Effort Modeling and Programmer Participation in Open Source Software Projects

Stefan Koch
Institute for Information Business
Vienna University of Economics and Business Administration
Augasse 2-6, A-1090 Vienna, Austria
stefan.koch@wu-wien.ac.at

Abstract

This paper analyses and develops models for programmer participation and effort estimation in open source software projects. This has not yet been a centre of research, although any results would be of high importance for assessing the efficiency of this model and for various decision-makers. In this paper, a case study is used for hypotheses generation regarding manpower function and effort modeling, then a large data set retrieved from a project repository is used to test these hypotheses. The main results are that Norden-Rayleigh-based approaches need to be complemented to account for the addition of new features during the lifecycle to be usable in this context, and that programmer-participation based effort models show significantly less effort than those based on output metrics like lines-of-code.

1 Introduction

Open source software development (Feller & Fitzgerald, 2002; Raymond, 1999) has generated increasing interest in the last years, both from the business and academic world. As some projects in different application domains like Linux together with the suite of GNU utilities, GNOME, KDE, Apache, sendmail, bind, and several programming languages have achieved huge success in their respective markets, new business models have been developed and tested by businesses. Academic interest into this new form of collaborative software development has arisen from very different backgrounds including software engineering, sociology, management or psychology.

The main ideas of this development model are described in the seminal work of Raymond (Raymond, 1999), '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, 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, thus forming an example for 'egoless programming' as proposed by Weinberg already in 1971 (Weinberg, 1998). In order to allow for this to happen and to minimize duplicated work, the source code of the software needs to be accessible, and new versions need to be released often. To this end, 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 and some others have been developed. One model for such licenses is the Open Source Definition, which lists a number of requirements for specific licenses (Perens, 1999). The most prominent example which fulfils these criteria while still being even more stringent, is the GNU General Public Licence (GPL), developed by the GNU project and advocated by the Free Software Foundation (Stallman, 2002).

While the differences between the decentralized open source process and traditional software engineering practices have been debated (Bollinger et al., 1999; McConnell, 1999; Vixie, 1999), and also quantitative studies of development projects and communities have been undertaken (Dempsey et al., 2002; Dinh-Trong & Bieman, 2005; Ghosh & Prakash, 2000; Koch & Schneider, 2002; Koch, 2004; Krishnamurthy, 2002; Mockus et al., 2000, 2002), some points remain to be explored. One of the most important questions remaining is the effort for developing open source software, which is not known even to the leaders of the respective project, and thus the efficiency of this model. As software engineering has dealt with the problem of estimating the effort for a software project for decades and has produced a multitude of methods to this end, their use seems a natural answer. These include the well-known COCOMO (Boehm, 1981), which offers an algorithmic formula for estimating the effort based on a quantification of the lines-of-code. This model has lately been modified and updated with the publication of COCOMO II (Boehm et al., 2000). Other options for effort estimation include the software equation by Putnam (Putnam, 1978), also forming the base for the proprietary SLIM, approaches based on the function point metric (Albrecht & Gaffney, 1983), diverse machine-learning approaches or proprietary models like ESTIMACS. Both the COCOMO approaches and Putnam's work are based on a general formulation of a development project introduced by Norden (Norden, 1960), termed the Norden-Rayleigh model. This model is based on the derivation of a manpower function giving the number of people participating in a development project at a given time. Therefore the effort estimation leads to another important question, whether the participation in open source projects can be modeled and predicted using approaches created in the context of traditional software development, or whether new models have to be developed. The answer to these questions will, besides showing insights into open source software development, give important input to effort estimation and the applicability of different effort models in this context.

The work on effort estimation on open source software development project has to date been very limited (Koch & Schneider, 2002; Koch, 2004; Wheeler, 2005; Gonzalez-Barahona et al., 2004), and for the most part applied only very basic models like original COCOMO (Boehm, 1981) without further discussion or analysis.

The results offered by performing an effort estimation have several uses, with one of them, ex-post estimation, being in addition to the normal uses of effort estimation results in commercial development. While in commercial organizations normally data on the effort of finished projects exist, this is not the case in open source projects. Normally, not even the project leaders or members of the inner circle know this data. Therefore this effort having been expended for finished or at least far progressed projects has to be estimated. Naturally, some of the problems inherent in ex-ante effort estimation do not apply in this case, but some problems persevere. While in commercial contexts the main use of effort estimates is in planning and continuous management, ex-post estimation does not offer any immediate benefits. In an assessment of the open source development model, these results are nevertheless paramount. While there are many discussions and claims about this development model (Bollinger et al., 1999; McConnell, 1999; Vixie, 1999), the question of efficiency can not be answered without assessing the inputs, i.e. effort, going into the development. Any discussion of issues like quality (Dinh-Trong & Bieman, 2005; Koru & Tian, 2004; Stamelos et al., 2002; Zhao & Elbaum, 2000), or development speed seems incomplete without relation to costs. Therefore estimating the effort having been expended in open source projects is an important, if not the most important activity in comparing this development model to others, helping to decide whether this model should be pursued, abandoned or combined with traditional approaches into hybrid-models (Sharma et al., 2002).

While this ex-post estimation seemingly has great merits, more traditional applications in the line of planning and control exist in open source software projects as well, which also have stakeholders who could be interested in such results at early stages or during a project. These include the community itself, especially, dependent on the organizational form, the owner/maintainer, inner circle or committee (Fielding, 1999; Raymond, 1999), which need to monitor progress and plan for release dates, and programmers considering whether to join or to remain in a project. Maybe these persons have only a limited time slot available, but want to remain in the project until a certain progress has been achieved. Further possible interested parties are current or prospective users, who need the functionality at a given date or with a given maturity level, especially corporations which are intending to pursue a business models based on this software, need it for their operations or plan to incorporate it in their products or services.

The work described in this paper tries to evaluate and establish effort modeling for open source software development projects. To this end, a two-step approach will be used in the analysis: First, a case study of a large and successful project, GNOME, will be analyzed in-depth to generate research hypotheses, then similar data from SourceForge.net, an ecology of both small and large, successful and failed projects will be used to test these hyptheses on a larger data base. Prior to this analysis, the method used for retrieval of data on the projects, and the data sets together with general results will be described in the following section.

2 Method and data set

2.1 Software repository mining

Software development repositories contain a plethora of information on the underlying software and the associated development processes (Cook et al., 1998; Atkins et al., 1999). Studying software systems and development processes using these sources of data offers several advantages (Cook et al., 1998): 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 (Kemerer & Slaughter, 1999).

Depending on the tools used in a project, possible repositories available for analysis include source code versioning systems, bug reporting systems, or mailing lists. Many of these have already been used as information sources for closed source software development projects. For example, Cook et al. (1998) present a case study to illustrate their proposed methodology of analyzing in-place software processes. They describe an update process for large telecommunications software, analyzing several instances of this process using event data from customer request database, source code control, modification request tracking database and inspection information database. Atkins et al. (1999) use data from a version control system in order to quantify the impact of a software tool, a version-sensitive editor, on developer effort. Kemerer and Slaughter (1999) employed data from change logs in their study on software evolution. Dutoit and Bruegge (1998) retrieved a set of communication metrics from electronic bulletin boards in addition to product metrics to analyze differences in software processes due to changes in the applied development methodology.

In open source software development projects, repositories in several forms are also in use, in fact form the most important communication and coordination channels, as the participants in any project are not collocated. Therefore only a small amount of information can not be captured by repository analyses because it is transmitted inter-personally. As a side effect, the repositories in use must be available openly and publicly, in order to enable as many persons as possible to access them and to participate in the project. In addition, also the general stance towards openness and free information within this community enhances this trend. As both necessarily due to lack of resources and because of ideological reasons only open source software itself is adopted for operating the necessary repositories, the variety over several projects is not enormous, thus allowing for construction of automated retrieval and analysis tools which can be re-used for several projects. For example, a tool for automated retrieval and analysis of data from a version-control system, CVS, termed CVSAnalY has been developed (Robles et al., 2004b), as has been the GlueTheos approach (Robles et al., 2004a), a modular system automating the retrieval and analysis processes from several kinds of repositories including source code control or mailing list.

