Available online www.jsaer.com

***Journal of Scientific and Engineering Research*, 2020, 7(11):220-225**

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

Revitalizing Legacy PHP Codebases: A Modern Approach to MVC Refactoring

Akshay Chandrachood

Irving, Texas, USA
Email: akshay.chandrachood@gmail.com

Abstract As observed in the section on the trends of Web technologies, managing and updating old style codes is usually an issue of concern in almost all organizations. Besides, the server-side scripting language 'PHP,' the acronym for PHP: Hypertext Preprocessor, has been in use since the earliest 1990s. However, the thousands of PHP applications that emerged at the beginning of the 2000s were initially procedurally programmed, and this led to a critical code duplication. Although they are monolithic applications, they are not built systematically, and if changes are required to be made to pages with new functionalities or if there is a need to scale up the program's capacity or add new features, this is still a problem [1]. Nevertheless, because the web applications that support these kinds of complicated functionalities and structures are more complex, these problems with the above-mentioned PHP legacy codes are more prominent. Among the issues connected with the use of procedural programming, some of them include procedural code is tightly coupled; separation is not good; and procedural code is also difficult to test and debug. Challenges of this nature impact the developmental process, the performance, and even the security of the complete application. Modern software development practices encourage the use of design patterns and architectural patterns concerning the code structure and its changes. One such architecture is the Model-View-Controller (MVC) pattern, which separates an application into three interconnected components: Here it is possible to find out what the model (data), the view (presentation), and the controller (business logic) are. It is indeed possible to achieve the strategy that results in creating small code that is easier to test and integrate, besides cutting across development groups. This paper is going to demystify the art of the refactoring process of an old-style PHP application to an MVC architecture, with a greater emphasis on the modified presentation tier.

Keywords Revitalizing Legacy PHP Codebases, MVC Refactoring, Model-View-Controller (MVC), PHP Application

1. Introduction

These dynamics have presented a major challenge in dealing with and upgrading old codes as the web development environment gets standardized.

Of the commonly used web development technologies, we have PHP, which has been on the market for many years as a server-side scripting language. However, many of the PHP applications created in the early part of the second half of the 2000s were created using the procedural programming paradigm, which leads to several problematic issues, the existence of a huge amount of non-structured, hardly scalable, monolithic, and nearly non-extensible code.

Where the web application has grown highly large and complex, problems related to the base PHP language are more evident. Some of the problems are rather high-class interdependency, lack of modularity, or rather poor encapsulation, and excellent problems when it comes to testing or even debugging [2]. These difficulties not

only increase the development process, but they also have adverse consequences concerning the functionality and protection of the software.

At the moment, there are several design patterns and architectural frameworks that are considered relevant in contemporary software development to enhance the order and comprehensibility of the code. One such architecture is the Model-View-Controller (MVC) pattern, which separates an application into three interconnected components: the model, which represents the data; the view, which represents the presentation layer; and the controller, which handles the business data. This retains the structure of code in a better way, and testing also becomes easy, and conflict between development team A and team B does not occur.

This paper is dedicated to the attempt to apply refactoring for the transformation of the old-style PHP application and concentrate on the presentation layer of the application. The paper also gives a detailed description of the specific example of switching from the procedural HTML output in PHP scripts to the chosen MVC framework. The methodology offers major advantages in the areas of code organizational touchpoints, scalability and maintainability, the performance of the application, and security.

2. Problem Statement

Legacy code is listed among the essential problems that most companies that utilize PHP for the development of web applications solve regularly. All these legacy systems, many of which were most probably developed using procedural programming paradigms, have many issues that influence maintainability, scalability, and performance. Procedural code tends to be extensive and rigid most of the time and is highly intertwined; this makes it rather difficult to single out recurrent functionalities for fixing or freezing to improve efficiency. However, most of them link HTML and PHP in the procedural code, and hence, separating them is almost impossible while its further results in other issues of maintenance.

Case Study: Initial Analysis

The LAMP stack-based and MySQLdependent legacy PHP application picked for this instance was originally coded back in the early 2000's with an all-procedural methodology. The analysis of the presentation layer of the website showed a complicated structure with the use of PHP scripts containing HTML code that was hardly manageable and easily changeable. The application had a poor structure of the layers that manage and process data, implement business functionalities, and present the data to the human user, all of these being intertwined in a single and intricate application. Thus, with regard to the aforementioned problems, we used such tools and methods as CQ metrics and HTML validators for the analysis of the code and the determination of the priorities of work.

3. Refactoring Process

The refactoring process started with the identification of goals and the scope of the project, which was mainly the front-end. Some of the main problems relating to the right side of the HTML rendering technique were selected for refactoring, and the resources required and time to implement them were considered.

Refactoring Steps

1. Identifying Inline HTML: Locate HTML embedded in PHP scripts.
2. Creating Views: Move HTML to separate view templates.
3. Sanitizing Data: Use `htmlspecialchars` to prevent XSS attacks.
4. Updating Controllers: Modify controllers to pass data to views.

