# QORAL: An External Domain-Specific Language for Mining Software Repositories

Hiroki Nakamura, Rina Nagano, Kenji Hisazumi, Yasutaka Kamei, Naoyasu Ubayashi, Akira Fukuda
*Graduate School and Faculty of Information Science and Electrical Engineering*
*744 Motooka Nishi-ku, Fukuoka 819-0395, Japan*
*Email: {hiroki, rina}@f.ait.kyushu-u.ac.jp, nel@slrc.kyushu-u.ac.jp, {kamei, ubayashi, fukuda}@ait.kyushu-u.ac.jp*

***Abstract*—The mining software repositories (MSR) field integrates and analyzes data stored in repositories such as source control and bug repositories to provide support to practitioners. In order to provide useful information to practitioners, MSR researchers need to perform tasks iteratively; these tasks include extracting data from repositories, transforming them into specific data formats, and loading them into the statistical analysis tool. These tasks require a significant amount of man hours to implement and execute according to the requirements of the researchers. This paper proposes an external domain-specific language (DSL) called *QORAL* to facilitate the performance of multiple iterations and environment development . The results from a questionnaire used to evaluate QORAL indicate that it is easy to understand and modify source code.**

## I. Introduction

Software repositories (e.g., version control systems and bug tracking systems) store considerable measurable information about the revision history of source code and issue reports. It is believed that these software repositories contain much useful knowledge about projects, since the real history of a project is stored in them [5]. This idea has motivated researchers to carry out data mining in software repositories (mining software repositories; MSR). Nowadays, performing MSR supports decisions of developers in a project [1]. For example, Shihab et al. [8] reported that frequently modified files tend to have bugs and should be tested carefully. The number of bug fixes in a large file points to the existence of bugs, which has a significant effect on the quality of software.

Generally, MSR is performed in accordance with the procedure depicted in Figure 1. MSR researchers perform multiple iterations from steps 1 to 3 until they obtain useful knowledge pertaining to the development of the software.

1. **Data Extraction:** Extraction of the data from the software repositories
2. **Data Transformation:** Parsing of data and transforming it into a specific data structure that can be easily handled
3. **Data Loading:** Data Loading: Loading of data from step 2 into a statistical analysis tool (e.g., R [4] and WEKA [2]) and analyzing it

One of the major problems in the MSR field is the fact that the steps require a significant amount of man hours to implement and execute according to requirements of the researchers. This is because there are many factors to examine and compare in a typical MSR analysis. For example, comparing particle sizes (e.g., a package versus a file of a unit of source code) and comparing periods (e.g., version 1.0 versus version 2.0). We believe that a light-weight domain-specific language (DSL) to specify requirements in the MSR steps and an environment in which to execute them would help reduce the turnaround time of these steps. However, to the best of our knowledge, there is no such lightweight language and environment to support MSR researchers.

The goal of this study is to resolve the problem How can researchers perform multiple iterations easily? To this end, we propose and implement a lightweight external DSL called the Quick ORes Acquisition Language (*QORAL*) and an environment to automatically generate executable source code from a QORAL description. Specifically, we create a script language for easily implementing, understanding, and modifying source code for MSR. QORAL enables us to execute the MSR steps with fewer lines of code than general-purpose programming languages (e.g., Java and Python).

To evaluate whether researchers can understand and modify QORAL descriptions easily, we sent questionnaires to thirteen experienced software developers without explaining QORAL grammar to them. To evaluate their ability to understand the language, we showed the subjects source code written in QORAL, and they responded by indicating how well they understood the source code. We then used the average number of persons who answered correctly. To evaluate their ability to modify source code, we showed them a description of QORAL, and asked them to modify the source code by following a set of instructions. We then used the average number of subjects who had modified the source code correctly. At the end of the experiment, it was found that the average number for understandability was 9.33, while the average for ability to modify the source code was 10.50. These results indicate that QORAL facilitates understanding and easy modification.

The rest of this paper is organized as follows: Section II describes our proposed external DSL QORAL, while Section III evaluates and discusses it. Section IV discusses threats to the validity of the results in this paper, and Section V presents our conclusions.

