

Performance-based Refactoring of Web Application: A Case of Public Transport

Anna Derezińska^a and Krzysztof Kwaśnik

Institute of Computer Science, Warsaw University of Technology, Nowowiejska 15/19, Warsaw, Poland

Keywords: Web Application, Performance Evaluation, Quality Improvement, Single Page Application, Web of Public Transport System.

Abstract: Performance issues are, among other quality attributes, important factors of web applications devoted to public services. Performance-based refactoring concerns program quality improvement when functional requirements but also selected non-functional requirements, such as clarity, user-friendliness, and security issues, remain preserved. We have examined three independent web applications supporting card processing for public transport widely used in different provinces. Based on the experience, a new web application has been developed. While using the Single Page Application approach it has been aimed at easing a client interaction. Further refactoring helped in the performance improvement. The general performance has been compared to those three applications. Benefits of the refactoring have been evaluated and discussed.

1 INTRODUCTION

Software refactoring has been primarily concerned with a systematic manipulation of object-oriented programs in order to improve their quality (Fowler, 2018). The distinguishing feature of any refactoring approach is preservation of functionality realized by an application of concern. Refactoring can be aimed at avoiding unwanted features, as bad smells, but also at achieving certain goals, for example, applying design patterns (Kierievsky, 2004). A challenging task could be refactoring directed at the improvement of various quality features, in particular an application performance. This problem has been addressed in our research and a case study performed. In this paper, an improvement of the performance measures has been discussed as primary effects of a web application refactoring. Other quality attributes, also related to the SQuaRE model (ISO/IEC, 2010, 2015), have been studied in (Derezińska and Kwasnik, 2020).


It should be stressed that the refactoring cannot be applied independently to other nonfunctional features, for example, security requirements. Therefore, we have to address the basic issues that should be preserved in a refactoring process. In the broader sense, different frozen spots (fixed

invariants) and hot spots (changeable features) can be considered in a refactoring process.

In this paper we focus on web applications that are devoted to public services. After examining different applications supporting card processing for public transport in three different provinces of our country, we have found several weaknesses. Moreover, all these application have been based on the Multi-Page Application model (MPA), whereas the Single-Page Application model (SPA) supposed to be more suitable in this domain.

A prototype Public Transport Web (PTW) has been developed using a different model, SPA instead of MPA. A set of activities has been performed, treating PTW as an object in a performance-based refactoring process. Experiments have been conducted to evaluate PTW in regard to other applications, and to compare PTW before and after refactoring. While focusing on the performance, the following research questions have been considered: (i) is really SPA more appropriate, (ii) how much we can benefit from the performance-based refactoring.

This paper is organized as follows: Section 2 introduces briefly MPA and SPA approaches. Related work is reported in Section 3. Section 4 describes the case study. The experimental results are presented and discussed in Section 5. Finally, Section 6 concludes the paper.

^a <https://orcid.org/0000-0001-8792-203X>

2 WEB APPLICATION MODELS

There are two main models of web applications: MPA (Multi-Page Application) and SPA (Single-Page Application).

An SPA works inside a browser and does not require page reloading during use. There are many advantages of this model. The application resources, i.e. HTML, CSS and Script files, are loaded once, therefore, the transmission overhead is reduced. The development is simpler than for MPA. However, SPA is exposed to XSS risks (Cross-Site Scripting), which lowers its security (Gupta and Gupta, 2015).

In the MPA model, each page sends a requests to the server and updates all its data, consequently it puts more requirements on performance. In order to limit the traffic between the browser and the server, additional methods could be used to prevent transmission of small amount of data, as e.g. AJAX. Using this model is simpler to make Search Engine Optimization (SEO). However, MPA is less suitable to build a mobile application.

The SPA approach can be supported by many libraries that wrap JavaScript language and implement Virtual DOM (Document Object Model) (Zou et al., 2014). V-DOM is a local copy HTML DOM that makes possible to recalculate actualization data. In this way, operations of the original HTML DOM can be omitted. The following frameworks of this kind belong to the most widely used:

React.js – developed and shared by Facebook in 2013. This mostly used framework does not cover all project utilities and has to be accompanied by intermediate layer as AJAX engine.

Angular.js – created and maintained by Google. It is a very comprehensive framework also supporting solutions of routing or asynchronous web application model. It requires usage of a domain language - TypeScript.

