Collecting, Integrating and Analyzing Software Metrics and Personal Software Process Data

Alberto Sillitti
DIST – Università di Genova
alberto@dist.unige.it

Andrea Janes
Libera Università di Bolzano
Andrea.Janes@unbz.it

Giancarlo Succi
Libera Università di Bolzano
Giancarlo.Succi@unbz.it

Tullio Vernazza
DIST – Università di Genova
tullio@dist.unige.it

Abstract

Measures represent important data in all engineering disciplines. This data allows engineers to understand how things work and how to make changes to produce desired results. In software engineering, it is difficult to collect useful measures because developers do not consider it an important activity, compared to coding. Moreover, manual collected data is often affected by errors, making it unusable.

The shortage of automated tools for collecting and analyzing measures does not contribute to the evolution of software engineering.

This paper presents PROM (PRO Metrics), an automated tool for collecting and analyzing software metrics as well as personal software process (PSP) data. The tool uses an architecture based on plug-ins that automatically collects data from development tools.

1. Introduction

There are two kinds of measures in engineering: product metrics and process metrics. The former describes product's qualities such as dimensions, physical data, etc; the latter describes process' qualities such as effort required, production time, etc. In software, the former includes code length, complexity, reusability, maintainability, etc. [4]; the latter includes editing time, number and type of changes in a class or in a file, etc. [7]

At present, tools for retrieving such data in software exist but require some human effort and/or collect only one of the sets previously described. Moreover, these tools allow data acquisition but they do not provide any advanced and integrated analysis.

In software engineering, measures are difficult to collect due to two main problems [3] [9]:

1. collecting metrics is a time expensive task. This is a problem because software projects are often late and there is no time to spend in activities that do not produce immediate benefits
2. manual data collection is an unreliable activity. Too many errors or missing data badly affect the analysis process. These errors appear mostly in critical periods, when the data correctness should help to understand better the situation such as during high stress working periods.

Correlation between such acquired data and its meaning in terms of product's costs, time of delivery, maintenance costs, etc. are not well known. The analysis of such correlation is difficult also because the low quality of data that is often acquired manually by programmers. A completely automated acquisition tool should be able to both improve data quality and reduce to zero the acquisition effort done by programmers.

Software development is an activity mainly based on human effort. For this reason, costs highly depend on the time spent doing each activity. Tracing the time spent in each activity is useful to implement accounting methodologies like activity-based costing (ABC) in such environment [2].

This paper presents PROM (PRO Metrics), an automated tool for collecting and analyzing software metrics and PSP data. The paper is organized as follows: section 2 describes the importance of software metrics and personal software process data to make software engineering a true engineering discipline; section 3 presents the PROM architecture; section 4 analyzes data and communication models used in PROM; section 5 presents use scenarios; section 6 presents similar tools and focuses on the comparison of PROM and Hackystat; finally, section 7 draws the conclusions.

2. Product and process metrics

Software engineering is very different from other engineering disciplines because there are no consolidated practices and, sometimes, practices that work in one context do not work in another, causing budget, schedule, and quality problems.

There is no standard way to measure and associate a meaning to both code and process.

Proceedings of the 29th EUROMICRO Conference "New Waves in System Architecture" (EUROMICRO'03)
1089-6503/03 $17.00 © 2003 IEEE

Engineers need to measure relevant variables of a process to understand and control it. This also happens in software engineering. The most used resources in software companies are human resources. For this reason, software engineers are mainly interested in: human effort needed to build a product, quality, easiness of maintenance, cost, and time required.

Most of software development costs are human resources costs: experience, skills, etc. Moreover, the productivity of very good programmers is ten times better than average [11].

For these reasons, it is very important to understand how top developers work and to encourage all developers to adopt a process that helps them to achieve the best possible results.

The Personal Software Process (PSP) [7] provides a rigorous methodology to monitor software development from the early stages of a project to the shipped product. This helps software engineers to keep track of their work that than can be easily analyzed at any development stage. This continuous monitoring helps developer to trace their own performances, compare them to the planned schedule, and find out whether specific elements (i.e. environment, behaviors, interactions, etc.) affect them. Identification of inefficiencies is the first step to improve productivity as stated in lean production environments [12] [17].