Given this situation, repository data have already been used in research on open source software development. This includes in-depth analyses of small numbers of successful projects (Gallivan, 2001) like Apache and Mozilla (Mockus et al., 2000, 2002), GNOME (Koch & Schneider, 2002), FreeBSD (Dinh-Trong & Bieman, 2005) or Linux distributions (Gonzalez-Barahona et al., 2004) using mostly information provided by version-control-systems, but sometimes in combination with other repository data like from mailing list archives. Large-scale quantitative investigations spanning several projects going into software development issues are not yet as common, and have mostly been limited to using aggregated data provided by software project repositories (Crowston & Scozzi, 2002; Hunt & Johnson, 2002; Krishnamurthy, 2002), meta-information included in Linux Software Map entries (Dempsey et al., 2002), or data retrieved directly from the source code itself (Ghosh & Prakash, 2000).

In this paper, we will follow this approach of using publicly available data from software repositories to study the open source development process, focusing on estimation of the expended effort and programmer participation. Using a two-step approach, first a detailed case study on one project, GNOME, will be undertaken, then a large data set retrieved from a project hosting site, SourceForge.net, will be used to validate the results. Next, we will describe the data collection method and general results for both data sets.

2.2 The GNOME Project

For this case study research into open source software development, data from existing and publicly available repositories was retrieved for a single project. The project considered is GNOME, the GNU Network Object Model Environment, 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).

The main data source was the source code control system, in the case of the GNOME project the most widely used one in the open source community (Fogel, 1999), CVS (Concurrent Versions System), whose aim is to coordinate a number of participants working together on source code, trying to maximize efficiency by allowing for concurrent work on checked out copies and providing for later merges if conflicts occur. The underlying database stores each change to the code committed by a participant, thereby allowing reconstruction of prior states and comparisons between versions of source code files. In addition, meta-data on the work of the programmers within the project by submitting (checking in”, committing”) files are stored. Especially the changes in the lines-of-code, associated programmer name, file identification, date and further information are saved with each commit. 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 connections using a web-based interface (Fielding, 1999).

Repositories storing all e-mails sent to the different project discussion lists were identified as an additional source of information especially on communication and coordination besides the CVS-repository.

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: There exist 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, 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 can be checked in to the CVS-system by a programmer. The CVS-repository then records this checkin with the changes in the lines-of-code and further information. The definition of this often disputed metric (Humphrey, 1995; Park, 1992) is taken from the CVS-repository and therefore includes all types of lines-of-code, e.g. also commentaries (Fogel, 1999). In addition, any line-of-code changed is counted as one line-of-code added and one line-of-code deleted. A posting is a message to a discussion list pertaining to the GNOME project.

As a first step, the web interface of the CVS-repository as offered by Bonsai was used to retrieve the necessary data concerning checkins. This data included for every checkin programmer, file, date, LOC added and deleted, revision number and some comment. This was done using a Perl-script which generated successive queries simulating a browser-based input form. Each query spanned an interval of one day in the history of the CVS archive, starting with the earliest entries at the start of the project. The requests were distributed over a four day period in order not to overload the server. The result of each individual query was a HTML page which was subsequently parsed extracting the necessary attributes conforming to the data model. This information was then stored in a database. The necessary queries were then performed and the output analyzed 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, subject, time and complete text. For the analysis of the posting behavior of the programmers, the short name each programmer uses for checkins had to be matched to the full name or e-mail address used for postings. For 175 persons this has been possible using several regular expressions with human check-up.

In the GNOME project, 301 programmers were identified, who differ significantly in their effort for this project, with a majority contributing only a quite small amount to the total work done, a result also found in other studies (Dempsey et al., 2002; Dinh-Trong & Bieman, 2005; Mockus et al., 2002; Ghosh & Prakash, 2000; Gonzalez-Barahona & Robles-Martinez, 2003; Hertel et al., 2003). For example, Mockus et al. (2002) have shown that the top 15 of nearly 400 programmers in the Apache project added 88 per cent of the total lines-of-code. A similar distribution was found in a community of Linux kernel developers by Hertel et al. (2003). Also the results of the Orbiten Free Software survey (Ghosh & Prakash, 2000) are similar, the first decile of programmers was responsible for 72 per cent, the second for 9 per cent of the total code. In contrast, the top 15 programmers for the GNOME project were responsible for 48 per cent, while the top 52 persons were necessary to reach 80 per cent. This is similar to the results of Dinh-Trong & Bieman (2005) for the FreeBSD project, who found that fewer than 50 top developers contribute 80 percent. In the GNOME project, a clustering of the programmers based on lines-of-code added hinted at the existence of a smaller group of 11 programmers within this larger group, which were still more active, a number still allowing for easy communication and cooperation. 1 881 different posters have been identified, with the mean number of messages for identified programmers significantly higher than for all posters, and more productive programmers also more active participants in the discussions. This community structure with a small inner circle, one order larger number of programmers and still one order larger number of total participants has been found to be very similar to the results of Mockus et al. (2000, 2002) and Dinh-Trong & Bieman (2005).

The total size of the GNOME project in lines-of-code has experienced a steady increase up to the size of 1 800 000 LOC at the end of the observed time period, with 1 230 000 LOC being the size at the time it became operational (first major release in March 1999). During this time, the number of active programmers has seen a staggering rise between November 1997 and the end of 1998. A programmer is defined as active in a given month if he performed at least one checkin during this month. During the year 1999 this number has been roughly constant at around 130 persons. A correlation of 0.932 was found between total of lines-of-code added and number of active programmers each month, which confirms the usability of this number for effort estimation. Another interesting finding is that productivity (defined as the mean number of lines-of-code per programmer) is strongly positive correlated with number of active programmers in each month, thus violating Brooks' argument of increasing communication costs, which is one reason for the famous Brooks' Law (Brooks, 1995). Further results of the analysis, e.g., concerning participants, postings, cooperation on file level and progression over time can be found in Koch and Schneider (2002).

2.3 SourceForge.net: A Project Ecology

For further analysis, validation and calibration of effort and programmer participation models, a large data set covering a diverse population of large, small, successful und failed projects was necessary. SourceForge.net, the well-known and largest software development and hosting site, was chosen as the source of data. 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. While SourceForge.net publishes several statistics, e.g. on activity in their hosted projects, this information was not detailed enough for the proposed analysis. For example, Crowston and Scozzi used the available data for validating a theory for competency rallying, which suggests factors important for the success of a project (Crowston & Scozzi, 2002). Hunt and Johnson have analyzed the number of downloads of projects occurring (Hunt & Johnson, 2002), and Krishnamurthy used the available data of the 100 most active mature projects for an analysis (Krishnamurthy, 2002).

The data collection method utilized was similar to the case study for the GNOME project, with several exceptions. In this case, data from the web pages and especially the source code control system, again in the form of CVS, of the projects hosted was retrieved.

The retrieval process started differently than for the case study, as information on the project population available was necessary. Therefore the first step included inspecting the SourceForge.net homepage for the published number of currently hosted projects (at the relevant date 23,000). All of these were selected as candidates for analysis. As not all projects are both still hosted and have the CVS service enabled, the CVS information web page of each project hosted at SourceForge.net was queried for the information necessary for the data retrieval, i.e. project and server name. This resulted in 21,355 candidate projects with enabled CVS service. In addition, the development status indicator for each project assigned by the project's administrator was retrieved from its respective web page. This indicator assigned by the project administrator aims at reflecting the phase of a project in the development lifecycle.

Using the CVS web interface page provided for each project with enabled CVS service, 8,791 projects were identified which actively use this service. Together with the CVS server name information, this information was used to retrieve the necessary data from the CVS servers. This process was mostly managed by Perl scripts for web page querying and generating a shell script for CVS server access, in each case allowing for ample sleeping periods so as not to delay services for other users. In contrast to the GNOME case study, CVS was accessed directly. The output of each step was again parsed by Perl scripts for the relevant data, which was stored in a database. Analyses were performed by queries to this database and subsequent processing with a statistics package. The following analyses are based on the 8,621 projects for which all relevant information could be retrieved. Projects which do not have CVS enabled, which are no longer hosted or which never have actively used the CVS repository are not included, as there are several possible explanations for this, not necessarily project failure.