Vue.js – created by E. You, drawn on experience of Angular.js. In this case, usage of TypeScript is not obligatory.

These frameworks are open source, support JavaScript, and have been applied in professional solutions by many companies. An application structure is more flexible in the case of Vue.js. An advantage of this framework is also its simplicity.

3 RELATED WORK

The basic catalog of code refactoring transformations has become popular after the seminal work of

(Fowler, 2018). Notion of refactoring has also been further associated with different activities.

Various kinds of non-source code refactoring are surveyed in (Rochimah, Arifiani, and Insanittaqwa 2015). They have focused on refactoring of other software artifacts, such as UML models, requirements, software architecture, and refactoring in code with non-conventional detection techniques. Refactoring activities that are discussed in this paper, could be partly classified as refactoring of software architecture. However, no performance-based or refactoring aimed directly on a non-functional feature has been discussed in this survey. Moreover, it is not a universal remedy on development shortcomings, as “refactoring activities affect quality attributes in an inconsistent manner” (Kaur, Kaur and Kaur, 2019).

A great challenge to refactoring is ability to automate the transformation as much as possible (Baqais and Alshayeb, 2019). The most of research refers to automated code refactoring, while some also to model refactoring. There are also approaches to search-based refactoring (Mariani and Vergilio, 2017). However, automating of operations discussed in this paper is still an open issue.

Different activities can be undertaken to improve performance of a web application.

A set of methods to accelerate a process of an SPA application has been presented in (Stępniaak and Nowak, 2017). The authors pointed at the beneficial utilization of the following methods, *minification* of JavaScript, removing unused CSS rules and concatenation of resources. These outcomes are similar to our findings. The main difference is usage of another framework, AngularJS, and discussion of mechanisms specific to this solution.

Also SPA using AngularJS was a subject of research discussed in (Jadhav, Sawant and Deshmukh, 2015). The paper presents the main architectural features of SPA, and SEO (Search Engine Optimization) in SPA. However, no performance issues are concerned.

SEO expresses a position that could be reached by a website in search engines of web browsers (Gudivada, Rao and Paris, 2015). It is a crucial feature of many kinds of web applications, especially those supposed to be widely accessible in e-community. Any kind of refactoring transformation should take into account the impact on SEO.

Performance analysis of parallel web applications has been argued in (Verdu and Pajuelo, 2016). In our case, no JavaScripts of this kind have been considered, although, they could be counted to an extended set of refactoring actions in the future.

Experiments with the Jmeter performance evaluation tool have been reported (Kiran, Mohapatra and Swamy, 2015). They confirmed usefulness of the tool, referred to different technologies, so the detailed results are not comparable to ours.

The HTTP/2 protocol and its prioritization ability have been studied in (Wijnants, Marx, Quax, and Lamotte, 2018). They have found that HTTP/2 prioritization influence web performance, especially in terms of page load time, though the results depend strongly on the browser type.

Different evaluation tools have been used to assess software quality of web application, including also performance attribute (Kaur, Kaur, and Kaur, 2016), (Martinez-Fernandez et al., 2019). Some of the tools have also been used in the experiments reported here. Different metrics could also be applied for this purposes (Lew, Olsina, Becker, and Zhang, 2012).

There is a lot of research related to web application security. Selected threats have to be considered in the application discussed here, such as Cross-Site Cripting (XSS) (Gupta and Gupta, 2015), SQL Injections (Srivastava, 2014).

4 PUBLIC TRANSPORT WEB

In this section, we discuss the case study background, requirements, architecture, and refactoring.

4.1 Background

City cards are commonly used to integrate services related to public transport in big cities and whole regions. They offer modern access to the transport for citizens and visitors of the area under concern.

Cards of this kind are supported by web applications, which make possible to buy tickets of different tariff variants and associate them with the card. A typical functionality of the web also covers basic administration features, like registration of new clients, user login/logout, storing of history of transactions related to the card, updating of personal data, collecting of client comments, etc.

We have investigated public transport web applications that have been independently developed and applied in three different provinces of Poland: MKA – for Krakow and other cities of Malopolska province, <https://mka.malopolska.pl/en>, PEKA – for Poznan and its satellite towns, <https://www.peka.poznan.pl/SOP/login.jspb>, Waw Card – for Warsaw and its satellite towns, <https://www.wtp.waw.pl/en/>.

The applications are used by many clients, of various skill levels. The biggest reference area is of the Waw Card and includes above two millions of citizens and commuters.