PSP requires the collection of detailed metrics of the development time, bugs discovered and corrected at all development stages, as well as software size. Then all collected data is analyzed statistically. These results provide software engineers sets of historical data mainly used to:

1. make reliable estimates on variables such as time schedule, quality, etc. of on going projects
2. find out how to improve the development process
3. highlighting problems

Hackystat is a framework that allows individuals to collect and analyze PSP data automatically [8].

PROM is designed to help both developers and managers to keep projects under control. To achieve this goal, the system provides two kinds of reports to users basing on their role inside the project.

Developers can access only their own data including software metrics, PSP data and analysis results. This information helps them to highlight their inefficiencies and improve their approach to software development. A developer can grant access to his personal data to another user to help him to achieve the best results as possible.

The data collection procedure is completely automated to address the two main problems in acquiring data discussed in section 1. Moreover, the process is transparent for users that do not require interrupting their main activity to collect data solving the *context switching* problem identified by Johnson *et al.* [9].

Many companies are adopting an accounting method called Activity-Based Costing (ABC) to manage costs. It is very difficult to apply such methodology in software development because human activities are hard to track. PROM helps managers in applying ABC collecting data. It performs automatically tracking for each project collecting data for both developers and managers that use the system.

Managers cannot access single developer's data due to privacy issues but they can access to the same information in an aggregated form that summarize the status of the whole project. This approach helps manager to focus on important data without looking at too many details that are not useful for management purpose and affect developer's privacy. Usually, managers are interested in costs and schedules.

2.1. Hackystat

Hackystat is the third generation of PSP data collection tools developed at University of Hawaii. That framework allows developers to collect process data and analyze them automatically. The system collects data through *sensors* attached to development tools that communicate with a centralized server using the SOAP protocol [14].

The tool focuses on individual data collection and individual use of such data. This structure allows a high level of privacy, very important inside industries, but limiting advantages for the workgroup and for the entire company.

The architecture of Hackystat is very lightweight and it is completely based on open-source components and standard protocols such as Apache Tomcat, XML, SOAP, etc. It does not use any back-end database; it uses XML files to store information [6] [18].

3. Architecture of PROM

PROM is tool for automated data acquisition and analysis that collect both code and process measures. The tool focuses on a comprehensive data acquisition and analysis to provide elements to improve products. Collected data include a wide range of metrics including all PSP metrics, procedural and object oriented metrics, ad-hoc developed metrics to trace activities rather than coding such as writing documents with a word processor.

The tool collects and analyzes data at different levels of granularity: personal, workgroup and enterprise. This differentiation takes a picture of the whole software company and preserve developers' privacy providing to managers only aggregated data.

PROM is component-based. In particular, it is based on the Package-Oriented Programming development technique [16]. As an example, the plug-ins approach allows developers to use mass-market development,

design, and documentation tools integrated in the data acquisition process.

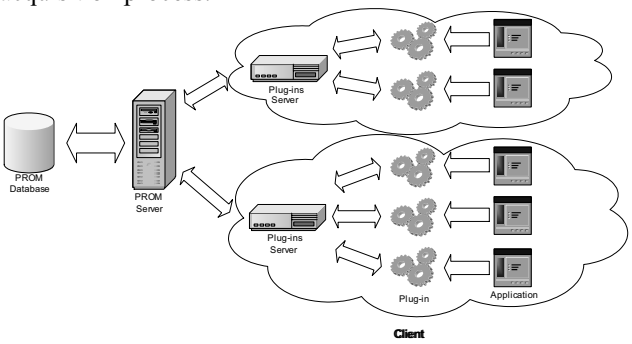

Figure 1: PROM data acquisition architecture

The architecture design has three main constraints:

1. the architecture should be extensible to support new IDEs, kind of data, and analysis tools.
2. IDE dependent plug-ins need to be as simple as possible
3. developers can work off-line

All these main constraints are satisfied in the PROM architecture.

The PROM core is completely written in Java using open source technologies and standard protocols such as XML and SOAP. SOAP is a lightweight protocol based on XML that implements remote procedure calls (RPC) over HTTP channels.

Developers can write plug-ins in any language but they have to communicate to the Plug-ins Server using the SOAP protocol.

PROM exposes a set of functions that are available inside an Intranet or on the Internet as web services. At present, the system is designed to run inside an Intranet or on the Internet through a VPN (Virtual Private Network) because PROM communications are not encrypted. Future versions will implement secure communications.

