Cloning in Popular Server Side Technologies using Agile Development: An Empirical Study

Aisha Khan[1], Hamid Abdul Basit[2], Syed Mansoor Sarwar[1*], and Muhammad Murtaza Yousaf [1]

1. *Punjab University College of Information Technology (PUCIT), University of the Punjab, Lahore, Pakistan*
2. *Department of Computer Science, School of Science and Engineering, Lahore University of Management Sciences (LUMS), Lahore, Pakistan*

* **Corresponding Author:** Email: syed.sarwar@pucit.edu.pk

Abstract

Several types of clones exist in software systems due to the copy-paste activity, developer limitations, language restrictions, and software development lifecycle. This work studies the issues of cloning in server side technologies for web applications. We studied 11 different reasonable size (average over 22K LOC) web development projects coded in C#, Java, Ruby-on-Rails (ROR), and PHP based on the same set of requirements. We identified and analyzed simple and structural clones present in these systems in order to compare the different technologies in terms of number of clones, clone size, clone coverage, reasons behind creation of clones, and the ratio of refactorable and non-refactorable clones. Our study focused only on the base languages of these server side technologies. Our analyses show that C# has the highest number of clones and ROR has the lowest. C# also has the highest and ROR has the lowest percentages of refactorable clones. PHP has the highest clone coverage and ROR has the lowest. Average clone size for all projects ranges from 49.8 to 77.2 tokens. In terms of clone size, there are no significant differences across projects in the same technology. The project size, project architecture, and developer approach dictate the percentage of clones present in a software project. The use of frameworks and design patterns helps control generation of clones.

Key Words: Code Clones; Clone Coverage; Web Technologies; Refactoring.

1. Introduction

Clones (or code clones) are code segments of considerable size that are similar to each other, based on some similarity criteria. In software development, copying a piece of code and then reusing it, with or without changes, is called cloning. The piece of copied code is called a clone. Clones are undesirable from the software maintenance point of view because they may result in bug propagation. Although, many researchers consider code clones harmful [13, 14], some say that clones are not always harmful [5, 17]. Kasper and Godfrey [5] argue that code clones can be used as a principled software engineering tool.

Tools and techniques are available for detecting clones in software [20, 21]. Clone detection is helpful in plagiarism detection, origin analysis, finding usage patterns, software evolution research, and bug detection [11]. Mondal et al. [22] performed the study with the intention of finding the categories of clones that tend to introduce bugs in the software systems. It also helps in software maintenance, because there may be more chances of update anomalies if a project contains multiple clones of the same code fragment. For example, it is common for developers to copy-paste the same piece of code multiple times with the intention to reuse it, but such activities make maintenance difficult. An in-depth study on the effort required to maintain a cloned code can also be found at [24].

The presence of code clones increases the likelihood of bug propagation, difficult to maintain design, and higher maintenance cost [12]. If there is a change in a clone, making the same change in all of its instances may be time consuming and chances of accidently missing some fragment are significant. Thus, clone detection is important, because techniques like refactoring can then be used to unify similar clones. However, the important to know is the types of clones that have been detected. Along with the detection of clones that are exact copies of each other, most probably the copy-paste ones, similar code fragments and larger similar structures in software as well as in design can be more useful to detect. Design level similarity detection may also help in better understanding of the software architecture and improving design. It may also be helpful in reengineering legacy systems.

There have been studies on the analysis of clones detected in software applications [1]. Attempts to detect code clones at different granularity levels also exist [25]. Several types of clones are present in software systems. Sometimes these clones are accidental [4] while at other times they are caused by the simple copy-paste activity with the software reuse intention, developer limitations, or even language limitations in creating suitable generic abstractions. Rajapakse and Jarzabek [2] analyze clones in web applications, but their analyses are for exact clones only. Other studies [1] are not specific to web technologies. Koschke et al. [23] targeted open source projects written in C/C++ to investigate software clone rates. Ours is the first attempt to study the issues of cloning in server side technologies for web applications.

Currently, different technologies are in use for web development, including Java, .Net, ROR, and PHP. Software developed in all of these technologies usually contains clones, but some might have more clones than others. Studying cloning characteristics of web development languages is a meaningful study, but no such study is available in the literature. The purpose of our study was to detect and analyze clones in the systems developed with the above-mentioned web technologies and observe their various characteristics as well as to know which technologies produce more clones and why.

This study not only focuses on simple clones but also another category called "structural clones". Identification of structural clones adds to the benefits of knowing simple clones. As structural clones cover larger parts of code, they are more meaningful.

A. What are structural clones?

Structural clones are "recurring patterns of simple clones" usually due to the design level similarities [9]. They embody such large-granularity, design-level similar program structures that often map to design or application domain concepts. Large granularity, design-level similarity patterns often create opportunities for reuse of design solutions within a given system, or even across similar systems. This form of reuse is natural and enhances the current architecture-centric, component-based reuse methods. These are often induced by the analysis pattern and design techniques used by the developers [7].

For an in-depth study of the cloning characteristics in the above-mentioned technologies, we analyzed 11 social networking projects developed in these technologies. We used Clone Miner [9] for clone detection because it identifies simple and structural clones, which was the focus of this work. Clone Miner is a widely used and one of the most cited tools [26]. This tool works on the token-based clone detection technique. It uses token-based simple clone detector [18]. It also identifies structural clones, for which it implements a structural clone detection technique. This technique works with the information of simple clones [19].

