Preface

EVALUATION AND SELECTION OF FREE AND OPEN SOURCE SOFTWARE: A REVIEW AND PORTFOLIO PLANNING PROPOSAL

ABSTRACT
This preface reviews the importance of and approaches for arriving at an assessment and evaluation of open source projects. It then proposes a solution based on an analysis of their growth rates in several aspects. These include: code base, developer number, bug reports, and downloads. Based on this analysis and assessment, a well-known portfolio planning method, the BCG matrix, is employed for arriving at a very broad classification of open source projects. While this approach naturally results in a loss of detailed information, a top-level categorization is in some domains necessary and of interest.

INTRODUCTION
The adoption and evaluation of free (Stallmann, 2002) and open source (Perens, 1999) projects has gained increasing interest, both from an academic and business perspective (Norris and Kamp, 2004). In the context of several business model and approaches based on the idea of open innovation, organizations increasingly seek to get outside assistance and knowledge as embodied in software in preparing their own products or services. Due to the importance that such use together with sometimes continued collaboration has gained, it becomes a vital task and skill to choose the correct project and community. This has led to the development of assessment schemes like OpenBRR (Open Business Readiness Rating) (Wassermann et al., 2005), Open Source Maturity Model (Petrinja et al. 2010a), QSOS by Atos Origin (2009), OpenBQR (Taibi et al., 2007), and similar achievements (Petrinja et al., 2010). Most of these approaches are based on detailed scoring of open source products, and aggregation using some form of weightings. While some consider that features of the underlying community form an important part of an evaluation (Robles et al., 2006), this is not generally acknowledged. In addition, while in some approaches the use of real data, both on community and the software product itself is planned, some rely on personal rating or data entry of many features. Some of the approaches are also overly complex for the context of managerial decision-making, or the aggregation of different aspects to higher levels of abstraction is poorly and problematically handled.

The approach first proposed by Koch and Stix (2008) tries to both rely on actual data, and to provide a top-level, aggregate classification scheme. They proposed to base such an effort on well-known portfolio planning techniques, especially the BCG matrix, and use growth rate analysis as axis of such a matrix.
DATA GATHERING AND ANALYSIS

For any evaluation and assessment based on actual data and not expert opinions, the information contained in software development repositories will be used. Software development repositories contain a plethora of information on software and underlying, 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): It is very cost-effective, as no additional instrumentation is necessary, and does not influence the process under consideration. In addition, longitudinal data are available, allowing for analyses that consider the whole project history. Depending on the tools used in a project, possible repositories available for analysis include source code versioning systems (Atkins et al., 1999; Kemerer and Slaughter, 1999), bug reporting systems, and mailing lists.

In open source software development projects, repositories in several forms are also in use, in fact form the most important communication and coordination channels. Therefore only a small amount of information cannot 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. Therefore open source software development repositories form an optimal data source for studying the associated type of software development.

Prior studies have included both in-depth analyses of small numbers of successful (Crowston et al., 2006) projects (Gallivan, 2001) like Apache and Mozilla (Mockus et al., 2002), GNOME (Koch and Schneider, 2002), or FreeBSD (Dinh-Tong and Bieman, 2005), and also large data samples, such of those derived from Sourceforge.net (Koch, 2004; Long & Siau, 2007). Primarily, information provided by version control systems has been used, but also aggregated data provided by software repositories (Crowston and Scozzi, 2002; Hunt and 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).

Although this data is available, the task is made more complicated by the large size and scope of the project repositories or code forges, and the heterogeneity of the projects being studied (Howison and Crowston, 2004; Robles et al., 2009; Syeed et al., 2011; Sunindyo et al., 2011). Therefore, in the last years RoRs (“repository of repositories”) have been developed, which collect, aggregate, and clean the targeted repository data (Sowe et al., 2007; Berger et al., 2010). Two examples are FLOSSMetrics and FLOSSmole. These RoRs usually hold data collected from project repositories, and some of them also store some analysis and metrics calculated on the retrieved data. The results (raw data, summary data, and/or analyses) will be stored in a database and accessible to the rest of the research community. The researcher therefore does not need to collect data independently.