There are four main components in PROM (Figure 1):

1. *PROM Database*: it stores all acquired data and information on users and projects. 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. This interface hides completely the DBMS and the low-level data model. This server provides all its functionalities through web services. This allows all web services enabled clients (i.e. Microsoft Excel 2002 with web services extensions) to access collected data and perform custom analysis. The implementation is based on the Apache Tomcat application server, to execute Java Servlets, integrated with the Apache Axis SOAP server and on the JDBC technology for the database access.
3. *Plug-ins Server*: it collects data from all plug-ins, makes some pre-processing reducing redundant data and providing a cache. Then it sends the source code to the WebMetrics tool for metrics extraction [15] and collects results (Figure 2). The *Plug-ins Server* sends these results to the *PROM Server* that stores them into the database. This component also provides off-line working features: it collects all data in the cache and sends them to the *PROM Server* when it is connected. This feature allows developers to keep track automatically of their work in any situation, even if they use a laptop. Moreover, the system needs this component to simplify the structure of plug-ins that are application dependant, so each tool supported by PROM needs a new plug-in implementation.

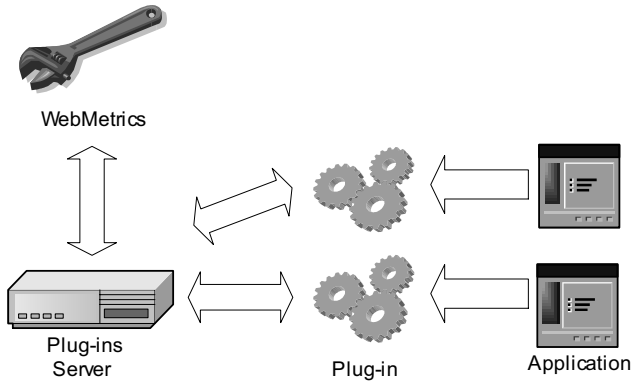

Figure 2: PROM client architecture

4. *Plug-in*: it is an IDE dependant plug-in that listens to application-generated events as file open, file save, edit, etc. It collects and sends all these data to the *Plug-ins Server* adding timestamp information and providing user authentication features to identify users that are working. This feature is different from traditional ones: multiple logins are allowed at the same time to support new development techniques like pair programming (two developers work together on the same machine). *Plug-ins* communicate with the *Plug-ins Server* using the SOAP protocol. This choice helps plug-in developers to find a ready-to-use and compatible communication library because there are many of them (both commercial and open source) available for a wide range of languages and platforms.

The architecture also includes data analysis and visualization (Figure 3). The *PROM Server* performs simple data analysis retrieving data from the database and aggregating them. At present, the system performs analysis regarding individual and workgroup data. These data show human effort spent to write classes and

correlate it with metrics extracted from the same piece of source code to determine its properties. The analysis considers also code evolution keeping track of versions of the source code. This feature provides a comprehensive view of the development process to developers.

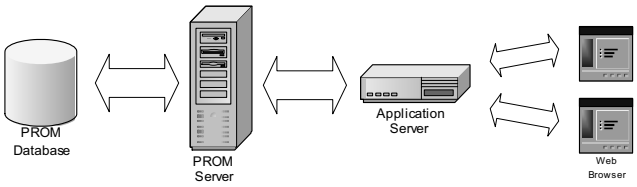

Figure 3: PROM data analysis and visualization architecture

Moreover, managers take advantages of the system accessing aggregated data that is useful to manage a project. The PROM system takes care of developers' privacy not providing to managers data related to a single developer but only project level data.

A more advanced analysis is possible integrating into the system an advanced data analysis tool that can perform exhaustive and time-consuming calculation.

Dynamically generated web pages perform data visualization. The chosen application server is Apache Tomcat: it runs Java servlets that generate, at runtime, both tables and bitmaps containing graphs.

PROM administration is managed through web pages too (Figure 4). Administration includes: user and project management (add, delete, and update users and projects; assign users to projects), event collection administration, system status reports, etc.