2. Related Work

There have been studies discussing the clones in different technologies [1, 2]. Roy et al. discussed clone management in detail along with future research directions [27]. Studies on the evaluation of different clone detection tools also exist [28][29]. However, the work most relevant to our study is described in [1], [2], and [3]. Rajapakse and Jarzabek [2] explain cloning in web applications of different sizes, developed using a range of web technologies, and serving diverse purposes. However, they consider simple clones and analyze the files of the projects under study for identifying clones as simple text. Their initial results show cloning rates of up to 63% in both newly developed and already maintained web applications.

Roy and Cordy [1] describe a similar analysis performed on the various open source systems written in C, Java, and C# to identify near-miss clones in them. Near-miss clones are clones where the copied fragments are very similar to the original ones, but are not exactly the same. Editing activities such as changing comments and layouts, changing the position of the source code elements through blanks and new lines, and changing identifiers, literals, and macros might have been applied to such clones. The results of this study show a large number of exact function clones in these open source systems, but the number of near-miss clones is even greater. Furthermore, they found more exact clones in object-oriented Java and C# systems than in C systems. The study also found that the cloning characteristics are not affected by the size of a system.

In [3], Roy and Cordy describe whether the observations made in [1] also apply to scripting languages. For this purpose, they analyzed some open source projects built in Python and compared their results to the results obtained for the projects built in C, C#, and Java [1]. They found out that the cloning characteristics of scripting languages are similar to those of the traditional imperative (compiled) languages.

Ours is an in-depth study of clones in web applications to analyze simple and structural clones. However, to keep the study focused, we chose only a set of popular server side web technologies, including ROR, PHP, C#, and Java. C# and Java are used together with ASP.Net and JSP, respectively, because currently these are among the primary technologies being used for the development of commercial web applications.

This work is focused on web applications, particularly, from the social networking domain. We analyzed clones and identified those that could be refactored [6]. We, however, did not study the applications written in scripting languages, including JSP and ASP.Net. However, based on the conclusions of [1], we expect that the cloning characteristics of the code in JSP, ASP.Net, JavaScript, Python, HTML, etc. would be similar to those of the associated base languages, i.e., Ruby, PHP, C#, and Java.

3. Research Questions

In order to compare the above stated popular server-side technologies, our study mainly concentrated on, but not limited to, four research questions. Our research questions focus on identifying cloning characteristics of the target technologies and their comparison on the basis of the several factors, including number of clones, clone-to-code ratio and clone sizes. We also investigated structural clones. The research questions are:

RQ1: Which technology produces more simple and structural clones, and what are the causes of clone production?

RQ2: Are there some structural clones that are present across multiple systems? Are they technology dependent?

RQ3: Do the use of language frameworks, design patterns, etc. affect clone production?

RQ4: What is the refactorable to non-refactorable clones ratio in each technology? Which clones could or could not be refactored?

These research questions concentrate on the quantitative comparison of clone production, calculating the clone-to-code ratio and average clone size metrics. These metrics can also be useful in similar studies. During our study, we observed that some similar type of clones exist in more than one projects. So, we not only studied clone production within projects but also looked for the clones present across multiple systems. These results will help analyze whether code clones are just caused by the underlying technology or there are some types of clones produced irrespective of underlying technology. The effect of development approach on cloning is also studied. As clones are duplicated or similar pieces of code, some of them can be removed by restructuring the code, called refactoring [30]. However, this type of code restructuring may not be possible in all cases. The last research question addresses the refactoring of clones in the chosen technologies.

4. Methodology

We analyzed the source codes of 11 social networking projects targeting the same set of requirements and implemented in C#, Java, PHP, and ROR. The basic architecture of the projects is also quite similar.

Table 1: Percentage of base language code

Project	Language Code	Average
PHP1	67%	77%
PHP2	87%	
J1	90%	90%
CS1	65%	44%
CS2	47%	
CS3	36%	
CS4	29%	
ROR1	6%	16%
ROR2	18%	
ROR3	25%	
ROR4	14%	

We limited our analysis to the code sections written in the base language only, ignoring the HTML code and the code segments written in scripting languages. For example, in one of the ROR projects, different source files include JavaScript, CSS, YAML, HTML, DOS Batch, Make, Python, and XML files. However, we analyzed only the Ruby files with the .rb extension. Similarly, only files with .cs, .java, and .php extensions were considered for analysis in the projects implemented in C#, Java, and PHP, respectively. Table I shows the percentage of code written in the base language for each project. After the detection of clones, we manually analyzed the projects for finding refactorable pieces of code.

4.1 Projects Details

We selected a set of 11 social network projects/applications developed in imperative, server side languages C#, Java, PHP, and ROR. The details of these applications, including the metrics related to their size, are given in Table 2.

The students in a graduate course offered at the Lahore University of Management Sciences (LUMS) developed the applications. The student teams comprised skillful persons with prior relevant industry experience of about two years on average.

