



ELSEVIER

Available at

www.ElsevierComputerScience.com

POWERED BY SCIENCE @ DIRECT®

Journal of Systems Architecture xxx (2003) xxx–xxx

**JOURNAL OF
SYSTEMS
ARCHITECTURE**

www.elsevier.com/locate/sysarc

Measures for mobile users: an architecture

Alberto Sillitti ^{a,*}, Andrea Janes ^b, Giancarlo Succi ^b, Tullio Vernazza ^a

^a Department of Communication, Computer and System Sciences, University of Genoa, Via Opera Pia 13, I-16145 Genoa, Italy

^b Faculty of Computer Science, Free University of Bozen, Piazza Domenicani 3, I-39000 Bolzano, Italy

Received 20 March 2003; received in revised form 18 August 2003; accepted 8 September 2003

Abstract

Software measures are important to evaluate software properties like complexity, reusability, maintainability, effort required, etc. Collecting such data is difficult because of the lack of tools that perform acquisition automatically. It is not possible to implement a manual data collection because it is error prone and very time expensive. Moreover, developers often work in teams and sometimes in different places using laptops. These conditions require tools that collect data automatically, can work offline and merge data from different developers working in the same project. This paper presents PROM (PRO Metrics), a distributed Java based tool designed to collect automatically software measures. This tool uses a distributed architecture based on plug-ins, integrated in most popular development tools, and the SOAP communication protocol.

© 2003 Published by Elsevier B.V.

Keywords: Development monitoring; Process metrics; Product metrics; Java

1. Introduction

It is difficult to extract measures from both software [7] and software development process [11] due to a shortage of focused tools that perform such tasks automatically. In particular, tracking the entire development process manually is time expensive and error prone [5,14]. Moreover, these errors are more frequent when a reliable tracking is more important to understand if and how the

process should be improved. For example, that happens when developers are under pressure because of an approaching deadline.

Metrics data are important to find out the correlation between objective measurable data and software qualities like complexity, reusability, cost of maintenance, effort required, etc.

A completely automated tool that performs such data acquisition without any effort by developers should help both developers and managers to improve software quality and shipping times. The system has to take care of developers' privacy allowing managers and colleagues to access only aggregated data at different levels.

Collected data can help managers to implement a popular accounting technique called activity-base costing (ABC) [3]. Usually, this technique is

* Corresponding author. Tel.: +39-010-3532173; fax: +39-010-3532154.

E-mail addresses: alberto@dist.unige.it (A. Sillitti), andrea.janes@unibz.it (A. Janes), giancarlo.succi@unibz.it (G. Succi), tullio@dist.unige.it (T. Vernazza).

44 difficult to implement in a software company be-
45 cause nearly all costs are human costs and keeping
46 track of time spent in each activity is very difficult.
47 The proposed system provides such data auto-
48 matically.

49 This paper presents the architecture and the
50 implementation of PROM (PRO Metrics), an au-
51 tomated and distributed Java tool for collecting
52 and analyzing software metrics and personal
53 software process (PSP) data. The paper is orga-
54 nized as follows: Section 2 and subsections present
55 an overview of the metrics collection problem;
56 Section 3 describes the state of the art; Section 4
57 presents the architecture of PROM; Section 5 de-
58 scribes the implementation; finally, Section 6
59 draws the conclusions.

60 2. Metrics collection

61 2.1. Metrics overview

62 Software metrics is an area that tries to quantify
63 different aspects of software development. “You
64 cannot control what you cannot measure” [4], this
65 is the main reason to measure software. The goal
66 of software metrics is finding a way to control
67 software project that too often are over time and
68 over budget.

69 Measuring is a process through which numbers
70 or symbols are assigned to entities of the real
71 world in order to describe it [7].

72 A measure is not only required to control, but
73 also to verify, estimate, and help decision makers.
74 Hardly a qualitative approach is enough, without
75 a quantitative approach is difficult to compare
76 experiences, collect results, and perform analysis.

77 Through measures, engineers can evaluate a
78 product that already exists or estimate features of
79 a product that do not exist. In both cases, a model
80 of the features of the artefact under analysis helps
81 to identify relationships between artefact attri-
82 butes. The COCOMO model [1] can be used in this
83 sense. As instance, the equation $E = aS^b$, if E and
84 S are known, can be used to evaluate the position
85 of the current project compared to the average
86 from which the constants a and b are calculated; if
87 S is estimate, than the model can be used to esti-

mate E . A predictive system includes a model and
a set of procedures to predict the unknown pa-
rameters of the model. The COCOMO model in-
clude a model described by the previous equation
and a set of procedures to predict parameters a , b ,
and S .

In an industrial context, software is character-
ized by:

1. product quality, 96
2. process quality. 97

A product is the result artifact of the software
production; a process is the set of activities re-
quired to produce intermediate products and the
final product. In the software context, the meaning
of *quality* is not trivial. The *IEEE Standard for
Glossary of Software Engineering Terminology* [12]
defines quality as the whole set of features of a
product or service able to satisfy a specific need.

