Language Interaction and Quality Issues: An Exploratory Study

Antonio Vetro', Federico Tomassetti, Marco Torchiano, Maurizio Morisio

Politecnico di Torino,
Dept. of Computer and Control Engineering
C.so Duca degli Abruzzi 24
Torino, Italy
{firstname.lastname}@polito.it

ABSTRACT

Most software systems are complex and composed of a large number of artifacts. To realize each different artifact specific techniques are used resorting to different abstractions, languages and tools. Successful composition of different elements requires coherence among them. Unfortunately constraints between artifacts written in different languages are usually not formally expressed nor checked by supporting tools; as a consequence they can be a source of problems. In this paper we explore the role of the relations between artifacts written in different languages by means of a case study on the Hadoop open source project. We present the problem introducing its terminology, we quantify the phenomenon and investigate its relation with defect proneness.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics – *product metrics.*

General Terms

Measurement, Experimentation, Languages.

Keywords

Languages interaction, cross language modules, polyglot programming.

1. INTRODUCTION AND BACKGROUND

Most software projects nowadays are polyglot, i.e. files written using different languages interact with each other. Wampler et al. [1] introduced a special issue on this topic writing "Most teams are by necessity MPP [Multi-Paradigm programming] teams now. No one writes in a single language anymore. Even trivial applications have a general-purpose language, SQL, JavaScript, CSS, and dozens of frameworks, each of which includes an external DSL [Domain Specific Language] (usually in XML) that is its own mini language (the syntax is XML, but the XMLSchema defines the semantics)".

Given this scenario our team seek to study the effects of language interaction and eventually evolve development techniques and supporting tools to consider these aspects. Nowadays tools used by developers help them only to verify the consistency internal to a language, i.e. consistency within a set of artifacts written in the same language. For example, editors check that an expression in Java code invokes a Java method which exists in the codebase, either in the same file or in another Java file. On the other hand there are major limitations in verifying the consistency across the language boundaries. For example can tools help the developer to understand immediately if a piece of XML code used for configuration refers to a really existing Java class? Normally currently available tools cannot do this because they are not aware of the cross-language semantics.

While the issue of language interaction is already very relevant today, the appearance of language workbenches [2] let us suppose that this issue is going to become even more important in the future. For example, with Xtext [3] and GMF [4] we can create, textual and graphical DSLs with custom editors integrated in the Eclipse platform with a minimal effort. Other tools like Intentional Software [5] and the Meta-Programming System [6] fully support the Language Oriented Programming paradigm [7] and are based on projectional editing. The existence of these tools and their usage in industrial projects [8] seem to indicate that the interaction between languages in projects will increase in the future.

Pfeiffer et al. [9] conducted a study related to language interaction. They realized a tool named *GenDeMoG* to mine inter-languages interaction based on text analysis. Their work was motivated by observing the amount of errors introduced by undocumented relations that cross the language border (i.e., they involve modules written in different languages) and the resulting complexity.

Our hypothesis is that in the long run we need to support cross language development, including design, modeling, and validation. To reach this goal we first need to start understanding the effects of languages interaction: this work is intended as a first step in that direction.

2. DEFINITIONS

Before stating our goals and translating them into actionable research questions, we define how we do identify and measure the languages interaction. We provide here a list of definitions used throughout the rest of the paper.

Module: we considered a module each single file.

Antonio Vetro', Federico Tomassetti, Marco Torchiano, and Maurizio Morisio. 2012. Language interaction and quality issues: an exploratory study. In Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement (ESEM '12). ACM, New York, NY, USA, 319-322. DOI=10.1145/2372251.2372309 http://doi.acm.org/10.1145/2372251.2372309

We consider a commit[1] as a unit of work, consequently we suppose that files committed together are related.

Intra-language commit (*ILC*): a commit containing a set of modules with the same extension.

Cross-language commit (*CLC*): a commit containing modules with different extensions.

Cross-language commit for an extension (CLC_{ext}): a CLC containing that includes modules with the extension *ext*.

Defect fix: a commit executed to fix a defect.

We consider a module to be *cross language* when it is related to modules written in a different language (e.g., a Java file loading the configuration from an XML file). To measure how much a module is cross language we analyze its history: if the module was frequently committed with files written in other languages we consider that as an indicator of interaction between the module and those files. This interaction is measured through different variants of the *cross language ratio* (CLR).