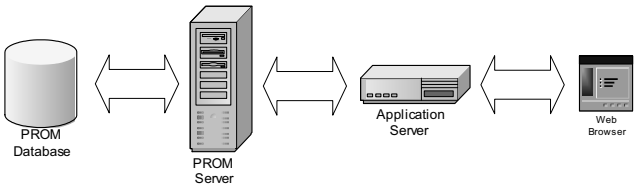

Figure 4: PROM administration architecture

Most of the Plug-ins under development focuses on development IDEs like Microsoft Visual Studio 6 and .NET, NetBeans, Eclipse, and Emacs. Plug-ins for Microsoft Office and Together Central support the design and documentation process as well. Support for other tools (e.g. Rational Rose and JBuilder) is planned.

The tool also supports manual data insertion through a web page. This modality of data collection supports non computer-oriented activities like reading manuals or paper documentation. Manual data collection is a well known error-prone activity but if it is very focused and reduced to just a few items it should be correct enough and then useful. To support this thesis, a monitoring about how developers use this particular feature will be started as soon as the tool will be used in a real environment. However, to prevent result errors, the analysis tool can discard such data.

4. Data and communication models

Data collection model and data analysis model are the two components of the PROM data model.

The former, showed in Figure 5, models the core system. It contains data regarding users, projects, collected events, and metrics. Relations between elements are also stored here. The PROM administration tool affects data contained in this part of the data model, except the *event* and the *metrics* tables. Plug-ins store automatically collected data into these two tables.

The analysis tool builds the latter data model component dynamically (not showed), mainly analyzing the *events* and the *metrics* tables to provide ad hoc views that contain pre-elaborated information. These are not fixed views, but their structure changes according to the specific analysis task. Pre-processed data is further analyzed, and then users access results through an application server that generates HTML reports.

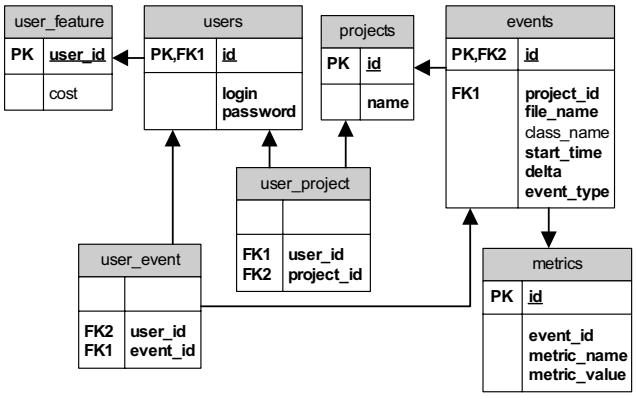

Figure 5: PROM data model overview

PROM components communicate using the SOAP protocol (except database access that uses JDBC). This protocol allows *plug-ins* to communicate with *Plug-ins Servers* and *Plug-ins Servers* with the *PROM Server* (Figure 1).

The *PROM Server* is a SOAP server that accepts connections from *Plug-ins Servers*. Moreover, *Plug-ins Servers* accept connections from *plug-ins*. The system uses the same communication scheme (Figure 6) for both *PROM Server – Plug-ins Servers* communications and *Plug-ins Servers – Plug-ins* communications.

The schema is based on the command design pattern. The two servers expose only one function (*executeCommand*) that accepts a *Command* and returns a *Result*. The *Command* can contain a *Parameter*, both of them are generated at the same time through the

CommandBuilder class that implements the builder design pattern [5].

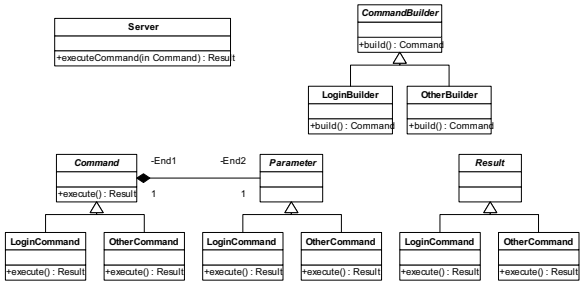

Figure 6: PROM command structure

Generally, the two servers support two different set of commands, but some of them are nearly the same as the login command. In this case, the *Plug-ins Server* works as a proxy for the request.

Moreover, inside PROM there is the core of the *WebMetrics* tool that performs metrics extraction. Language parsers, that extract software metrics and are command-line executable, compose this core. For this reason, the interaction is managed through a file exchange mechanism. The *Plug-ins Server* creates a file containing the source code; then, calls a parser to analyze it; finally, extracts results from the generated file and sends them to the *PROM Server* that stores them into the database.

