On the Relationships between Domain-Based Coupling and Code Clones: An Exploratory Study

Md Saidur Rahman*, Amir Aryani†, Chanchal K. Roy*, Fabrizio Perin‡
*University of Saskatchewan, Canada
{saeed.cs, chanchal.roy}@usask.ca
†Australian National University, Australia
amir.aryani@anu.edu.au
‡University of Bern, Switzerland
perin@iam.unibe.ch

***Abstract*—Knowledge of similar code fragments, also known as code clones, is important to many software maintenance activities including bug fixing, refactoring, impact analysis and program comprehension. While a great deal of research has been conducted for finding techniques and implementing tools to identify code clones, little research has been done to analyze the relationships between code clones and other aspects of software. In this paper, we attempt to uncover the relationships between code clones and coupling among domain-level components. We report on a case study of a large-scale open source enterprise system, where we demonstrate that the probability of finding code clones among components with domain-based coupling is more than 90%. While such a probabilistic view does not replace a clone detection tool per se, it certainly has the potential to complement the existing tools by providing the probability of having code clones between software components. For example, it can both reduce the clone search space and provide a flexible and language independent way of focusing only on a specific part of the system. It can also provide a higher level of abstraction to look at the cloning relationships among software components.**

I. INTRODUCTION

Code clones constitute a significant fraction of code (7% to 23% [11]) and it is important to be aware of clones from several maintenance perspectives. One of the important maintenance concerns is the propagation of existing bugs and introduction of new bugs due to inconsistent updates to duplicate code blocks [5], [6]. Given the importance, many tools and techniques for detecting clones have been proposed [11]. However, cross-language and source code independent clone analysis are still open research challenges. Although complete solutions to these open problems are yet to come, artifacts and information beyond the source code are potentially useful to formulate complementary solutions for identifying dependencies and change propagation between software components [2].

In contrast to a great deal of research about code clones, there is little information available about the relationships between clones and other aspects of software systems such as the domain level artifacts. Recently, relationships among software components at domain level, termed as *domain-based coupling* [3], have been shown useful in tracing dependencies at the source code and database levels. In this paper, we focus on the domain-based coupling and its relationships with code clones. We hypothesize that coupling at the domain level can approximate clones in the source code.

We evaluate our hypothesis with a case study on a large scale enterprise system called ADEMPIERE. We derive domain-based coupling from the domain information available at the user interface level. Code clones, on the other hand, are detected using the clone detection tool. The relationships between domain-based coupling and code clones are then determined based on the extent of co-existence of both relationships between all pairs of software components. Our results from the case study show that the coupling between domain level components can approximate clones in the associated code with high precision without even accessing the source code. In particular, we focus on the following research questions in conducting the exploratory study.

RQ1 *What is the probability of finding code clones where there is a domain-based coupling?*

RQ2 *How efficiently can domain-based coupling assist software maintainers in discovering code clones?*

While such a probabilistic approach of predicting the existence of clones does not replace the traditional clone detection tools, it certainly has the potential to complement the existing state of the art tools by providing the probability of having clones among software components. Moreover, the approach is independent of program source code. Consequently, it can be used for the analysis of hybrid systems developed with multiple programming languages and legacy systems with missing or obsolete source code. In addition, based on the domain-based coupling, it provides a flexible way of focusing only on the relevant parts of the systems and thus reduces the clone search space. Once the relevant parts are identified, one can either use the existing tools for actual clone detection or opt for manual analysis where feasible. This essentially helps predicting goal-driven clones without being overwhelmed with hundreds of other irrelevant clones.

The rest of this paper is organized as follows: Section II describes the domain-based coupling analysis. Section III discusses the code clones analysis. Section IV presents the evaluation results. Section V discusses the potential threats to the validity of our results, Section VI describes the related

978-1-4673-3076-3/13 © 2013 IEEE

ICSE 2013, San Francisco, CA, USA
New Ideas and Emerging Results

Accepted for publication by IEEE. © 2013 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/ republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

works, and finally Section VII concludes this paper with a discussion on the outcomes and future areas of investigation.