In the project population, a total of 7,734,082 commits have been made, with 663,801,121 LOCs having been added and 87,405,383 having been deleted. The projects consist of 2,474,175 single files, and an overall number of 12,395 distinct programmers have contributed with at least one commit.

The distribution of both the assets available, i.e. the programmers, and the resulting outcome, i.e. commits, lines-of-code and project status within the project ecology is very skewed (see Table 1).

TABLE 1.

Descriptive statistics for project variables from SourceForge.net data set (n=8,621)

	Min.	Max.	Mean	Std. Deviation	Median
Number of programmers	1	88	1.86	2.61	1
Commits	1	133,759	897.12	3,840.90	192
LOC added	0	12,951,218	76,998.16	458,975.28	10,801
LOC deleted	0	3,846,863	10,138.66	73,493.03	373
Files	1	42,674	285.46	1,317.74	69
Development Status	0	6	2.67	1.77	3

The vast majority of projects have only a very small number of programmers (67.5 per cent have only 1 programmer), only 1.3 per cent have more than 10 programmers. This number of programmers can be shown to follow a power law (or Pareto or Zipf) distribution (Koch, 2004), like Hunt and Johnson (2002) have also found for the number of downloads of projects. These numbers also correspond to the findings of Krishnamurthy (2002), who showed that most of the projects had only a small number of participants (median of 4). Only 19 per cent had more than 10, 22 per cent only 1 developer. While this percentage is much smaller than found here, this is not surprising as Krishnamurthy only used the 100 most active projects.

Regarding the output of the projects, a similar situation can be seen, the vast majority of projects achieves only a small number of commits and is of small size, leading to the assumption that input and output of projects a correlated, i.e. that projects with a small number of programmers only achieve small numbers of commits and lines-of-code. This intuitive relationship can be ascertained. For example, the total number of programmers of a project correlates positively at significance 1 percent with coefficients 0.472 with number of commits and 0.408 with total lines-of-code added (Koch, 2004). For starting time of a project, significant negative correlations with both input and outputs can be found (Koch, 2004).

The software evolution (Belady & Lehman, 1976; Lehman & Ramil, 2001) of open source projects has been shown by Godfrey and Tu (2000), who have analyzed the Linux operating system kernel and found a super-linear growth rate, to contradict the laws of software evolution. These 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. Paulson et al. (2004) have used a linear approximation, and have on the other hand not found any differences in growth behaviour between open and closed-source software projects. For the SourceForge.net project ecology, while 61 per cent of the projects can be shown to exhibit a decreasing growth rate, the rest show super-linear growth (Koch, 2005). Small (but significant) relationships with size and number of programmers can be found, indicating that larger projects with a higher number of participants might be more often able to sustain super-linear growth (Koch, 2005).

Regarding the situation within projects, most prior studies as cited (Dempsey et al., 2002; Dinh-Trong & Bieman, 2005; Mockus et al., 2002; Ghosh & Prakash, 2000; Hertel et al., 2003) have found a distinctly skewed distribution of effort between the participants. Similar results can also be found at the project ecology under consideration. The top decile is responsible for 79 per cent of the total SourceForge.net code base, the second decile for additional 11 per cent. Using a simple measure for the inequality of work distribution within the development team (Koch, 2004; Robles et al., 2004b), rather small, but positive correlations with total number of commits and sum of lines-of-code added show up. There is no correlation with the age of the project. Of course, the direction of the relationship is not ascertained, so the results do not necessarily indicate that more activity in projects is caused by a more unequal distribution of contributions, as the other way would also give a possible explanation, i.e. as the project grows, the inequality grows as a result.

The next possible influence on productivity in a project is the number of active programmers, following the reasoning of Brooks (1995). Therefore, the number of active programmers and the achieved progress in each project was analyzed on a monthly basis. Although the number of active programmers has a significant and positive relationship with the output, the coefficient is much smaller than previously found for the GNOME project. A correlation of 0.932 between active programmers and number of lines-of-code added showed up, while in the project ecology considered here coefficients are only 0.072 for lines-of-code added and with 0.194 slightly higher for number of commits. The communication burden as argumented by Brooks itself is next to non-existent, as the correlation between the number of active programmers and the mean output in a period is (although negative) only -0.013 with both measures (significant at 5 per cent). Additional and more detailed analyses of this data set can be found in Koch (2004).

3 Effort modeling and Programmer participation

3.1 Effort estimation for open source projects

A general discussion of the applicability of current effort estimation models hints at the existence of several problems. These show up in addition to the problems inherent in effort estimation, e.g., difficulty to predict complexity, that plague commercial software development project estimation as well. The first problem might be posed by the voluntariness of people's participation. On the one hand, sufficient staffing to ensure progress could be unavailable, on the other hand, the turnover of personnel could be faster than in commercial projects. This would limit learning effects and thus decrease overall productivity (Brooks, 1995). Contrary to this, empirical data have shown that productivity is not necessarily declining due to people joining the programming team. In addition, Gonzalez-Barahona & Robles-Martinez (2003) have shown that the core team in open source projects, while not constant over time, is not changing at a rapid pace. Therefore new entrants can slowly be trained to assume more and more responsibility. This learning process is governed in each project by certain rituals, which can be very elaborate like for example in the Debian project (Coleman & Hill, 2004), or more informal. These joining scripts" (von Krogh et al., 2003) need to be followed by possible entrants. Due to the fact that most parts of these scripts are to be performed by the applicant, e.g., by writing software to demonstrate his or her technical prowess, the overall productivity is not decreased. Nevertheless, modeling the participation of developers in open source projects forms a necessary basis for effort models, and therefore is explored in this analysis.

As an additional problem for performing effort estimation, Vixie (1999) mentions the lack of a formal design and requirements definition. Naturally, the amount of information necessary depends on two distinct factors: Whether an ex-ante or ex-post estimation is to be done, and which effort estimation approach is chosen. Therefore this point will be discussed in the context of each approach considered.

In addition, there are problems associated with some special effort estimation models, which pose restrictions or assumptions that might inherently be violated by the open source model. Such problems might limit the applicability of these models in the context of open source projects. One example is the original COCOMO 81 (Boehm, 1981), which assumes a good management by both software producer and client, development following a waterfall-model and permanence of the requirements during the whole process. If the main ideas of open source software development are analyzed, the first assumption still holds. As there is no distinction between producer and client in open source projects, instead the whole team is (at least in theory) composed of developer-turned users, no conflicts of interest are possible. The other two assumptions on the other hand are inherently violated: The requirements are neither written down (Vixie, 1999) nor constant over time, and the software development follows are more spiral type of approach, having been termed micro-spirals (Bollinger et al., 1999). These theoretical points alone make COCOMO 81, although one of the most widely known and used models in commercial environments, problematic to use for open source projects. Nevertheless, due to its simplicity, it is still nearly the only model currently employed for a rough estimate of development effort. Wheeler (2005) uses COCOMO 81 on Red Hat Linux 7.1 as a representative GNU/Linux distribution, resulting in an development effort of nearly 8,000 person-years priced at over $1 billion. Gonzalez-Barahona et al. (2004) use the same model on several releases of both Red Hat Linux and Debian, resulting in up to 26,835 person-years valued at $3,625 million for Debian 3.0. COCOMO II (Boehm et al., 2000) on the other hand does not formulate these restricting assumptions but tries to account for a more modern, prototype-oriented type of development. It also incorporates the function point sizing method (Albrecht & Gaffney, 1983). Function points can also be directly used for deriving an effort estimation, if a relationship to effort is either taken from literature, or is calibrated using an existing data set. In this case, the method also does not contain any assumption concerning the process model, but especially aims at being technology-independent and taking the user's viewpoint. The most basic model was formulated by Norden (1960), termed the Rayleigh-Norden model, and it forms the basis for most of the other approaches, including COCOMO (Boehm, 1981) or Putnam's model (Putnam, 1978). In its very general approach, it also does not pose any explicit restrictions on any development model used. Therefore, pending further analysis, all of these models do seem to be applicable to open source software development estimation on a general level.