Different people are interested in measure and
control different variables.

Managers are interested in: 108

- cost: to determine the price of the product, 109
- productivity: to built a team with the right size, 110
- product quality: to compare different products, 111
- effectiveness of methodologies and tools: to 112
choose the best ones.

Developers are interested in: 114

- product quality: to evaluate the status of a pro- 115
ject,
- process quality: to improve it through ad hoc 117
changes.

The analysis of the measure of several attributes
determines an estimation of the quality of a soft-
ware product [7]. These attributes include: 121

1. internal attributes: features measurable through 122
a static analysis of the product,
2. external attributes: features requiring a compre- 124
hensive measurement of the product and its
run-time behavior.

127 External attributes such as reliability, perfor-
128 mances, cost, etc. are difficult to measure. For this
129 reason, they are often estimated using some,
130 strictly related, internal attributes. The measure-
131 ment of internal attributes during a project helps
132 to identify quality problems in the final product.

133 Software metrics aim at quantifying properties
134 of artifacts generated during the software lifecycle,
135 of the development process, and of the associated
136 resources [7].

137 The idea of industrializing the software devel-
138 opment process is centered on the ability to iden-
139 tify and institutionalize advantageous software
140 development practices. These “best practices” can
141 be the cornerstone of the effective and efficient
142 development of sound software systems.

143 Unfortunately, most of the proposed “best
144 practices” still lack a sound, quantified assessment.
145 Major problems include the identification of a
146 substantial amount of relevant industrial data to
147 analyze with a sound application of modern sta-
148 tistical techniques. Most of the studies refer to
149 student data. The way students develop software
150 cannot be considered representative of profes-
151 sional development. Statistical techniques are of-
152 ten used without a careful determination of their
153 applicability.

154 Major theoretical and practical problems pre-
155 vent the drawing of general conclusions applicable
156 in different contexts even in the few cases when
157 individual projects have evidenced the pros of a
158 methodology or tool. In principle, generalization
159 of findings coming from a particular study requires
160 that we must make sure that the samples of de-
161 velopers, software, and the history of the software
162 are randomly chosen from their respective global
163 populations. This is seldom the case, since most of
164 the individual studies in software engineering are
165 performed for a specific software development
166 environment.

167 2.2. Data collection

168 In software engineering, measures are difficult
169 to collect due to two main problems [5,14]:

170 1. Collecting metrics is a time expensive task. This
171 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. 174
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.

Understanding how an enterprise really works 180
is not an easy task and requires a huge amount of 181
time spent interviewing both managers and em- 182
ployees. Managers of specific enterprise functions 183
should know, in a very detailed way, how tasks are 184
performed. Often, they know the big picture but 185
their knowledge of the details is very different from 186
how tasks are really performed. That could hap- 187
pen due to several reasons. Among them, there are 188
the following: 189

- There are no or limited quantity of data regard- 190
ing how people work.
- People claim to work in a certain way but they 192
actually work differently.

The former is possible because inspections are 194
very expensive, do not produce immediate benefits 195
and too many errors or missing data badly affect 196
the analysis process. These errors appear mostly in 197
critical periods, when the data correctness should 198
help to understand better the situation such as 199
during high stress working periods. 200

The latter is possible when management intro- 201
duces wrong or too complex processes that people 202
hardly follow but they claim to do. In this way, 203
there is a mismatch between the real process and 204
managers’ knowledge. 205

Unfortunately, it is difficult to collect useful 206
data in software development environments be- 207
cause managers and developers do not consider 208
measurement an important activity, compared to 209
coding. Moreover, it is hard to collect data man- 210
ually and there are only few, very expensive tools 211
to collect the data semi-automatically. Semi-auto- 212
matically means that still the human intervention 213
is heavily involved, especially in the critical task of 214
collecting process measures. 215

216 Such tools suffer from severe limitations. (A)
217 Mostly, they deal with product measures;
218 (semi)automatic collection of process measures is
219 nearly always ignored. (B) Often, they are not in-
220 tegrated in the “usual” working context of devel-
221 opers and managers; the developer is required to
222 invoke such tools explicitly. This lack of integra-
223 tion affects the precision of the collected data.
224 Sometimes, it even happens that measures are
225 collected later in the process than expected, just to
226 comply with given process guidelines; this results
227 in spurious data. (C) Even when some of the tools
228 a developer or a manager may use supports the
229 collection of a few, mostly product-oriented mea-
230 sures, such measures are not automatically com-
231 bined with the measures collected from other tools
232 the user or the manager may be using. This still
233 requires the developer and the manager to store
234 explicitly the data with the consequent problems
235 mentioned above; therefore, it limits the possibility
236 of extracting sound, sensitive information from the
237 combined analysis of measures.

