MAGISTER: Quality Assurance of Magic Applications for Software Developers and End Users

Csaba Nagy, László Vidács
Rudolf Ferenc, Tibor Gyimóthy
University of Szeged, Hungary
Department of Software Engineering
Research Group on Artificial Intelligence
{ncsaba,lac,ferenc,gyimi}@inf.u-szeged.hu

Ferenc Kocsis, István Kovács
SZEGED Software Zrt.
Hungary
{kocsis.ferenc,kovacs.istvan}@szegedsw.hu

***Abstract*—Nowadays there are many tools and methods available for source code quality assurance based on static analysis, but most of these tools focus on traditional software development techniques with 3GL languages. Besides procedural languages, 4GL programming languages such as Magic 4GL and Progress are widely used for application development. All these languages lie outside the main scope of analysis techniques.**

In this paper we present *MAGISTER*, which is a quality assurance framework for applications being developed in Magic, a 4GL application development solution created by Magic Software Enterprises. *MAGISTER* extracts data using static analysis methods from applications being developed in different versions of Magic (v5-9 and uniPaaS). The extracted data (including metrics, rule violations and dependency relations) is presented to the user via a GUI so it can be queried and visualized for further analysis. It helps software developers, architects and managers through the full development cycle by performing continuous code scans and measurements.

***Index Terms*—Magic 4GL, Reverse Engineering, Quality Assurance, Metrics, Static Analysis**

I. Introduction

Fourth generation languages (4GLs) are also refered to as Very High Level Languages (VLLs) [1]. A developer who develops an application in such a language does not need to write 'source code', but he can program his application at a higher level of abstraction and higher statement level, usually with the help of an application development environment. These languages were introduced and widely used in the mid-1980s. At that time many 4GLs were available (such as Oracle, FOCUS, RAMIS II and DBASE IV), but today most of the information systems are developed in third generation languages. However, large systems developed earlier in a 4GL are still evolving and there is still a continuous need for RADD (Rapid Application Development and Deployment) tools, which are usually based on these higher level languages.

Since the appearance of 4GLs, large software systems have evolved and the role of quality assurance of these systems is of increasingly importance. Unfortunately, the main focus of current QA tools and techniques is on the more popular 3GL languages.

In the literature few papers are available considering the software quality of 4GLs languages. When they became popular, many studies were published in favour of their use. These studies tried to predict the size of a 4GL project and its development effort, for instance by calculating function points [2], [3] or by combining 4GL metrics with metrics for database systems [4]. Today, some tools are available for testing purposes and for optimization purposes like Magic Optimizer [5].

In this paper, with ***MAGISTER*** we propose a novel quality assurance framework for Magic 4GL, namely, an adapted methodology for the QA of 4GL languages based on the continuous supervision of product metrics, rule violations and changes in Magic programs.

At the heart of ***MAGISTER*** lies the widely used, multi-layer, product-oriented Columbus reverse engineering technology [6], which performs continuous source code analysis and the tracking of changes. In addition, there are other components as well. The components of ***MAGISTER*** built on top of one another, may be used separately. A large part of the components may be run in an automated manner, supported by various tools, whereas performing manual steps requires expert knowledge.

II. Quality Assurance on Magic

A. *Magic specialties*

Magic 4GL was introduced by Magic Software Enterprises (MSE) in the early 80's. It was an innovative technology to move from code generation to the use of an underlying meta model within an application generator. The resulting application was run on popular operating systems including DOS and UNIX. Since then newer versions of Magic have been released called *eDeveloper* and *uniPaaS*. Recent versions support novel technologies including RIA (Rich Internet Applications), SOA (Service Oriented Architecture), XML and SAP.

The heart of a Magic application is the Magic Runtime Engine (MRE), which allows one to run the same application on different operating systems. When one develops an application in Magic, one actually programs the MRE using the unique meta model language of Magic, which is – at a higher level of abstraction – closer to business logic. This meta model is what makes the development in Magic unique and what really makes Magic a RADD (Rapid Application Development and Deployment) tool.

Magic was invented to develop business applications for data manipulating and reporting, so it comes with many GUI screens and report editors. Hence the most important elements of its meta model language are the various entity types of business logic, namely the data tables. A table has its columns and a number of programs (consisting of subtasks) that manipulate it. The programs or tasks are linked to forms, menus, help screens and they may also implement business logic using logic statements e.g. for selecting variables (virtual variables or table columns), updating variables, conditional statements and expressions.