978-0-7695-4866-1/12 $26.00 © 2012 IEEE
DOI 10.1109/IWESEP.2012.20

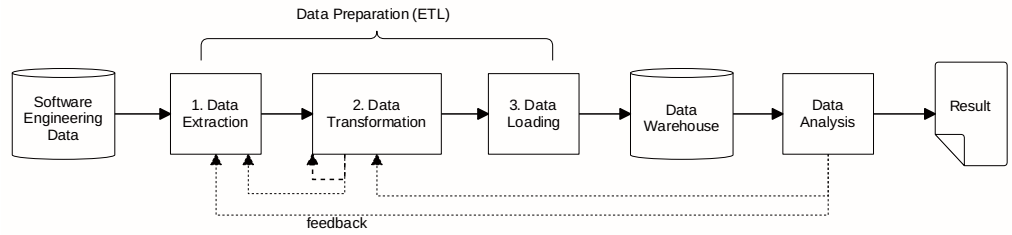


Figure 1. Procedure of MSR [7]

## II. QORAL : QUICK ORES ACQUISITION LANGUAGE

In this paper, we propose an external DSL QORAL and an environment for automatically generating executable source code from QORAL description in order to resolve the problem,• •How can researchers perform multiple iterations easily?• •DSL is a language that is specialized to a specific domain. Source code written in a DSL is adapted to the specific domain and is at a high abstraction level. As a result, productivity, reliability, and reusability are improved when a DSL is used. The DSL user can implement the source code by using technical terms from the domain, and so it is easy for domain experts to implement, understand, and modify the source code. Therefore, we consider a DSL to be an effective way to resolve the problem.

QORAL enables MSR researchers to express the metrics (the data to calculate) using less source code. In other words, QORAL has a high abstraction level. Therefore, MSR researchers can implement the source code easily without having to know how to parse the text data or the algorithm to calculate metrics.

### A. QORAL Overview

QORAL is a lightweight language for MSR studies with little expression. More specifically, it is a script language that MSR researchers can use to easily implement, understand, and modify source code for MSR studies. In this study, we constructed an environment for writing QORAL descriptions and for generating executable source code from the QORAL descriptions, as shown in Figure 2.

Because MSR's objective is to extract pertinent information and/or uncover relationships or trends about a particular evolutionary characteristic, two kinds of outputs are needed

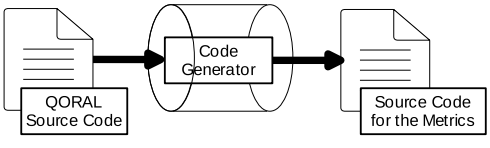


Figure 2. Code Generator

Table I
TARGET METRISCS

| Metrics | Keywords |
|---|---|
| File name | EditedFiles |
| Package name | EditedPackages |
| Author name | Authors |
| Commit | CommitNum |
| Time unit | edit Time EACH *TimeUnit* |

$TimeUnit$:Year, Month, Day, Hour, Minute or Second

Table II
AMOUNT METRICS

| Metrics | Keyword |
|---|---|
| # of packages | EditedPackages |
| # of files | EditedFiles |
| # of authors | Authors |
| # of lines added | AddedLines |
| # of lines deleted | DeletedLines |
| # of editing | EditCount |
| # of commit | CommitNum |

[1]. Therefore, we define two outputs: The first is a list consisting of certain factors, while the other is an intensive value in QORAL. For example, in the list consisting of factors, it is possible to output a list showing who edited files, which files were edited, and how many times each file was edited. In the intensive value output, it is possible to output the number of times a file was edited by a specified author. In QORAL, when MSR researchers obtain the list consisting of factors, they can express the metrics in Table I. We call them• •Target Metrics.• •

On the other hand, when MSR researchers obtain the intensive value output, they can express the metrics in Table II. We call them• •Amount Metrics.• •

Table III lists the metrics defined in QORAL. Each row shows the Amount Metrics for calculation while each column shows the Target Metrics for setting the condition. The point where they intersect shows the value of the metrics in the condition. An asterisk• •*• •field in the table indicates that MSR researchers want to know the value of that field. For example, the number of files in the row for each author is a meaningful value. However, the number of files in each edited file is 1. Thus, there are fixed numbers in some

combinations of the conditions.

We define the combination of the value 0 or 1 in Table III as an error in QORAL. Therefore, a case where metrics include the combination expressed is an error. For example, the combination of authors in each package and commit is an error since the combination of authors in each commit is always 1, since a commit is performed by a single author in general log data.

### B. QORAL Grammar

To improve implementation facility, a row is made to express a list or a value to an output.