The applications have been examined observing their normal operation and using dedicated tools to website quality evaluation (Derezinska and Kwasnik, 2020). The main drawback of the applications is a high number of activities required by a client to acquire rights to travel. Other problems referred to configuring some kinds of tickets, duplicating of data introduction, or difficulties of finding necessary information and links to perform ticket purchasing.

The websites are overwhelmed with many information of public transport organisation mixed with client panel and purchasing module. All applications are also based on the MPA approach.

4.2 Prototype Application

Based on the above experience, a Public Transport Web (PTW) prototype has been developed that delivers the basic services similarly to the discussed applications. The main requirements of ticket purchasing refer to the following activities:

- Long-term ticket can be bought.
- Parking voucher can be bought.
- A ticket can be configured, specifying travel zone, period, discount, combining with a parking voucher, etc.
- Ticket price should be updated according to the ticket configuration.
- All ticket options should be configured in one view of the website (no view change necessary).
- Before ticket purchasing, the complete configuration and the final price should be confirmed.
- A feedback about successful or unsuccessful purchasing should be provided to a user.

Activities supported by an application of this kind should be simple and straightforwardly operable by any client type. Client interactions should be quick and their number limited. A transition between views should not require loading additional data from a server. Expected data should be presented to a user after a delay no longer than 7 sec. This requirement follows the results of (Dennis and Taylor, 2006).

4.3 PTW Security Requirements

Apart from performance issues, the application should meet various security requirements. In this section, we address several topics to be concerned in

PTW. The identified general problems can be treated as frozen spots of the current or future refactoring, while some selected methods or technology could be still a subject of substitution. Additionally, fulfilling security requirements could have a considerable impact on the application performance.

4.3.1 Client Registration

Registration of a client requires delivery of a set of personal data. Structure of some data, as e-mail address or telephone number should be verified. Registration service should be separated from other events of the application. At least typical security concerns should also be devoted to processing a client password, like hiding, encryption, re-typing, strength verifying, secure transmitting to the data base. In PTW, a password is encoded with jsSHA (currently SHA-512) and stored in the data base.

4.3.2 Client Authentication

Client authentication is used for a client verification, usually based on a login and password but also an e-mail could be used. In PTW, authentication is performed by the server, where an encoded password is delivered. Successful authentication initiates a client session with a time limit, managed by vue-session of Vue.js.

4.3.3 Application Responsiveness

It is preferred that an application accommodates to desired sizes of all devices it is displayed on. In PTW, operations on styles are used for this purpose. CSS scripts are loaded when certain conditions are fulfilled.

A MobileFirst approach could also be used, when a website is designed first for a mobile device, and then for other device types. The website is then displayed the most quickly on a mobile device.

An alternative solution is development of a separate web application dedicated for the mobile devices. In this case, however, the SEO attribute would be lower, because a single site address is assessed higher than two adapted sites (Gudivada, 2015). The higher SEO the better is a site position in a browser and a better client accessibility.

4.3.4 Data Security

There are different security threats related to data manipulation or interception, that should be taken into account in PTW.

XSS (Cross-Site Scripting) attacks should be avoided (Gupta and Gupta, 2015). It concerns introduction of unwanted JavaScript code to a webpage from a client site, to get access to sensitive data (Gupta, Gupta, Gangwar, Kumar, and Meena, 2015). While developing an application based on Vue.js, Reflected XSS and DOM Based XSS have not to be taken into account, as no information has to be made available nor fetched from a client URL. However, because of usage of a dynamically refreshed DOM code, other XSS attacks should be considered. A pure HTML code can be directly introduced into a view component. This feature (so called v-html) should not be applied on data delivered by a user. Moreover, in PTW, the number of data transmitted by user are strongly limited, which prevents delivery of complex JavaScripts.

The application should also be resistant against SQL Injections (Srivastava, 2014). Potential threat concerns transmission of data to the data base. In PTW, an additional EntityFramework is used to send values of queries as parameters to the data base instead of direct queries, which could have been manipulated.

4.4 PTW Architecture

The PTW prototype has been developed as SPA using the Vue.js framework. The application is placed on a Cloud server and consists of the following components: a view, an intermediate layer and a database project. The view is created by Vue.js and communicates with the intermediate layer. The second component is based on the MVC pattern (Leff and Rayfield, 2001). It transfers data between the data base project and the view. The database project communicates with an external data base (Figure 1).