The projects were built with tight deadlines in order to simulate the real-world experience of web development in a software industry, following the agile development methodology. The developers were provided with a set of features to be implemented every week and were evaluated against a functionality checklist. Figure 1 shows a few examples of the functionality points.

Functionality Points
1. User can upload a picture
2. User can create albums
3. Users can post comments on pictures
4. Users can 'like' pictures

Figure 1: Functionality points

The applications were developed on the basis of the same set of requirements. Further, the requirements were not vague; they were based on the popular social network application, Facebook. The design requirements for the projects were also well defined. Almost all projects followed the Model View Controller (MVC) design pattern in some form. Some of the projects were based on the standard MVC frameworks, e.g., CakePHP used by PHP1, Code Igniter used by PHP2, and others implemented their own MVC. However, CS2 used 2-tier architecture. This made the architecture of the projects similar to a considerable extent.

For the ROR and PHP applications, we considered only the app directories, which mostly contain the developer written code and, in some cases, also contain auto-generated code. Other directories, including plug-ins and CSS files, were not considered for this study.

Table 2 shows that, on average, PHP applications contain the maximum code, Ruby projects have the minimum code, and C# and Java lie in the middle. One of the main factors contributing towards the large code size in case of PHP is the use of standard frameworks: PHP1, PHP2, and CS3 used CakePHP, CodeIgniter, and Object Relational Mapping (ORM), respectively. The other factor is the total functionality implemented. Functionality count is the total number of functions implemented.

Table 3 shows that the total functionality points implemented by every project. ROR1 has the smallest size and it is so because this project has the minimum functionality point count. Functionality point count is the count of the number features implemented in a project.

C# projects have a mean of 306 functionality points with standard deviation of 16 whereas the ROR projects have a mean of 244 functionality points and standard deviation of 101.

4.2 Introduction to Tool

We used Clone Miner [9] for our work. This tool was initially developed to work with Java but was adapted to works with C#, Ruby and also PHP. It is a token-based clone detection tool that can detect clones on the simple clones basis and is able to generate structural clones based on the simple clones detected in the code. It detects simple clones in groups of simple clone sets (SCS). Simple clones detected during the first step are rearranged to generate structural clones.

Clone Miner detects clones within and across methods, files, and directories. It stores its output in the form of simple text files, filled with numbers describing features of the different types of cloning abstractions found in the analyzed system. For example, for *simple clones*, Clone Miner assigns an ID to each simple clone class and then associates the number of instances belonging to it as well as the length in tokens of each clone

Table 2: Project metric details

Project	**Language**	**No. of Files**	**LOC**	**Comment LOC**	**Input Size (In Tokens)**
PHP1	PHP	415	49,640	18709	521628
PHP2		226	44,524	12539	256801
J1	Java	88	10,494	1905	55044
CS1	C#	288	27,392	10,411	175526
CS2		129	28,418	8,976	164759
CS3		443	51,121	20,346	220408
CS4		213	16,182	2,038	102051
ROR1	Ruby	41	1,513	410	5530
ROR2		135	3,622	649	27255
ROR3		151	8,671	1209	14278
ROR4		63	3,426	315	15784

Table 3: Tokens to functionality ratio

Project	Functionality Count	Input Size (In Tokens)	Tokens/functionality (AVG)
PHP1	286	521628	1824
PHP2	366	256801	702
J1	185	55044	298
CS1	286	175526	614
CS2	321	164759	513
CS3	316	220408	697
CS4	301	102051	339
ROR1	144	5530	38
ROR2	374	27255	73
ROR3	269	14278	53
ROR4	188	15784	84

Table 4: Simple clone statistics

Project	SCC	Clone Instances	Instances/ SCC (AVG)	Maximum Clone Size	Average Clone Size
PHP1	1564	7294	4.7	850	69
PHP2	943	3438	3.6	577	54.9
J1	107	474	4.4	299	63.5
CS1	735	1947	2.6	977	71
CS2	659	3550	5.4	893	77.2
CS3	644	2779	4.3	337	67
CS4	348	1441	4.1	356	58
ROR1	6	13	2.2	96	49.4
ROR2	58	512	8.8	304	67.5
ROR3	37	77	2.1	160	51.8
ROR4	76	188	2.5	182	52.1
Average	**470.6**	**1973.9**	**4.1**	**457.4**	**61.9**

Table 5: Structural clones, MCC, and FCC

Project	SCS	SCS within File	SCS across File	MCC	MCC by File	MCC across File	FCC
PHP1	1365	374	924	294	267	122	74
PHP2	855	280	222	155	57	90	14
J1	98	47	20	35	16	6	2
CS1	538	138	253	87	77	25	9
CS2	641	130	272	82	68	27	11
CS3	375	221	383	76	90	35	24
CS4	324	91	174	64	57	14	7
ROR1	4	3	2	2	2	0	0
ROR2	46	43	26	6	38	2	1
ROR3	28	11	10	8	8	1	1
ROR4	67	13	28	15	10	2	3
Average	**395**	**123**	**210**	**75**	**63**	**29**	**13**

Table 6: PHP2 clone percentage among directories

Project Directories	Clone Count	Clone Percentage (%)
Controllers	110	13
Models	191	22
Libraries	12	1
Libraries/SimpleTest	102	12
Libraries/Extensions	7	1
Libraries/SimpleTest/Test	424	50
Views	9	1

instance. For each simple clone instance, it also indicates the file ID that contains the instance and the location of the instance in the file by stating the start and end line numbers of that instance.