238 The lack of fully automated process and prod-
239 uct measures collection tools affects severely the
240 PSP and other similar approaches. Empirical in-
241 vestigations on the PSP evidence that measures
242 data are usually not collected at all or with the
243 required precision to produce meaningful results
244 [13].

245 Automated data collection, integrated in pop-
246 ular software, helps to solve both problems and
247 introduce further benefits such as help to imple-
248 ment the Activity-Based Costing (ABC) as ac-
249 counting technique [3]. These automated data
250 collection is possible wherever people spend most
251 of their time performing computer-based activities
252 such as software development.

253 Engineers need to measure relevant variables of
254 a process to understand and control it. That also
255 happens in software engineering. The main re-
256 source in many companies, such as software
257 companies, is human resource. For this reason,
258 software engineers are mainly interested in: human
259 effort needed to complete a task, quality, cost, and
260 time required.

261 Process metrics describe process qualities such
262 as effort required, production time, steps of a task,
263 etc. These qualities can be evaluated through the

264 acquisition of measurable properties such as edit- 264
265 ing time, number and type of changes in a file, 265
266 usage patterns, etc. 266

267 Most of management costs are human resources 267
268 costs: experience, skills, etc. Moreover, produc- 268
269 tivity of very good employee is tens times better 269
270 than average [20]. 270

271 For these reasons, it is very important to un- 271
272 derstand how top developers work and force all to 272
273 adopt a process that helps them to achieve the best 273
274 results possible. 274

275 The Personal Software Process (PSP) [11] is a 275
276 process for development improvement on a per- 276
277 sonal level. It is a self-improvement process to 277
278 control, manage, and improve one’s work. There is 278
279 a structured framework of forms, guidelines, and 279
280 procedures for software development. 280

281 PSP uses the idea of feedback to achieve im- 281
282 provement. Developers define, measure, and track 282
283 metrics such as code size, defects, effort, defect- 283
284 removals, and so on. They evaluate and learn 284
285 based on their defined processes and measure- 285
286 ments. The measurements help to improve a de- 286
287 veloper’s ability to estimate code size and effort 287
288 before starting. They also aid the developer in 288
289 becoming more efficient in doing reviews. 289

290 Humphrey stresses the use of checklists to aid in 290
291 design and code reviews. A checklist makes the 291
292 review process more complete, formalized, and 292
293 efficient. As new types of defects are found in the 293
294 development process, they are appended to the 294
295 checklist. The checklist moves the effort of fixing 295
296 defects up to an earlier phase, where the repair is 296
297 magnitudes less costly. 297

298 The greatest goal of the PSP is for developers to 298
299 discover the methods and practices that work for 299
300 their abilities and adapt the process for their own 300
301 use. Humphrey claims that improvement on the 301
302 personal level ultimately makes developers better 302
303 team members. 303

304 PSP include several phases to help developers to 304
305 trace a data set that includes, among others, 305
306 working times and defects. At the end of each 306
307 phase, collected data are ordered, related to fur- 307
308 ther information such as code metrics, and stored 308
309 in ad hoc forms (plan summary forms) in a Post- 309
310 mortem Phase. At the end of all phases, developers 310

311 update their private database of historical infor-
312 mation regarding the productivity.

313 PSP and several other research initiatives evi-
314 dence the importance of collecting such data in
315 software engineering and cross-analyzing them to
316 have a benchmark for improvement. In addition,
317 Extreme Programming and the other agile meth-
318 odologies rooted in lean management inherit from
319 lean management the critical importance of mea-
320 suring [16].

321 PSP requires the collection of detailed metrics
322 of the development time, bugs discovered and
323 corrected at all development stages, and software
324 size. Then all collected data are analyzed through
325 statistical methods. These results provide software
326 engineers sets of historical data mainly used to:

- 327 1. make reliable estimates on variables such as
- 328 time schedule, quality, etc. of on going projects,
- 329 2. find out how to improve the development pro-
330 cess identifying problems.

331 PSP requires the collection of the following
332 data:

- 333 • Editing time: time spent performing a task such
334 as coding, writing a document, etc.
- 335 • Tool name: the name of the tool used.
- 336 • File name: the name of the file edited.
- 337 • Class name: the name of the class edited (if pos-
338 sible).
- 339 • Project: the name of the project to which the
340 document belongs.
- 341 • Size: total size of the document.
- 342 • Differential size: size of the document compared
343 to the previous version.
- 344 • Defects: defects identified in the source code.

345 Due to the amount of information required,
346 manual data collection is time expensive and error
347 prone. An automated data collection is the only
348 way to collect reliable data without affecting the
349 process under analysis.

350 2.3. Metrics correlation

351 Code and process metrics measures different but
352 related attributes. Code metrics are able to identify

potential problems inside the source code but they
do not provide any data regarding how they have
been generated. On the contrary, process metrics
trace the behavior of developers but they do not
collect information regarding the modification in-
cluded in the source code.

A comprehensive approach is the best way to
collect data regarding how developers spend their
time and the effects of this effort on the source
code. As instance, a huge effort spent in a single
class can produce a remarkable improvement of
the code or not. The only way to collect this in-
formation is through the comparison of code and
process metrics data.

