Understanding Software Evolution: The Maisqual Ant Data Set

Boris Baldassari
SQuORING Technologies
Toulouse, France
boris.baldassari@squoring.com

Philippe Preux
SequeL, LIFL, CNRS, INRIA
Université de Lille, France
philippe.preux@univ-lille3.fr

ABSTRACT

Software engineering is a maturing discipline which has seen many drastic advances in the last years. However, some studies still point to the lack of rigorous and mathematically grounded methods to raise the field to a new emerging science, with proper and reproducible foundations to build upon. Indeed, mathematicians and statisticians do not necessarily have software engineering knowledge, while software engineers and practitioners do not necessarily have a mathematical background.

The Maisqual research project intends to fill the gap between both fields by proposing a controlled and peer-reviewed data set series ready to use and study. These data sets feature metrics from different repositories, from source code to mail activity and configuration management meta data. Metrics are described and commented, and all the steps followed for their extraction and treatment are described with contextual information about the data and its meaning.

This article introduces the Apache Ant weekly data set, featuring 636 extracts of the project over 12 years at different levels of artefacts – application, files, functions. By associating community and process related information to code extracts, this data set unveils interesting perspectives on the evolution of one of the great success stories of open source.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics

General Terms

Measurement

Keywords

Data mining, Software Engineering, Software Metrics

MSR'14, May 31 – June 1, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2863-0/14/05...$15.00
http://dx.doi.org/10.1145/2597073.2597136

1. INTRODUCTION

In the last 30 years, software has become ubiquitous both in the industry (from Internet to business intelligence to supply chain automation) and in our daily lives. As a consequence, characteristics of software such as reliability, performance or maintainability have become increasingly important – either for stakeholders, developers, or end-users. Research on software metrics was introduced a long time ago to help and support the field, but despite some remarkable advances there are still many critics (most notably from Fenton [5] and Kaner [8]) as to the scientific approach and overall mathematical rigour needed to build scientific methods.

Most of the studies that cover software engineering concerns use their own retrieval process and work on unpublished, non-verifiable data. This fact definitely influences the credibility of studies, and lends credence to criticism about the relevance, reproducibility and usability of their conclusions. As an alternative, authors may rely on public data sets like the Comets, Helix or Promise series. The proposed data set intends to establish a bedrock for upcoming studies by providing a consistent and peer-reviewed set of measures associated to various and unusual characteristics extracted from heterogeneous sources: mails, configuration management and coding rules. It provides a set of metrics gathered on a long-running, real-world project, and states their definitions and requirements. It is designed to be easily imported to any statistical program.

In Section 2, we describe the structure and contents of the data set, and give information on the project history. Section 3 enumerates the various types of measures retrieved and how they are integrated, and Section 4 lists the coding rules checked on code. Section 5 proposes a few examples of usage for the data set. Finally, we state our on-going and future work regarding these concerns in Section 6.

2. DATA SET DESCRIPTION

2.1 The Ant Project

The early history of Ant begins in the late nineties with the donation of the Tomcat software from Sun to Apache. From a specific build tool, it evolved steadily through Tomcat contributions to be more generic and usable. James Duncan Davidson announced the creation of the Ant project on the 13 January 2000, with its own mailing lists, source repository and issue tracking.

There have been many versions since then: 8 major releases and 15 updates (minor releases). The data set ends in

Table 1: Major releases of Ant.

Date	Version	SLOC	Files	Functions
2000-07-18	1.1	9671	87	876
2000-10-24	1.2	18864	171	1809
2001-03-02	1.3	33347	385	3332
2001-09-03	1.4	43599	425	4277
2002-07-15	1.5	72315	716	6782
2003-12-18	1.6	97925	906	9453
2006-12-19	1.7	115973	1113	12036
2010-02-08	1.8	126230	1173	12964

July 2012, and the last version officially released at that time is 1.8.4. Table 1 lists major releases of Ant with some characteristics of official builds as published. It should be noted that these characteristics may show inconsistencies with the data set, since the build process extracts and transforms a subset of the actual repository content.

Ant is arguably one of the most relevant examples of a successful open source project: from 2000 to 2003, the project attracted more than 30 developers whose efforts contributed to nominations for awards and to its recognition as a reliable, extendable and well-supported build standard for both industry and the open source community.

An interesting aspect of the Ant project is the amount of information available on the lifespan of a project: from its early beginnings in 2000, activity had its climax around 2002-2003 and then decreased steadily. Although the project is actively maintained and still brings regular releases the list of new features is decreasing with the years. It is still hosted by the Apache Foundation, which is known to have a high interest in software product and process quality.