Input parameters for Clone Miner include minimum similarities for simple clone classes, minimum similarities for method clone classes, and minimum similarities for file clone classes. Minimum similarity refers to the minimum size in tokens of similar code fragments that may be considered as valid clone.

We detect clones in individual projects separately, as shown in figure 2, and use the final results for the clones to answer our research questions.

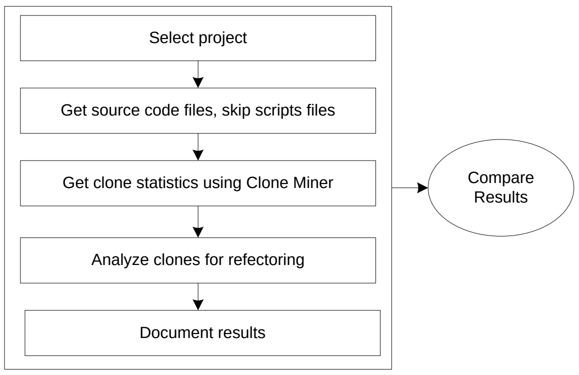
Select project
Get source code files, skip scripts files
Get clone statistics using Clone Miner
Analyze clones for refectoring
Document results
Compare Results

Figure 2: Methodology for study

4.3 Metrics

Simple Clone Classes (SCC): A simple clone class refers to simple clones detected in different files or methods. Clone Miner provides the number of classes of simple clones and instances of each class in the files, i.e. the number of times a clone is repeated in the project.

Structural Clones: Clone Miner uses the data about simple clones to identify structural clones. SCS can form structural clones.

Method Clone Classes (MCC) and File Clone Classes (FCC): Method and file clones are similar methods and files across a project. They are found through clustering of SCS.

Clone Coverage (CC): Projects are analyzed to find Clone Classes and Clone Instances. We modified Clone Miner to provide CC, which is the clone to code ratio.

$$\mathrm{CC} = \frac{\text{Cloned number of tokens}}{\text{Total number of tokens}} \tag{1}$$

Clone Size (CS): Clone size is the average length in tokens of clone instances in a given project. For each project, we recorded CS and the sizes of longest and shortest clones.

4.4 Parameters

We performed our study according to following parameters:

$\text{Clone size } > 30 \text{ tokens}$

$\text{Minimum similarities}_{\text{simple clone classes}} = 30$

$\text{Minimum similarities}_{\text{method clone classes}} = 30$

$\text{Minimum similarities}_{\text{file clone classes}} = 50$

5. Analyses Outcomes

As shown in Table 4, PHP and C# projects had the highest number of clones and ROR projects had the smallest. Note that for PHP and ROR projects, we ignored the default library/framework code that developers had to copy into their project directories. For example, PHP2 used the Cake PHP framework and in order to use this framework developers had to copy the framework files (that contain the PHP code) into their project directories. We ignored such framework code, as it was neither specific to the project nor written by the project developer. However, if some framework code was generated specific for a project, we considered it part of the project while searching for clones. For example, in CS1 the database mapping language file is used, which contains an object relation mapping of the project DB. Since this code is specific to the project, therefore, we considered it as project code.

Table 4 and Table 5 show that PHP projects contain the largest number of simple clone classes

(SCC), structural clones, method clone classes, and file clone classes. As shown in Table 2, the sizes of the PHP projects are also the largest because of the number of features implemented in these projects on average, as shown in Table 3.

ROR2 implements the maximum number of features and, as expected, also has the highest clone coverage. But, if we compare ROR2 and ROR3 with the PHP projects, it is clear that both are very close in terms of the number of features implemented by them. However, lines of code and the number of clones are considerably different in the two technologies. One reason for this difference is the use of standard frameworks in the PHP projects. Another reason is that ROR projects involve many scripting and markup languages along with the Ruby language, and we have focused on the Ruby language code only.

Yet another reason behind this difference is developer's programming approach. For example, there is a feature of comments in the social networking applications and this feature is used in multiple modules including posts, photo albums, and videos. In CS2, this code is copied and pasted across modules, whereas in CS3, the developer created a common control for this feature and used it instead of using copy-paste. The later approach reduces the code size as well as the number of clones in the code. Similarly, CS1 uses OR, which generates a lot of code automatically, whereas CS2 uses common functions and SQL queries.

Table 4 shows that clones are larger in C# in terms of their size. However, our analysis show that clone size is larger for the projects that involve some language provided framework, including LINQ-to-SQL and CakePHP, but most of these clones could be considered as non-refactorable clones.

Now, we discuss answers to our proposed research questions.

RQ1: Which technology produces more simple and structural clones, and what are the causes of clone production?

As shown in Table 4, the number of clones is largest in the PHP and C# applications, while ROR applications contain the smallest number of clones.

All projects considered together have a mean of 470.6 simple clone classes with a standard deviation of 493. However, C# projects have a mean of 596 simple clone class with standard deviation of 170 and ROR projects have a mean of 44 with standard deviation of 30. One reason for the small average for ROR projects is code size, which is smallest for the ROR projects, as shown in Table 2.