The meta model of a Magic application serves as a 'source code' that can be analyzed for quality assurance purposes. Using this model we can describe the main characteristics of an application and we can locate potential coding problems or structures which may indicate bugs or bad design.

B. Quality supervision

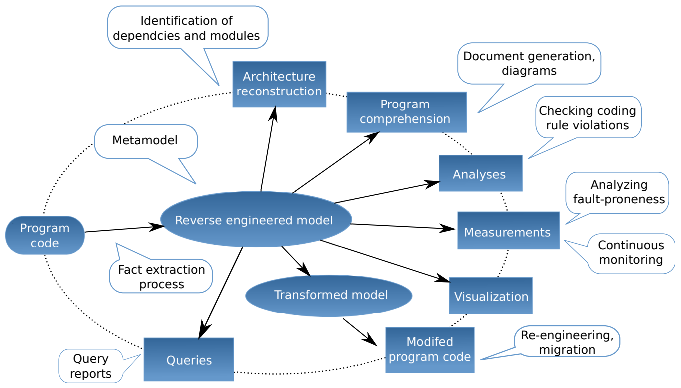


Figure 1. Columbus methodology adapted in the Magic environment.

Elements of the Columbus methodology, successfully applied on object-oriented languages, are re-used and adapted in the Magic environment (Figure 1). Quality supervision covers the most influential areas of the software life cycle with the following goals [7]:

- Decrease the number of post-release bugs
- Increase maintainability
- Decrease development/test efforts
- Assure sustainability
- Assure continuous measurement and assessment

Goals are targeted with continuous monitoring: scheduled analysis, data processing, storing and querying, visualization, evaluation. Regarding these activities, based on the global solution, quality assessment services may be offered: programming rule violations; maintainability, testability reports based on software product metrics, duplicated code; architecture reconstruction [8]; concrete suggestions for quality improvement.

III. Overview of ***MAGISTER***

MAGISTER can be easily incorporated into the processes of a Magic developer company. A typical environment is shown in Figure 2. The project leader or manager gives orders to developers and controls their progress. A developer does the 4GL programming in the Magic/uniPaaS environment and commits stable states to a configuration management system. To take control of the process, the program under development is checked out by the BuildEngine on a regular basis, and given to the analyzers. The final results are uploaded to the database, and the first checks are performed to produce reports and notify the leaders about any new programming problems. Notification can be customized to report bad tendencies in complexity or other properties of the program as well.

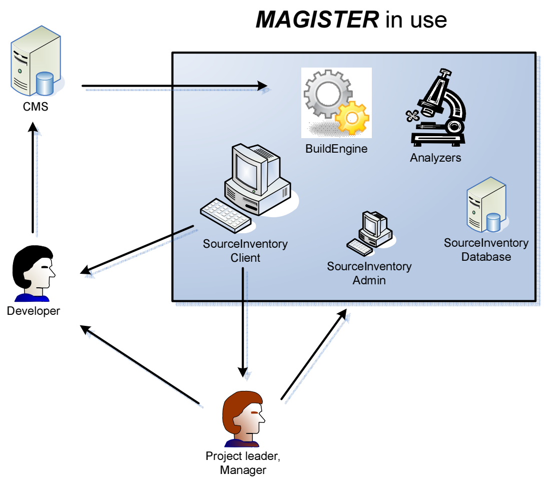


Figure 2. *MAGISTER* in a typical development environment.

Analysis results can be investigated using the SourceInventory Web client. The developer may look for programming problems like invalid references, missing clauses; he may check for changed program elements since the last measurement; he may use references to reveal dependencies among programs, tasks and logic units; he may check metric data to find hot spots in the program. The project leader/manager receives the email report about problematic points and may get further information on the programs in question; he may check the overview of changed elements or analyze the summarized views of program-related metrics. The reported data serves as a guideline for decisions and provides objective reasons for taking action against bad developer habits if needed.