3. State of the art

At present, there are several tools to address the
metrics collection problem. Some of them are
commercial products such as MetricCenter [15] or
ProjectConsole [17]; others are research tools such
as Hackstat [10] developed at the University of
Hawaii. Often, these tools uses proprietary solu-
tions, domain dependant, or focusing on a specific
aspect of data collection or analysis.

The basic idea of the Hackstat system is very
close to PROM. Both are based on open source
software and use plug-ins to collect data, but the
aim of the two tools is very different. Hackstat
focuses on the coding activity and target users are
developers; PROM focuses on the entire develop-
ment process including both code activities and
non-code activities such as writing documentation,
project management, etc. The target users include
all members of the development team (including
developers, managers, etc.).

In addition, PROM offers both detailed data
for developers and high-level views, at project le-
vel, for managers helping them to monitor and
improve the whole development process.

One of the goals of the PROM system is the
collection of data and the retrieve of important
information through a wide range of devices, in-
cluding laptops and PDAs. The system is able to
collect data even if the user is working offline using
a synchronization mechanism when that system is
online again.

Table 1
Features 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
Support to mobile users	Built-in	-

398 Table 1 briefly compares the main features of
399 the two systems.

400 4. Distributed architecture

401 PROM (PRO Metrics) is a distributed Java tool
402 designed to collect different set of software data:
403 software metrics [7] and Personal Software Process
404 data [11]. The former set includes code length,
405 inter-class and inter-function dependencies, reus-
406 ability, etc. The latter includes time spent in each
407 activity, number of changes per class, etc.

408 To collect such data, the architecture has three
409 main components (Fig. 1):

410 1. *PROM core*: this is a Java based architecture
411 that includes three main components: a data-
412 base that stores all data, a centralized PROM
413 Server, and many distributed Plug-ins Servers
414 (one per client). It provides web administration
415 tools and access to collected data through dy-
416 namically generated statistics.

2. *Tool-specific plug-ins*: they listen to events gen- 417
erated by third-parties tools and send relevant
data to the PROM core [19]. Plug-ins are tool
dependant, for this reason they are written us-
ing different languages. A present, in available
or under development plug-ins, these languages
are: Java for NetBeans, Eclipse, Borland
JBuilder, and Together Central; C++ for Mi-
crosoft Visual Studio 6.0, C# for Microsoft Of-
fice XP and Visual Studio.NET.

3. *Third-parties tools*: the system uses popular 427
tools for collecting data. Such tools include
IDEs (Microsoft Visual Studio, NetBeans,
Eclipse, Borland JBuilder, etc.), design tools
(Rational Rose, Together Central, etc.), and of-
fice automation tools (Microsoft Office, Sun
Staroffice, Open Office, etc.).

The Hackystat system developed at University 434
of Hawaii [13] partially inspired this tool but the 435
two systems has different targets: the former pro- 436
vides the most benefits to developers focusing on 437
PSP; the latter has a wider target, it provides in- 438
formation to both managers and developers using 439

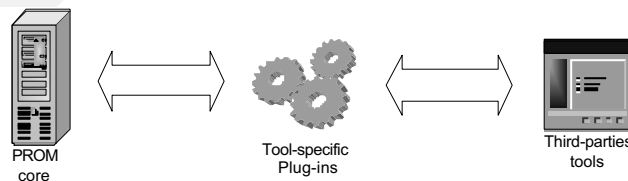


Fig. 1. Architecture components.

440 different views on collected data that include
441 software metrics and PSP data.

442 The architecture of PROM fulfills these main
443 constraints:

- 444 1. it is extensible to support new third-parties
445 tools,
- 446 2. the development of plug-ins for third-parties
447 tools are as simple as possible,
- 448 3. it supports both online and offline users,
- 449 4. it provides data to both managers and develop-
450 ers preserving privacy,
- 451 5. it supports different kind of clients (desktop,
452 laptops, and handler),
- 453 6. all collected data are accessible through the
454 SOAP protocol [18].

455 To fulfill such requirements the architecture is
456 presented from different points of view. Fig. 1
457 describes the architecture of PROM from the point
458 of view of data collection fulfilling the first con-
459 straint. Fig. 2 focuses on the same architecture as a
460 layered structure satisfying constraints 4, 5, and 6.
461 From this point of view, the architecture includes
462 three layers: a database (layer 1), a server (layer 2),
463 and a set of clients (layer 3).