II. Domain-Based Coupling

In this section, we describe the methodology for domain-based coupling analysis, and how it is implemented for ADEMPIERE, the system under analysis.

A. Notation and Definitions

Our domain analysis model is derived from the models proposed by Aryani *et al.* [3], [1], [2]. The three key elements at the domain level are modelled as follows:

- A *domain variable* is a variable unit of data that has a clear identity at the domain level. *Domain variables* are modelled by a finite set V, called *variable symbols*.
- A *domain function* represents a domain-level behaviour of the system associated with at least one domain variable as an input or output. *Domain functions* are modelled by a finite set F, called *function symbols*, and the binary relation $USE \subseteq F \times V$ represents the relation between functions and variables as the input-output of the functions.
- A *user interface component (UIC)* is a system component which directly interacts with users, and contains one or more domain functions. *UICs* are modelled by a finite set C, called the *component symbols*, and $HAS \subseteq C \times F$ represents the relation between components and functions.

Definition 1. *The conceptual connection relation $CNC \subseteq C \times C$ is defined by*

$$CNC = HAS.USE.USE^{-1}HAS^{-1}$$

The domain-based coupling is derived based on the following measurements:

Definition 2. *The number of common variables among two UICs is modelled by the function $\vartheta : C \times C \to \mathbb{N}$ where*

$$\vartheta(c, c') = |c.HAS.USE \cap c'.HAS.USE|$$

Note that the definition of common domain variables is symmetric, i.e., $\vartheta(c, c') = \vartheta(c', c)$.

Definition 3. *The domain-based coupling graph of a system is a symmetric weighted graph, $G = (C, CNC \backslash ID, \omega)$ where ID indicates the identity relation and ω indicates the coupling weight function $\omega : C \times C \to [0..1]$ by*

$$\omega(c, c') = \frac{\vartheta(c, c')}{|c.HAS.USE \cup c'.HAS.USE|}$$

In the scope of this paper, by *domain-based coupling* we refer to ω. The next example demonstrates how to measure ϑ and ω for two example UICs.

B. Example

In ADEMPIERE, *Daily Balances* (c_1) and *Accounting* (c_2) are two UICs each with one domain function *Query Daily Account Balances* and *Query Accounting Transactions* respectively. *Daily Balances* (c_1) contains 22 domain variables while *Accounting* (c_2) has all *Daily Balances'* domain variables and 11 others. Thus, there are total 33 domain variables used by either of these UICs; therefore,

$$\vartheta(c_1, c_2) = 22 \quad and \quad \omega(c_1, c_2) = 22/33 = 0.67$$

C. Domain Analysis in ADempiere

In this study, we consider a tab as an user interface component. The ($HAS.USE$) relationships between UICs and domain variables can be derived from manual analysis of the working software, user manuals, or system functional specification [2]. In the case of ADEMPIERE, this information is stored in a part of the database called *application dictionary*. We harvested this information using a SQL script resulting in 889 UICs and 2,359 domain variables, leading to 49,854 pairwise domain-based couplings relations.

III. Code Clone Analysis

For our study on ADEMPIERE we used the NiCad [9] tool for clone detection as NiCad is reported to have high precision and recall [8], [10]. We detected clones with a minimum size of 5 lines of code (LOC) for function granularity. At function granularity each function in the source code is considered as a unit of code for comparison and detection of clones.

To determine the number of the *clone class* (set of all fragments that are clones of each other) and clone fragments associated with each pair of UICs, all the clone classes are checked. We count only those clone classes which have at least one clone fragment from each of the source files behind the selected UIC pair. For each of these clone classes, all clone fragments originated from the source files behind the selected pair of UICs are counted. Definition 4 formally defines this procedure.