As can be seen, ***MAGISTER*** is divided into several layers. At the lowest level there are modules called *Magic analyzers* to extract all the necessary data from an application in question. These analyzers are able to parse control files or XML exports of Magic versions from 5 to 9, and *uniPaaS*, which is the most recent version. Analyzers calculate metrics, rule violations, dependency relations. The results of analyzers are uploaded to a database and visualized by *SourceInventory*, which has several other modules such as the *Uploader* to upload the results into a database, the *BuildEngine* to control regular analysis, the *AdminPages* for configuration purposes and the *SourceInventory applet* for the GUI. The results of *Magic analyzers* can be queried via another module called *MagiQuery*. This module was implemented as a utility package to help developers during their work. It has evolved over years to fulfill the technical requirements of developers. For instance it supports parameterizable cross-reference queries, the automatic generation of program documentations and migration (*e.g.* converting control files from Magic versions 5 to 9).

IV. Usage examples

Here we present example usage scenarios that demonstrate some of the main features of ***MAGISTER***. The scenarios, among many other features (except for the email warnings), can be reproduced and we kindly invite the reader to try them at home using our demo applet, reachable through the homepage http://www.szegedsw.hu/magister.

In the first scenario we learn how to query sample metrics of the tasks with top LLOC metric values. In addition to querying metric values, it shows how we can locate the longest tasks or programs of the system in question. These tasks usually play a central role in the application as they implement a relatively big part of the business logic. They usually have a relatively high complexity as well and they are coupled to many other tasks, as can be seen in Figure 3.

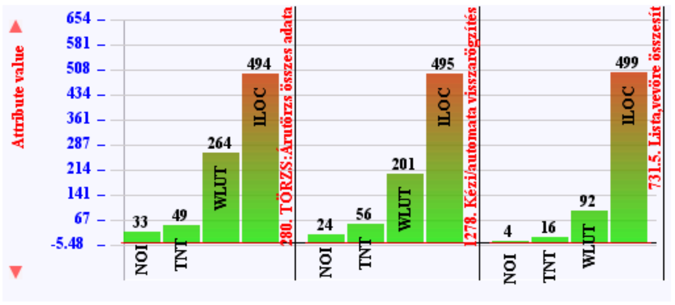
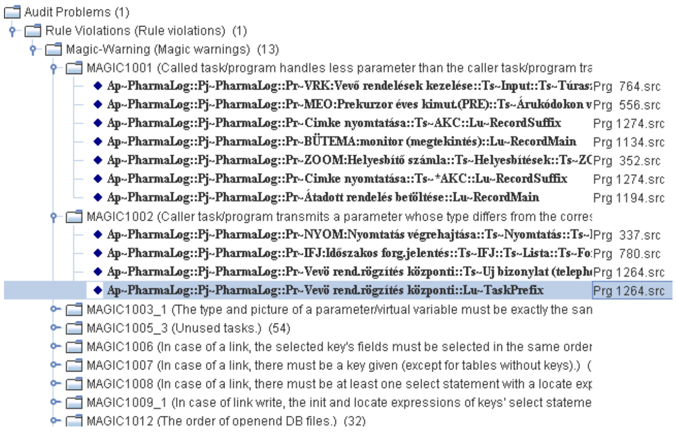


Figure 3. Bar chart showing the metrics of tasks with top LLOC metric values.

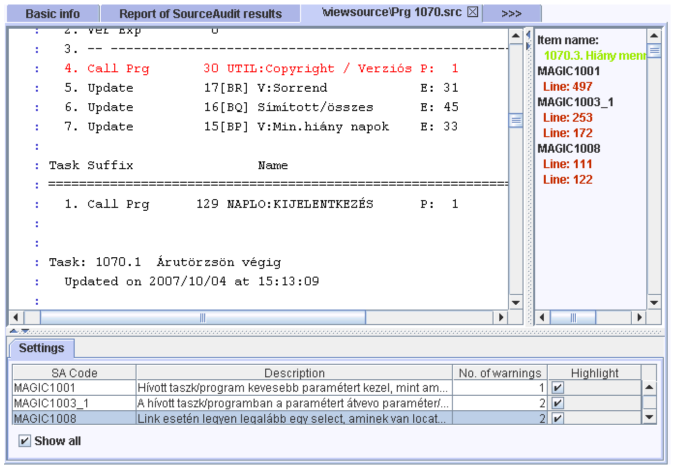
The full scenario is described in a step-by-step way in Table I. In Figure 3 the NOI (Number of Outgoing Invocations), TNT (Total Number of Tasks), WLUT (Weighted Logic Unit per Task) and the LLOC metrics can be seen. WLUT is a special complexity metric to measure the complexity of a task and is similar to WMC (Weighted Methods per Class), which measures the complexity of a class in object oriented languages. LLOC has a special meaning in magic too: it measures the number of non-remark (non-comment) logic lines (statements) in a task. Besides the sample metrics, we defined about 50 metrics and grouped them into size, complexity or coupling categories.