Cross language ratio of a module (CLR_m): the CLR of a module *m* is the fraction of cross-language commits in which *m* was involved with regard to the total number of commits regarding the module (both intra-language and cross-language):

$$CLR_m = \frac{\#\ CLC}{\#\ CLC + \#\ ILC}$$

Cross language ratio of a module with regard to an extension ($CLR_{m,ext}$): the CLR of a module *m* considering as CLC only the commits involving *m* and a module with extension *ext*:

$$CLR_{m,ext} = \frac{\#\ CLC_{ext}}{\#\ CLC_{ext} + \#\ ILC}$$

Cross language ratio of an extension (CLR_{ext}): for each extension *ext* we compute its cross language ratio as the mean of the CLR_m considering all modules having extension *ext*:

$$CLR_{ext} = \frac{\sum CLR_m\ , m \in ext}{\#\ *.ext}$$

Cross language ratio of an extension *extA* with respect to an extension *extB* ($CLR_{extA,extB}$): the mean of $CLR_{m,extB}$ among all modules *m* with extension *extA*:

$$CLR_{extA,extB} = \frac{\sum CLR_{m,extB}\ , m \in extA}{\#\ *.extA}$$

Cross Language Module (CLM): a module is cross language if its CLR is $\geq t_{CLM}$%, where t_{CLM} is a threshold to be defined.

Intra Language Modules (ILM): a module is intra language if its CLR is $< t_{ILM}$%, where t_{ILM} is a threshold to be defined.

3. GOALS, RESEARCH QUESTIONS AND METRICS

The goal of this preliminary study is two-fold. Firstly we investigate the level of languages interaction in a common project. Secondly, we verify whether the level of interaction is related to quality problems. We look at defects as a proxy of software external quality. We identify two research questions related to the first goal.

RQ1 How much interaction is there among the languages present in a project?

The interaction is computed as the percentage of CLC among a set of commits. First we consider all type of commits (RQ1.1), then (RQ1.2) we consider separately the commits related to a particular activity (e.g., improvement, bug fixing, new feature).

Once we have defined the size of the phenomenon by answering to RQ1, we will go deeper considering the behavior of each single extension.

RQ2 Which extensions interact more?

The second research question is answered at two levels, i.e. firstly investigating the relationship between one extension versus all the other extensions (RQ2.1), then analyzing the most interacting pairs of extensions (RQ2.2).

We answer RQ2.1 computing the CLR_{ext} for each extension, while we answer RQ 2.2 computing the $CLR_{extA,extB}$ for all pairs of extensions.

The last research question is related to the second goal, i.e. investigating whether a high interaction between languages might result in higher defect proneness.

RQ3 Are Cross Language Modules more defect-prone?

We answer RQ 3 computing the number of Cross Language Modules (CLM) with and without defects, and the number of Intra Language Modules (ILM) also with and without defects. Then we compare the two proportions with/without defects by means of the F-test to see whether the proportion of Cross Language Modules with defects is different from the one of Intra Language modules.

This metric is computed at three granularity levels:

- considering all files regardless of their extension (RQ3.1),
- considering for each single extension its level of interaction with all the other extensions as aggregate (RQ3.2),
- considering interaction between specific ordered pairs of extensions (RQ3.3).

4. CASE STUDY

This exploratory study aims at understanding the phenomenon of language interaction and derived quality issues. We also use it to investigate whether the methodology defined above is applicable. We selected as a case study Apache Hadoop[2], which is a set of libraries to support distributed data processing. We selected Hadoop because it is a mature project (it is supported since April 2006) and it is used in many industrial applications (e.g., Yahoo, and Facebook).

Our methodology for computing the metrics defined above is based upon the fact that Hadoop uses SVN[3] to manage artifacts versions and JIRA[4] to track not only defects but any other activity that can be associated with software artifacts. Those elements are called "JIRA issues", and each project has its own set of issues. Example of JIRA issues are the implementation of a new feature, a single implementation task, a bug report, and so on. Hadoop developers established links between commits in the SVN code repository to JIRA issues by systematically including issue ids in their SVN commit comments.

We downloaded the SVN log from the Hadoop repository (last revision retrieved is the 1233090, from 01/18/2012, the first

[1] We refer to the term commit as used in the context of version control systems.

[2] http://hadoop.apache.org

[3] http://subversion.tigris.org/

[4] http://www.atlassian.com/software/jira/overview

Table I. Percentage of cross language commits (RQ 1)

All	Bug	Improvement	New Feature	Sub task	Task	Test
0.53	0.12	0.26	0.30	0.45	0.26	0.05

Table II. CLR_{ext} (RQ 2.1)

CLR_{ext}	Nr files	Extension
0.96	49	c
0.87	114	sh
0.72	75	properties
0.71	320	xml
0.59	4328	java

Table III. $CLR_{extA,extB}$ (RQ 2.2)

extA/extB	C	Java	Properties	Sh	XML
C	-	0.51	0.10	0.50	0.83
Java	0.01	-	0.28	0.04	0.48
Properties	0	0.54	-	0.36	0.46
Sh	0.09	0.22	0.24	-	0.47
Xml	0.04	0.52	0.43	0.24	-

Table IV. Odds ratio of the defectivity in respect to the relation between pairs of extensions (RQ 3.3)

	C	Java	Properties	sh	XML
C	-	**Inf**	**0**	**0**	**Inf**
Java	2.79	-	0.32	0.43	**0.96**
Properties	**Inf**	**1**	-	12.08	**0.94**
Sh	**3.55**	4.45	17.17	-	7.44
Xml	3.83	**0.95**	3.22	4.73	-

available revision is the 776174 from 5/19/2009). We also extracted all JIRA issues from the Apache JIRA database.