Definition 4. *Let $c, c' \in C$ be any pair of user interface components with their associated source code files S and S'. The number of cloned code fragments $\xi : C \times C \to \mathbb{N}$ associated with c, c', is defined by*

$\xi(c, c') = \sum_i | \{f : (f \in F_S \vee f \in F_{S'}) \wedge f \in CC_i \wedge (CC_i \cap F_S) \neq \phi \wedge (CC_i \cap F_{S'}) \neq \phi\} |$.

where F_S and $F_{S'}$ are the sets of all code fragments in S and S' respectively for a defined granularity, and CC is the set of clone classes in the system.

We observed that out of 889 UICs in ADEMPIERE there are 813 UICs with their associated source code, and the other 76 UICs are created at runtime based on predefined business rules. We consider exact clones for these 813 UICs with source code for the evaluation of our study.

IV. Evaluation

In this section, we provide empirical evidence on the relationships between code clones and domain-based coupling, and demonstrate how these relationships can be used to assist software maintainers by answering the defined research questions.

A. RQ1: Probability of Finding Clones

For our analysis, we use Fruchterman and Reingold's [4] force-based graph layout (known as spring layout) to visualize the domain-based coupling graph (Fig. 1a). Now, let us consider $E \subseteq C \times C$ to be the set of domain-based coupling relations including all combinations of pairs between UICs. Our query is $Q = \{(c, c') | c, c' \in C, \omega(c, c') > 0\}$, the set of the couplings connecting UICs by coupling weight ω, and $A = \{(c, c') | c, c' \in$

$C, \xi(c, c') > 0\}$ is the expected answer, the set of UIC pairs with one or more clones in their code behind.

There are 813 UICs in our analysis with source code (Section III), leading to $|E|$ =330,078 disjoint UIC pairs out of which $|A|$ =68,563 UIC pairs have clones in their code behind it. Again, the domain-based coupling relation contains $|Q|$ =44,163 couplings, representing UIC pairs with $\omega > 0$, from which $|Q \cap A|$ =39,850 have exact clones in their code behind. The probabilities of finding code clones between UIC pairs with and without domain-based coupling represented by P_Q and $P_{\bar{Q}}$ respectively and measured as

$$P_Q = \frac{|Q \cap A|}{|Q|} = 90.23\% \qquad P_{\bar{Q}} = \frac{|A \cap E \backslash Q|}{|E \backslash Q|} = 10.04\%$$

The higher value of probability P_Q, also can be considered as *precision*, implies that domain-based coupling can be used to provide a precise prediction of the places in the source code that have cloned code (or vice versa). In addition, the high probability of co-existence of domain-based coupling and code clones suggests that domain-based coupling can be used to single out a subset of codebase having code clones with a selected component for a goal-driven clone analysis.

Summary: *The probability of finding code clones where there is domain-based coupling is more than 90%.*

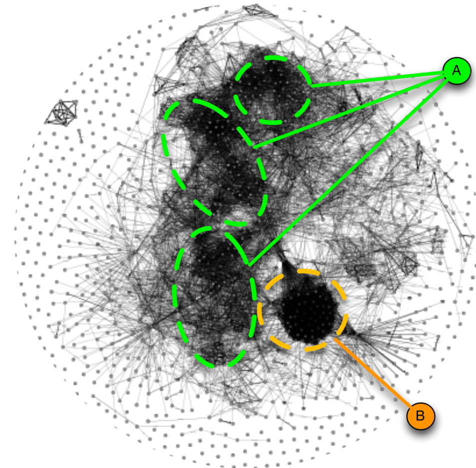
B. RQ2: Discovering Code Clones

Clone detection in the source code is often computationally expensive and it may generate an exhaustive list of clones. To carry out maintenance, software maintainers need precise and concise suggestions about the locations of code clones, as inconsistent updates to clone fragments may introduce bugs in the system. Instead of reviewing an exhaustive list of all clones in the system, it is more useful to have a short list of higher abstraction level components that most likely contain code clones. The maintainers can then confidently look at those selected components for their intended task. Using domain-based coupling we can provide such a short list based on the top most coupled UICs, and which might in turn help in prioritizing the components to work on.