3.2 Hypotheses generation: The GNOME Project case study

The first approach to modeling the effort for the GNOME project uses data on project participation, not on the resulting product. It is therefore based on a metric for the input to the development process, not the output, like for example lines-of-code. The approach uses the general work of Norden (1960) and its extension by Putnam (1978). Norden models any development project 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 assumed 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, represented by the difference between total effort K and cumulated effort to this date $C(t)$ in person-years. This results in the following differential equation, which can be solved to arrive at the cost function $C(t)$:

$$dC(t)/dt = p(t)\ [K - C(t)]. \tag{1}$$

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 in years since project start $p(t)=2at$ which governs the application of effort. Therefore the cumulative manpower effort is null at the start of project and grows monotonically towards the total effort. Following, the manpower function at a given time represents a Rayleigh-type curve governed by a parameter a which plays an important role in the determination of the peak manpower. This manpower function $m(t)$ can be derived by differentiating $C(t)$ relative to t:

$$m(t) = 2\ K\ a\ t\ exp\ (-\ a\ t^2). \tag{2}$$

By deriving the manpower function relative to the time and finding the zero value, the relationship between time of peak manning and this parameter can be found. Furthermore, the value of the peak manning can be obtained by substituting this term in the manpower function. Using this relationship, the total manpower required can be determined once peak manning has been reached. As the manpower distribution for the GNOME project, depicting the number of active programmers in each month, can be retrieved from the data (see Fig. 1) and seems to follow a Rayleigh-type curve at least until a certain point, this information can be used for estimating the effort. The peak manning of active programmers seems to have been reached between November 1998 and September 1999. Therefore the time elapsed between the beginning of the project (using January 1997) and the peak manning is set to 2.25 years, taking the middle of this range. The peak manning is set to 131.8 persons, again using the mean staffing during this interval. As the Rayleigh-Norden model assumes full-time employees, this number needs to be adapted, or else the results would also be scaled to open source developer-years. For this conversion, some value for the time actually invested in the project is necessary. The study of Hertel et al. (2003) reports a number of 18.4 hours per week spent on development, which is based on the answers of 69 active Linux kernel developers to a questionnaire. As the other results of this study show at several points similar characteristics to the data retrieved from the GNOME project, for example regarding the distribution of effort within the community, this number is applied here. This results in a peak manning of 60.6 persons. Using these values in the model results in a total effort of 224.8 person-years. The projected manpower function derived is also shown in Fig. 1 (depicted as VAR1). As the manpower distribution retrieved from the data shows a small level of activity until October 1997, a second model was computed using this point as start of the project. The time of peak manning then becomes 1.42 years and the total effort is estimated as 141.9 person-years. The resulting manpower function is again shown in Fig. 1 (as VAR2).

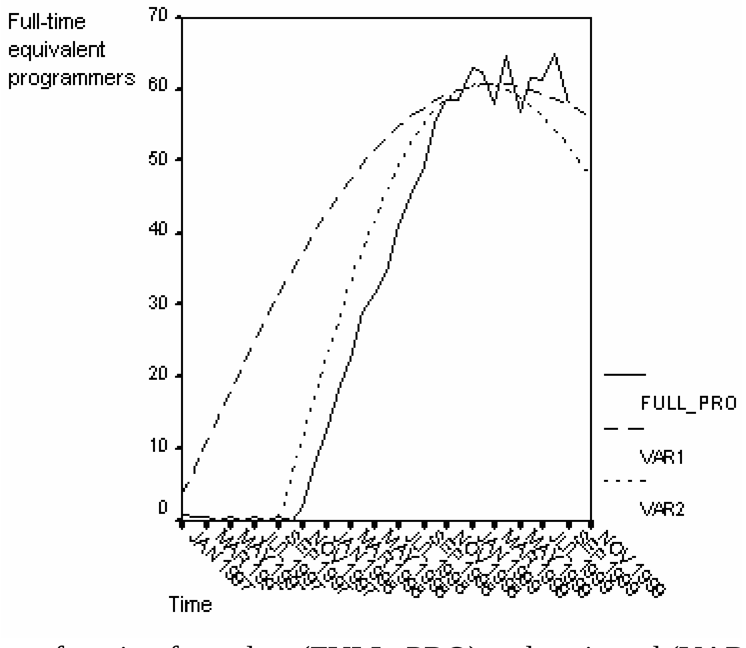

Fig. 1. Manpower function from data (FULL_PRO) and projected (VAR1 and VAR2).

As Putnam (1978) has shown, the time of peak manning is close to the time the software becomes operational, i.e. is released, while effort thereafter is expended for modification and maintenance. The first major release of GNOME has been in March 1999, which coincides with the peak manning empirically determined. The cumulative effort expended until this date is estimated as 88.4 person-years by the first model, as 55.8 by the second model computed. The results of the effort estimation for the total project presented above therefore include modification and maintenance. One problem resulting from using this model might be that as the requirements are not fixed in open source projects over time, but are expanded according to the requests of programmers and users, leading to further releases incorporating not only bug fixes, but also new functionality. Therefore the estimation presented might not give a complete forecast, as it assumes a decrease in the manpower function after the release date, which indeed can not yet be seen in the data. While Norden postulates a finite and fixed number of problems, additional requests will lead to the generation of new problems to be worked on. Therefore, in addition to the Poisson process governing resolution of problems, another stochastic process might be necessary to model this fact. While this effect might be small to negligible until time of operation, it might be the driving factor later on. Additional problems in using the Norden-Rayleigh model are the definition of the start of the project, and the linear learning rate assumed (which can only be safely assumed if there is no major turnover in participants without overlapping between groups to allow for knowledge transfer). Besides this, the fact that the Rayleigh-curve proposed for commercial projects decades before closely fits the curve for a contemporary open source project (at least until time of operation) is astonishing and hints at the fact that a self-regulating community follows the theory for efficient manpower application as well (or maybe even better) than commercial management. In addition, this model constitutes the foundation of several other estimation methods including COCOMO, which therefore might also be applicable in this context. These results lead to the formulation of the first hypothesis:

Hypothesis H1: *The manpower distribution in open source projects until time of operation is not different from commercial projects. Therefore the Norden-Rayleigh approach can be used to model the manpower function. To cover complete open source projects, the addition of new functional requests needs to be incorporated.*

For contrast, a second approach to estimating the effort for the GNOME project will be explored, this time based on output metrics of the software produced. The first model of this category to be applied is the original COCOMO (Boehm, 1981). While severe problems with using this model due to violated assumptions have already been discussed, it is still employed for comparison to other models and with existing studies (Gonzalez-Barahona et al., 2004; Wheeler, 2005). Analogous to these studies, basic COCOMO is employed using organic development mode. As COCOMO does not consider extensions, the size in lines-of-code of the GNOME project at first major release, 1,230,000 lines-of-code, is used, resulting in 351 person-years of effort. In semi-detached development mode, this number would increase to 722 person-years, in embedded mode to 1,531 person-years. Based on the theoretical discussion as presented above, COCOMO II (Boehm et al., 2000) seems to be more suitable in the context of open source software development. Again using the size of the GNOME project at time of operation, COCOMO II is first applied with nominal values for all parameters, resulting in 612.5 person-years. Using realistic parameters, e.g. high precedentedness, extra high development flexibility, low architecture/risk resolution, extra high team cohesion and low process maturity, results in a significantly lower estimation of 296.8 person-years. A similar approach to COCOMO, due to also being based on an output metric, is the function point method (Albrecht & Gaffney, 1983). This method in general offers several advantages, most importantly the possibility to quantify the function points relatively early in the development based on analysis and design, technology-independence as the user-viewpoint is considered, and no assumptions concerning the underlying software process. For an ex-post estimation, the first advantage is naturally irrelevant, but also an ex-ante estimation might face some problems due to missing documentation of requirements in open source software development. Independence of implementation language and technology used seems to be of high importance, as function points offer a possibility for comparison of productivity and efficiency based on software functionality produced. For estimating the effort for the GNOME project, it is difficult, especially for an outsider, to correctly quantify the function points, even after delivery. This would necessitate intimate knowledge of the software, and access respectively existence of relevant documentation of analysis and design. Another possibility to arrive at the number of function points is using the opposite way to converting a function point count to lines-of-code (Albrecht & Gaffney, 1983; Boehm et al., 2000). For this conversion, literature yields mean numbers of lines-of-code necessary to implement a single function point in a given programming language. In GNOME, the most employed language is C, followed by Perl and C++. Therefore an overall conversion factor is estimated by using the factors from Boehm et al. (2000) for these languages with a weight of 0.7, 0.2 and 0.1, respectively, resulting in 100.5 lines-of-code per function point. The size of GNOME at the time of operation thus corresponds to approximately 12,200 function points. In order to arrive at an effort estimation based on the function point count, either this measure is converted to lines-of-code and an estimation model like COCOMO is employed, or a relationship between function points and effort derived from finished projects is used. As the first approach has already been employed above, the second is taken at this stage. Using the equation provided by Albrecht and Gaffney (1983) results in an effort of 353.8 person-years. Different equations are provided by Kemerer (1987) resulting in 336.3 person-years, and by Matson et al. (1994) with their linear model resulting in 101.9 and their logarithmic model in 82.3 person-years. It is interesting that the newer models result in significantly lower effort estimates. This might be caused by the larger database containing larger projects employed by Matson et al. (1994), and also the date of their study which allows for newer practices to be included in their results. This might allow for stronger similarities to open source development processes.

