Non-invasive collection of software metrics: some issues and experiences

Marco Scotto, Alberto Sillitti, Giancarlo Succi, and Tullio Vernazza

Abstract

The configuration of a metrics collection tool often represents a significant obstacle in the adoption of a measurement program inside a software company. This paper describes the main issues encountered in establishing a measurement program inside some Italian small and medium enterprises (SMEs). In particular, the discussion focuses on the issues related to the automatic collection of product metrics. Mainly, such issues depend on two major factors: the evolution of programming languages and the software configuration. The paper also proposes a solution based on the package-oriented software engineering paradigm.

1 Introduction

Many times prior literature has highlighted the importance of non-invasive and automatic data collection procedures. In particular, Pfleeger [18] discusses in detail the importance of smooth and non-intrusive data collection procedures in determining the success of a metrics program. Daskalantonakis [4], Offen and Jeffrey [16] also make similar points. Optimized data collection procedures improve metrics programs in two ways. First, they do not significantly reduce programmers' productivity, which typically creates resistance and reluctance to establish metrics programs. Second, they increase the accuracy and reliability of the data, thereby increasing the confidence of managers who use this information. Such an approach is partially inspired by the Lean Management [25], a management technique devised initially by Toyota [17], in which the measurement phase of the process is totally integrated with the process itself and it requires a limited human effort. However, there is still lack of tools and techniques able to introduce, in a systematic and efficient way, a system for measuring and controlling the software product [19]. This paper describes the main issues encountered in establishing a measurement program inside some Italian small and medium enterprises (SMEs). Mainly, the discussion focuses on the issues related to the automatic

* Corresponding author. Tel.: +39 010 353 2183; fax: +39 010 353 2154. E-mail address: scotto@dbs.unige.it (M. Scotto).
collection of product metrics. The article is organized as follows: section 2 introduces the different kind of product metrics; section 3 introduces PROM (PRO Metrics), an automatic tool for collecting both process and product metrics; section 4 describes the main issues related to the automatic collection of product metrics; section 5 presents the approach we use to address the collection of product metrics. Finally, section 6 draws the conclusions.

2 Product metrics

Product metrics [6] measure any artifact or document that derives from the software development process. Products are not restricted but the management is committed to deliver to the customer. Any artifact or document produced during the software life cycle can be measured. There are two kinds of product metrics:

1. **Dynamic metrics**, which are collected measuring a running program;

2. **Static metrics**, which are collected measuring system representations such as design, source code, or documentation.

In the literature, there are many proposal of product metrics that match different programming languages and paradigms [7, 8, 12, 13]. This paper only focuses on the issues related to the collection of static metrics, in particular it deals with source code metrics collected directly by parsing and analyzing the source code of a software project.

3 PROM (PRO Metrics)

PROM [21] is a distributed architecture designed to collect different set of software data: process metrics and product metrics. Process metrics include Personal Software Process data [9] and ad-hoc metrics. Product metrics include both object-oriented metrics [3] and procedural metrics [7, 8, 13].

PROM provides plug-ins for mass-market applications such as office automation, popular IDEs, etc. In this way, process metrics collection is completely transparent and users do not have to learn new applications. The architecture design fullfills three main requirements:

1. the architecture should be extensible to support new IDEs, new programming languages, kind of data, and analysis tools;

2. IDE dependent plugins should be as simple as possible;

3. developers should be able to work also off-line.

The PROM core is completely written in Java using open source technologies and standard protocols such as XML and Web Services related protocols [22].
4 Metrics collection issues

Some high level product metrics can be extracted before the development phase, for instance, the Chidamber and Kemerer's metrics suite [3] can be calculated directly from UML class diagram. In many cases, product metrics are calculated from the source code of a software project. The automatic collection of product metrics from source code presents some issues that mainly depend on two factors:

- the evolution of programming languages
- the configuration of metrics collection tools.

Standard languages, such as C, are not likely to change but others, like Java, C#, are subject to change quite often. For instance, the last version of Java (Tiger) has introduced the support to generic types [10]. This means that language parsers for extracting product metrics have to be updated accordingly. Moreover, the configuration of product metrics collection tools metric introduces many difficulties. If, in fact, as it happens in most cases, product metrics are extracted directly from the source code, this requires an accurate and complete configuration in order to set up the metrics collection system. For example, considering C and C++, essential information needed to extract product metrics are contained in header files, whose content could depend on the system configuration, this also requires the presence of preprocessing techniques. If we consider object-oriented languages such as Smalltalk, C++, Java, C#, and object-oriented extensions of scripting languages, the necessary information can be even distributed in the whole class hierarchy. In these cases, techniques such as reflection or run time type information are the only way to calculate some product metrics.