*1) List Consisting of Some Factors:* List Consisting of Some Factors: The grammar is as follows:

$$FileName = List\ TargetMetrics; \quad (1)$$

$FileName$: name of generated files
$List$: a set phrase (which is needed to get lists)

It is possible to set an arbitrary number of metrics in Target Metrics, and to add Amount Metrics for them. Let us now look at an example of how to obtain the list consisting of some factors. The following QORAL description is given to obtain the list of who edited, which files were edited, and how many times they were edited.

$$\mathrm{resultA} = \mathrm{List\ Authors,\ EditedFiles\ EACH\ EditCount}; \quad (2)$$

From this description, a file called resultA.cu is generated by the QORAL environment. If researchers run the generated file (i.e., resultA.cu), they obtain the list shown in Table IV.

*2) One Intensive Value:* The grammar is as follows.

$$FileName = AmountMetrics; \quad (3)$$

In addition, researchers can obtain the value under certain conditions if they want to add the following optional factors:

- Files of packages:
  By adding the keyword "in file xx/xx/xx/xx/xx.java," it is possible to target the file xx/xx/xx/xx/xx.java. If the keyword "in package xx/xx" is added instead of the keyword to set a file, it is possible to target all the files in package xx/xx. The keyword we provided is essential for calculating AddedLines or DeletedLines.
- Metrics co-changed with a file or a package:
  By adding the keyword • •Co-change WITH xx, • •it is possible to obtain metrics values that have been co-changed with a file or a package.
- Editing type:
  By adding the keyword• •typeof BugFix,• •it is possible to target the commit whose editing type is BugFix. In this study, the editing types in the log data we use are BugFix and unknown.
- Authors:
  By adding a keyword as• •by author xx,• •it is possible to target the files edited by author xx.
- Scope of time:
  By adding a keyword as • •before yyyy-MM- dd, hh:mm:ss, • •it is possible to target files that were edited before a specified time. Similar to the keyword • before, • •by adding a keyword such as• •after yyyy-MM-dd, hh:mm:ss,• •it is possible to target the files that were edited after a set time. The input is year, month, day, hour, minute, and second in that order. This input can also be written in terms of only year, month, and day. Furthermore, by using both• •before• •and• •after,• • it is possible to target the files before a specified time and after another set time.

The following QORAL description is used to obtain the number of files in the package src/home edited by Ken after January 1, 2012.

$$\mathrm{resultB} = \mathrm{EditedFiles\ in\ package\ src/home\ by\ Author\ Ken}$$

$$\mathrm{after\ 2012\text{-}1\text{-}1}; \quad (4)$$

### C. Implementation

In this section, we present an environment for generating executable source code from QORAL descriptions. We parse a QORAL description and generate executable source code that is in accordance with it.

QORAL is implemented using Xtext, which supports the development of an external DSL running on Eclipse [10]. Using Xtext, an external DSL can be developed based on grammar such as Backus?Naur Form (BNF). If the DSL grammar is defined, it is possible to open an editor tool to support the grammar. In the editor, users can write source code by following the grammar. As per Eclipse • s• functionality, keywords used on the grammar appear colored. Furthermore, the source code is automatically checked to verify whether it satisfies the grammar.

We developed the code generator using Xtend that works with Xtext. Xtend is a programming language that determines how to generate a deliverable from source code written in grammar defined using Xtext [9]. The source code written in the defined grammar is not executable because developers define only the grammar. The source code written in grammar defined using Xtext contains keywords that have to be written and arbitrary strings or numbers that the DSL user inputs. In Xtend, developers define how to generate source code corresponding to these keywords. They also embed arbitrary strings or numbers written in the source code in the source code being generated. By defining in this way, the executable source code is generated.

Using these tools, developers are able to output arbitrary source code. The environment is established using only the installation in Eclipse. Figure 3 shows a screenshot of the development environment for QORAL that was developed using Xtext and Eclipse. The source code generated from QORAL calls the library for MSR. The library was made

Table III
VALUES FOR EACH FACTOR

| Number of | EditedPackages | EditedFiles | Authors | EditedLines | EditCount | CommitNum |
|---|---|---|---|---|---|---|
| each EditedFile | 0 | 1 | • • | • • | • • | • • |
| each EditedPackage | • • | • • | • • | • • | • • | • • |
| each Author | • • | • • | 1 | • • | • • | • • |
| each Commit | • • | • • | 1 | • • | • • | 1 |
| each TimeUnit | • • | • • | • • | • • | • • | • • |
| each typeof | • • | • • | • • | • • | • • | • • |
| each Co-change | • • | • • | 1 | • • | • • | • • |
| Time Unit | • • | • • | • • | • • | • • | • • |

*Edited Lines*:AddedLines or DeletedLines

Table IV
EXAMPLE OF LIST

| File | Author | AddedLines |
|---|---|---|
| a/app/apple.java | Ken | 53 |
| a/app/apple.java | Tom | 125 |
| c/co/context.java | Tom | 234 |
| m/mobile.java | Ken | 50 |

by our research group [6]. It is used to parse the log data and to calculate the metrics by parallel processing with GPGPU (General-Purpose computing on GPU).