For characterising the past development, and also gain an understanding of possible future developments, growth rates can be computed for several aspects like source code, contributors, bug reports, mailing list postings or downloads. All of these might give some insight, while of course the growth in size (of source code) is most often cited (and normally understood under the term software evolution). It should be noted that there are some interrelations between these aspects, for example quality of open source projects (Dinh-Trong & Bieman, 2005; Stamelos et al., 2002; Zhao & Elbaum, 2000; Ruiz & Robinson, 2011) and community. Prior research used characteristics of quality for which diverse metrics from software engineering like McCabe’s cyclomatic complexity (McCabe, 1976) or Chidamber and Kemerer’s object-oriented metrics suite (Chidamber & Kemerer, 1994) can be employed. Koch and Neumann (2008) have attempted such an analysis using Java frameworks, and found that a high number
of programmers and commits, as well as a high concentration is associated with problems in quality on class level, mostly to violations of size and design guidelines. This underlines the results of Koru and Tian (2005), who have found that modules with many changes rate quite high on structural measures like size or inheritance. If the architecture is not modular enough, a high concentration might show up as a result of this, as it can preclude more diverse participation. The other explanation is that classes that are programmed and or maintained by a small core team are more complex due to the fact that these programmers ‘know’ their own code and don’t see the need for splitting large and complex methods. One possibility in this case is a refactoring (Fowler, 1999) for a more modular architecture with smaller classes and more pronounced use of inheritance. This would increase the possible participation, thus maybe in turn leading to lower concentration, and maintainability together with other quality aspects. Underlining these results, MacCormack et al. (2006) have in a similar study used design structure matrices to study the difference between open source and proprietary developed software, without further discrimination in development practices. They find significant differences between Linux, which is more modular, and the first version of Mozilla. The evolution of Mozilla then shows purposeful redesign aiming for a more modular architecture, which resulted in modularity even higher than Linux. They conclude that a product’s design mirrors the organization developing it, in that a product developed by a distributed team such as Linux was more modular compared to Mozilla developed by a collocated team. Alternatively, the design also reflects purposeful choices made by the developers based on contextual challenges, in that Mozilla was successfully redesigned for higher modularity at a later stage. On project level, research found a distinct difference: Those projects with high overall quality ranking have more authors and commits, but a smaller concentration than those ranking poorly. Thus, on class level a negative impact of more programmers was found, while on project level a positive effect. This underlines a central statement of open source software development on a general level, that as many people as possible should be attracted to a project. On the other hand, these resources should, from the viewpoint of product quality, be organized in small teams. Ideally, on both levels, the effort is not concentrated on too few of the relevant participants.

For computing and characterising the growth rates to capture these effects, the following methodology will be adopted. This is taken from a prior study of one of the project participants (Koch, 2007) on growth in size. The first step is to analyse whether a linear or other growth pattern is present in the data. To this end, both a linear and a quadratic model are computed for each project, taking the size in lines-of-code $S$ as a function of the time in days since the first commit $t$, which is used as project start date, and using one month as time window. Therefore model A is formulated simply as $S_A(t) = a \ast t + b$ and model B as $S_B(t) = a \ast t^2 + t \ast b + c$. The necessary parameters are to be estimated using regression techniques. As a next step, it is necessary to explore whether the growth rate is decreasing over time. This can be done by analysing the second derivative of the quadratic model $S_B(t)'$, or directly the coefficient of the quadratic term $a$.

The sharp distinction between two groups of projects might prove too inflexible. A new group is therefore introduced representing linear growth in contrast to sub- and super-linear rates. This group is defined as those projects having either a better fit for the linear than the quadratic model, or a coefficient of the quadratic term between -0.1 and 0.1, thus being very near to zero. This allows for arriving, for each project and each aspect of interest, at a classification for the evolutionary behavior as being either sub-linear, linear, or super-linear.
Portfolio planning methods have been applied in strategic decision making for over 20 years (Armstrong and Brody, 1994; Wind and Mahajan, 1981), although they have little theoretical support. They are presented in the literature as diagnostic aids and as prescriptive guides for selecting strategic options (Kotler, 1984). The general idea is to classify positions of products along two dimensions to form a matrix: attractiveness of the market and ability of the product to compete within that market, and to derive insights into strategic actions in this way. Managers often neglect to use a rational economic approach, instead applying unstructured judgmental processes. They may base their decisions on power or emotional factors, which might lead to many of their decisions as being irrational. Thus, portfolio planning methods, such as the BCG matrix, may lead managers to make decisions that are less irrational.

Maybe the most well-known portfolio planning method is the Boston Consulting Group (BCG) method (Day, 1986), the most widely used portfolio method in US firms (Armstrong & Brody, 1994). It is based on measuring market attractiveness by market growth rate, and it assesses the firm’s ability to compete by its relative market share. The BCG matrix assumes a causal relationship between market share and profitability. It is based on product life cycle theory that can be used to determine what priorities should be given to different products. To ensure long-term value creation, a company should construct a portfolio using products that contains both high-growth products in need of cash inputs and low-growth products that generate cash. Each of the two axes is normally divided into a high and a low portion, resulting in four different quadrants. Each quadrant is assigned both a catching name and a general strategy (see also Figure 1).