In the second example scenario we learn how to examine coding rule violations in the system being analyzed (see Table II). First we query a report called 'Problems report' to see all the rule violations in the system, and then we examine specific coding problems using the source view of SourceInventory. There are 22 rules implemented in ***MAGISTER*** based on experiences of developers who have been developing in Magic over 10 years. Figure 6 shows one example rule violation (*MAGIC1001*) where a task calls another subtask with more parameters than the callee can handle.



```
Audit Problems (1)
  Rule Violations (Rule violations)  (1)
    Magic-Warning (Magic warnings)  (13)
      MAGIC1001 (Called task/program handles less parameter than the caller task/program tra
        Ap~PharmaLog::Pj~PharmaLog::Pr~VRK:Vevő rendelések kezelése::Ts~Input::Ts~Túras:Prg  764.src
        Ap~PharmaLog::Pj~PharmaLog::Pr~MEO:Prekurzor éves kimut.(PRE)::Ts~Árukódokon vPrg  556.src
        Ap~PharmaLog::Pj~PharmaLog::Pr~Cimke nyomtatása::Ts~AKC::Lu~RecordSuffix        Prg 1274.src
        Ap~PharmaLog::Pj~PharmaLog::Pr~BÜTEMA:monitor (megtekintés)::Lu~RecordMain      Prg 1134.src
        Ap~PharmaLog::Pj~PharmaLog::Pr~ZOOM:Helyesbítő számla::Ts~Helyesbítések::Ts~ZCPrg  352.src
        Ap~PharmaLog::Pj~PharmaLog::Pr~Cimke nyomtatása::Ts~*AKC::Lu~RecordSuffix       Prg 1274.src
        Ap~PharmaLog::Pj~PharmaLog::Pr~Átadott rendelés betöltése::Lu~RecordMain        Prg 1194.src
      MAGIC1002 (Caller task/program transmits a parameter whose type differs from the corres
        Ap~PharmaLog::Pj~PharmaLog::Pr~NYOM:Nyomtatás végrehajtása::Ts~Nyomtatás::Ts~]Prg  337.src
        Ap~PharmaLog::Pj~PharmaLog::Pr~IFJ:Időszakos forg.jelentés::Ts~IFJ::Ts~Lista::Ts~Fo:Prg  780.src
        Ap~PharmaLog::Pj~PharmaLog::Pr~Vevö rend.rögzítés központi::Ts~Uj bizonylat (telephPrg 1264.src
        Ap~PharmaLog::Pj~PharmaLog::Pr~Vevö rend.rögzítés központi::Lu~TaskPrefix       Prg 1264.src
      MAGIC1003_1 (The type and picture of a parameter/virtual variable must be exactly the san
      MAGIC1005_3 (Unused tasks.)  (54)
      MAGIC1006 (In case of a link, the selected key's fields must be selected in the same order
      MAGIC1007 (In case of a link, there must be a key given (except for tables without keys).)  (
      MAGIC1008 (In case of a link, there must be at least one select statement with a locate exp
      MAGIC1009_1 (In case of link write, the init and locate expressions of keys' select stateme
      MAGIC1012 (The order of openend DB files.)  (32)
```

Figure 5. Problems report.



```
:   2. Ver Exp            0
:   3. -- -----------------------------------------------------------
:   4. Call Prg         30 UTIL:Copyright / Verziós P:   1
:   5. Update           17[BR] V:Sorrend              E: 31
:   6. Update           16[BQ] Simított/összes        E: 45
:   7. Update           15[BP] V:Min.hiány napok      E: 33
:
: Task Suffix                   Name
: ==============================================================
:   1. Call Prg       129 NAPLO:KIJELENTKEZÉS         P:   1
:
:
: Task: 1070.1  Árutörzsön végig
:   Updated on 2007/10/04 at 15:13:09
:
```

```
Item name:
 1070.3. Hiány menı
MAGIC1001
 Line: 497
MAGIC1003_1
 Line: 253
 Line: 172
MAGIC1008
 Line: 111
 Line: 122
```