The MVC project has been implemented using the .NET framework, the controller actions in C#, and the view in HTML with VS extensions.

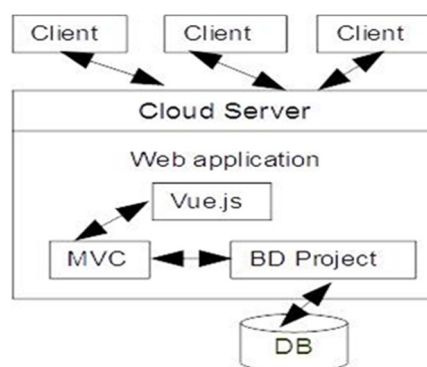


Figure 1: PTW architecture.

4.5 PTW Refactoring

PTW has been refactored in order to improve its quality attributes, especially performance. The refactored application should support the same functionality and other non-functional requirements; in particular security constraints and usefulness facilities, should be preserved or improved. Below, we review modifications that have been introduced.

Different resources have been combined together into one request, lowering in this way a total number of requests (Grigorik, 2013).

The content and appearance of the website have been modified, in order to better fit to the resolution of a target device.

Data compression has been performed for the selected types of files. The gzip compression has been switched on in the configuration file of the application that is used by the Cloud server.

Superfluous CSS rules have been deleted, and, therefore, files to be delivered to a browser have been reduced. Such reduction can be automatically performed by browser tools, or an external tool (Grunt-uncss, 2020).

File minification has been applied to all CSS and JS files. During this action, unnecessary characters such as spaces, tabs, blank lines, and comments have been removed. We used a tool (UglifyJS 3, 2020).

The protocol HTTP/2.0 has been used instead of HTTP/1. In the primary solution, each request was queued, which influenced the duration of the website loading time. The newer protocol provides multiplexing of requests, therefore, communication between the server and the browser can be completed in shorter time (Wijnants Marx, Quax, and Lamotte, 2018). Moreover, usage of this protocol positively influences the SEO attribute.

The file types have been recognized, namely: those that have to be loaded directly at the beginning and which can be postponed and handled asynchronously. In result, some scripts do not block the website processing. This has been realized using *async* and *defer* attributes in JavaScript.

All pictures used in the application have been compressed. The substantial size reduction was obtained using the tool (WebP, 2020).

Data transmission has been performed with the https protocol, an encoded version of http, improving the data transmission security.

A cache memory has been used to store files. The appropriate configuration parameters have been changed to allow storing of the selected types of files.

Adaptations have been performed in order to better adjust the application to mobile devices.

5 EXPERIMENTS

Various performance features of PTW before and after refactoring, and performance of the public transport applications (MKA, PEKA and WAW Card) have been experimentally evaluated. General performance and other quality factors could have been measured on all applications (Sec. 5.1.1). These kind of experiments are comparable and do not depend on parameters of a local client computer.

Other experiments referred to performance tests on PTW variants only (Sec. 5.1.2.) – (Sec. 5.1.4). They required direct access to an application under development, not only to its website version available for a user. Results of tests from Sec. 5.1.2 and 5.1.3 depend on parameters of a local computer. The experiments have been carried out using a computer with processor Intel Core i5, 2.30 GHz, RAM 16 GB, Windows 10x64, and integrated graphics. Transmission speed of collecting data was about 20 Mb/s. We used the Microsoft Azure cloud service .

5.1 Experiment Results

Experiment results have been delivered by different tools: Site Analyzer and Website Grader (Sec. 5.1.1), developer tools of Google Chrome v.74.0.3729.131 (64-bit) (Sec. 5.1.2, 5.1.3), and JMeter (Sec. 5.1.4).

5.1.1 Performance Reported by Quality Evaluation Tools of Web Applications

Quality of three reference applications, the PTW prototype, and PTW after refactoring have been calculated by different quality evaluation tools. Functionality taken into account in all five applications covered the same basic requirements (Sec. 4.2). Two quality evaluation tools returned also performance among other measured attributes. Site Analyzer assessed performance with a range from 0 to 100 points. Another tool, Website Grader used a scale from 0 to max 30 points. The results are given in Table 1.

Table 1: Comparison of performance.