As can be seen from the estimates given above, severe differences show up between different estimation models, especially between those based on output metrics and the Norden-Rayleigh model based on programmer participation. Therefore, a comparison between COCOMO in both versions and the Norden-Rayleigh model, as the first is based on the latter, seems of special interest. This relationship allows for a mathematical comparison of both approaches. Londeix (1987) has detailed how an estimation in COCOMO can be transferred to the model by Putnam (1978), i.e. how the Rayleigh-curve corresponding to a given COCOMO estimation can be determined. In this case the other direction is employed to try to find a parameter set in COCOMO corresponding to the Rayleigh-curve derived from programmer participation. This is impossible for basic COCOMO, as even organic development mode results in a much larger value. Intermediate COCOMO offers both the development mode and the values of a number of cost drivers as parameters, so clearly no single solution can be found. But nevertheless, even if organic mode is assumed, the influence of the cost drivers would have to be more favorable than possible in the model given the parameter space. Therefore the development of GNOME can not be modeled using original COCOMO, which leads to the conclusion that from this model's viewpoint the development is more efficient than even theoretically possible. Far more output, i.e. lines-of-code, is produced than assumed given the programmer effort put into the project. Therefore the successor COCOMO II (Boehm et al., 2000) seems to be a better choice, as it allows for both increasing and decreasing economies of scale, a prototype-oriented software process and flexibility in the requirements. When possible parameters are explored based on the Norden-Rayleigh estimation, the result is that once again this project is seen as very efficient as the both cost drivers and scale factors replacing the modes of development in COCOMO II have to be rated rather favorably to obtain the estimated effort from the Rayleigh-curve, but this time the resulting combinations are within the specified range. If realistic values for the scale factors are used, the resulting necessary value for the effect of the cost drivers is still within possible range.

In comparing the Norden-Rayleigh results to estimates derived using function point equations, the results from function point are higher than those from programmer participation in all cases, although the difference is less poignant when using newer models.

In general, results from applying result-based estimation models relying on product metrics are considerably higher than those derived from programmer participation. Whether this is a general trend or only applies to the case study is to be tested, using hypothesis H2.

Hypothesis H2: *Effort estimates for open source projects based on programmer participation are lower than those based on output (product) metrics.*

This difference might be caused by several reasons: The open source development model might constitute a more efficient way of producing software, mostly due to self-selection outperforming management intervention. Participants might be able to more accurately determine whether or not they are able to productively work on the project overall, or on which tasks. In addition, overhead costs are very much reduced in this model of software development. The second explanation would be that the difference between programmer participation-based and product-based estimates is caused by non-programmer participation, i.e. people participating by discussing on mailing lists, reporting bugs, maintaining web sites or similar. If the Norden-Rayleigh and COCOMO 81 estimates are compared, COCOMO 81 results are more than eightfold those from programmer participation. If it were assumed that this difference would only come from the effort invisibly expended by these participants, their effort would be enormous. It would account for about 88 per cent of the effort, translating to about 7 persons assisting each programmer. As has been shown, the number of participants other than programmers is about one order of magnitude larger than the number of programmers (Dinh-Trong & Bieman, 2005; Mockus et al., 2000, 2002; Koch & Schneider, 2002), but their expended effort was implicitly assumed to be much smaller. Decades ago, Mills (1971) has proposed the ‘chief programmer team organization’ (Baker, 1972), also termed ‘surgical team’ by Brooks (1995), in which system development is divided into tasks each handled by a chief programmer who is responsible for the most part of the actual design and coding, supported by a larger number of other specialists like a documentation writer or a tester. Brooks (1995) mentions a team size of ten persons, but most of these would not be working on the team full-time. Therefore a possible explanation for the differences in effort estimates could be seen in the effort expended by these team members.

3.3 Evaluating the hypotheses using SourceForge.net data

For testing hypothesis H1, i.e. whether the Norden-Rayleigh approach can be used to model the manpower function of open source projects and whether it needs to be amended for the addition of new requests, several approaches are computed and compared. The first one is the standard Norden-Rayleigh approach, deriving the necessary parameters from time of peak manning and number of active programmers. As an in-depth analysis including consideration of release dates analogous to the case study is impossible for more than 8,000 projects, the point in time of peak manning and the respective number of active programmers are retrieved for each project. From this, the Rayleigh-curve is constructed automatically. As a second approach, the necessary parameters are not computed using the equations given by Norden, but are derived from non-linear regression.

As a third group of models, the Norden-Rayleigh manpower function is adapted to reflect the introduction of additional requests during the life cycle, which will lead to the generation of new problems to be worked on, thus necessitating more manpower than the model accounts for. For including this effect, the general ideas of Norden are applied: During each time interval, besides the problems left from the fixed starting set to be worked on by the manpower, additional problems have been introduced to be worked on. Therefore an additive term is introduced. In the first line of thought, the number of additional problems introduced is proportional to the number of problems in the starting set. Similar to Norden, it is assumed that team learning contributes to the capability to solve, and also uncover problems.

Therefore, the same linear learning rate $p(t)=2at$ is applied to the number of additional problems. This gives the first new differential equation

$$dC(t)/dt = p(t)\,[K - C(t)] + p(t)\,K \tag{3}$$

and a resulting manpower function:

$$m(t) = 2\,K\,a\,t\,\exp(-\,a\,t^2) + 2\,K\,a\,t. \tag{4}$$

As a further approach, a different linear learning rate $q(t)=2bt$ is assumed for the introduction of new problems, thus introducing an additional parameter b:

$$dC(t)/dt = p(t)\,[K - C(t)] + q(t)\,K \tag{5}$$

$$m(t) = 2\,K\,a\,t\,\exp(-\,a\,t^2) + 2\,K\,b\,t. \tag{6}$$

As both of these approaches would eventually lead to unlimited projects, as the linear term would be increasing without boundary, we also compute a model using a quadratic learning rate $r(t)=bt^2+ct$:

$$dC(t)/dt = p(t)\,[K - C(t)] + r(t)\,K \tag{7}$$

$$m(t) = 2\,K\,a\,t\,\exp(-\,a\,t^2) + K\,b\,t^2 + K\,c\,t. \tag{8}$$

In the second line of thought, the number of problems introduced is not assumed to be proportional to the number of problems in the starting set, but to the number of problems already solved, i.e. the cumulated effort $C(t)$ already applied. Again, it is assumed that team learning contributes to the capability to solve, and also uncover problems. Analogous to above, either the same learning rate $p(t)$, resulting in:

$$dC(t)/dt = p(t)\,[K - C(t)] + p(t)\,C(t) \tag{9}$$

$$C(t) = a\,K\,t^2 \tag{10}$$

$$m(t) = C'(t) = 2\,K\,a\,t \tag{11}$$

or a different linear learning rate $q(t)$, resulting in:

$$dC(t)/dt = p(t)\,[K - C(t)] + q(t)\,C(t) \tag{12}$$

$$C(t) = -\,[a\,K\,(-1 + \exp[(-a + b)\,t^2])]\,/\,(a - b) \tag{13}$$

$$m(t) = C'(t) = -\,[2\,K\,a\,t\,(-a + b)\,\exp[(-a + b)\,t^2]]\,/\,(a - b) \tag{14}$$

are applied. Using a quadratic learning rate $r(t)$ as above does not yield any usable solutions to the differential equation and is therefore discarded. For all of these models, the manpower function, i.e. equations (4), (6), (8), (11) and (14) are fitted to the data using non-linear regression to determine the necessary parameters.