SA Code	Description	No. of warnings	Highlight
MAGIC1001	Hívott taszk/program kevesebb paramétert kezel, mint am...	1	✓
MAGIC1003_1	A hívott taszk/programban a paramétert átvevo paraméter/...	2	✓
MAGIC1008	Link esetén legyen legalább egy select, aminek van locat...	2	✓

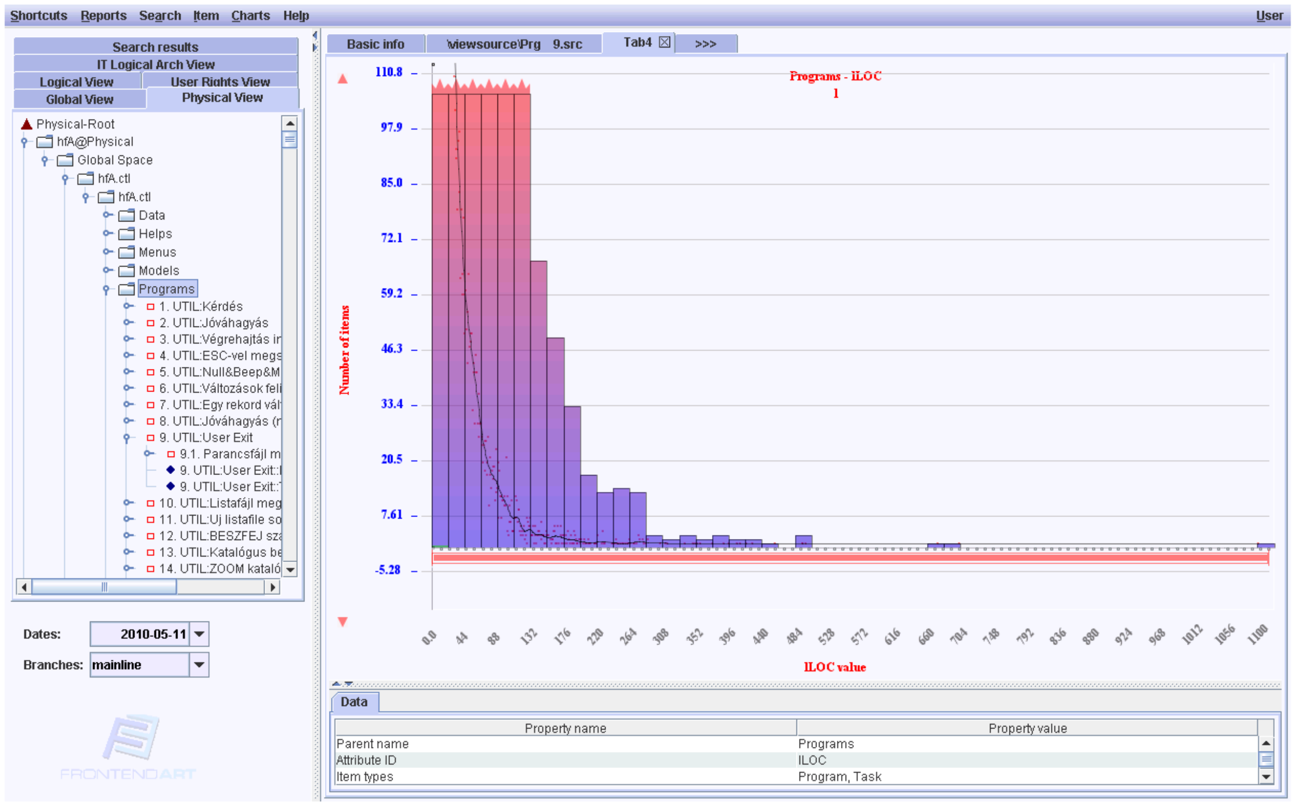
☑ Show all

Figure 6. Source view for Magic.

In the third scenario we learn how the notification emails are sent automatically when a new rule violation appears or when an important metric (*e.g.* complexity) increases in the system (see Table III). First we set up baselines as thresholds for certain metrics and then we configure SourceInventory (where and how to send the notification emails). A sample notification email can be seen in Figure 7.