Table 5 shows the total number of structural clones, as well as structural clones across and within files. Structural clones are almost in the same ratio as simple clones among different technologies. These are due to the requirements composed of some of the already implemented requirements, as well as due to the software architecture and the coding approach followed by the developers.

Table 6 shows the clone percentages among different directories of a PHP project, making it visible that a large number of clones is detected in unit tests directory.

Table 7 shows clone-to-code ratio, i.e., clone coverage, for all projects. C# and PHP projects have the highest clone coverage. However, if we ignore clones due to auto generated code, code for unit testing, etc., C# comes at the top and ROR at the bottom. Reason for excluding these is that unit tests are the code snippets generated due to unit testing performed by developers, but has nothing to do with the features of functionality implemented in an application. It is also observed that a smaller piece of code in ROR can produce more functionality when compared with code written in Java, C#, or PHP.

The developer methodology is one of the causes of clone production. As the projects under our study were developed using the agile development approach, developers were provided with the requirements set every week to be implemented during the same week. It seems that because of the short deadlines, the developers mostly copied and pasted code fragments from the previously written code, instead of generalizing and refactoring it. This development technique resulted in a lot of clones. This is a reason which exists independent of underlying technology but it is a commonly observed reason for clones production. For example, the requirements for all projects included a 'comments' feature. This feature was repeated in different forms, e.g., post comments, photo comments, video comments, link comments, and question comments. Most of the development teams copied and pasted this code at all the places it was needed.

However, one of the teams created a generic control for comments once and then reused it where needed, instead of copying and pasting the whole feature code. This approach reduced such type of clones in their project. The overall clone count in this project is still high because of the use of framework and a lot of framework generated code. This team used a sort of 2-tier architecture, instead of using MVC, resulting in a lot of copied and pasted code.

Table 7: Clone coverage

Project	Clone Coverage
PHP1	37.50%
PHP2	50.80%
J1	33.70%
CS1	40.80%
CS2	50.51%
CS3	36.60%
CS4	40.45%
ROR1	11.51%
ROR2	51.65%
ROR3	21.815
ROR4	38.61%

Another observation is that there is a significant difference among projects developed using the same technologies. For example, there is a clear difference in the number of clones for the two C# projects: CS2 has 50.5% clone coverage and CS3 has 36.6% clone coverage. Similarly, the clone coverage for the two ROR projects, ROR2 and ROR3, are 51.65% and 21.81%, respectively. Reasons for these differences are wrong design and coding approach used by the developers, including not following standards, violation of architectural guidelines and rules, and not exploiting reuse, and, perhaps, lack of knowledge about a framework or technology. Some clones are due to similar requirements for different features also.

As shown in Table 4, clone sizes are larger in C# in terms of the maximum size, compared to clones in other technologies. Our analyses show that clone sizes are larger for projects that involve the use of some language provided frameworks, including an ORM in CS1 and CakePHP in PHP1. However, most of these clones could be considered non-refactorable. If some large clones are not due to the use of a framework rather its developer written code, developers tend to copy paste this piece of code if required somewhere else. One reason can be the effort and time required to look again into this large piece of code in making it reusable.

RQ2: Are there some structural clones that are present across multiple systems? Are they technology dependent?

There are structural clones that are present across systems. They are mostly due to the similar software architectures. Those generated by a framework are limited to that specific framework architecture and are technology dependent clones.

As MVC is an architectural pattern, following which an application is divided into three components, i.e. Model, View, and Controller. The study described in [16] reveals that there were more clones in the controller than in the model. Our study upholds these observations. One possible reason for this is that model is for dealing with data sources, e.g., database; changes in it do not cause much effect on the other components of application. Also, there are less frequent changes in a model as compared to a controller. Hence, if the model code is not auto generated, data source specific code should be added to the model whenever possible instead of adding it to the controller. We observed that if extra code is added to a controller, e.g., database related code; the chances of duplication of that piece of code are much higher.

Clones that are created due to the coding approach followed by the developers or due to the ease by copying and pasting, and not following the conventions of patterns, frameworks, etc., are independent of a specific language, as discussed above in case of the MVC architecture.

RQ3: Does the usage of language frameworks, design patterns, etc. affect clone production?

Some research has already been conducted on the effect of framework usage on cloning, but at a very limited level. [15] analyzes six web applications of different sizes developed in classic ASP.Net and ASP.Net MVC framework and tries to see the effect of the use of framework on the number of clones produced. The analyses indicate that the use of a framework affects cloning and there are significant differences with respect to the cloning level in web applications that are developed using frameworks in comparison to those developed without any framework. The author states applications developed using classic ASP.Net are more prone to cloning than those developed using ASP.Net MVC framework. However, the study is limited to only C# and the findings of this research may or may not hold for other languages.

In [16], the authors describe a study of two industrial dynamic web applications with distinct architectures to identify patterns of clones. One was developed using the traditional style with HTML and PHP and the other one used the MVC framework in PHP. Even though the two applications had different architectures, the paper reports that both applications had significant number of clones. However, the cloning patterns were different. The clones in the traditionally developed system were scattered in more files as compared to those found in the MVC based application. This study was limited to only two applications, both written in PHP.