We designed the code generator that generates the source code calling library from QORAL to facilitate implementation of the source code for MSR. In this design, when the library • s•content has changed and the library itself has changed, QORAL can correspond to such changes by only relating to the specific part in the code generator. Therefore, if there are changes in the library, the user can rewrite QORAL grammar.

## III. EXPERIMENTAL EVALUATION

### A. Overview

In order to evaluate the ability of users to understand and modify source code, we sent out questionnaires to persons experienced in developing software, since persons would need to write some source code in order to perform MSR studies. To evaluate whether the subjects could easily understand and modify QORAL descriptions, we gave them no information about the grammar. We asked students in our laboratory as well as our friends on Facebook to answer the questionnaire. We used Facebook as a new experiment because Social Networking Service (SNS) is rapidly spreading nowadays.

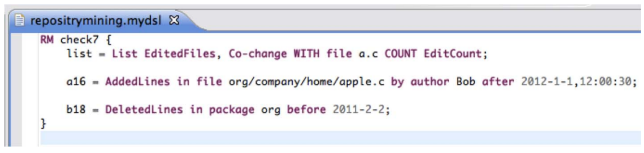


Figure 3. An example of source code written in QORAL

### B. Questionnaire

In the questionnaire, because the subjects were not MSR specialists, we used all the metrics except those unique to MSR. For example, the files or commits co-changed with a certain file. Specifically, the subjects answered the following questionnaire:

Q1:Describe your understanding of the result of the following description:

$$\text{lines} = \text{AddedLines by author Nakamura}$$
$$\text{before 2011-11-27;} \quad (5)$$

Q2: Describe your understanding of the result of the following description:

$$\text{authorsNum} = \text{Authors in file apple.java;} \quad (6)$$

Q3: Describe your understanding of the result of the following description:

$$\text{list} = \text{ListEditedFiles, Authors, EditTime EACH Month}$$
$$\text{after 2011-11-27;} \quad (7)$$

Q4:Modify the following source code to target the files edited after 12th March 1990.

$$\text{count} = \text{EditCount by author Sonoda}$$
$$\text{before 2000-12-31;} \quad (8)$$

Q5:Modify the following source code to calculate the number of authors, and not the number of the lines added.

$$\text{num} = \text{AddedLines in package src/example/program}$$
$$\text{before 2000-12-31;} \quad (9)$$

The objective in using Q1, Q2, and Q3 was to evaluate the understanding of the subjects, and so we asked the subjects to evaluate how easily they understood the descriptions and gave them a choice of five grades. Q4 and Q5 were used to evaluate how well the subjects could modify source code. We made the same requests of the subjects as before.

### C. Results and Discussion

In this section, we show the results obtained for the questionnaires in Tables V and VI. Each table shows the right and wrong answer in each line and the evaluation ranging over five grades in each column. The intersection point shows the number that applies to the right/wrong answers and the evaluation. Empty fields signify 0.

Table VII gives a summary of the results. It shows the percentage of persons who answered correctly, of the thirteen subjects. The number in the evaluation column is the average score evaluated by the thirteen subjects.

First, let us focus on the facility to understand. The results of Q1, Q2, and Q3 are that more than nine subjects answer correctly. In the evaluation, we focused on the number that answered in the affirmative and gave grades above three in the evaluation. In Q1 and Q2 about one intensive value, the results were 10 and 9. In Q3 about the list consisting of some factors, the result was 6. Therefore, the facility to understand is not low but still needs improvement. Next, let us focus on the facility to modify. The results for both Q4 and Q5 are that more than ten subjects answer correctly. Almost all the subjects evaluated it above grade three. Therefore, we can conclude that QORAL facilitates modification. However, there is still room for improvement• just as in the facility to understand.

From these results, it can be seen that QORAL has good facility for understanding and modification. However, we can improve these numbers by referring to other languages and looking at the keywords used in their grammar and again referring to other papers for MSR.

