2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering

Towards Detection and Analysis of Interlanguage Clones for Multilingual Web Applications

Yuta Nakamura*, Eunjong Choi*, Norihiro Yoshida†, Shusuke Haruna* and Katsuro Inoue*
*Osaka University, Japan
{n-yuuta@ist, ejchoi@osipp, haruna@ist, inoue@ist}.osaka-u.ac.jp
†Nagoya University, Japan
yoshida@ertl.jp

***Abstract*—In this position paper, we introduce an instance of interlanguage clones in a multilingual web application system and then discuss research challenges on the detection and analysis.**

I. Motivation

The number of web applications have rapidly increased over the last decades. In order to speed up web application development, developers frequently write code clones [4]. As a result, many web applications contain more than 50% code clones [2][3][4]. Since code clones can be one of the factors that increase maintenance costs in general, code clones in a web application system should be detected and managed to alleviate maintenance costs [2][4].

Web application systems are typically written in multiple programming languages (e.g. *HTML* and *JavaScript*) where different languages rely on each other [2][4]. This causes interlanguage clones, each of which spans different program files that are written in different programming languages in the development of web application systems. In this position paper, we define an interlanguage clone as a set of code fragments S_o that satisfies the following conditions:

- S_o span multiple files F (where $|F| \geq 2$)
- multiple programming languages are used in files F
- each code fragment $c_i \in S$ is callee or caller of a set of the other code fragments S_a (where $|S_a| \geq 1, S_a \in S_o, c_i \notin S_a$)
- $\exists S_c (S_c \equiv S_o)$ where $\equiv$ means that each code fragment in S_o has its clone in S_c and the same callee/caller relations exist between the code fragments in S_c.

To give a clear idea of interlanguage clones across multiple program files in web application systems, we introduce a pair of interlanguage clones in Webogram[1], an open-source telegram web application (Fig. 1). This pair of interlanguage clones is spread across view and logic files implemented by *HTML* and *JavaScript*. Each of the code fragments in the *HTML* files (Fig. 1(a) and 1(c)) calls the functions updateGroup (Fig. 1(b)) and updateChannel (Fig. 1(d)) in the 'controller.js' file, respectively. The pair of interlanguage clones (a, b) and (c, d) in Fig. 1 is considered to be maintained simultaneously (e.g. enhancement, bug fix). For example in Fig. 1, if developers change a view of the HTML file (Fig. 1(a)) of the interlanguage clone (a, b), they have to consider if the HTML file (Fig. 1(c)) of its clone pair, (c, d), should be modified or not. For another example, if developers add an exception handling to updateGroup (Fig. 1(b)), they have to determine that they should add the code to updateChannel (Fig. 1(d)).

Assuming that a clone pair in the 'html' files (Fig. 1(a), Fig. 1(c)) and the 'js' file (Fig. 1(b), Fig. 1(d)) is detected, respectively, developers should check these two clone pairs. However, if a clone pair is detected as a larger-granularity code clones, they only check a clone pair (Fig. 1 (a) and (b), Fig. (c) and (d). As a result, they can save time and efforts for checking all the detected code clones. Therefore, we expect that detecting these code clones from web applications helps developers to easily perform maintenance.

Although some previous approaches detect code clones from multilingual software systems [1][2], the granularity of the detection is a code fragment written in only one language. Therefore, they are against our purpose of detecting interlanguage clones which detection granularity is a code fragment spanning multiple languages.

In order to detect these code clones from web applications, we discuss a method to detect interlanguage clones (i.e., larger-granularity code clones spread throughout different programming languages). We expect that larger-granularity can decrease efforts to check all of the detected code clones from web applications.

II. Research Challenges

Target programming languages: Web applications are mainly written in multiple programming languages. In order to detect interlanguage clones from web applications, we can adopt a language-independent approach. However, we need to detect code clones using a language dependent approach because program dependent approach is more precise than an independent one. In order to use a program dependent approach, we should decide on the target programming languages. To solve this problem, we investigated frequencies of co-used programming language combinations used in web applications. In detail, at first, we randomly selected 109,547 open source web applications that were developed until 2014 from GitHub[2]. After that, we excluded applications written

[1]https://web.telegram.org

[2]https://github.com/

978-1-5090-1855-0/16 $31.00 © 2016 IEEE
DOI 10.1109/SANER.2016.55

```
<div class="md_simple_modal_wrap" my-modal-position>
 <div class="md_simple_modal_body">
            :
</div>
 <div class="md_simple_modal_footer">
  <button class="btn btn-md" ng-click="$dismiss()" my-i18n="modal_cancel"></button>
  <button class="btn btn-md btn-md-primary" ng-class="{disabled: group.updating} "
   ng-click="updateGroup() "
   ng-bind="group.updating ? 'group_edit_submit_active' : 'group_edit_submit' | i18n "
   ng-disabled="group.updating"></button>
 </div>
</div>
```