464 The *PROM Database* (layer 1) stores all col-
465 lected data, results of analysis, project details, and
466 data regarding users. The *PROM Server* (layer 2)
467 provides access to all information stored inside the
468 database and make them available to clients in
469 very different ways. Users can access data ac-

470 cording to their role (developers or managers).
471 Developers can access their own data; managers
472 can access only aggregated data at different levels
473 to preserve privacy of developers. Moreover, users
474 can allow colleagues to access their own data.
475 Clients (layer 3) include different kinds of devices:
476 desktops, laptops, and PDAs. The first two set of
477 devices can fully access to the system and collect
478 data (software metrics and PSP data) through
479 plug-ins attached to third-parties tools; PDAs
480 cannot collect such data but they can provide in-
481 formation to managers even if they are mobile
482 users accessing reports and statistical data impor-
483 tant for the decision process.

484 Devices that collect data (desktops and laptops)
485 require a set of three entities to satisfy require-
486 ments 2 and 3 (Fig. 3): a third-party tool, a plug-
487 in, and a Plug-ins Server.

488 Developers use third-parties tools to design,
489 write software or documentation. Tool-specific
490 plug-ins listen to events generated inside those
491 tools and collect relevant information. A Plug-ins
492 Server collects such data and sends them to the
493 PROM Server. Each client collecting data needs a
494 Plug-ins Server to simplify plug-ins development.
495 Plug-ins are tool dependant, so a specific imple-
496 mentation is required for each supported tool,
497 while the Plug-in Server is written once because it
498 is not tool dependant. For this reason plug-ins
499 delegate as much as possible to the Plug-ins Server.
500 This server provides a set of services to plug-ins:

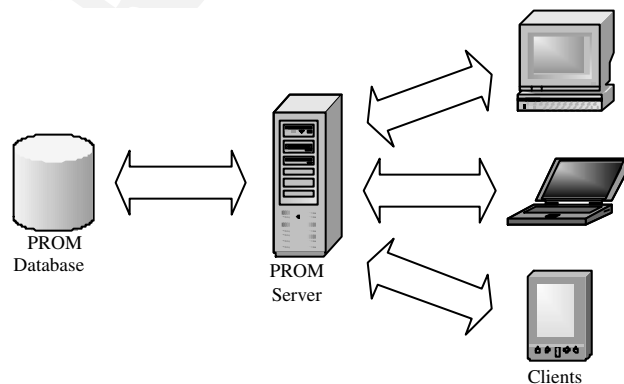


Fig. 2. Architecture overview.

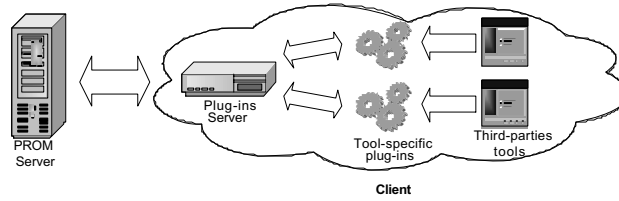


Fig. 3. Architecture of clients.

- 501 1. users authentication,
502 2. data caching to support offline users,
503 3. data storing into the PROM Database through
504 the PROM Server.

505 System administration and reports are available
506 through dynamically generated web pages that an
507 application server provides retrieving data through
508 the PROM Server (Fig. 4).

509 System administration includes: users and pro-
510 jects management (add, remove, update, assign
511 users to projects, etc.), event collection parameters
512 (types of events, frequency of collection, etc.),
513 system status reports, etc.

514 The PROM system provides many different
515 views on data. Detailed ones are designed for de-
516 velopers; aggregated views at project level are
517 useful for managers. A deeper analysis is possible
518 integrating into the system a specific analysis tool
519 that can access data through the SOAP protocol
520 and perform custom analysis.

521 The architecture also includes data analysis and
522 visualization (Fig. 4). The PROM Server performs
523 simple data analysis retrieving data from the da-
524 tabase and aggregating them. The analysis con-
525 sideres also code evolution keeping track of
526 versions of the source code stored into a CVS re-

pository. When a piece of code is stored into a 527
version control system, the Plug-in Server invokes 528
the metrics extractor (WebMetrics) that collects 529
metrics data and stores them into the PROM 530
Database. This feature provides a comprehensive 531
view of the development process to developers. 532

Data visualization is performed in two ways: 533

- 534 1. through web pages: a user can retrieve informa- 534
tion using customizable but predefined queries. 535
The user select a specific query, such as total 536
time spent in a project, than inserts some pa- 537
rameters such as time constraints, and the sys- 538
tem returns a web page with the required 539
data. Among these predefined elaborations 540
there are: effort spent in a project by users, ef- 541
fort spend by a user considering all projects, 542
metrics-specific graphs (i.e. length of code, com- 543
plexity, coupling, etc.);
- 544 2. using third-parties tools: in this case the system 545
provides all the data to a specific tool that can 546
perform user-defined analysis. 547

4.1. WebMetrics 548

WebMetrics is an extensible tool able to extract 549
code metrics from source code files. At present, 550

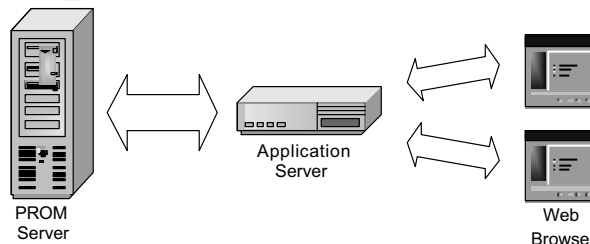


Fig. 4. PROM administration.