Gamma and Helm [22] also describe that design patterns define abstraction in systems and this way help reduce the complexity of a system. These patterns can be considered reusable blocks contributing to the overall architecture of the system.

Our analyses say that the use of frameworks and design patterns affects clone production in all four technologies that we studied. For example, we found out that CS2 did not follow any standard software architecture and the developers had simply tried to loosely divide their code into two layers. Although they implemented generic controls for common features and used those instead of copying and pasting large fragments of feature code, yet the clone coverage for CS2 came out to be 50.51%, which is the highest among all C# projects. The primary reason for such high clone coverage is that the software architecture was not properly designed and implemented. Even in the generic controls, which were developed to support reusability, code clones were detected.

According to our analyses, there were almost 40% clones in the generic controls. C# projects have most clones and database access file in C# code have the highest percentage of clones, simply because of not following frameworks or architectural standards. Similarly, in the Java project also, most of clones are in the database access files. The reason is that connection to database, and then statements for specifying query, execution etc., are repeated in almost all functions. The common steps could be specified once and reused in each function.

The clone coverage in the Java project is 33%, which is less than those of C# projects. The main reason for this is the use of proper architecture. The project team used the struts MVC framework and made use of design patterns whenever required. A major reason of clone production in this project is the improper use of structs.

Similarly, ROR projects are MVC based and have the least clone coverage, whereas most C# projects do not follow architectures properly, which is a major cause of larger clone coverage in these projects.

Another type of clones that is significant in any project, irrespective of the underlying technology, is repeatedly doing initializations. Unnecessarily creating an object multiple times can be avoided by following design patterns.

However, there are some cases where the use of a framework itself introduces clones, and such clones are usually non-refactorable. For example, ROR projects include clones that are to facilitate requests from XML and HTML. These types of clones cannot be refactored. Similarly, clones are due to database level similarities too cannot be removed through code refactoring.

Thus, the usage of frameworks standardizes code, which causes fewer clones as compared to the code written from scratch without following any pattern of framework. However, framework may also cause clones. But the percentage of clones produced due to the use of a framework is lesser than the percentage of clones avoided by its use.

RQ4: What is the ratio of refactorable and non-refactorable clones in each technology? Which clones could (or should) be and which could (or should) not be refactored?

Refactoring is used to restructure code such that functionally remains the same, but code design improves [8]. As stated previously, code clones are a potential cause of greater maintenance cost, so it is usually advisable to detect and refactor them. But before attempting any refactoring, we should consider concerns such as software stability, code ownership, and design clarity.

Some of the refactoring techniques are:

- Extract Classes
- Extract Methods
- Replace Parameter with a Method
- Generalize Type

Clones help in identifying the code fragments that may be considered for refactoring. Most of the automatically generated code is non-refactorable. Generally, clones due to following reasons are not refactored:

- Database design
- Technology or framework limitations
- Maintenance benefits

Some frameworks automatically generate code on the basis of the database schema, e.g., ROR. In case of automatically generated code, if the database contains unnecessary or repetitive fields and improper relationships, this should not be refactored from code; rather the code should be regenerated after the database schema modifications.

Some of the clones cannot, or should not, be refactored. If we try to refactor them, they may cause the following problems:

Property Definitions: Property definitions are detected as clones but these cannot be refactored, because each property is used for a different purpose.

Language Provided Framework: Modification of the code auto-generated by a language framework is usually not recommended, because the framework uses it later for different actions. If we refactor such code fragments, the software architecture maybe disturbed, causing difficult-to-track bugs.

LINQ Queries: There are also some large LINQ (Language Integrated Query) queries that are identified as clones, but they also cannot be refactored because of framework-imposed limitations.

Table 8 shows the percentage of simple clones (as shown in Table 4) that can be refactored. As stated earlier, PHP contains the highest number of clones, most of which are in the library directory that includes extensions and unit tests. However, as shown in Table 8, C# projects contain the highest percentage of refactorable clones, followed by Java. ROR has the smallest percentage of refactorable clones.

Table 8: Refactorable clones

Project	**Refactorable Clones (%)**	**Average**
PHP1	30%	32.50%
PHP2	35%	
J1	63%	63%
CS1	62%	61%
CS2	72%	
CS3	48%	
CS4	61%	
ROR1	38%	41%
ROR2	23%	
ROR3	63%	
ROR4	38%	

Some of the reasons for the generation of refactorable clones are developer laziness, software architecture followed, and non-usage of technology provided features that help avoid clones, e.g., helper methods and ORM. Most of the code generated in PHP and ROR is automated because of the use of frameworks. The clones that cannot (or should not) be refactored are mostly due to language or technology dependence. Also, in the case of high coupling there are higher chances of introduction of errors in the code.

We analyzed manually simple clones in order to differentiate between the clones that could be refactored and those that could not be refactored. For each project, we processed a random sample of 25% SCC (simple clone classes) to identify the clones that could be refactored. Also, we looked for the reasons why some clones could not be refactored.

CS1 uses an ORM framework. In this project, 35% of the clones are in the sidekick.designer.cs file, which is an ORM file auto-generated by the language. Thus, the remaining 65% of the code in CS1 is considered for refactoring, out of which 62% is refactorable. However, CS2, CS3, and CS4 do not use ORM generated or any other such large amount of auto-generated code. Therefore, the whole software of these projects can be considered for refactoring decisions.