For comparison to these Norden-Rayleigh-based manpower functions, both a standard linear and a quadratic model are computed as well. This results in an overall number of 9 different models for the manpower function. For all models, squared residuals, chi-squared test-statistic and R-squared values have been computed to compare their goodness of fit (see Table 2).

TABLE 2.
Results from manpower function estimation for SourceForge.net projects

		N	Min.	Max.	Mean	Median
Norden-Rayleigh (equation-based)	Squared residuals	8620	.00	19574.93	8.00	.24
	Chi-squared	8620	.00	1.5+295	2.9+291	.44
	R-squared	2243	-37.08	1.00	-1.37	-.84
Norden-Rayleigh (regression)	Squared residuals	2243	.00	1599.02	8.76	1.88
	Chi-squared	2243	.00	6.5+269	2.9+266	2.06
	R-squared	2243	-15.25	1.00	-.14	.12
Norden-Rayleigh mod. equation (4)	Squared residuals	944	.00	8.4+307	8.9+304	9.63
	Chi-squared	944	-1.5+154	745.41	-1.6+151	18.18
	R-squared	943	-9+307	.91	-9.6+304	-1.21
Norden-Rayleigh mod. equation (6)	Squared residuals	943	.20	2.3+307	2.4+304	3.92
	Chi-squared	943	-6.8+153	188.86	-7.2+150	3.64
	R-squared	943	-2.1+306	.97	-2.2+303	.16
Norden-Rayleigh mod. equation (8)	Squared residuals	648	.00	2.5+307	3.8+304	4.35
	Chi-squared	648	-7+153	383.72	-1.1+151	3.46
	R-squared	648	-2.3+306	1.00	-3.5+303	.27
Norden-Rayleigh mod. equation (11)	Squared residuals	1743	.00	1595.89	20.14	6.40
	Chi-squared	1743	.00	3362.35	34.37	13.91
	R-squared	1743	-16.94	1.00	-1.95	-1.63
Norden-Rayleigh mod. equation (14)	Squared residuals	754	.00	5365.17	25.82	7.29
	Chi-squared	754	.00	6.7+50	8.9+47	10.04
	R-squared	754	-11.22	1.00	-.54	-.34
Linear model	Squared residuals	2243	.00	1619.79	6.76	1.07
	Chi-squared	2243	-47.51	113.09	2.26	.75
	R-squared	2243	.00	1.00	.35	.22
Quadratic model	Squared residuals	2243	.00	1002.32	5.18	.72
	Chi-squared	2243	-3.6+15	191.73	-1.6+12	.54
	R-squared	2243	.00	1.00	.5608	.52

All of these different quality indices have been used for comparing the models. A Wilcoxon signed-rank test has been employed, due to the fact that all variables are not normally distributed (which can be verified using a Kolmogorov-Smirnov test, significance in all cases lower than 1 per cent). All models have been tested against each other, first for the complete data set, then for two validation subsets. As a first subset, only those 1,636 projects with status of production/stable and mature are used, to counter any effects of different progression in the lifecycle. As a second subset for validation, only the 43 largest projects, defined as having at least five developers and 500,000 lines-of-code, have been selected. The results are very consistent and do not depend on the quality index used. Also the changes between different test sets are very small. Overall, in the full data set, both the simple quadratic and the modified Norden-Rayleigh-function using a different, quadratic learning rate as given by equation (8) significantly outperform all other models (applying Bonferroni correction for 36 independent tests which reduces the maximum accepted *p*-value to .000278 at the .01 level). The difference between these two is not statistically significant. This is followed by the simple linear model, again not statistically significantly distinguishable from the modified Norden-Rayleigh-function using a different, linear learning rate as given by equation (6). Afterwards, the standard Norden-Rayleigh model with regression-derived parameters, the modified model based on cumulative effort given by equation (14), the modified model with same learning rate as given by equation (4), the standard Norden-Rayleigh with parameters derived from using Norden's equations and the model described by equation (11). In using the first subset, i.e. only mature projects, the worst performing three models can no longer be statistically significantly distinguished. In the second subset, when only the largest projects are considered, also the top four models are no longer different. From these results, the following conclusion can be generalized: Norden-Rayleigh-based approaches, while they can not be dismissed, are not able to significantly outperform other, simple models for arriving at a manpower function for open source projects. Therefore hypothesis H1 is, with limitations, confirmed. When considering Norden-Rayleigh-based approaches, the standard models without incorporating the addition of new features during the life cycle perform badly over complete project lifespans. For incorporating this effect, a different proportionality factor, i.e. learning rate, has to be assumed. Models using the same learning rate perform very badly. As a further conclusion, the features added seem to depend on the starting problem set more than on the cumulative effort expended until the respective time, as those class of models consequently outperforms the latter one. In general, a quadratic function seems to be better suited to modeling both the manpower function, and also the addition of new problems. Only in the dataset containing the largest projects is this difference no longer statistically significant.

Next, hypothesis H2 concerning the difference between programmer-participation and output (product) metric based estimation results will be explored. For arriving at estimates for the project population using the Norden-Rayleigh model, the same approaches as detailed for testing hypothesis H1 are used. First, the necessary parameters are extracted from point in time of peak manning and the respective number of active programmers automatically using the equations by Norden, and for a second approach non-linear regression is used to estimate these parameters. As the total effort to be expended is also used as parameter in the modified Norden-Rayleigh models as described above, these parameter values derived from regression are also used. The resulting effort data are scaled to open source programmer-years, so again using the mean number of working hours of open source programmers, these measures are converted. For equation-based estimation, the mean effort for a single project is 0.69 and median 0.19 person-years. All results for these approaches are given in Table 3 and give an estimation of the total effort expended during the project lifecycle.

As an additional data point, effort expended in the projects until the end of inspection was calculated from the available data. This was performed by cumulating the number of active programmers per month from project start until last data point. Again, the result is converted from open source programmer-years into full-time employee equivalents. Naturally, results are lower overall than those reached by Norden-Rayleigh estimations, which include effort to be expended in the future (see also Table 3).

For comparison, product-based estimates are computed using several COCOMO and function point equations analogous to the case study (all results can be found in Table 3). Using COCOMO 81 basic model and organic development mode for each project, the mean for each project is about 18.6 person-years (with median 2.02). Results for COCOMO II, using in a first approach nominal values for all parameters, and in a second attempt realistic parameters identical to the case study results in a mean effort of 31.84 (median 2.76) respectively 15.67 (median 1.69) person-years. For function point estimation, the same conversion factor as applied in the case study has been used, again due to the difficulty of inspecting more than 8,000 projects. Resulting size in function points of the projects considered therefore ranged up to 130,000 function points, with mean 665 and median 90 function points. Results for effort estimation are generally lower than COCOMO estimates in all variants, but contain negative values due to projects with small size using the equations provided by Albrecht & Gaffney (1983) and Kemerer (1987). Both approaches therefore need to be discarded, at least for estimations using the complete project space. Results for these estimations can also be found in Table 3. It should be noted that these estimations use the size of the software systems at the time of data retrieval, therefore reflect the effort estimated to be necessary to produce this final product and do not incorporate any effort to be expended in the future for additional releases.