551 implementation of the C/C++ and Java parsers are
552 available.

553 This tool is able to extract the following metrics
554 [7]:

- 555 • Procedural code metrics,
- 556 • Object-Oriented design metrics.

557 Procedural code metrics measure internal at-
558 tributes of functions and procedures, so they are
559 applicable to almost all programming languages.
560 In particular, the following metrics are considered:

- 561 • Lines of Code (LOC),
- 562 • McCabe Cyclomatic Complexity,
- 563 • Halstead Volume,
- 564 • Fan-In and Fan-Out.

565 In addition, metrics have also been proposed
566 for designs, especially object-oriented (OO) de-
567 signs. Since OO designs can be well-defined and
568 specified, as with a language like the Unified
569 Modeling Language (UML), measures about a
570 system's class structure, coupling, and cohesion
571 can be easily derived.

572 The most well-cited OO metrics are the Chid-
573 amber and Kemerer (CK) suite of OO design
574 metrics [2]. This is a set of six metrics which cap-
575 ture different aspects of an OO design, including
576 complexity, coupling, and cohesion. These metrics
577 were the first attempt at being OO metrics with a
578 strong theoretical basis. The metrics are listed be-
579 low:

- 580 • Weighted Methods per Class (WMC),
- 581 • Depth of Inheritance Tree (DIT),
- 582 • Number of Children (NOC),
- 583 • Coupling Between Object Classes (CBO),
- 584 • Response For a Class (RFC),
- 585 • Lack of Cohesion in Methods (LCOM).

586 WebMetrics includes two main modules: a web
587 interface and a backend. These modules are very
588 tightly coupled and they can be used separately.
589 The former is not useful inside the PROM archi-
590 tecture due to the more advanced interface re-
591 quired by the whole system; the latter is a
592 command line tool that requires a set of source

code files and produces a file with the data of the
analysis. Due to this architecture, the integration
of WebMetrics tool in the PROM architecture is
through the exchange of files.

5. The implementation

The PROM system includes the PROM core
and tool-specific plug-ins (Fig. 1). The former is
entirely written in Java, the latter are developed
using different languages because Java is not al-
ways supported to write tools extensions (e.g. in
Microsoft's products).

The PROM core is completely based on open
source technologies (e.g. PostgreSQL, Apache
Tomcat, etc.) and standard protocols (e.g. XML
[9,21], SOAP [18], etc.). It includes:

1. *PROM Database*: it is implemented using the
open source DBMS PostgreSQL.
2. *PROM Server*: it provides access to the data-
base and exposes its functionalities as web ser-
vices. It is based on Java Servlets [6] and
implemented with the Apache Tomcat extended
with the Apache Axis SOAP server.
3. *Plug-ins Server*: it collects data from plug-ins
and performs data preprocessing. It caches in-
formation and sends data to the PROM Server
when the client is online. It is implemented in
Java and uses the Apache Axis libraries to com-
municate with plug-ins and with the PROM
Server.

The PROM Server architecture (Fig. 5) runs on
the Apache Tomcat 4.1 Servlet engine integrated
with the Apache Axis 1.0 to manage SOAP con-
nections. The server includes a *Core* and a set of
pluggable *Commands*. All the functionalities of the
system are managed through specific *Commands*
(i.e. authentication, data collection, etc.) while the
Core provides only support to the execution of the
commands. The *Core* is implemented as a Java
Servlet that launches the execution of a specific
command using the parameters passed through the
SOAP connection. The commands are simple Java
classes implementing the *Command* interface that

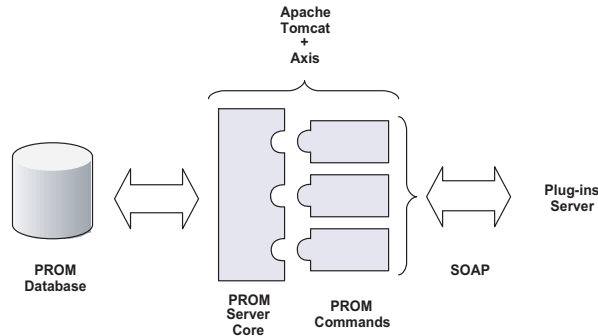


Fig. 5. PROM Server architecture.

635 can be added to the system without any change to
636 the *Core*.

637 The SOAP protocol implements all communi-
638 cations between PROM components except be-
639 tween the PROM Database and the PROM Server
640 that uses the JDBC protocol. The PROM server
641 exposes a set of services that are accessible inside
642 an Intranet or on the Internet as web services. The
643 available implementation of the system is designed
644 to run inside an Intranet or on the Web using a
645 Virtual Private Network (VPN) because no en-
646 cryption is implemented in the communication yet.
647 PROM Server-Plug-ins Servers communi-
648 cations use the same scheme based on the *Command*
649 design pattern (Fig. 6) [8].