2.2 Structure of the Data Set

The complete data set is a consistent mix of different levels of information corresponding to the application, files and functions artefact types. Hence three different subsets are provided. The **application** data set has 159 measures composed of 66 metrics and 93 rules extracted from source code, configuration management and communication channels. Each record is a snapshot of the application. The **files** data set has 123 measures composed of 30 metrics and 93 rules extracted from source code and configuration management. Each record is a Java file with the .java extension. The **functions** data set has 117 measures composed of 24 metrics and 93 rules extracted from source code only, of which each record is a Java function with its arguments.

Table 2: Sizing information for the CSV data sets.

	App	File	Func
Size of flat files	312KB	232MB	2.4GB
Size of compressed files	68KB	12MB	89MB
Number of records	636	654 696	6 887 473

Each data set is composed of 636 exports of the Ant project, extracted on the Monday of every week since the beginning of the project until end of July, 2012. The format of the files is plain text CSV and the separator used for all data sets is `!` (exclamation mark). Some key sizing information is provided in table 2.

2.3 Retrieval Process

Data is retrieved from the project's official repositories:

- Source code is extracted from the Subversion repository's trunk at specific dates. Only files with a `.java` extension have been analysed. *Code metrics* are computed using SQuORE [2], and *rules violations* are extracted from SQuORE, Checkstyle [9, 3] and PMD [1, 4].
- Configuration management metadata is extracted from Subversion's `svn log -v` command executed on trunk and parsed with custom scripts.
- Communication measures are computed from the mailing lists' archives in `mbox` format.

To ensure consistency between all artefact measures we rely on SQuORE, a professional tool for software project quality evaluation and business intelligence [2]. It features a parser, which builds a tree of artefacts (application, files, functions) and an engine that associates measures to each node and aggregates data to upper levels. Users can write custom parsers to analyse specific types of data sources, which we did for the analysis of configuration management and communication channels.

3. METRICS

The measures presented here are intended as real-world data: although they have been cross-checked for errors or inconsistencies, no transformation has been applied on values. As an example, the evolution of line counting metrics shows a huge peak around the beginning of 2002 due to some configuration management large-scale actions which impacted many metrics. We deliberately kept raw data because this *was* actually the state of the subversion repository at that time.

Migrations between tools often make it difficult to rely on a continuous measure of the characteristics. Some information of the former repository may not have been included or wrongly migrated, and the meaning of metadata may have heavily differed depending on the tool. An example of such issues lies in erroneous dates of migrated code in the new configuration management system.

Another point is there may be a huge difference between the source code of an official release and the actual configuration management state at the release time. The build and release process often extracts and packages source code, skipping some files and delivering other (potentially dynamically generated) information and artefacts. As an example, the common metrics shown in table 1 for Ant official releases cannot be confirmed by the repository information available in the data set.

The set of metrics for each artefact type is shown in tables 3, 4 and 5. Some metrics are available only at specific levels – *e.g.* distinct number of operands in a function, while others are available on more than one level – *e.g.* SCM Commits. Please note that the relationship between levels varies: summing line counts on files gives the line count at the application level, which is not true for commits – since a commit often includes several files. Practioners should check ecological inference[11] to reduce bias when playing with the different levels of information.

3.1 Source Code

Most source code measures are borrowed from the literature: artefact- and line-counting metrics have their usual definition[1], VG is from McCabe [10], DOPD, DOPT, TOPD, TOPT are from Halstead [7]. LADD, LMOD and LREM are differential measures that respectively count the number of lines added, modified and removed since last analysis.

Some of the metrics considered have computational relationships among themselves. They are:

$$\begin{aligned}
\textsc{sloc} &= \textsc{eloc} + \textsc{brac} \\
\textsc{lc} &= \textsc{sloc} + \textsc{blan} + \textsc{cloc} - \textsc{mloc} \\
\textsc{lc} &= (\textsc{eloc} + \textsc{brac}) + \textsc{blan} + \textsc{cloc} - \textsc{mloc} \\
\textsc{comr} &= ((\textsc{cloc} + \textsc{mloc}) \times 100)/(\textsc{eloc} + \textsc{cloc})
\end{aligned}$$

Table 3: Source code metrics.