5. Scenarios of use

PROM provides useful information for both developers and managers.

Developers work with traditional tools integrated with an ad hoc developed plug-in that automatically collects data. Then, the system analyzes such data and makes different views according to role of users.

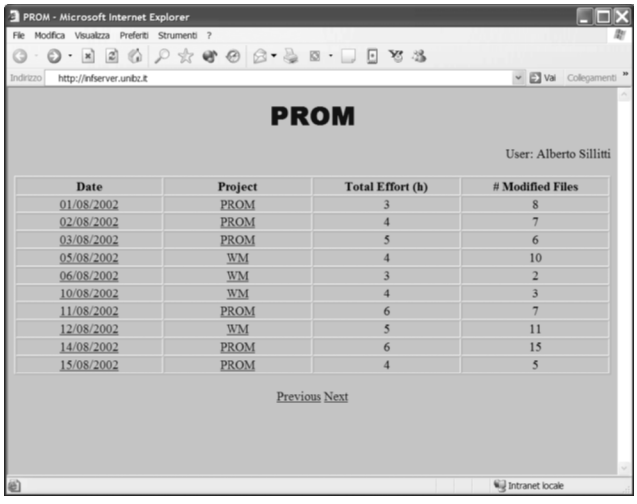

Figure 7: Statistics for a user

PROM users include developers and managers. The former group can access only their own data and statistics to improve personal performances. Figure 7 shows one of the data views available to developers.

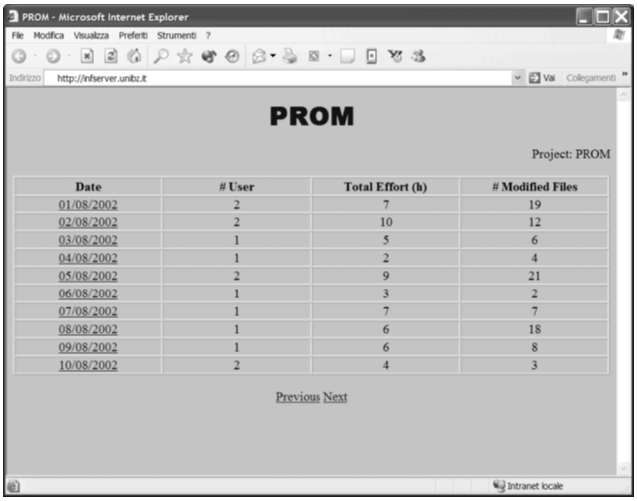

Figure 8: Statistics for a project

The latter group can access only aggregated data at project level. In fact, managers do not need to know projects details do to their job. This choice allows managers to keep under control projects respecting developers' privacy. Figure 8 shows one of the data views available to managers.

The chosen architecture allows both managers and users to maximize benefits of data acquisition. Main benefits include: code and process improvements for developers; easier costs management and control on project evolution for managers.

6. Related tools

There are several commercial tools for metrics collection and analysis such as MetricCenter [10] and ProjectConsole [13] but most of them a domain dependant, use proprietary solutions, and often require some human effort for data collection.

Moreover, metrics data exchange is becoming an important issue. To address this set of problems, metrics exchanging protocols are emerging such as Simple Metric Data Exchange Format (SIMDEF) [1]. At present, these formats are not included in the PROM architecture due to their immaturity but a complete support will be provided as soon as these formats become more stable.

Among all metrics collection tools, Hackystat is the closest to PROM. A detailed analysis is summarized in the section 6.1.

6.1. PROM - Hackystat comparison

Table 1 briefly summarizes the main features of PROM and Hackystat.

Table 1: PROM – Hackystat comparison

Feature	PROM	Hackystat
Supported languages	C/C++, Java, Smalltalk, C# (planned)	Java
Supported IDEs	Eclipse, JBuilder, Visual Studio, Emacs (planned)	Emacs, JBuilder
Supported office automation packages	Microsoft Office, OpenOffice	-
Code Metrics	Procedural, object oriented and reuse	Object oriented
Process Metrics	PSP	PSP
Data aggregation	Views for developers and managers	Views for developers
Data Management	Project oriented	Developer oriented
Business process modeling	Under development	-
Data analysis and visualization	Predefined simple analysis and advanced customized analysis (both in beta)	Predefined simple analysis

Looking at these data the different target of the two applications emerges. The aim of PROM is the collection and analysis of the whole development process including activities not strictly related to coding and target users include all members of the development team (including developers, managers, etc.). Hackystat focuses on the coding activity and target users are developers.