650 The chosen communication scheme provides an
651 easy mechanism for extending the functionalities
652 implemented through servers. The server exposes
653 only one method (*executeCommand*) that takes a
654 *Command* as input and produce a *Result* as output.
655 Subclasses of *Command* implement the desired

656 functionality and use *Parameter* subclasses for
657 parameterization purposes.

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

668 The *PROM Server* offers a set of basic com-
669 mands such as:

- 670 • *AuthenticateCommand*: provides support to au-
671 thenticate users,
- 672 • *ProcessEventCommand*: collects all process met-
673 rics,
- 674 • *ProcessEventMetricsCommand*: collects all code
675 metrics,

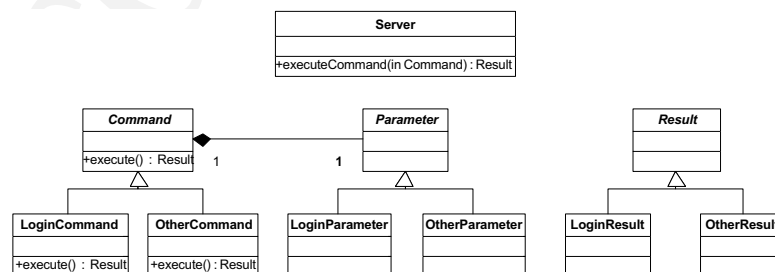


Fig. 6. Command architecture.

- 676 • *ProcessSQLCommand*: queries the database
 677 with a user-defined query and returns a result
 678 set,
 679 • *ProjectMembershipCheck*: checks if a user be-
 680 longs to a specific project,
 681 • *ReadProjectsCommand*: returns the name of all
 682 projects.

683 The PROM architecture supports third-parties
 684 tools for a more accurate data analysis. The
 685 PROM Server can provide data to advanced
 686 analysis tools that support the SOAP protocol for
 687 data exchanges.

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

The manual data collection is performed 700
 through web forms (Fig. 7) that collect a set of 701
 information very close to the one collected 702
 through plug-ins: users, project name, activity, 703
 starting time, and length. 704

6. Conclusion 705

This paper presented the architecture and the 706
 Java implementation of a distributed tool for col- 707
 lecting and analyzing software metrics and PSP 708
 data. 709

The collection of such data should help both 710
 developers and managers. The formers can keep 711
 track of their performances and compare them 712
 with other people in the same working group; the 713
 latter can easily keep track of projects evolution 714
 and accounting information. 715

The system is still under development but plans 716
 for the next version are nearly completed. That 717
 new release will provide a secure communication 718
 protocol between PROM Server and Plug-ins 719
 Servers to allow Internet use. Moreover, improved 720
 analysis tools will be integrated into the system. 721

The screenshot shows a web browser window titled "PROM - Microsoft Internet Explorer". The address bar shows the URL "Z:\Progetti\MAPS\Deliverable\Unige\Manual.htm". The main content area displays the "PROM Manual Data Collection Form".

User(s):	Alberto Sillitti	Add User																																										
Project Name:	PROM	Add Project																																										
Activity:	<input type="text"/>																																											
Start Time:	<div style="display: flex; align-items: center;"> <div style="margin-right: 10px;">lug</div> <div style="margin-right: 10px;">2003</div> <div style="border: 1px solid black; padding: 2px;"> <table border="1" style="font-size: small;"> <tr><td>30</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td></tr> <tr><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td></tr> <tr><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td></tr> <tr><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td></tr> <tr><td>28</td><td>29</td><td>30</td><td>31</td><td>1</td><td>2</td><td>3</td></tr> <tr><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td></tr> </table> </div> <div style="margin-left: 10px;"> Hour Minute 15 30 </div> </div>	30	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	1	2	3	4	5	6	7	8	9	10	
30	1	2	3	4	5	6																																						
7	8	9	10	11	12	13																																						
14	15	16	17	18	19	20																																						
21	22	23	24	25	26	27																																						
28	29	30	31	1	2	3																																						
4	5	6	7	8	9	10																																						
Length:	Hours: 1 Minutes: 15																																											

Operazione completata intranet locale

Fig. 7. Manual data insertion form.

722 **Acknowledgements**

723 This work has been co-funded by the Italian
724 Government through the Funds for Basic Re-
725 search (MAPS project, an Italian acronym for
726 Agile Methodologies for Software Production).
727 We would like to thank out project partners for
728 their stimulating input.

729 The first author tanks Professor Philip Johnson
730 and his research staff at the University of Hawaii
731 at Honolulu for showing him the Hackystat sys-
732 tem and answering his many questions.