V. Experiences and lessons learned

We got the first real-life experiences with ***MAGISTER*** from SZEGED Software, a Magic developer company, which is an active user of the system; in addition to the continuous monitoring of the development process, they use it to support



Dates: 2010-05-11

Branches: mainline

Property name	Property value
Parent name	Programs
Attribute ID	ILOC
Item types	Program, Task

Figure 4. Histogram showing the frequency distribution of LLOC metric values of tasks.

Table I
EXAMPLE SCENARIO HOW TO QUERY SAMPLE METRICS OF THE TASKS WITH TOP LLOC METRIC VALUES

Steps	**Expected Outcome**	**Comments**
Step 1: First select the root node in the item tree (on the left hand side of the applet). Then in the wizard of 'Charts/Create histogram' menu, select the LLOC metric, and the Task as 'selected item'.	A histogram showing the frequency distribution of LLOC metric values of tasks (see Figure 4).	Besides the frequency distribution of the LLOC values, other information can be seen in the diagram: average LLOC value, number of items, variance, etc. The critical tasks implementing most statements for business logic can also be easily identified on the right hand side of the diagram.
Step 2: Double click on a bar on the right hand side of the histogram, on the left hand side of the screen the list of the items related to the selected bar will appear. In the 'Charts/Bar chart' menu select McCC, NL, NOI and any other metrics you are interested in.	A bar chart showing the metrics of tasks with top LLOC metric values. Figure 3 shows a screenshot of this.	

migration tasks. The company has over 15 years of Magic application development experience and has excellent professional knowledge. Their complex logistics system specialized for medicine wholesalers has attained outstanding references and the system's market share is over 60% in Hungary. Since we first deployed ***MAGISTER*** at the company, we have had to deal with many technical issues and we have received a lot of positive feedback and helpful advice from the developers. Here we will try to sum up the lessons learned from all of this.

Developers were open-minded and showed special interest in using the tools. Despite scepticism at first, product metrics revealed interesting attributes of their system. There are also recommendations on how to add new Magic-specific metrics to better fit developers views of Magic programs. Surprisingly, the well-known McCabe complexity metric seems to be less useful than with object-oriented languages.

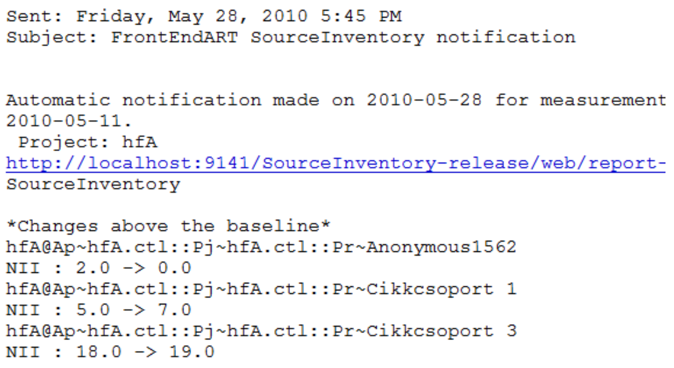
Checking rule violations is without doubt helpful to them. As they admitted, there were more violations than they had expected, and the majority of the problems found had to

Table II
EXAMPLE SCENARIO OF HOW TO QUERY THE CODING RULE VIOLATIONS OF TASKS

Steps	**Expected Outcome**	**Comments**
Step 1: First select the root node in the item tree, and then select the 'Reports/Problems' menu.	A list of coding rule violations in the system being analyzed (Figure 5).	In the problems report wizard you can specify the coding rules you are interested in and you can narrow down your search to specific elements. The maximum number of results can be also specified.
Step 2: Select one rule violation and double click on its location.	Source view (Figure 6).	The source view presents the selected element (usually a task) in a similar way to the application development environment. Hence the developer sees the full context of the coding rule violation and he can easily examine it using ***MAGISTER*** and later fix it within the development environment.
Step 3: Click the 'Show all' checkbox at the bottom of the screen and select those rules which you are interested in. On the right hand side of the screen the list of selected rule violations appear. By clicking on one of them, the source view will move to the exact location of the selected coding violation.	Context of the coding violation. (Highlighted red line in Figure 6.)	