Some other clones, if refactored, can make software more complex and difficult to understand. Such clones are also not refactored in order to achieve maintenance benefits.

Summary, Conclusion, and Future Work

In this paper, we have presented the results of a study to compare popular web technologies in terms of cloning: C#, PHP, Java, and ROR. Our analyses focused on the base languages only. However, each of our projects involved the use of several other technologies as well, including JavaScript, CSS, ASP.Net, and JSP, but we did not consider the code segments in these technologies for our study. We chose a set of systems that all belonged to the same domain and were developed by groups of developers with more or less the same level of expertise, in the same time frame, and with the same set of requirements and deadlines. The analyses of our results give us insight about the technologies that are easier and effective from the maintenance point of view and how software written in those technologies could be made even more maintenance friendly.

Here is the summary of our findings and conclusions:

1. The number of clones in software depends on several factors including project size, project architectures, and developer approach.

2. Clone statistics for the software projects in the same technology vary due to the use of frameworks and the programming methodology used by the developer. Clone sizes also vary but not by much.

3. The clone count is highest in C# and lowest in ROR. The number of clone instances ranges between 77 and 7294, where PHP projects (5366 average clone instances) and C# projects (2430 average clone instances) contain the largest number of clone instances, while ROR projects (790 average clone instances) contain the smallest number of clone instances.
4. Clone coverage ranges from 11.51% to 51.65%.The PHP projects have the highest clone coverage with 44.15% average, followed by C# with 42.09% average clone coverage. However, ignoring the clones due to auto generated, technology dependent, and unit test code fragments brings C# at the top and ROR with 30.89% average at the bottom.
5. Average clone sizes (in tokens) for the different technologies range between 49.8 and 77.2. Clones in C# are larger as compared to other technologies with an average size of 68.3 tokens. ROR has the smallest clones with average clone size of 55.2 tokens. C# also had the largest clone of 977 tokens and ROR had the smallest clone of 96tokens.There are no significant differences across projects of the same technology with respect to clone size.
6. Structural clones are almost in the same ratio as simple clones, i.e., high in PHP and C#.
7. Some clones are language independent such as architecture dependent clones (e.g., due to MVC) and clones due to the coding approach followed and not following proper coding conventions.
8. Programming languages provide support for clone prevention, e.g., helper functions in PHP and ROR help improve design by normalizing repetitive pieces of code. Also, language IDEs have integrated refactoring features, thereby helping remove code clones.
9. The use of frameworks and design patterns mostly helps in preventing clones.
10. The percentage of refactorable clone ranges between 23 and 72. Java, with an average of 63% refactorable clones, contains the highest percentage of refactorable clones followed by C# with an average of 61% refactorable clones. ROR has the least percentage of refactorable clones, with an average of 41%.
11. Generally, clones generated due to the following reasons should not be refactored:
    a. Database design
    b. Technology or framework limitations
    c. Maintenance benefits
12. ROR provides most ease of maintenance because it has the smallest number of clones, smallest sized clones, and smallest percentage of refactorable clones. It can, therefore, be concluded that object-oriented ROR that has features of both imperative and functional languages, is much more expressive, and is better than pure imperative languages for maintenance purposes.

In future, we will extend this work to study the cloning characteristics of scripting languages and other technologies involved in developing web applications, in addition to base languages. Study can further be extended to analyze some open source projects that are being used and maintained professionally, instead of considering academia level projects. Moreover, we will work on the refactoring aspects in more detail.

Threats for validity:

Some of the threats to validity of the study are:

- All of the projects considered for study are based on the same domain, i.e., social networking web applications.
- There is just one project in Java language. So, it is hard to generalize the results of our study conclusions for Java.
- Server side scripting languages, including JSP, ASP.Net, HTML, and Java script are not considered in our study.

References

[1] Roy, C. K., & Cordy, J. R. (2010). Near-miss Function Clones in Open Source Software: An Empirical Study. Journal of Software Maintenance and EvolutionResearch and Practice - Working Conference on Reverse Engineering, , 22(3), 165-189.

[2] Rajapakse, D. C., & Jarzabek, S. (2005). An Investigation of Cloning in Web Applications. Int. Conf. on Web Engineering, Syndney, 924-925.

[3] Roy, C. K., & Cordy, J. R. (2010). Are Scripting Languages Really Different?. Proc. IWSC 2010, ICSE 4th International Workshop on Software Clones,Cape Town, South Africa, 17-24.

[4] Al-Ekram, R., Kapser, C., & Godfrey, M. (2005). Cloning by Accident: An Empirical Study of Source Code Cloning Across Software Systems. In ISESE, 2005. 376-385.

[5] Kapser, C., & Michael, W. G. (2008). “Cloning Considered Harmful” Considered Harmful. Patterns of Cloning in Software, Empirical Software Engineering. 13(6), 645-692.

[6] Yoshiki, H., Toshihiro, K., Shinji, K., & Katsuro, I. (2004). Refactoring Support Based on Code Clone Analysis. PROFES’04, Kansai Science City, Japan, April 2004.