Moreover, different sets of views and data management approaches characterize each tool: PROM offers both detailed data for developers and high-level views, at project level, for managers; on the contrary, Hackystat offers only detailed data for a single developer.

PROM is useful to monitor and improve the whole development process; Hackystat focus on the improvement of the performances of the single developer.

7. Conclusions and future work

This paper proposed a tool for acquiring and analyzing software metrics and PSP data without any human effort.

At present, the core system is working, a small set of plug-ins are available for most popular IDEs and others are under development. At this stage of development, an internal use in software companies is possible.

The current version of the system includes a very simple analysis tool; the integration with party specialized tools will provide a relevant enhancement.

At present, not enough data regarding the use of the system in a production environment is available. According to preliminary data from an internal evaluation, the tool succeeds in collecting data without any human effort but only a wider experiment can prove the real effectiveness of PROM.

To provide a more professional tool rather than an academic exercise, next version of the system will include cryptographic protected communications over the Internet and it will integrate acquired data with a bug-tracking tool such as Bugzilla. This integration will provide a correlation between software metrics and PSP data on one hand and code bugs on the other.

Acknowledgements

The first author thanks Professor Philip Johnson and his research staff for showing and explaining him the Hackystat system and answering his many questions.

References

[1] M. Auer, "Measuring the Whole Software Process: A Simple Metric Data Exchange Format and Protocol, *6th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2002)*, Malaga, Spain, June 2002.

[2] G. Cokins, *Activity-based Cost Management: An Executive's Guide*, John Wiley & Sons, 2001.

[3] A.M. Disney, P.M. Johnson, "Investigating Data Quality Problems in the PSP", *Sixth International Symposium on the Foundations of Software Engineering (SIGSOFT'98)*, Orlando, FL, USA, November 1998.

[4] N.E. Fenton, S.H. Pfleeger, *Software Metrics: a Rigorous and Practical Approach*, Thomson Computer Press, 1994.

[5] E. Gamma, R. Helm, R. Johnson, J. Vlissidies, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.

[6] C.F. Goldfarb, P. Prescod, *The XML Handbook*, 3rd edition, Prentice Hall Computer Books, 2000.

[7] W. Humprey, *A Discipline for Software Engineering*, Addison-Wesley, 1995.

[8] P.M. Johnson, "You can't even ask them to push a button: Toward ubiquitous, developer-centric, empirical software engineering", *The NSF Workshop*

for New Visions for Software Design and Productivity: Research and Applications, Nashville, TN, USA, December 2001.

[9] P.M. Johnson, A.M. Disney, "A critical analysis of PSP data quality: Results from a case study", *Journal of Empirical Software Engineering*, December 1999.

[10] MetricCenter, Distributive Software – website: http://www.distributive.com/

[11] S. McConnell, *Rapid Development: Timing Wild Software Schedules*, Microsoft Press, 1996.

[12] T. Ohno, *Toyota Production System: Beyond Large-Scale Production*, Productivity Press, 1988.

[13] ProjectConsole, Rational Software Corporation – website: http://www.rational.com/

[14] SOAP (Simple Object Access Protocol) – specifications: http://www.w3.org/TR/SOAP/

[15] G. Succi, C. Bonamico, L. Benedicenti, E. Liu, T. Vernazza, R. Wong, "Supporting Electronic Commerce of Software Products through Pay-Per-Use Rental of Downloadable Tools", book chapter in *Internet Commerce and Software Agents: Cases, Technologies and Opportunities* (Rahman, Syed M. and Robert J. Bignall editors), IDEA Group Publishing, 2000.

[16] G. Succi, W. Pedrycz, E. Liu, J. Yip, "Packadge-Oriented Software Engineering: a Generic Architecture", *IEEE IT Pro*, April 2001.

[17] J.P. Womack, D.T. Jones, *Lean Thinking: Banish Waste and Create Wealth in Your Corporation*, Simon & Schuster, 1998.

[18] XML (Extensible Markup Language) 1.0 – specifications: http://www.w3.org/TR/2000/REC-xml-200010