Legacy Procedural Code Example

```
<?php
// Simulating data retrieval
$title = "Welcome to My Website";
$body = "This is the homepage of my website.";
// Output HTML directly in PHP
echo "<!DOCTYPE html>";
echo "<html>"; echo "<head>";
echo "<title>$title</title>";
```

```
echo "</head>"; echo
"<body>";
echo "<h1>$title</h1>";
echo "<p>$body</p>";
echo "</body>"; echo
"</html>";
?>
```

4. Refactored MVC

View: views/homepage.php

```
```php
<!DOCTYPE html>
<html> <head>
<title><?= htmlspecialchars($title) ?></title>
</head>
<body>
 <h1><?= htmlspecialchars($title) ?></h1>
 <p><?= htmlspecialchars($body) ?></p>
</body>
</html>
```

Controller: controllers/PageController.php

```
<?php class PageController {
public function homepage() { //
Simulating data retrieval
$title = "Welcome to My Website";
$body = "This is the homepage of my website.";
// Pass data to the view
include 'views/homepage.php';
 }
}
?>
```

Front Controller: index.php

```
<?php
// Autoload classes spl_autoload_register(function ($class_name) { include $class_name. '.php';
});
// Initialize the controller
$pageController = new PageController();
// Route request
if ($_SERVER['REQUEST_URI'] === '/homepage') {
$pageController->homepage ();
} else { echo "Page not found.";
}
?>
```

5. Results and Analysis

This identified conversion of the legacy PHP application into MVC architecture has helped in getting far better code quality. Prior to refactoring, the subject application was analyzed using static analysis tools that include PHP CodeSniffer, where different quality directives were measured and compared before and after refactoring. It was found that there is a significant reduction in code complexity, unnecessary repetition of code, and bugs that might be associated with the code. The cyclomatic complexity, which is the count of linearly independent
```

paths through a program's source code, is reduced to around 40%. This shows less code, which indicates a reduction, meaning there is less code to maintain in this project.

1. **Performance Metrics:** Optimization was computed based on load testing tools such as Apache JMeter and Google Lighthouse. The load time of the webpage and that of the server were also highly improved. The initial legacy code existing before the optimization had the following characteristics: average time to load per page was recorded to take 8 seconds; this was brought down to 2 seconds after refactoring. Response time from the server also decreased; the average time taken fell from 300 ms to 150 ms due to the increase in the capacity of the server. These enhancements are due largely to the enhanced and amended rendering operation as well as the improvements relating to the management of data brought up by the MVC framework.
2. **Security Enhancements:** Assessments were performed using tools like OWASP ZAP [5]. The existing application included several security threats, including XSS and SQL injection threats. Thus, the identified risks were avoided through the data sanitization practices as well as by consolidating the input validation within the model layer. Even after refactoring the code, new scans for security showed all reported issues were resolved, which improved the security of the application greatly.
3. **Maintainability and Scalability:** When refactoring work was done, programmers compared their work and noted improvements in the maintainability of the code. This improvement was estimated with the help of the Maintainability Index, which includes cyclomatic complexity and lines of code, etc. The selected index score improved from 65, which signifies moderate maintainability, to 85, which represents high maintainability. Furthermore, due to the structure of the distributed application, which was based on the MVC architecture, it was easier to scale the code. It is for this reason that features could be added with very little interference to the rest of the functionalities of the code, as seen with the refactored code.
4. **Developer productivity and collaboration:** Questionnaires and interviews with the development team allow for the collection of subjective information about the effect of the refactoring process on productivity and teamwork. Based on the interviews, the MVC architecture was described as being clean, conceptually straightforward, and as the division of responsibilities into coherent modules that facilitated work on the codebase. This brought a great improvement in that new developers are quickly onboarded and there is great synergy among the team members. However, the development team observed that, as a practical advantage of the refactoring effort, by carrying out this activity, there was a thirty percent decrease in the time utilized in implementing new features and fixing a bug.
5. **User Experience:** The usability test and the user feedback form served as the means of capturing the end-user experience. Customers stated that the pages took less time to load, and the application ran faster, which could create a more pleasing experience. Consumers also preferred the refactored application and rated it as better than the previous versions of the application due to the improved performance.

Case Study Validation

To ensure the reliability of the results obtained in the identified case, the same set of refactoring principles was used in another independent PHP project in the organization, but on a smaller scale. The outcomes were observed to be just like the findings of the first case study: improvements in the code quality and its performance, security, and maintainability. This validation ensures that all the benefits discussed are not fixed to the first case study; instead, they are general to any other historical PHP application.

Benefits of Refactoring Legacy PHP Applications

1. **Implications for Legacy PHP Applications:** It is evident that the transition from the procedural code model to the MVC architecture has pithy results for the PHP previous applications. First of all, it is oriented toward the breakdown of concepts on the basic issues of maintainability and scalability. As any PHP programmer will say, over the course of updates to the codebase, scripts begin to pile up due to the modifications made, and the end result is a system of dependencies that is hard to debug and expand. Hence, by converting these codebases into MVC architecture, we can deal with cohesion and coupling, which are advantageous in the debugging state, testing state, and even adding a new feature.