733 **References**

- 734 [1] B. Boehm, B. Clark, E. Horowitz, C. Westland, R.
735 Madachy, R. Selby, Cost models for future software life
736 cycle processes: COCOMO 2.0, *Annals of Software Engi-*
737 *neering* 1 (1) (1995).
- 738 [2] S.R. Chidamber, C.F. Kemerer, A metrics suite for object
739 oriented design, *IEEE Transactions on Software Engineer-*
740 *ing* 20 (6) (1994).
- 741 [3] G. Cokins, *Activity-based Cost Management: An Execu-*
742 *tive's Guide*, John Wiley & Sons, 2001.
- 743 [4] T. DeMarco, *Controlling Software Projects—Manage-*
744 *ment, Measurement & Estimation*, Yourdan Press, 1982.
- 745 [5] A.M. Disney, P.M. Johnson, Investigating data quality
746 problems in the PSP, in: *Sixth International Symposium on*
747 *the Foundations of Software Engineering (SIGSOFT'98)*,
748 Orlando, FL, USA, November 1998.
- 749 [6] B. Eckel, *Thinking in Java*, second ed., Prentice-Hall, 2000.
- 750 [7] N.E. Fenton, S.H. Pfleeger, *Software Metrics: A Rigorous*
751 *and Practical Approach*, Thomson Computer Press, 1994.
- 752 [8] E. Gamma, R. Helm, R. Johnson, J. Vlissidies, *Design*
753 *Patterns: Elements of Reusable Object-Oriented Software*,
754 Addison-Wesley, 1994.
- 755 [9] C.F. Goldfarb, P. Prescod, *The XML Handbook*, third
756 ed., Prentice Hall Computer Books, 2000.
- 757 [10] Hackystat, University of Hawaii—website: [http://](http://csdl.ics.hawaii.edu/Research/Hackystat/)
758 csdl.ics.hawaii.edu/Research/Hackystat/.
- 759 [11] W. Humphrey, *A Discipline for Software Engineering*,
760 Addison-Wesley, 1995.
- 761 [12] IEEE, *IEEE Standard for Glossary of Software Engineer-*
762 *ing Terminology*, IEEE Std 828, 1983.
- 763 [13] P.M. Johnson, You can't even ask them to push a button:
764 Toward ubiquitous, developer-centric, empirical software
765 engineering, *The NSF Workshop for New Visions for*
766 *Software Design and Productivity: Research and Applica-*
767 *tions*, Nashville, TN, USA, December 2001.
- 768 [14] P.M. Johnson, A.M. Disney, A critical analysis of PSP
769 data quality: Results from a case study, *Journal of*
770 *Empirical Software Engineering* (1999).

- [15] MetricCenter, *Distributive Software*—website: [http://](http://www.distributive.com/)
771 www.distributive.com/.
- [16] S. McConnell, *Rapid Development: Timing Wild Software*
773 *Schedules*, Microsoft Press, 1996.
- [17] ProjectConsole, Rational Software Corporation—website:
775 <http://www.rational.com/>.
- [18] SOAP (Simple Object Access Protocol)—specifications:
777 <http://www.w3.org/TR/SOAP/>.
- [19] G. Succi, W. Pedrycz, E. Liu, J. Yip, Package-oriented
779 software engineering: a generic architecture, *IEEE IT Pro*
(2001).
- [20] J.P. Womack, D.T. Jones, *Lean thinking: banish waste and*
782 *create wealth in your corporation*, Simon & Schuster, 1996.
- [21] XML (Extensible Markup Language) 1.0—specifications:
784 <http://www.w3.org/TR/2000/REC-xml-200010>.



A. Sillitti is a PhD Student in Electrical and Computer Engineering at the University of Genoa, Italy, where he received a Laurea degree (summa cum laude) in 2001. In 2000 he developed a real-time simulator of an Italian highway, and then he was involved in two EU founded projects related to software components and Agile Methodologies. His research areas include software engineering, component-based software engineering, integration and measures of web services, and agile methodologies.



A. Janes is a PhD Student in Economics at the University of Klagenfurt, Austria, and Research Assistant at the Free University of Bozen, Italy. He achieved his Master of Science in 2002 at the Technical University of Vienna, Austria. His research areas include Software Engineering, Agile Methodologies, Information Visualization and Economics-Driven Software Engineering.



G. Succi, Ph.D., PEng is Professor of Software Engineering and Director of the Center for Applied Software Engineering at the Free University of Bozen. His research areas include empirical software engineering, software product lines, software reuse, software engineering over the Internet. He is author of more than 100 papers published in international conferences and journals, and of one book.



T. Vernazza received a degree in Electronic Engineering from the University of Genoa in 1971, he worked on the design of the central processor for a digital telephone exchange in Telettra from 1971 to 1974. Then he joined the Faculty of Engineering of Genoa University as Assistant Professor in 1974 and as Associate Professor in 1983. Actually he is holding the courses of Calcolatori Elettronici and Calcolatori Elettronici II at the third and fourth year of the Laurea Degree in Computer and Control Engineering,

University of Genoa.

UNCORRECTED PROOF