Web application	Perform. by Site Analyzer [max 100]	Perform. by Website Grader [max 30]
MKA	61.0	12
PEKA	61.0	12
Waw Card	73.2	14
PTW	73.2	24
Refactored PTW	73.2	30

5.1.2 General Performance Evaluated by Developer Tool

Evaluation of quality attributes of the software under development can also be performed by developer tools available in a browser, e.g. Chrome. Results of assessment of the performance attribute of PTW before and after refactoring are shown in Table 2.

Table 2: Performance by developer tool of Chrome.

Web application	Performance by developer tool of Chrome [max 100]
PTW	7
Refactored PTW	91

5.1.3 Performance of Website Loading

Other performance tests have been carried out to evaluate mean time that is necessary to present a website to a user. The measurements were performed with the developer tool contained in Google Chrome. All tests were repeated 30 times, and average values calculated. The results are given in Table 3.

Table 3: Comparison of web loading performance.

Measurement	PTW	Refact PTW without cache	Refact PTW with cache
Mean time of DOM tree loading [s]	4.01	0.76	0.40
Mean time of web loading until starting of JS execution [s]	5.35	0.96	0.72
Average total loading time of all web elements [s]	12.48	1.38	0.88
Average downloaded data/web size	4.55 mb / 4.55 mb	740 kb / 1.5 mb	15kb / 1.5mb
Average number of requests	44	42	42

One of the refactoring options was enabling usage of local cache of browser to store files. Therefore, the measurements have been conducted twice: without cache and with cache used.

5.1.4 Performance Testing with Jmeter

Web application performance tests that could identify workload problems can be performed with the Jmeter tool (Kiran, 2015). Simulations affecting a server performance and an application under test can be executed for different application types and protocols.

For PTW, the performance tests have been conducted with the following parameters: number of

threads that simulate number of users (7000), rump-up period, i.e. the time interval during which all the threads are initiated (500 s), loop count – the number of repetitions of each thread (5).

Each thread reported its execution time, latency, size and status, i.e. whether the request was successfully delivered to the server. The exemplary results of a testing session are shown in Table 4.

Table 4: Results of performance testing with Jmeter.

Label	HTTP Request
#Samples	5998
Average	2024
Median	1873
90% line	4177
95% line	4410
99% line	4768
Min	14
Max	6519
Error [%]	40.76 [%]

5.2 Discussion of Results

According to the Website Grader tool, PTW has better performance than the other web applications even before refactoring (Table 1). After refactoring this attribute reached the maximum value (30/30).

Evaluation with another tool (Site Analyzer) also showed the best result in comparison to other applications. However, no performance improvement was reported after refactoring of PTW. Contrasting this result to other tools, we have doubts whether the performance results of Site Analyzer are worthwhile.

Measurements conducted by the developer tools of the Chrome browser showed a very big increase in performance (from 7 to 91 points), which reached a level close to maximum (100 points).

Moreover, besides the numeric values, the quality evaluation tools have returned some hints on the application improvement. Remarks of Website Grader and tools from Chrome were beneficial, whereas those from Site Analyzer were too general and not practically applicable.

A considerable improvement has been observed in loading times (Table 3.). The average time of loading all website elements has dropped from 12.48 s to 1.38 s, or to 0.88 s with the cache support. This was mainly due to lowering of the site size because of file minification and compression of picture and other data. Selection of files to be loaded first resulted in shortening time delay before starting JavaScript, from 5.35 s to 0.96 s or 0.72 s, accordingly.

The PTW application was placed in the cloud server and, therefore, some limitations have been

observed during the performance tests with Jmeter. The server requests have been successfully handled up to the limit (almost 6000 users sending requests in 500 s), conforming the sufficient performance. There was no decline in the website performance observed. But afterwards, the remaining requests have been blocked by the server due to the limits exceed. Those have been classified as error requests in Table 4.

5.3 Threats to Validity

In this section we examine the threats to validity of this study as described by (Wohlin et al., 2000).

Considering construct validity, we should look at various tools used in experiments.

Performance of web site loading have been measured by developer tools supported by the Google Chrome browser. In order to alleviate generalisation bias, each experiment has been repeated 30 times with the same workload, and the final result calculated as an average value. The tests have been conducted using Google Chrome, and using a different browser we can get some diverse outcomes. These measurements depend also on the connection speed, which could influence time of getting results. Additional factors affecting this time are also parameters of computer on which the experiments have been carried out, including computer processor, graphical card, and RAM memory. However, the essential goal of the measurement was not getting some absolute values, but comparison of the application performance before and after refactoring.