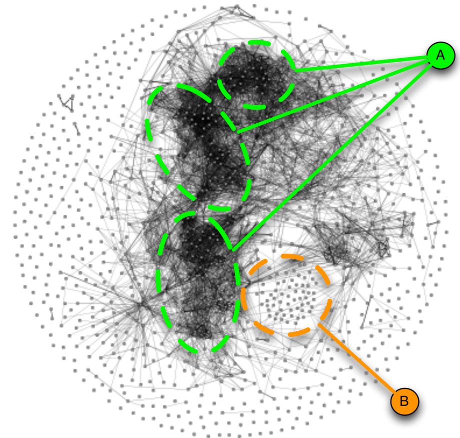
For a given UIC, $c \in C$, we derive the short list of $Q_{c,n} \subseteq C$ which contains the top n UICs with highest domain-based coupling to c. To evaluate how useful such a short list is, we measure the *likelihood* of finding one or more clone fragments in the code behind of UICs in the short list. More formally, if $A_{c,k} \subseteq C$ represents the set of UICs with at least k number of clone fragments to c, then the *likelihood* measure is defined as

$$L_{n,k} = \frac{|\{c | c \in C, Q_{c,n} \cap A_{c,k} \neq \emptyset\}|}{|\{c | c \in C, Q_{c,n} \neq \emptyset\}|}$$

Table I shows the results for the likelihood of finding code clones in the short lists of UICs that include top 3, 5 and 10 highly coupled UICs. The likelihood of finding 10 code clones or more in the top 3 UICs is 3.94%. The likelihood increases by reducing the threshold for code clones, up to 88% for k =1. In addition, by increasing the size of the short list one can find more code clones. This is more notable for the higher thresholds. For the threshold of k =10, increasing the short list size from three to ten, almost doubles the chance of finding code clones, while for the threshold k =1, increasing the size of the list improves the likelihood from 88.81% to 91.64%. This is a trade off between accuracy and cost, since it is less expensive and more convenient for software maintainers to review a list of three UICs compared to a list of ten.



(a) Domain-based coupling graph



(b) Filtered graph by code clones

Legend: Nodes are the UICs of ADEMPIERE in both graphs. Top: Edges represent domain-based coupling between UICs with $\omega > 0.1$. Bottom: Edges show that there are both code clones and domain-based coupling.

Fig. 1: Domain-based coupling vs clones

TABLE I: LIKELIHOOD (%) OF FINDING CLONES IN TOP n RESULTS

k	1	3	5	10
Top 3	88.81	38.75	17.47	3.94
Top 5	90.53	44.77	21.40	5.78
Top 10	91.64	54.40	24.97	7.75

Legend: k is the threshold for minimum number of clones.

Summary: *The likelihood of finding code clones in the efficient short list of top three results is more than 88%.*

C. Visual Comparison

To visualize the relationships between domain based coupling and code clones the comparison graphs (Fig. 1b) are created by the following steps: First, the domain-based coupling graph is generated (Fig. 1a) with a given threshold of $\omega > 0.1$. The value of the threshold is derived by a heuristic approach based on the distribution of UIC pairs in the graph to filter out weak couplings. Second, all the edges with no code clones, $\xi = 0$ are filtered out from the graph. The result is presented in Fig. 1b. The remaining edges show the UIC pairs with both strong domain-based coupling and code clones.

The comparison between these graphs in Fig. 1 shows that from four clusters in the domain-based coupling graph, three of them have high number of clones (tagged with A), and there is one cluster with no clones (tagged with B). The qualitative analysis of this cluster shows that it is composed of 76 UICs which implement the multilingual aspect of ADEMPIERE. These UICs are generated at runtime based on the predefined business rules and there is no code behind these UICs.

V. Threats to Validity

The code clone patterns and behaviour might be influenced by the inherent features of a particular programming language. Similarly, the distribution and patterns of clones may vary from system to system. We aimed to address these issues by selecting a case study system which typifies a majority of enterprise systems. However, further studies are required with diversified experimental settings to draw a generalized conclusions. In this study, the quality of derived domain variables is dependent on authors' knowledge about the system. We addressed this issue by cross checking the derived domain variables with the help descriptions of the system.