(a) chat_edit_modal.html

```
$scope.updateGroup = function () {
   if (!$scope.group.name) {
    return;
   }
            :
$scope.group.updating = true;
   var apiPromise;
            :
   return apiPromise.then(function (updates) {
    ApiUpdatesManager.processUpdateMessage(updates);
    var peerString = AppChatsManager.getChatString($scope.chatID);
    $rootScope.$broadcast('history_focus', {peerString: peerString});
   })['finally'](function () {
    delete $scope.group.updating;
   });
            :
```

(b) controllers.js

```
<div class="md_simple_modal_wrap" my-modal-position>
 <div class="md_simple_modal_body">:
                        :
</div>
 <div class="md_simple_modal_footer">
  <button class="btn btn-md" ng-click="$dismiss()" my-i18n="modal_cancel"></button>
  <button class="btn btn-md btn-md-primary" ng-class="{disabled: channel.updating} "
   ng-click="updateChannel()"
   ng-bind="channel.updating ? 'channel_edit_submit_active' : 'channel_edit_submit' | i18n "
   ng-disabled="channel.updating"></button>
 </div>
</div>
```

(c) channel_edit_modal.html

```
$scope.updateChannel = function () {
   if (!$scope.channel.title.length) {
    return;
   }
   var promises = [];
            :
   $scope.channel.updating = true;
   return $q.all(promises).then(function () {
    var peerString = AppChatsManager.getChatString($scope.chatID);
    $rootScope.$broadcast('history_focus', {peerString: peerString});
   })['finally'](function () {
    delete $scope.channel.updating;
   });
  };
```

(d) controllers.js

Fig. 1: An Example of Clone Pair in Webogram

in only single language and then found 66,825 applications using multiple languages. Finally, we applied a pattern mining approach to these applications to identify frequently co-used program languages. As a result, we found out that *Html* and *JavaScript* are the most frequently co-used (22.9%) languages for web applications. They are followed by *PHP* and *JavaScript* (6.5%), *C* and *C++* (6.1%), and *HTML* and *Ruby* (6.0%). These results imply that an approach that detects code clones across *HTML* and *JavaScript* files is necessary.

Approach for detecting clones: Web applications contain source code written in co-dependent multiple programming languages. Therefore, detecting interlanguage code clones from web applications is a challenging issue. One of the solutions for detecting interlanguage code clones from web applications is merging co-dependent code clones detected from each programming language. In detail, after detecting code clones from files written in each programming language, the code clones are merged into an interlanguage clone based on call relationship or data flows among the code clones.

In order to detect code clones from each programming language of web applications, we can use existing code clone detection tools [5]. However, as shown in Fig. 1, some code clones might have gaps such as addition or insertion of statements. Therefore, tools that detect Type-3 code clones can be a good candidate for detecting code clones[5].

However, we have one problem on this domain. Some files are written in two or more languages (e.g., an *HTML* file including *JavaScript* between `<script>` and `</script>` tags). These tangled files are an obstacle to analyze for clone detection tools described above. Therefore, it is necessary for our approach to separate them to files written in only one language.

Management for detected clones: After code clones are detected from web applications, results of detected code clones should be reported. However, because code clones might be spread throughout web applications, we face the challenge of how detected code clones should be presented to the developer and in what format. These design decisions will affect management of detected code clones.

ACKNOWLEDGMENT

This work was supported by JSPS KAKENHI Grant Numbers 25220003, 26730036 and 15H06344.

REFERENCES

[1] N. Kraft, B. Bonds, and R. Smith. Cross-language clone detection. In *Proc. of SEKE 2008*, pages 54–59, 2008.

[2] T. Muhammad, M. F. Zibran, Y. Yamamoto, and C. K. Roy. Near-miss clone patterns in web applications: An empirical study with industrial systems. In *Proc. of CCECE 2013*, pages 1–6, 2013.

[3] D. C. Rajapakse and S. Jarzabek. An investigation of cloning in web applications. In *Proc. of ICWE 2005*, pages 252–262, 2005.

[4] D. C. Rajapakse and S. Jarzabek. Using server pages to unify clones in web applications: A trade-off analysis. In *Proc. of ICSE 2007*, pages 116–126, 2007.

[5] C. K. Roy, J. R. Cordy, and R. Koschke. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Sci. Comput. Program.*, 74(7):470–495, 2009.