Table VIII shows the result of using QORAL as opposed to not using QORAL in terms of the Lines of Code (LOC) incurred and the implementation time. It is clear to see that implementation is easier when QORAL is used.

### D. Related Work

Shang et al. [7] reported that most of the problems in the step to prepare data for MSR have already been faced on the World Wide Web (WWW). On the WWW, platforms have been developed to prepare and facilitate the processing of large datasets. For example, Pig and Hadoop. Shang et al. used Pig to scale up MSR. They demonstrated the effect of using Pig by comparing it with the use of Hadoop. The study done using Pig needed special programing [7]. In contrast, using QORAL enables the analysis of different metrics repeatedly and easily because it is easy to understand and modify QORAL descriptions, as evidenced in Section III.

Hindle and German [3] pointed out that the defect in the MSR field is the difficulty handling revision histories stored in software repositories. To handle these histories, researchers have to transform them to a model or a diagram. They then presented a demonstration model to express their proposed Source Control System (SCS) and defined a query language, Source Control Query Language (*SCQL*), using it. They also ran three queries on five projects to evaluate SCQL and to demonstrate its effect.

It is easy to understand SCQL • s• grammar because it corresponds to logical equations. For example, the following logical equation means that there is an author whose changes stay within one directory:

$$\begin{aligned} &\forall a \in \mathbb{A}uthors\ s.t. \\ &\forall f \in isAuthorOf(a, f) \Rightarrow \\ &\forall f_2 \in isAuthors(a, f_2) \Rightarrow \\ &directory(f) = directory(f_2) \end{aligned} \tag{10}$$

This logical equation is equivalent to the following SCQL code:

Listing 1. SCQL example

```
E(a, Author) {
    A(f, author.files) {
      A(f2, author.files) {
        eq(f.directory, f2.directory)
      }
    }
}
```

SCQL has two kinds of outputs: a check to see if the metrics exists, and a calculation of the number of metrics. There is not a lot of output as MSR • s•purpose is to extract pertinent information and/or uncover relationships or trends about a particular evolutionary characteristic [1].

In contrast, in QORAL, researchers can express the output as a list consisting of some factors and one intensive value; so it is better.

## IV. Limitations and Threats to Validity

QORAL supports the metrics shown in Section II only. In the MSR field, many metrics have been proposed [1], so it is not possible to implement all these metrics. To generalize our study, we need to design an environment that is able to extend the language easily.

The executable source code generated by QORAL description and the types of usable repositories depends on the library for MSR [6]. Therefore, QORAL covers only the commit log of the CVS repositories.

## V. Conclusion

This paper proposed two means of resolving the MSR problem: QORAL, a lightweight DSL for MSR, and the QORAL environment. We showed that the MSR problem lies in the fact that a lot of time is spent on steps taken in MSR to perform multiple iterations, to implement and modify the source code, and to extract, transform, and load the data. To resolve this problem, we developed the DSL QORAL and an environment where people can easily implement, understand, and modify source code. To evaluate

Table V
DISTRIBUTION OF RIGHT/WRONG ANSWERS AND EVALUATION IN Q1 TO Q3 FOR THE FACILITY TO UNDERSTAND

| | distribution | 5 | 4 | 3 | 2 | 1 | sum | % of right answers | each average of evaluation | average of evaluation |
|---|---|---|---|---|---|---|---|---|---|---|
| $Q1$ | right | 3 | 3 | 4 | | | **10** | | 3.85 | |
| | wrong | | 1 | 1 | 1 | | 3 | | $(= \frac{5\times3+4\times4+3\times5+2\times1+1\times0}{13})$ | |
| $Q2$ | right | 1 | 3 | 3 | 2 | | **9** | 71.8 | 3.08 | 3.49 |
| | wrong | | | 2 | 2 | | 4 | $(= \frac{10+9+9}{13+13+13} \times 100)$ | $(= \frac{5\times1+4\times3+3\times5+2\times4+1\times0}{13})$ | $(= \frac{3.85+3.08+3.54}{3})$ |
| $Q3$ | right | | 4 | 2 | 1 | 2 | **9** | | 3.54 | |
| | wrong | | | | 3 | 1 | 4 | | $(= \frac{5\times0+4\times4+3\times2+2\times4+1\times3}{13})$ | |