TABLE 3.
Results from effort estimation for SourceForge.net projects

	N	**Min.**	**Max.**	**Mean**	**Std. Deviation**	**Median**	**Sum over all projects**
Norden-Rayleigh (equation-based)	8,620	0.06	200	0.69	3.72	0.19	5,965
Norden-Rayleigh (regression)	2,192	0.10	11,337.49	29.07	299.47	0.74	63,711
Norden-Rayleigh mod. equation (4)	944	-2,996.65	1,001.53	-117,62	210.23	0.11	-111,037
Norden-Rayleigh mod. equation (6)	944	-508.47	59,692.56	396,11	2,422.63	0.46	373,925
Norden-Rayleigh mod. equation (8)	649	-295,987.41	14,288.15	-2,197.98	21,352.57	0.35	-1,426,491
Norden-Rayleigh mod. equation (11)	1,743	-14.01	25.77	3.06	4.47	3.07	5,341
Norden-Rayleigh mod. equation (14)	755	-23.44	35.00	1.44	3.31	0.66	1,089
Active programmer effort years	8,620	0.10	55	0.56	1.39	0.12	2,229
COCOMO 81	8,621	0.00	4,159	18.56	133.66	2.02	160,020
COCOMO II (nominal parameters)	8,621	0.00	8,156	31.84	254.73	2.76	135,112
COCOMO II (realistic parameters)	8,621	0.00	3,539	15.67	113.55	1.69	274,482
Function Point (Albrecht & Gaffney)	8,621	-7.34	3,808	12.36	126.51	-4.67	106,528
Function Point (Kemerer)	8,621	-10.17	3,650	8.73	121.35	-7.61	75,249
Function Point (Matson et al., linear)	8,621	0.32	1,069	5.84	35.42	1.07	50,316
Function Point (Matson et al., log.)	8,559	0.00	869	4.52	28.93	0.62	38,695

For testing hypothesis H2, a Wilcoxon signed-rank test is employed, due to the fact that all variables are not normally distributed (which can be verified using a Kolmogorov-Smirnov test, significance in all cases lower than 1 per cent, again applying Bonferroni correction for the number of independent tests). Estimates derived from Norden-Rayleigh modeling are tested against each other method. Results clearly support hypothesis H2, the differences are in all cases significant (with significance below 1 per cent for all comparisons). This holds true also for Norden-Rayleigh estimates derived from non-linear regression, or from modified models. In comparison to the estimates for effort expended until now, Norden-Rayleigh results are significantly higher (with significance below 1 per cent), which is not surprising due to different time frame. Naturally, product-based estimates are significantly higher than this effort as well, although in this case size and effort consider the same time interval.

For validation of these results, again the same two subsets of the projects are selected and tested separately. First, only the projects with status of production/stable and mature are used, to counter any effects of different progression in the lifecycle. As has been detailed before, Norden-Raleigh estimation gives the total effort needed to finish the software, including maintenance and enhancements, given that the peak manning seen as yet is the peak manning for the whole project, and also the time of first operation. COCOMO estimation on the other hand uses the size of the software at time of operation as input, and returns effort until this point in time. Therefore the results from Norden-Rayleigh should in general be larger than COCOMO results, but as current size is used as input for the latter, some degree of enhancements will also already be included in this estimation. Both will underestimate the effort for projects not progressed very far yet, COCOMO because it does not assume any further increase in lines-of-code, Norden-Rayleigh because the peak manning would not have been reached. Results from using these 1,636 projects does not change results from comparing effort estimates, so hypothesis H2 is still supported. As a second subset for validation, only the 43 largest projects, defined as above, are selected. Within this group, all estimation methods give positive results throughout, therefore are included into the test statistics. Again, a Wilcoxon signed-rank test is employed to test for differences between Norden-Rayleigh estimates derived from all techniques and each other estimates. Also in this validation sample, results are consistent, Norden-Rayleigh equation-based estimates are, with the exception of one project in comparison to COCOMO II with realistic parameters and function point estimation using equations from Matson et al. (1994), lower. Norden-Rayleigh regression and modified equation results are also significantly lower, with a few more exceptions. Significance levels of the tests remain below 1 per cent in all cases. Therefore hypothesis H2 is supported also in all validation subsets.

4 Conclusions

This paper has taken up the subject of effort and programmer participation modeling for open source software development projects, which has to this date not been extensively covered, leading to the application of only moderately advanced techniques from the toolset offered by software engineering research. We have tried to motivate establishing effort estimation for open source projects from two major ideas: One application is ex-post estimation of finished or active projects, in order to be able to accurately assess the efficiency of this new development model, while also results from traditional, ex-ante estimation would lend important information to diverse stakeholders in and around open source projects, e.g. companies intending to pursue a related business model or planning to incorporate it in their products or services.

For our research, the method applied relies on the analysis of software development repositories, which offers several advantages in studying software processes. Especially in open source software development projects, repositories in several forms form the most important communication and coordination channels and are available openly and publicly. Already several studies have shown that these sources can give important information about the underlying development processes. In this paper, we have followed this approach, focusing on estimation of the expended effort. First a detailed case study on one project, GNOME, has been undertaken based on mailing list and source code repository data, then

Sourceforge.net, a project hosting and community site, was used to gather a large sample of diverse projects for validation.

Analysis of the case study has led to the formulation of two hypothesis: First, the number of active programmers has been found to closely follow the Norden-Rayleigh model (Norden, 1960). Therefore hypothesis H1 concluded that the manpower distribution in open source software development projects is not different from commercial projects and therefore can be modeled using the Norden-Rayleigh approach. One major difference to traditional software engineering processes is that the requirements are not fixed in open source projects over time, but are expanded according to the requests of programmers and users. It was assumed that this generation of new problems might be an important extension to the standard model. Applying several estimation models to the case study, significant differences between result-based estimation models relying on product metrics and those derived from programmer participation showed up. This led to formulation of hypothesis H2, saying that effort estimates for open source projects based on programmer participation are lower than those based on output (product) metrics. Possible reasons for this effect include the open source model being more efficient in using self-selection for task management, or an extremely high amount of non-programmer participation, which would lead to see these projects as ‘chief programmer teams’ (Mills, 1971; Baker, 1972) or ‘surgical teams’ (Brooks, 1995).

In testing both hypotheses, the full sample of about 8,000 projects from Sourceforge.net was used, with two subsets for additional validation of results. Hypothesis H1 was partly supported by computing and comparing nine different manpower functions, as Norden-Rayleigh-based approaches, while they can not be dismissed, are not able to significantly outperform other, simple models for open source projects. When considering Norden-Rayleigh-based approaches, the standard models without incorporating the addition of new features during the life cycle perform badly. This confirmed the notion described above. For incorporating this effect, a different proportionality factor, i.e. learning rate, has to be assumed, as models using the same learning rate perform very badly. As a further conclusion, the features added seem to depend on the starting problem set more than on the cumulative effort expended until the respective time.

Regarding hypothesis H2, this was unconditionally supported. Estimates derived from programmer-participation based methods show significantly less effort than those based on output metrics like lines-of-code or function points.

Overall, this paper demonstrated several points: Effort modeling has to be seen as an important topic also in the context of open source software development, and needs to be explored further. Data retrieved from software repositories offers several advantages and can form the base for this research, as this paper has demonstrated. While some similarities to traditional projects regarding the manpower distribution can be found, the addition of new features based on user requests during the life cycle features much more prominently and necessitates incorporation into newly developed models. Results from effort estimation show that programmer-participation based methods show significantly less effort than those based on output metrics. This leads to the conclusion that either the open source model is really a more efficient way of producing software, or that an extremely high effort is contributed by non-programmers.

Note

Please note that all data like effort estimates for the projects are available from the author, and are not reproduced in the paper due to the large size of the data set which encompasses nearly 9,000 projects.

References

Albrecht, A.J. & Gaffney, J.E. (1983). Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. *IEEE Transactions on Software Engineering*, 9(6), 639-648.

Atkins, D., Ball, T., Graves, T. & Mockus, A. (1999). Using Version Control Data to Evaluate the Impact of Software Tools. *Proc. 21st International Conference on Software Engineering*, pp. 324-333.

Baker, F.T. (1972). Chief Programmer Team Management of Production Programming. *IBM Systems Journal*, 11(1), 56-73.