Metric name	Mnemo	App	File	Func
Blank lines	BLAN	X	X	X
Braces lines	BRAC	X	X	X
Control flow tokens	CFT	X	X	X
Number of classes	CLAS	X	X	X
Comment lines of code	CLOC	X	X	X
Comment rate	COMR	X	X	X
Depth of Inheritance Tree	DITM	X		
Distinct operands	DOPD			X
Distinct operators	DOPT			X
Effective lines of code	ELOC	X	X	X
Number of files	FILE	X		
Number of functions	FUNC	X	X	
Lines added	LADD	X	X	X
Line count	LC	X	X	X
Lines modified	LMOD	X	X	X
Lines removed	LREM	X	X	X
Mixed lines of code	MLOC	X	X	X
Non Conformities	NCC	X	X	X
Maximum nesting	NEST			X
Number of parameters	NOP			X
Number of paths	NPAT			X
Acquired practices	ROKR	X	X	X
Source lines of code	SLOC	X	X	X
Number of statements	STAT	X	X	X
Number of operands	TOPD			X
Number of operators	TOPT			X
Cyclomatic number	VG	X	X	X

3.2 Configuration Management

All modern configuration management tools propose a log retrieval facility to dig into a file history. In order to work on several different projects, we needed to go one step further and define a limited set of basic information that we could extract from all major tools (*e.g.* CVS, Subversion, Git): number of commits (SCM_COMMITS), committers (SCM_COMMITTERS), and files committed (SCM_COMMITS_FILES, including any file type and directories). The SCM_FIXES measure counts the number of commits that have one of *fix*, *issue*, *problem* or *error* keywords in their associated commit message.

Table 4: SCM metrics.

Metric name	Mnemo (SCM_*	App	File	Func
SCM Fixes	FIXES	X	X	
SCM Commits	COMMITS	X	X	
SCM Committers	COMMITTERS	X	X	
SCM Commit. Files	COMMIT_FILES	X		

Measures are proposed in four time frames, by counting events that occured during last week (SCM_*_1W), during last month (SCM_*_1M), during last three months (SCM_*_3M), and since the beginning of the project (SCM_*_TOTAL). These enable users to better grasp recent variations in the measures, and give different perspectives on its evolution. Configuration management metrics are listed in table 4.

3.3 Communication Channels

Open-source projects usually have at least two mailing lists: one for technical questions about the product's development itself (*i.e.* the developer mailing list) and another one for questions relative to the product's usage (*i.e.* the user mailing list). Historical data was extracted from old mbox archives, all of which are available on the project web site.

Table 5: Communication metrics.

Metric name	App	File	Func
Number of authors in developer ML	X		
Median response time in developer ML	X		
Volume of mails in developer ML	X		
Number of threads in developer ML	X		
Number of authors in user ML	X		
Median response time in user ML	X		
Volume of mails in user ML	X		
Number of threads in user ML	X		

Available measures are the number of distinct authors[2], the volume of mails exchanged on the mailing list, the number of different threads, and the median time between a question and its first answer on the considered time frame. These measures are proposed in three time frames spanning on last week (COM_*_1W), last month (COM_*_1M) and the last three months (COM_*_3M) of activity. Communication metrics are listed in table 5.

4. RULES

Rules are associated to coding conventions and practices. They deliver substantial information on the local customs in use in the project, and are usually linked to specific characteristics of quality. In the data sets, conformity to rules is displayed as a number of violations of the rule (nonconformity count or NCC) for the given artefact.

Many of these rules are linked to coding conventions published by standardisation organisms like CERT (Carnegie

[1] A complete definition of the metrics is available on the Maisqual web site: maisqual.squoring.com/wiki/-index.php/Ant_data_set.

[2] Distinct authors have different email adresses; no persona unification has been achieved on the data set.

Mellon's secure coding instance), which usually give a ranking on the remediation cost and the severity of the rule. There are also language-specific coding conventions, as is the case with Sun's coding conventions for the Java programming language [12].

4.1 SQuORE Rules

We identified 21 rules from SQuORE 2013-C, targeting the most common and harmful coding errors. Examples of checked rules include fall-through in switch cases, missing default, backwards goto, and assignment in condition. The following families of rules are defined: fault tolerance (2 rules), analysability (7 rules), maturity (1 rule), stability (10 rules), changeability (12 rules) and testability (13 rules). The full rule set is described on the Maisqual project wiki[3].