VI. Related Work

Studies show that domain knowledge is useful in understanding the functionality of the code [7]. Domain knowledge has been incorporated into reverse engineering and software maintenance by a number of researchers. Rugaber [12] showed how domain knowledge can be useful in program comprehension. Given the potentials of domain information, Aryani *et al.* [1], [3] recently showed how domain information can be used to identify dependencies among software components to support change propagation.

Clone detection, on the other hand, is not a new topic and there have been a great many tools and techniques available in the literature [11]. Some of these clone detection techniques are measuring textual similarity, finding common subsequences of tokens, finding the similar sub-trees in Abstract Syntax Tree (AST), comparing matrix values computed for source code blocks and finding isomorphic sub-graphs [11]. Although there are wide variety of clone detection tools and techniques, these tools lack in generalized application to detect clones from hybrid systems with source code of different languages. This exploratory study examines one potential application of domain information to predict clones in the program source code without even accessing the source code.

VII. Conclusion and Future Work

In this paper, we examined the relationships between domain based coupling and code clones and investigated how this relationship can support software maintenance. Unlike the traditional clone detection approach it evaluates the feasibility of using domain-based coupling to complement the existing clone detection tools by providing the probability of the existence of code clones between software components. The results based on one of the largest open source enterprise systems, ADEMPIERE, demonstrate that code clones could be predicted using solely domain information with more than 90% precision. In addition, we presented how domain-based coupling can be used to give efficient and precise suggestions about code clones to software maintainers. The likelihood of finding clones in such suggestions is more than 88%.

In future, we plan to extend our approach by using alternative sources of domain information and other types of clones. In this work, we used visible data fields to derive the domain-based coupling between user interface components. However, this approach can be extended to other data sources such as user manuals, software built-in help descriptions, and requirement documents. In addition, we used only one coupling metric based on the number of common domain variables among software components. In future, we plan to improve this approach by using hybrid metrics by combining domain-based and other coupling metrics.

References

[1] A. Aryani, I. D. Peake, and M. Hamilton. Domain-based change propagation analysis: An enterprise system case study. In *Proc. ICSM*, pages 1–9, 2010.

[2] A. Aryani, I. D. Peake, M. Hamilton, H. Schmidt, and M. Winikoff. Change propagation analysis using domain information. In *Proc. ASWEC*, pages 34–43, Australia, 2009.

[3] A. Aryani, F. Perin, M. Lungu, A. N. Mahmood, and O. Nierstrasz. Can we predict dependencies using domain information? In *Proc. WCRE*, pages 55–64, 2011.

[4] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, 1991.

[5] E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner. Do code clones matter? In *Proc. ICSE*, pages 485–495, 2009.

[6] M. Mondal, C. K. Roy, M. S. Rahman, R. K. Saha, J. Krinke, and K. A. Schneider. Comparative stability of cloned and non-cloned code: An empirical study. In *Proc. ACM SAC*, pages 1227–1234, 2012.

[7] M. Petrenko, V. Rajlich, and R. Vanciu. Partial domain comprehension in software evolution and maintenance. In *Proc. ICPC*, pages 13 –22, 2008.

[8] C. Roy and J. Cordy. A mutation/injection-based automatic framework for evaluating code clone detection tools. In *Proc. ICSTW*, pages 157 –166, 2009.

[9] C. K. Roy and J. Cordy. NiCad: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization. In *Proc. ICPC*, pages 172 –181, 2008.

[10] C. K. Roy and J. R. Cordy. Near-miss function clones in open source software: An empirical study. *Journal of Soft. Maintenance*, 22(3):165–189, 2010.

[11] C. K. Roy, J. R. Cordy, and R. Koschke. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of Comp. Prog.*, 74:470–495, 2009.

[12] S. Rugaber. The use of domain knowledge in program understanding. *Annals of Soft. Engg.*, 9:143–192, 2000.