Belady, L.A. & Lehman, M.M. (1976). A model of large program development. *IBM Systems Journal*, 15(3), 225-252.

Boehm, B.W. (1981). *Software Engineering Economics*, Englewood Cliffs, N.J.: Prentice-Hall.

Boehm, B.W., Abts, C., Brown, A.W., Chulani, S., Clark, B.K., Horowitz, E., Madachy, R., Reifer, D.J. & Steece, B. (2000). *Software Cost Estimation with COCOMO II*, Upper Saddle River, N.J.: Prentice Hall.

Bollinger, T., Nelson, R., Self, K.M. & Turnbull, S.J. (1999). Open-source methods: Peering through the clutter. *IEEE Software*, 16(4), 8-11.

Brooks jr., F.P. (1995). *The Mythical Man-Month: Essays on Software Engineering,* Anniversary ed., Reading, Mass.: Addison-Wesley.

Coleman, E.G. & Hill, B. (2004). The social production of ethics in Debian and free software communities: Anthropological lessons for vocational ethic. In: Koch, S. (ed.), *Free/Open Source Software Development*, pp. 273-295, Hershey, Pa.: Idea Group Publishing.

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

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

Dempsey, B.J., Weiss, D., Jones, P. & Greenberg, J. (2002). Who is an open source software developer?" *Communications of the ACM,* 45(2), 67-72.

Dinh-Trong, T.T. & Bieman, J.M. (2005). The FreeBSD Project: A Replication Case Study of Open Source Development. *IEEE Transactions on Software Engineering*, 31(6), 481-494.

Dutoit, A.H. & Bruegge, B. (1998). Communication Metrics for Software Development. *IEEE Transactions on Software Engineering*, 24(8), 615-628.

Feller, J. & Fitzgerald, B. (2002). *Understanding Open Source Software Development,* London: Addison-Wesley.

Fielding, R.T. (1999). Shared Leadership in the Apache Project. *Communications of the ACM*, 42(4), 42-43.

Fogel, K. (1999). *Open Source Development with CVS,* Scottsdale, Arizona: CoriolisOpen Press.

Gallivan, M.J. (2001). 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), 277-304.

Ghosh, R. & Prakash, V.V. (2000). The Orbiten Free Software Survey. *First Monday,* 5(7).

Godfrey, M.W. & Tu, Q. (2000). Evolution in Open Source software: A case study. *Proceedings. International Conference on Software Maintenance*, pp. 131-142.

Gonzalez-Barahona, J.M., Robles, G., Ortuno Perez, M., Rodero-Merino, L., Centeno-Gonzalez, J., Matellan-Olivera, V., Castro-Barbero, E. & de-las Heras-Quiros, P. (2004). Analyzing the anatomy of GNU/Linux distributions: methodology and case studies (Red Hat and Debian). In: Koch, S. (ed.), *Free/Open Source Software Development*, pp. 27-58, Hershey, Pa.: Idea Group Publishing.

González-Barahona, J.M. & Robles-Martínez, G. (2003). Unmounting the ‘code gods’ assumption. *Proceedings Workshop at XP2003 Conference : Making Free/Open-Source Software Work Better*.

Hertel, G., Niedner, S. & Hermann, S. (2003). Motivation of software developers in open source projects: An internet-based survey of contributors to the Linux kernel. *Research Policy*, 32(7), 1159-1177.

Humphrey, W.S. (1995). *A Discipline for Software Engineering*, Reading, Massachusetts: Addison-Wesley.

Hunt, F. & Johnson, P. (2002). On the pareto distribution of sourceforge projects. *Proceedings Open Source Software Development Workshop*, pp. 122-129.

Kemerer, C.F. (1987). An Empirical Validation of Software Cost Estimation Models. *Communications of the ACM*, 30(5), 416-429.

Kemerer, C.F. & Slaughter, S. (1999). An Empirical Approach to Studying Software Evolution. *IEEE Transactions on Software Engineering*, 25(4), 493-509.

Koch, S. & Schneider, G. (2002). Effort, Cooperation and Coordination in an Open Source Software Project: Gnome. *Information Systems Journal*, 12(1), 27-422002.

Koch, S. (2004). Profiling an open source project ecology and its programmers. *Electronic Markets*, 14(2), 77-882004.

Koch, S. (2005). Evolution of Open Source Software Systems - A Large-Scale Investigation. *Proceedings of the 1st International Conference on Open Source Systems (OSS 2005)*, pp. 148-153, Genova, Italy.

Koru, A.G. & Tian, J. (2004). Defect Handling in Medium and Large Open Source Projects. *IEEE Software*, 21(4), 54-61.

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

Lehman, M.M. & Ramil, J.F. (2001). Rules and Tools for Software Evolution Planning and Management. *Annals of Software Engineering*, 11, 15-44.

Londeix, B. (1987). *Cost Estimation for Software Development*, Wokingham, UK: Addison-Wesley.

Matson, J.E., Barrett, B.E. & Mellichamp, J.M. (1994). Software Development Cost Estimation Using Function Points. *IEEE Transactions on Software Engineering*, 20(4), 275-287.

McConnell, S. (1999). Open-source methodology: Ready for prime time? *IEEE Software*, 16(4), 6-8.

Mills, H.D. (1971). Chief Programmer Teams: Principles and Procedures. Report FSC 71-5108, IBM Federal Systems Division, Gaithersburg, Maryland.

Mockus, A., Fielding, R. & Herbsleb, J. (2000). A Case Study of Open Source Software Development: The Apache Server. *Proceedings 22nd International Conference on Software Engineering*, pp. 263-272.

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

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

Park, R.E. (1992). Software size measurement: A framework for counting source statements. Technical Report CMU/SEI-92-TR-20, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania.

Paulson, J.W., Succi, G. & Eberlein, A. (2004). An empirical study of open-source and closed-source software products. *IEEE Transactions on Software Engineering*, 30(4), 246-256.

Perens, B. (1999). The Open Source Definition. In DiBona, C. et al. (eds.), *Open Sources: Voices from the Open Source Revolution*, Cambridge, Mass.: O’Reilly & Associates.

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

Raymond, E.S. (1999). *The Cathedral and the Bazaar,* Cambridge, Mass.: O'Reilly & Associates.

Robles, G., González-Barahona, J.M. & Ghosh, R.A. (2004a). GlueTheos: Automating the Retrieval and Analysis of Data from Publicly Available Repositories. *Proceedings Mining Software Repositories Workshop*, 26th International Conference on Software Engineering.

Robles, G., Koch, S. & González-Barahona, J.M. (2004b). Remote analysis and measurement of libre software systems by means of the CVSAnalY tool. *Proceedings 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems*, 26th International Conference on Software Engineering.

Sharma, S., Sugumaran, V. & Rajagopalan, B. (2002). A framework for creating hybrid-OSS communities. *Information Systems Journal*, 12(1), 7-252002.

Stallman, R.M. (2002). *Free Software, Free Society: Selected Essays of Richard M. Stallman,* Boston, Mass.: GNU Press.

Stamelos, I., Angelis, L., Oikonomu, A. & Bleris, G.L. (2002). Code quality analysis in Open-Source software development. *Information Systems Journal*, 12(1), 43-60.

Vixie, P. (1999). Software Engineering. In DiBona, C. et al. (eds.), *Open Sources: Voices from the Open Source Revolution*, Cambridge, Mass.: O'Reilly & Associates.

von Krogh, G., Spaeth, S. & Lakhani, K.R. (2003). Community, joining and specialization in open source software innovation: A case study. *Research Policy*, 32(7), 1217-1241.

Weinberg, G.M. (1998). *The Psychology of Computer Programming,* Silver Anniversary Edition, New York: Dorset House Publishing.

Wheeler, D.A. (2005). More Than a Gigabuck: Estimating GNU/Linux's Size - Version 1.07 (updated 2002). Retrieved October 11, 2005, from http://www.dwheeler.com/sloc/redhat71-v1/redhat71sloc.html

Zhao, L. & Elbaum, S. (2000). A survey on quality related activities in open source. *Software Engineering Notes*, 25(3), 54-57.