Stars are located in the high growth and high market share area. Normally, the cash flow is rather balanced or even, but a position in stars should be maintained. Cash Cows are placed in low growth area coupled with high market share. Profits and cash generation will generally be high, with relatively small investment due to low growth, translating into a very desirable type. Dogs are placed in

*Figure 1. BCG matrix*
low growth and low market share quadrant, which are normally associated with a de-invest strategy. Question Marks are enjoying high growth but low market share, resulting in demand for cash but low profits. This kind of product over time might turn into either star or dog, so careful considerations is advised to invest or liquidate.

A review of previously published evaluations of the BCG matrix can be found in (Morrison & Wensley, 1991). Actual practical use of the BCG matrix is often found to be inhibited by difficulties in measurement of market growth rates and relative market shares. The results are highly sensitive on these measurements. As a result, different matrix methods are likely to yield different recommendations for the same situation.

OPEN SOURCE MATRIX AND CLASSIFICATION

We propose to adopt the BCG matrix approach for classifying open source projects. There are two main aspects to discuss and decide: The construction of the axes, and results of the classification. For constructing and measuring the axes, we propose to use the results of a growth rate analysis. The growth rate of an open source project is constructed using the growth rate in source code (which equals the software evolution viewpoint of software engineering research) and the growth rate of developers. For market share, we propose to use growth rates of bug reports and downloads. Bug reports normally are associated with usage of a product, especially by interested individuals, but might also signal a product with problems.

As each growth rate of the four types used here is classified in one of three steps, conversion into a single measure needed. We propose to use the mean of both rates, with source code respectively downloads taking priority in ties. Using this approach, an open source project matrix can be constructed, with the standard names having been replaced by possible representative release numbers (see Figure 2). Release numbers are usually coded based on major.minor release, with release 1.0 being a first fully functional release.

Figure 2. Open source project matrix
As can be seen, the classification results in four possible types of projects. This has to be translated into strategies. We need to differentiate between two possible uses, the first one being simple adoption, in which a company or individual wants to decide on which project within a given domain to adopt for a certain task. In the second case, a company wants to build a portfolio of projects. Possible reasons for this include a business model based on a range of software, cooperation with the open source world within a given area, or an application in marketing. Also a company that might pursue a development based on open source, but wants to keep open to several projects might pursue this idea. This leads to the following strategies assigned to the different types of projects.

1.0 (Question Marks) projects currently enjoy huge growth in size and participants, but have not achieved widespread adoption yet. Therefore they might become quite successful, but might fail. For a separate adoption decision, these projects pose considerable risk, while adding 1.0 projects to a portfolio might be interesting.

2.56 (Stars) projects enjoy both considerable growth and adoption. This makes them interesting candidates for any portfolio selection decision, and candidates for a singular adoption consideration.

5.13 (Cash Cows) projects have somewhat stabilized in their code and developer growth, but have achieved widespread adoption. This means that normally a mature solution has been found, with less emphasis on introducing new functionalities. This makes these projects prime candidates for consideration for a single adoption decision, and an interesting candidate for portfolio selection. On the other hand, there might be the need for maintenance at a later stage, maybe due to technological changes, but the community is not that active any more.

0.9 (Dogs) projects have neither huge growth nor adoption, meaning that they prove to be of interest to neither adoption or portfolio considerations. These projects could be termed as failed.

CONCLUSION AND FUTURE RESEARCH

This preface discussed the evaluation and selection of free and open source software projects. It also summarized an approach for constructing a top-level open source project classification based on growth rate analyses of several project aspects, and using portfolio planning techniques, especially the BCG matrix. Given current efforts of constructing the databases necessary to support the kind of analyses that form the basis of this approach, pursuing this road seems worthwhile. Portfolio planning approaches have been in business use for several decades, and despite some shortcomings, have provided value and are in wide-spread use. Given that software adoption decisions on organisational levels are often decisions made on a management level, adopting these common approaches might prove beneficial for communication between technical evaluation level and decision authorities.

In future research, an empirical evaluation of all possible approaches would be very important. This would include performing the necessary analyses based on a database, and presenting the results of categorization to decision-makers. The different approaches like OpenBQR or BRR etc. could then be evaluated according to effort needed, ease of use, understandability, or comprehensiveness. For the portfolio approach presented here, there are several possible ways of refinement of the approach that could be pursued: On the one hand, the construction of the axes could be discussed by adding or substituting aspects of projects or communities. On the other hand, there are other portfolio planning approaches besides the BCG matrix that could be explored.

Stefan Koch
Bogazici University, Turkey
REFERENCES