4.2 Checkstyle Rules

We identified 39 rules from the Checkstyle 5.6 rule set, corresponding to useful practices generally well adopted by the community. The quality attributes impacted by these rules are: analysability (23 rules), reusability (11 rules), reliability (5 rules), efficiency (5 rules), testability (3 rules), robustness (2 rules) and portability (1 rule). All rules are described on the Checkstyle web site[4].

4.3 PMD Rules

We selected 58 rules from the PMD 5.0.5 rule set. These are related to the following quality attributes: analysability (26 rules), maturity (31 rules), testability (13 rules), changeability (5 rules), and efficiency (5 rules). The full rule set is documented on the PMD web site[5].

5. POSSIBLE USES OF THE DATA SET

The introduction of data mining techniques in software engineering is quite young, and there is still a vast field of possibilities to explore. Data may be analysed from an evolutional or static perspective by considering either a time range or a single version. Since the different levels of data (application, file and function) are altogether consistent, one may as well consider studying relationships between them, or even the evolution of these relationships with time.

Communication and configuration management metrics give precious insights into the community's activity and process related behaviour in development. Time-related measures (LADD, LMOD, LREM, *_1W, *_1M, *_3M) are useful to grasp the dynamics of the project's evolution. Since rule violations are representative of coding practices, one may consider the links between the development practices and their impact on attributes of software.

The Maisqual project itself investigates the application of statistical techniques to software engineering data and relies on this data set series. Examples of usages are provided on the Maisqual web site, including evolution of metrics with time (*e.g.* time series analysis), analysis of coding rule violations, and basic exploration of a single version of the project (*e.g.* clustering of artefacts, principal components analysis).

We recommend the use of literate analysis tools like Sweave [6] and Knitr [13], which allow practitioners to embed R code chunks into LaTeX documents. These dynamic documents can then be safely applied to many sets of data with a similar structure to easily reproduce results of an analysis on a large amount of data. Another added value of literate data analysis is to situate results in a semantic context, thus helping practitioners and end-users to understand both the computation and its results.

6. SUMMARY AND FUTURE WORK

The contribution of the data set presented here is twofold: firstly the long time range (12 years) spans from the very beginning of the project to an apogee of activity and to a stable state of maturity. Secondly, the introduction of unusal metrics (rule violations, configuration management, mailing lists) at different levels opens new perspectives on the evolution of software, the dynamics of its community, the coding and configuration management practices.

Next versions of this data set will include new metrics, gathered on new sources (*e.g.* bug tracking system), with new data types (*e.g.* boolean, categorical) to foster usage of different algorithms working on non-numerical data. New projects will also be added from the open-source community (GCC, JMeter).

7. REFERENCES

[1] N. Ayewah, W. Pugh, D. Hovemeyer, J. D. Morgenthaler, and J. Penix. Using static analysis to find bugs. *IEEE Software*, 25(5):22–29, 2008.

[2] B. Baldassari. SQuORE: A new approach to software project quality measurement. In *International Conference on Software & Systems Engineering and their Applications*, Paris, France, 2012.

[3] O. Burn. Checkstyle, 2001.

[4] D. Dixon-Peugh. PMD, 2003.

[5] N. Fenton. Software Measurement: a Necessary Scientific Basis. *IEEE Transactions on Software Engineering*, 20(3):199–206, Mar. 1994.

[6] Friedrich Leisch. Sweave. Dynamic generation of statistical reports using literate data analysis. Technical Report 69, SFB Adaptive Information Systems and Modelling in Economics and Management Science, Vienna, 2002.

[7] M. H. Halstead. *Elements of Software Science.* Elsevier Science Inc., 1977.

[8] C. Kaner and W. P. Bond. Software engineering metrics: What do they measure and how do we know? In *10th International Software Metrics Symposium, METRICS 2004*, pages 1–12, 2004.

[9] P. Louridas. Static Code Analysis. *IEEE Software*, 23(4):58–61, 2006.

[10] T. McCabe. A complexity measure. *Software Engineering, IEEE Transactions on*, (4):308–320, 1976.

[11] D. Posnett, V. Filkov, and P. Devanbu. Ecological inference in empirical software engineering. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, pages 362–371. IEEE Computer Society, Nov. 2011.

[12] Sun. Code Conventions for the Java Programming Language. Technical report, 1999.

[13] Y. Xie. Knitr: A general-purpose package for dynamic report generation in R. Technical report, 2013.

[3] See http://maisqual.squoring.com/wiki/index.php/Rules.

[4] See http://checkstyle.sourceforge.net/config.html.

[5] See http://pmd.sourceforge.net/pmd-5.0.5/rules/.