceedings of the Mining Software Repositories Workshop*. 26th International Conference on Software Engineering (Edinburgh, Scotland), 2004.

[23] Madey, G,. Freeh, V. and Tynan, R.,'The Open Source Software Development Phenomenon: An Analysis based on Social Network Theory', in *Proceedings of the Americas Conference on Information Systems*, Dallas, Texas, 2002, pp. 1806-1813.

[24] Mockus, A., Fielding, R. and Herbsleb, J., 'A Case Study of Open Source Software Development: The Apache Server', in *Proceedings of the 22nd International Conference on Software Engineering*, Limerick, Ireland, 2000, pp. 263-272.

[25] Mockus, A., Fielding, R. and Herbsleb, J., 'Two case studies of open source software development: Apache and Mozilla', *ACM Transactions on Software Engineering and Methodology* 11(3): pp. 309-346, 2002.

[26] Norden, P.V., 'On the anatomy of development projects', *IRE Transactions on Engineering Management* 7(1): pp. 34-42, 1960.

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

[28] Putnam, L.H., 'A general empirical solution to the macro software sizing and estimating problem', *IEEE Transactions on Software Engineering* 4(4): pp. 345-361, 1978.

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

[30] Gregorio Robles, Jesús M. González-Barahona and Rishab Aiyer Ghosh, "GlueTheos: Automating the Retrieval and Analysis of Data from Publicly Available Repositories", in *Proceedings of the Mining Software Repositories Workshop*. 26th International Conference on Software Engineering (Edinburgh, Scotland). 2004.

[31] Gregorio Robles, Stefan Koch and Jesús M. González-Barahona, "Remote analysis and measurement of libre software systems by means of the CVSAnalY tool", in *Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS '04)*. 26th International Conference on Software Engineering (Edinburgh, Scotland), 2004.

[32] Turski, W.M. 'Reference Model for Smooth Growth of Software Systems', *IEEE Transactions on Software Engineering* 22(8): pp. 599-600, 1996.

[33] Wheeler, David A. 'More Than a Gigabuck: Estimating GNU/Linux's Size - Version 1.07', http://www.dwheeler.com/sloc/redhat71-v1/redhat71sloc.html, accessed Aug. 4., 2003.