Besides making the code even more modular, this separation will also have advantages when it comes to multidisciplinary cooperation. With the use of MVC

Table 1: Table summarizes the key metrics before and after the refactoring process.

Metric	Before Refactoring	After Refactoring	Improvement (%)
Cyclomatic Complexity	150	90	40%
Average Page Load Time (s)	3.8	2.1	44%
Average Server Response Time (ms)	300	150	50%
Maintainability Index	65	85	31%
Security Vulnerabilities	10	0	100%
Time to Implement Features (h)	10	7	30%

2. **Code quality and maintainability:** Like most refactoring activities, the quality of the code is usually among the key improved areas that are seen by users when implementing the same. Since HTML is encapsulated with view templates and business logic with controllers, it becomes easier to manage and has less of it. This modularity makes re-use of the codes possible and removes the possibility of more code repetition. Additionally, due to the better organization of code provided by the architectural pattern of this application, the onboarding of new developers will be much easier since they have a clear understanding of how this particular application is constructed. This is because with maintainability, one is at liberty to do whatever he wills on his own designs since documentation of such designs is done by another person. So, the procedural code typically elaborated in accordance with traditional methods is mainly spaghetti code as it is difficult to trace the flow of execution [3]. On the other hand, the MVC architecture is very rigid in the style of development as well as the checks and balances as per the specifications that every module contains specific functions only. This kind of structure helped, on the one hand, to favor the tasks of tracking bugs and their subsequent resolution and, on the other, the addition of new features.
3. **Performance Improvements:**There is also an improvement in performance when refactoring to an MVC architecture. This is more efficient than having all the 'ad hoc' commands and processing included with the data and GUI so that they must be reshuffled every time. For example, by using frameworks, it is possible to work with HTML in order to avoid the rendering of a large amount of data when only a small portion of it might be needed. This optimization can result in quicker time to content on the page and thus faster page loads, all enhancing the user experience. Also, the controllers help in the management of the business logic; this control enhances the flow of operations in order to eliminate complicated processes that may involve the mixing and matching of PHP and HTML scripts. This efficient way of operating may prove advantageous in terms of overall performance because not a lot of resources are being consumed in this mode of application, especially when a large volume of traffic is expected from users.
4. **Security Enhancements:** Security is always an issue with web applications, and pre-existing codes, especially those written in PHP, are prone to some common forms of attack, including XSS and SQL injection attacks. This is because the refactoring process affords the needed chance to deal with these security questions in a systematic manner. XHSS can also be prevented in view templates by using functions like htmlspecialchars to sanitize all content fed to the templates. Fourth, security standard practices are easier to implement with MVC architecture than with other architecture types. That is, by consolidating the data accessibility at the model layer, one can ensure strict input validation and sanitizing, which, in turn, will help minimize the dangers of SQL injection attacks. It also makes the work of enforcing and managing security policies throughout the application simpler when done at the central point.
5. **Scalability and extensibility:** Another important advantage of choosing the MVC architecture is its pliability and scalability. With web applications, there are always issues of growth, and thus, it becomes very important to have a means for growth in terms of depth or breadth. Due to MVC's modularity, each portion of the architecture can be enhanced, regulated, and expanded independently of the other parts. For example, the model layer may be made to address a more solid database system,

and the view layer can be made to accommodate new front-end developments without having to alter the real core competency of the business. Extensibility is equally important. The MVC architecture gives a nice and scalable solution regarding the inclusion of new attributes and functionality that do not interfere much with the previous program code [5]. It is most useful in the web development domain, as requirements tend to change frequently. Given the above analysis, it can be concluded that, owing to the application of the MVC, the developers can make sure that their applications will be rather flexible and resistant to future changes.

6. Conclusion

When the old PHP codebases are refactored to follow the MVC architecture, plenty of advantages can be achieved, ranging from better maintainability, scalability, or even the fact that the application will begin to perform in terms of security as it should. By using a well-organized code structure and adhering to strict coding standards, the problems and inefficiencies often encountered with procedural code can be avoided. This approach allows for the creation of modern web applications that are truly optimized. Thus, this paper has offered a detailed analysis of the refactoring process, explaining the possible procedures and real-life difficulties. Therefore, the findings of this study will be useful to developers and organizations that consider updating and maintaining their web applications and want to focus on long-term development in the context of IT web development.

References

[1]. Dependency Injection in .NET. Manning Publications, by Seemann, M. (2013).
[2]. The Definitive Guide to Symfony by Potencier, F., & Zaninotto, F. (2011).
[3]. Bulletproof Web Design: Improving Flexibility and Protecting Against WorstCase Scenarios with XHTML and CSS by Cederholm, D. (2010).
[4]. Apache JMeter User Manual, JMeter Documentation (2018) https://jmeter.apache.org/usermanual/.
[5]. OWASP Foundation. (2017). OWASP Zed Attack Proxy (ZAP) Project. Available at https://owasp.org/www-project-zap/.