We computed all modules CLR_m and observed their distribution: about 30% of modules have CLR_m between 0 and 0.1, and about 55% files have CLR_m between 0.9 and 1. Given these percentage and given that the remaining files have a positive (right) skewed distribution, we decided to use as thresholds $t_{CLM}=t_{ILM}=50\%$ to define CLM and ILM modules.

5. RESULTS AND DISCUSSION

Table I reports the percentage of cross language commits in the Hadoop repository: 53% of all commits (first column) are CLC, i.e. containing files of different languages. Looking at the portion of CLC related to the different activities (i.e., JIRA issues), we observe that their percentage varies with respect to the type of issue (from 2nd to last column in Table I). It goes from a minimum of 5% in commits related to Test up to a maximum of 45% in Sub Tasks (since not all issues are linked to JIRA issues, the mean "All" in the first column is not related to the other means in the following columns).

RQ 1.1 answer: *the 53% of commits in Hadoop are cross language.*

RQ 1.2 answer: *looking at the single activities, we derive that writing/modifying tests or fixing bugs are activities that involve mainly a single language, while adding new features is an activity that involves multiple types (or at least extensions).*

We now proceed to RQ 2.1 and 2.2. Table II contains the top 5 extensions in terms of number of files: c, sh, properties, xml and java. Among them, four extensions correspond to programming languages and one is used for configuration files. Subsequently, we compute the $CLR_{extA,extB}$ for all combinations of the five extensions . Table III reports the $CLR_{extA,extB}$.

RQ 2.1 answer: *all most common extensions in Hadoop are highly interacting with other extensions (i.e., $CLR_{ext} > 0.50$).*

RQ2.2 answer: *the most frequent interactions ($CLR_{extA,extB} \geq 0.50$) are: C-XML (0.83), Properties-Java (0.54), XML-Java (0.52), C-Java (0.51), C-sh(0.50). Border values are: Java-XML (0.48), sh-XML (0.47) Properties-XML (0.46), and XML-Properties (0.43).*

We observe that the only pairs with frequent interactions in both directions are Java-XML and Properties-XML. All the other pairs have frequent interactions in only one direction. For instance, CLR_{XML-C} = 0.04 and CLR_{C-XML}=0.83 means that most of the commits involving C contain also XML files, but not the other way around.

We now focus on the last RQ, i.e. on the relation between languages interaction and defect proneness. Table V contains metrics to answer RQ 3.1 (first line) and RQ 3.2 (from 2nd to last line). The following columns contain, in the order: the number of ILM with no defects and then with at least one defect, the number of CLM with no defects and then with at least one defect, the p-value of the F-test and finally the odds ratios (which is greater than 1 when CLM are more defect prone than ILM).

RQ 3.1 answer: *considering all extensions, ILM are more defect prone that CLM (about 5 times less).*

RQ 3.2 answer: *considering the five most common extensions, we observe that three extensions (XML, Properties and C) have CLM with higher defect proneness, while two extensions (Java and Sh) exhibit the opposite relation.*

Among the above differences, only *all extensions* and *Java* are statistically significant (p-value ≤ 0.05).

Finally, Table IV contains the odds for each pair of extensions to answer to RQ 3.3. We report in bold the values for which we obtained a p-value ≤ 0.05. We observe 7 pairs for which ILM are less defect prone than CLM, 12 pairs with CLM more defect prone than ILM and one pair with odds ratio =1. We consider only values with p-value ≤ 0.05 to answer RQ 3.3.

RQ 3.3 answer:

four extension pairs have CLM more defect prone then ILM (C-Java, C-XML, Properties-C, Sh-C),

five extension pairs have ILM more defect prone then CLM (C-Properties, C-sh, Java-XML, Properties-XML, XML-Java)

one extension pair have exactly same defect proneness (Properties-Java).

We notice that interactions where CLM results more defect prone involve always the C files. While interactions where ILM results more defect prone involve mainly XML, however C is also present. An interesting fact is that the pair Sh-C is in the first set, the pair C-sh is in the second.