A set of performance measurements were performed using the JMeter tool and an external server. The tests have been repeated 30 times, to get an averaged values. The time of a server response could depend on the traffic intensity on the server, but the measurement errors were not noticeable. Local factors, such as parameters of a local computer and of a transmission connection have no influence of this kind of measurement.

Evaluations delivered by the quality evaluation tools (Site Analyzer and Website Grader) should be at the lowest estimate endangered by measurement errors. They were assumed to be independent of external or local conditions. Nevertheless, this sort of experiments were repeated in different conditions (different local computers, under different workloads, during different daytimes, etc.) and the results were exactly the same, confirming our assumptions.

As for conclusion validity, we should be careful while discussing the same quality attributes but provided by different tools. The tools use different criteria for assessment of the attributes under the

same names. Therefore we have compared the results provided by each tool separately. Furthermore internal quality is bounded by the whole process measurements. Many refactoring activities impact on the given process as a whole; we were not assessing effects caused by the activities independently.

External validity is restricted, as the whole refactoring process undergoes only one application created in the selected technology.

6 CONCLUSIONS

The web application performance, and other quality factors, can be considerably improved. The general performance attribute of our SPA application was equal or higher than of the three MPA applications. Different refactoring activities have been shown to positively influence the performance results. The improvement reported by Chrome was very big (from 7 to 91), and all time characteristics of web loading increased significantly.

However, the conclusions are limited, as absolute outcomes vary in accordance to an application examined and the technology used. They could be different in the case of various SPA frameworks, as, for example, React.js, or Angular.js.

While considering this type of refactoring, should be carefully taken into account which application features have to be preserved, especially concerning its security, and other non-functional constraints. Another big challenge remains possibility of automating such kind of refactoring. Most of the operations do not refer to a direct code substitution, as in the traditional code refactoring, but relate to file manipulation, often with use of additional tools, changing of configuration parameters, or protocols.

REFERENCES

- Baqais, B.A.A., and Alshayeb, M., 2019. Automatic software refactoring: a systematic literature review. *Software Quality Journal*. 3 Dec. doi: 10.1007/s11219-019-09477-y
- Dennis, A.R. and Taylor, N.J., 2006. Information foraging on the web: The effects of “acceptable” Internet delays on multi-page information search behavior. *Decision Support Systems* 42(2), 810-824. doi: 10.1016/j.dss.2005.05.032
- Derezinska, A., Kwaśnik, K., 2020. Evaluation and improvement of web application quality – a case study. In: Zamojski W., Mazurkiewicz, J., Sugier, J., Walkowiak, T., and Kacprzyk, J., (eds) *New Advances in Dependability of Complex Systems: Proc. of 15th*