[7] Basit, H. A., & Jarzabek, S. (2009). A Data Mining Approach for Detecting Higher-level Clones in Software. IEEE Transaction on Software Engineering, 35(4), 497-514.

[8] Martin, F. (1999). Refactoring: Improving the Design of Existing Code. Addison-Wesely, 1999.

[9] Basit, H. A., & Jarzabek, S. (2005). Detecting Higher-level Similarity Patterns in Programs. In proceedingd 10th European Software Engineering Conference and 13th ACM SISOFT International Symposium on the Foundations of Software Engineering, ACM Press, , Lisbon, Portugal, Spetember 2005.

[10] Basit, H. A., Ali U., Haque S., & Jarzabek, S. (2012). Things Structural Clones Tell that Simple Clones Don’t. Int. Conference on Software Maintenance, ICSM’2012, Trento, Italy, September 2012, 275-28.

[11] Monzur, M. M., Arifur, R. M., & Salah, U. A. (2012). A Literature Review of Code Clone Analysis to Improve Software Maintenance Process. CoRR, May, 2012.

[12] Monden, A., Nakae, D., Kamiya, T., & Sato, S. (2002). Software quality analysis by code clones in industrial legacy software. Proceedings of the Eighth IEEE Symposium on Software Metrics (METRICSí02), Ontario, Canada, June 2002.

[13] Juergens, E., Deissenboeck, F., Hummel, B., & Wagner, S. (2009). Do code clones matter?. Proceedings of 31stIEEE International Conference on Software Engineering, Washington, DC, USA, 485–495.

[14] Lozano, A., & Wermelinger, M. (2008). Assessing the effect of clones on changeability. Proceedings of the 24th IEEE International Conference on Software Maintenance, Beijing, China, 2008, 227–236.

[15] Rakibul, I., Rafiqul I., Maidul, I., & Tasneem, H. (2011). A Study of Code Cloning in Server Pages of Web Applications Developed Using Classic ASP.NET and ASP.NET MVC Framework. Proceedings of 14th International Conference on Computer and Infonnation Technology (ICCIT 2011), Dhaka, Bangladesh.

[16] Tariq, M., Minhaz, F. Z., Yosuke Y., & Chanchal K. R. (2013). Near Miss Clone Patterns in web Applications: An Empirical Study with Industrial Systems. 26th Annual IEEE Canadian Conference onElectrical and Computer Engineering (CCECE),Regina, SK, May, 2013, 1-6.

[17] Rhaman F., Christian, B., & Premkumar D. (2010). Clones: What is that Smell. Proc. IEEE Working Conf. on Mining Software Repositories, 2010, 72-81.

[18] Basit, H. A., Puglisi, S., Smyth, W., Turpin, A., & Jarzabek, S. (2007). Efficient token based clone detection with flexible tokenization. In Proceedings of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC-FSE), September 2007, 513-516.

[19] Basit, H. A., & Jarzabek, S. (2009). A data mining approach for detecting higher-level clones in software. IEEE Transactions on Software Engineering, 35(4), 497-514.

[20] Bellon, S., Axivion, G. S., Koschke, R., Antoniol, G., & Krinke, J. (2007). Comparison and evaluation of clone detection techniques. IEEE Transactions on Software Engineering, 33(9), 577 – 591.

[21] Chanchal K. R., James R. C., & Rainer K. (2009). Copmarison and evaluation of clone detection tools and techniques: a qualitative approach. Science of Computer Programming Journal, 74(7), 470-495.

[22] Mondal, M., Roy, C. K., & Schneider, K. A. (2015). A Comparative Study on the Bug-Proneness of Different Types of Code Clones. IEEE International Conference on

Software Maintenance and Evolution (ICSME), Bremen, Germany, 91-100.

[23] Koschke, R., & Bazrafshan, S. (2016). Software-Clone Rates in Open-Source Programs Written in C or C++. IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER), Suita, Japan, 2016.

[24] Manishankar, M., Chanchal, K. R., & Kevin, A. S. (2017). Does cloned code increase maintenance effort?. IEEE 11th International Workshop on Software Clones (IWSC), Klagenfurt, Austria, 21-21 Feb. 2017.

[25] Yusuke, Y., Yoshiki, H., & Shinji, K. (2017). A technique to detect multi-grained code clones. IEEE 11th International Workshop on Software Clones (IWSC), Klagenfurt, Austria, 21-21 Feb. 2017.

[26] Rattan, D., Bhatia, R., & Singh, M. (2013). Software clone detection: A systematic review. Information and Software Technology, 55(7), 1165-1199.

[27] Roy, C., Zibran, M., & Koschke, R. (2014). The vision of software clone management: Past present and future (keynote paper). 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering, 18-33.

[28] Svajlenko, J., & Roy, C. K. (2015). Evaluating clone detection tools with bigclonebench. Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution ICSME ‘15.

[29] Svajlenko, J., & Roy, C. K. (2014). Evaluating modern clone detection tools. Proceedings of the 2014 IEEE International Conference on Software Maintenance and Evolution ICSME.

[30] Tsantalis, N., Mazinanian, D., & Krishnan, G. P. (2015). Assessing the refactorability of software clones. IEEE Transactions on Software Engineering, 41(11),1055–1090.