Table V. Relation between classification in ILM and CLM and presence of defects (RQ 3.1 and 3.2)

	RQ	MN	MY	CN	CY	P	Odds
all	**2**	**1891**	**225**	**2875**	**89**	**0.000**	**0.26**
c	**2.1**	**2**	**0**	**46**	**1**	**1.000**	**Inf**
java	**2.1**	**1692**	**201**	**2239**	**25**	**0.000**	**0.09**
properties	**2.1**	**19**	**1**	**45**	**7**	**0.429**	**2.92**
sh	**2.1**	**10**	**5**	**64**	**13**	**0.162**	**0.41**
xml	**2.1**	**96**	**11**	**184**	**24**	**0.851**	**1.14**

Besides these considerations, we do not have an unique answer for RQ3. However, we observe that having languages interacting with other languages is related to higher defect proneness for certain languages (mainly C) and specific interactions.

6. THREATS TO VALIDITY

Internal: in this exploratory case-study different aspects were not considered. In particular we did not examine all the possible confounding factors influencing the defect proneness of the modules. Among them the age and the size of modules (expressed in LOC, for example) are the most relevant ones.

We discriminated between modules on their names while the same module can change name in the course of the project. We grouped the files by their extension while a different extension could not always indicate a different language.

Construction: we are unable to measure directly the interaction between modules written in different languages and consequently we use as a proxy their concurrent presence in the same commits, which may be an imprecise approximation.

External: another threat is due to selection bias: we have no particular reason to believe that Hadoop is representative of other software projects. Of course having considered only one project generalization of the results presented is not possible at all.

7. CONCLUSIONS AND FUTURE WORK

Although we do not have unique answers, the results and observations from this exploratory study let us understand that the problem is worthy to be investigated. In fact we observed that more than half of the commits in Hadoop are cross language (at least according to our definition). However we also observed that this property depends on the type of the activities and the extensions of the modules.

Commits related to testing or fixing bugs involve mainly a single language, while adding new features or doing implementation sub-task are activities which involve multiple languages (or at least extensions).

Looking at the single extensions, we verified that the most common extensions are frequently changed together with files with different extensions. Frequent interactions are generally not symmetric, and many of them involve XML.

When we look at defect proneness, we observe that for Java modules the interactions with other languages (as an aggregate) is not problematic at all: we observed that Java CLMs files are ten times less defect prone than ILMs. However, when looking at single pairs of interactions, we notice that several pairs have CLM significantly more defect prone then ILM, especially C modules. Finally, the widespread interaction between Java and XML apparently is not related to defect proneness.

This study represents a first step in understanding the phenomenon of languages interaction. We should address in future work the threats that limit the scope and the validity of the study. However this study let us hypothesize that the interaction of languages might be problematic for specific languages interactions. We would like also to study other effects of languages interactions, for example on the development speed.

8. REFERENCES

[1] Wampler, D.; Clark, T.; Ford, N.; Goetz, B.; , "Multiparadigm Programming in Industry: A Discussion with Neal Ford and Brian Goetz," Software, IEEE , vol.27, no.5, pp.61-64, Sept.-Oct. 2010 doi: 10.1109/MS.2010.121

[2] Fowler, M. 2011. Domain Specific Languages. Addison Wesley Signature Series.

[3] Moritz Eysholdt and Heiko Behrens. 2010. Xtext: implement your language faster than the quick and dirty way. In Proc. of the ACM int. conf. Object oriented programming systems languages and applications companion (SPLASH '10). ACM, New York, NY, USA, 307-309. DOI=10.1145/1869542.1869625

[4] Fredrik Seehusen and Ketil Stølen. 2011. An evaluation of the graphical modeling framework (GMF) based on the development of the CORAS tool. In Proc.of the 4th int. conf. on Theory and practice of model transformations (ICMT'11).

[5] Charles Simonyi, Magnus Christerson, and Shane Clifford. 2006. Intentional software. SIGPLAN Not. 41, 10 (October 2006), 451-464. DOI=10.1145/1167515.1167511

[6] Markus Volter. 2011. From Programming to Modeling - and Back Again. IEEE Softw. 28, 6 (November 2011), 20-25. DOI=10.1109/MS.2011.139

[7] Sergey Dmitriev, 2004. Language Oriented Programming: the next programming paradigm http://www.jetbrains.com/mps/docs/Language_Oriented_Programming.pdf.

[8] Markus Völter and Eelco Visser. 2010. Language extension and composition with language workbenches. In Proceedings of the ACM int. conf. companion on Object oriented programming systems languages and applications companion (SPLASH '10). ACM, New York, NY, USA, 301-304. DOI=10.1145/1869542.1869623 .

[9] Rolf-Helge Pfeiffer and Andrzej Wąsowski: "Taming the Confusion of Languages" In: ECMFA 2011. Published in: ECMFA'11.