5 Our approach

This paragraph describes the approach used to address the issues about the configuration of a metrics collection tool. The basic idea is to obtain the configuration parameters in a non-invasive way, in particular, we want to develop specific components or plug-ins that collect information regarding the configuration of software projects in an automatic or semi-automatic way. This solution is based on package-oriented programming (POP) [23], which is a specialisation of component-based software development [24]. Basically, POP treats regular mass-market applications as large components taking advantage of the functionalities that they offer and user familiarity that comes with them. The configuration of a software project includes the entire set of libraries, components, header files, and settings needed to build the software project itself. For this purpose, we intend to develop specific plug-ins for both commercial and open-source development environments. This includes the integration with the most popular IDEs such as Eclipse [5], Netbeans [15], Microsoft Visual Studio [14], and Borland JBuilder [2].
Figure 1: Proposed architecture

Figure 1 shows the resulting architecture, the central element is *PROM Product Manager* (*PROM PM*), a tool that is able to track source code evolution by using a CVS repository. When a piece of code is stored into a version control system, the tool invokes the software metrics extractor (*WebMetrics*) that collects metrics data and stores them into the *PROM Database*. This feature provides a comprehensive view of the source code evolution throughout the development process. The architecture includes the following main components:

1. *PROM Database*: it stores all acquired product metrics. The implementation is based on the open-source DBMS PostgreSQL.

2. *PROM Server*: it provides an interface to the *PROM Database* through high-level commands exposed as SOAP services [22]. This interface hides completely the DBMS and the low-level data model. The implementation is based on the open-source SOAP engine Axis [1].

3. *PROM Product Manager* (*PROM PM*): this component orchestrates the whole process of product metrics acquisition. In particular, it checks out the last version of source code from the CVS repository, it invokes *WebMetrics* in order to calculate product metrics, and, finally, it uploads metrics to the *PROM Database* using the SOAP interface exposed by the *PROM Server*. 
The communication between the PROM Server and PROM PM uses the HTTPS protocol in order to guarantee confidentiality on metrics collected.

4. **IDE plugins**: these components represent the interface between the development environment and the metrics collection tool. In particular, they extract information about the configuration parameters of a software project. For instance, if we consider a project written in Java language, they provide to PROM PM parameters such as the classpath and command-line arguments needed to the Java compiler to build the project, etc.

5. **WebMetrics**: WebMetrics [20] is an automatic tool for product metrics collection. Metrics can be extracted from different source languages: C/C++, C#, Java and more to come. The system is completely written in Java and parsers have been developed using JavaCC [11]. The metrics collection process requires that all code must be syntactically correct and all include files or libraries used must be present.

6 Conclusions

This paper discussed the main issues related to the automatic collection of product metrics. The result is that metrics collection tools, to be effective, should be highly integrated with the development environment used by a development team. In fact, experience demonstrates that rarely programmers keep metrics tools in sync with the configuration of the project under development. Moreover, we plan to to integrate our architecture with tools popular in “agile” and open-source contexts. In particular, the integration with the following tools is planned:

1. Apache Ant (http://ant.apache.org)
2. Apache Maven (http://maven.apache.org)
3. CruiseControl (http://cruisecontrol.sourceforge.net)
4. CruiseControl.NET (http://ccnet.thoughtworks.com)

7 Acknowledgement

This work was partially supported by MAPS (Agile Methodologies for Software Production) research project, contract/grant sponsor: FIRB research fund of MIUR, contract/grant number: RBNE01JRK8.
References

[10] Java 1.5 (Tiger) - web site: http://java.sun.com/j2se/1.5.0/index.jsp


[22] Simple Object Access Protocol (SOAP) specification - web site: www.w3.org/TR/soap/


Marco Scotto  
DIST - Università di Genova  
Via Opera Pia, 13  
I-16145 Genova, Italy  
E-mail: scotto@dist.unige.it

Alberto Sillitti  
Free University of Bolzano/Bozen  
Piazza Domenicani, 3  
I-39100 Bolzano  
E-mail: Alberto.Sillitti@unibz.it

Giancarlo Succi  
Free University of Bolzano/Bozen  
Piazza Domenicani, 3  
I-39100 Bolzano  
E-mail: Giancarlo.Succi@unibz.it

Tullio Vernazza  
DIST - Università di Genova  
Via Opera Pia, 13  
I-16145 Genova, Italy  
E-mail: tullio@dist.unige.it