Table VI
DISTRIBUTION OF RIGHT/WRONG ANSWERS AND EVALUATION IN Q4 AND Q5 FOR THE FACILITY TO MODIFY

| | distribution | 5 | 4 | 3 | 2 | 1 | sum | % of right answers | each average of evaluation | average of evaluation |
|---|---|---|---|---|---|---|---|---|---|---|
| $Q4$ | right | 6 | 3 | 1 | | | **10** | | 4.31 | |
| | wrong | 1 | | 2 | | | 3 | 80.8 | $(= \frac{5\times7+4\times3+3\times3+2\times0+1\times0}{13})$ | 4.04 |
| $Q5$ | right | 4 | 3 | 4 | | | **11** | $(= \frac{10+11}{13+13} \times 100)$ | 3.77 | $(= \frac{4.31+3.77}{2})$ |
| | wrong | | | 1 | 1 | | 2 | | $(= \frac{5\times4+4\times3+3\times5+2\times1+1\times0}{13})$ | |

Table VII
THE RESULT OF THE FACILITY OF UNDERSTANDING AND MODIFYING

| the number of | % of right answers | evaluation |
|---|---|---|
| the facility of understanding | 71.8 | 3.49 |
| the facility of modifying | 80.8 | 4.04 |

Table VIII
COMPARISON

| Language | LOC | Time(min) |
|---|---|---|
| non-QORAL | 539 | 1140 |
| QORAL | 2 | 3 |

the facility to understand and modify using QORAL, we sent out questionnaires to thirteen experienced software developers. With regards to the facility to understand, the average number who replied correctly was 9.33, while those for the facility to modify was 10.50 on average. The average evaluation was 3.10 for the facility to understand and 3.42 for the facility to modify. From these results, it is clear that QORAL is effective in terms of the facility to understand and modify.

Our future work is as follows. We want to improve QORAL to be able to add new metrics easily because many metrics exist. We also want to enable the ability to make conditions for each metrics. For example, the number of modules edited more than 10 times, the number of authors less than 15, etc. Finally, we want to modify QORAL • s• grammar to improve the facility to understand and modify.

## ACKNOWLEDGMENT

This research was conducted as part of the Core Research for Evolutional Science and Technology (CREST) Program, Software development for post petascale supercomputing • •Modularity for supercomputing, • •2011, by the Japan Science and Technology Agency, and a Grant-in-Aid for Young Scientists (A), 24680003, 2012, by the Japan Society for the Promotion of Science.

## REFERENCES

[1] Cagatay Catal and Banu Diri. "A systematic review of software fault prediction studies". *Expert Systems with Applications*, Vol. 36, No. 4, pp. 7346–7354, 2009.

[2] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. "The WEKA data mining software: an update". *SIGKDD Explor. Newsl.*, Vol. 11, No. 1, pp. 10–18, November 2009.

[3] Abram Hindle and Daniel M. German. "SCQL: a formal model and a query language for source control repositories". *SIGSOFT Softw. Eng. Notes*, Vol. 30, No. 4, pp. 1–5, May 2005.

[4] Ross Ihaka and Robert Gentleman. "R: A Language for Data Analysis and Graphics". *Journal Of Computational And Graphical Statistics*, Vol. 5, No. 3, pp. 299–314, 1996.

[5] Huzefa Kagdi, Michael L Collard, and Jonathan I Maletic. “A survey and taxonomy of approaches for mining software repositories in the context of software evolution”. *Journal of Software Maintenance and Evolution Research and Practice*, Vol. 19, No. 2, pp. 77–131, 2007.

[6] Rina Nagano, Hiroki Nakamura, Yasutaka Kamei, Kenji Hisazumi, Naoyasu Ubayashi, and Akira Fukuda. “Using the GPGPU for scaling up mining software repositories”. *International Conference on Software Engineering(ICSE ’12)*, pp. 1435–1436, 2012.

[7] Weiyi Shang, Bram Adams, and Ahmed E. Hassan. “Using Pig as a data preparation language for large-scale mining software repositories studies: An experience report”. *Journal of Systems and Software*, 2011.

[8] Emad Shihab, Audris Mockus, Yasutaka Kamei, Bram Adams, and Ahmed E. Hassan. “High-impact defects: a study of breakage and surprise defects”. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, FSE ’11, pp. 300–310, 2011(to appear).

[9] “Writing a Code Generator With Xtend”. http://www.eclipse.org/Xtext/documentation.html#generator. Accessed: 27/06/2012.

[10] “Xtext - language development made easy”. http://www.eclipse.org/Xtext/. Accessed: 27/06/2012.