Table III
EMAIL WARNING ABOUT NEW RULE VIOLATIONS AND ABOUT METRICS THAT INCREASED SINCE LAST ANALYSIS (*e.g.* COMPLEXITY)

Steps	**Expected Outcome**	**Comments**
Step 1: Log in into the *admin pages* and set up the baselines of selected metrics.	Configured baselines.	Baselines serve as the limits of maximum allowed metric values. They can be specified based on developers experiences, on literature or on previous measurements.
Step 2: Set up notification SMTP and email address settings.	Configured notifications.	Notification email will be sent from now on every time the system is analyzed by the BuildEngine.
Step 3: Commit a complex task or a rule violation then wait for next analysis and check your mailbox.	Notification email (Figure 7).	



```
Sent: Friday, May 28, 2010 5:45 PM
Subject: FrontEndART SourceInventory notification


Automatic notification made on 2010-05-28 for measurement
2010-05-11.
 Project: hfA
http://localhost:9141/SourceInventory-release/web/report-
SourceInventory

*Changes above the baseline*
hfA@Ap~hfA.ctl::Pj~hfA.ctl::Pr~Anonymous1562
NII : 2.0 -> 0.0
hfA@Ap~hfA.ctl::Pj~hfA.ctl::Pr~Cikkcsoport 1
NII : 5.0 -> 7.0
hfA@Ap~hfA.ctl::Pj~hfA.ctl::Pr~Cikkcsoport 3
NII : 18.0 -> 19.0
```

Figure 7. Example notification email which shows how the NII metric of some tasks increased.

be corrected. They offered several suggestions for easing the handling of rule violations and for improving the quality of the checker. A critical issue is how the exceptions marked by the developers should be handled. The changes report was found very useful, for example in locating a newly appearing error by comparing the previous and the current versions; and in helping to keep track of changes.

When compared to current tools available for Magic developers, a significant advantage is its ability to calculate a wide range of metrics, to query changes and report any rule violations.

VI. CONCLUDING REMARKS

MAGISTER represents a full-fledged framework system that provides useful services for the developers and also users of Magic products. In addition to performing conventional quality assurance tasks (software code measurements using the product metric, coding conventions, violations control, etc.), the system also has a user interface (SourceInventory) that allows developers and project managers to readily evaluate the data obtained from reports, diagrams and so on.

The MagiQuery module assists in compiling queries concerning system objects and the relationship that holds among them, and also in automatically generating program documentation.

The results of ***MAGISTER*** are useful to companies involved in the development or use of Magic/uniPaaS applications; a quality assessment of the various applications may be ordered as a service if necessary.

Availability

MAGISTER is a proprietary software. Further information, including an online demo where the above usage scenarios can be reproduced, is available at http://www.szegedsw.hu/magister/.

Acknowledgements

This research was supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences and the Hungarian national grants GOP-1.1.1-07/1-2008-0081 and OTKA K-73688.

References

[1] J. Martin, *Application Development without Programmers*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1982.

[2] J. Verner and G. Tate, "Estimating size and effort in fourth-generation development," *IEEE Software*, vol. 5, pp. 15–22, 1988.

[3] G. Witting and G. Finnie, "Using artificial neural networks and function points to estimate 4GL software development effort," *Australasian Journal of Information Systems*, vol. 1, no. 2, 1994.

[4] S. MacDonell, "Metrics for database systems: An empirical study," *Software Metrics, IEEE International Symposium on*, vol. 0, p. 99, 1997.

[5] "Homepage of Magic Optimizer," http://www.magic-optimizer.com, last visited 2010.

[6] A. Beszédes, R. Ferenc, and T. Gyimóthy, "Columbus: A reverse engineering approach," in *Pre-Proceedings of IEEE Workshop on Software Technology and Engineering Practice (STEP 2005)*, 2005, pp. 93–96.

[7] T. Bakota, Á. Beszédes, R. Ferenc, and T. Gyimóthy, "Continuous software quality supervision using SourceInventory and Columbus," in *ICSE Companion*, 2008, pp. 931–932.

[8] L. Schrettner, P. Hegedűs, R. Ferenc, L. Fülöp, and T. Bakota, "Development of a methodology, software-suite and service for supporting software architecture reconstruction," in *CSMR '10: Procs. of the European Conference on Software Maintenance and Reengineering*, 2010.