- Inter. Conf. on Dependability and Complex Systems*. AICS, Springer, Cham. to appear.
- Fowler, M., 2018. *Refactoring: improving the design of existing code* (2nd ed.). Addison-Wesley.
- Grigorik, I., 2013. *High Performance Browser Networking: What Every Web Developer Should Know About Networking and Web Performance*. O'Reilly Media.
- Grunt-uncss, A grunt task for removing unused CSS from your projects with UnCSS. [Online] Available from: <https://github.com/uncss/grunt-uncss> [Accessed: 23 Feb 2020]
- Gudivada, V.N., Rao, D. and Paris, J., 2015. Understanding Search-Engine Optimization, *Computer*. 48(10) IEEE, pp. 43-52. doi: 10.1109/MC.2015.297
- Gupta, S. and Gupta, B.B., 2015. Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art, *International Journal of System Assurance Engineering and Management*. 8(1), pp. 512-530, doi: 10.1007/s13198-015-0376-0
- Gupta, B.B., Gupta, S. Gangwar, S., Kumar, M., and Meena, P. K., 2015. Cross-Site Scripting (XSS) Abuse and Defense: Exploitation on Several Testing Bed Environments and Its Defense. *Journal of Information Privacy & Security*; Abingdon 11(2), pp. 118-136. doi: 10.1080/15536548.2015.1044865
- ISO/IEC 25010:2011 Systems and software engineering. Systems and software quality requirements and evaluation (SQuARE). System and software quality models, 2010.
- ISO/IEC 25023:2016 Software engineering: software product quality requirements and evaluation (SQuARE): Measurement of system and software quality. 2015.
- Jadhav, M.A., Sawant, B.R., and Deshmukh, A., 2015. Single Page Application using AngularJS. *International Journal of Computer Science and Information Technologies*, 2876-2879. doi: 10.1.1.736.4771
- Kaur, S., Kaur, K., and Kaur, K., 2016. Analysis of Website Usability Evaluation Methods. In: *Proceedings of 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 1043-1046. IEEE, New York.
- Kaur, S., and Singh, P., 2019. How does object-oriented code refactoring influence software quality? Research landscape and challenges. *Journal of Systems and Software*. 157 (11), Elsevier. doi: 10.1016/j.jss.2019.110394
- Kierevsky, J., 2004. *Refactoring to patterns*. Addison Wesley.
- Kiran, S., Mohapatra, A., and Swamy, R., 2015. Experiences in performance testing of web applications with Unified Authentication platform using Jmeter. In: *International Symposium on Technology Management and Emerging Technologies (ISTMET)*, IEEE, pp. 74-78. doi: 10.1109/ISTMET.2015.7359004
- Leff, A., and Rayfield, J.T., 2001. Web-Application Development Using the Model/View/Controller Design Pattern. In: *Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference*, IEEE, pp.118-127, doi: 10.1109/EDOC.2001.950428
- Lew, P., Olsina, L., Becker, P., and Zhang, L., 2012. An integrated strategy to systematically understand and manage quality in use for web applications. *Requirements Eng.* 17, 299-330. doi: 10.1007/s00766-011-0128-x
- Mariani, T., and Vergilio, S. R., 2017. A systematic review on search-based refactoring. *Information and Software Technology*. 83(3) pp. 14-34. Elsevier. doi: 10.1016/j.infsof.2016.11.009
- Martinez-Fernandez, S., Vollmer, A. M., Jedlitschka, A., Franch, X., Lopez, L., Ram, P., Rodriguez, P., Aaramaa, S., Bagnato, A., Choraś, M., and Partanen, J., 2019. Continuously Assessing and Improving Software Quality with Software Analytics Tools: a Case Study. *IEEE Access* 7. doi: 0.1109/ACCESS.2019.2917403
- Rochimah, S., Arifiani, S., and Insanitaqwa, V.F., 2015. Non-Source Code Refactoring: A Systematic Literature Review. *International Journal of Software Engineering and Its Applications*. 9(6) pp.197-214. doi: 10.14257/ijseia.2015.9.6.19
- Srivastava, M., 2014. Algorithm to prevent back end database against SQL injection attacks. In: *International Conference on Computing for Sustainable Global Development (INDIACom)*, IEEE, pp. 754-757, doi:10.1109/IndiaCom.2014.6828063
- Stepniak, W., and Nowak, Z., 2017. Performance Analysis of SPA Web Systems. In: Borzemski L., Grzech, A., Świątek, J., and Wilamowska, Z., (eds) *Proceedings of 37th International Conference on Information Systems Architecture and Technology – ISAT 2016*. AICS vol 521. Springer, Cham, pp. 235-247. doi: 10.1007/978-3-319-46583-8_19
- UglifyJS 3: Online JavaScript minifier. [Online] Available from: <https://skalman.github.io/UglifyJS-online/> [Accessed 31 Jan 2020]
- Verdu, J., and Pajuelo, A., 2016. Performance scalability analysis of JavaScript applications with web workers. *IEEE Computer Architecture Letters*. 15(2), pp.105-108. doi: 10.1109/LCA.2015.2494585
- WebP A new image format for the Web. [Online] [Accessed 31 Jan 2020] Available from: <https://developers.google.com/speed/webp>
- Wijnants, M., Marx, R., Quax, P., and Lamotte, W., 2018. HTTP/2 Prioritization and its Impact on Web Performance. In: *Proceedings of the 2018 World Wide Web Conference, (WWW '18)* pp. 1755-1764, doi: 10.1145/3178876.3186181
- Wohlin, C., Runeson, P., Host, M., Ohlsson, M. C. Regnell, B. and Wessln, A., 2000. *Experimentation in Software Engineering - An Introduction*. Springer. Berlin Heidelberg.
- Zou, Y., Chen, Z., Zheng, Y., Zhang, X., and Gao, Z., 2014. Virtual DOM coverage for effective testing of dynamic web applications. In: *Proceedings of the 2014 International Symposium on Software Testing and Analysis, ISSTA 2014*. ACM New York, pp. 60-70, doi: 10.1145/2610384.2610399