Quality Rule Violations in SharePoint Applications: An Empirical Study in Industry

Apostolos Ampatzoglou[1], Paris Avgeriou[1], Thom Koenders[2], Pascal van Alphen[2], Ioannis Stamelos[3]

[1] Department of Computer Science, University of Groningen, Groningen, Netherlands
[2] SharePoint Department, Capgemini Netherlands, Utrecht, Netherlands
[3] Department of Computer Science, Aristotle University, Thessaloniki, Greece

apostolos.ampatzoglou@gmail.com, paris@cs.rug.nl, thomkoenders@gmail.com, pascal.van.alphen@capgemini.com, stamelos@csd.auth.gr

Abstract. In this paper, we focus on source code quality assessment for SharePoint applications, which is a powerful framework for developing software by combining imperative and declarative programming. In particular, we present an industrial case study conducted in a software consulting/development company in Netherlands, which aimed at: identifying the most common SharePoint quality rule violations and their severity. The results indicate that the most frequent rule violations are identified in the JavaScript part of the applications, and that the most severe ones are related to correctness, security and deployment. The aforementioned results can be exploited by both researchers and practitioners, in terms of future research directions, and to inform the quality assurance process.

Keywords: Quality assessment · Defect prediction · SharePoint

1 Introduction

Although some organizations have such unique business processes that urge for custom-made software solutions, in most cases, business processes are fairly typical; therefore, their needs can be accommodated using more standardized solutions (e.g., by reusing existing commercial-off-the-self—COTS software). This has led to a shift from developing custom-made solutions to standardized solutions that are based on modular development. A prominent way to develop such solutions is SharePoint, in which standardized COTS are combined to offer the required functionality. These standardized modules can be configured to target and facilitate the specific needs of diverse organizations. The configuration of such solutions can be performed by using a declarative programming language (e.g., XML). Furthermore, SharePoint can be extended with custom modules and code, so as to provide functionality that is not found in the offered COTS. This way, solutions can be created by combining standardized off-the-self components and organization-specific modules (developed using an imperative language—e.g., C#).

Similarly to conventional software development maintainability of SharePoint applications is of paramount importance, since maintenance of a software system is considered as the most effort-intensive part of the software development lifecycle

[11], urging organizations to increase maintainability to reduce the overall maintenance expenditures [1]. In this paper we study a particular aspect of maintainability, namely source code quality. Source code quality has been assessed in many ways in the literature: through software metrics [2], number of defects [10], number of vulnerabilities [3], etc. Focusing on the number of vulnerabilities (also known as quality rule violations) has certain benefits: (a) they are more easily interpreted compared to metrics, since they are targeting specific lines of code and the way to resolve them is simple; and (b) they can be handled at a pre-deployment phase (in contrast to defects), thus they do not reach the attention of the customer/end-user. A common practice for identifying such rule violations is code reviews [6], which can be performed manually (***code inspection***), or with tool-support (***tool-assisted code review***). The latter enables not only distributed reviewing, but also improves both the quality and the quantity of reviews [7].

The goal of this paper is to investigate the source code quality of SharePoint applications, through tool-assisted code reviews. Specifically, we aim at investigating which rules: (a) are more frequently violated; and (b) are the most severe ones depending on certain characteristics. The identification of the most frequently violated rules can provide insights into what are the most common "programming mistakes". Among those, special attention should be given to the rules that are most severe, as well as those that are more probable to lead to defects, as the number of defects is crucial for the success of a software system. To achieve this goal, we performed am industrial case study, based on the guidelines provided by Runeson et al. [9] (more details are available in Section 3).

2 Related Work

In this section we present research efforts that can be characterized as related work to our study. We note that, to the best of our knowledge, there are no available studies in SharePoint quality assessment. Therefore, we present related work that uses rule violations as a means for quality assessment. Quality rule violations have been extensively investigated as indicators of quality in the software engineering literature. For example, Misra and Bhavsar [8] have explored rule violations as indicators for correctness, and Zaman et al. [12] have explored them as indicators for security and performance. When using rule violations to quantify quality, it is a common practice to classify them into categories. Zaman et al. [12] classified rule violations according to their effect on specific QAs (e.g., security and performance). Therefore, to evaluate software projects with respect to their quality, one can perform static analysis by collecting the number of violated rules. One of the most established tools that is used for this purpose is FindBugs. FindBugs is capable of detecting vulnerabilities in software by using bug patterns [4], divided into five categories (in total 246 bug patterns) that can be mapped to: correctness, performance, and security. In this study, since FindBugs is not applicable for SharePoint applications, we used SPCAF (see Section 3.3). Other tools that perform such analysis for different programming languages are PMD and CPPcheck.

3 Case Study Design

Objectives and Research Questions. The goal of the study is described using the Goal-Question-Metric (GQM) approach, as follows: "*analyze* quality rule violations for SharePoint applications *for the purpose* of evaluation, *with respect to* their (a) frequency of occurrence, and (b) severity according to certain characteristics, *from the viewpoint of* software engineers, *in the context of* SharePoint application development". Based to the aforementioned goal, we derived two research questions that guide the design and reporting of the case study.

RQ_1: What are the most frequently violated rules in SharePoint applications?
This research question aims at identifying the rules that are most frequently violated in real-world SharePoint applications. Existing tools for SharePoint quality assurance are capable of assessing approximately 400 predefined quality rules, so we will explore which ones are more frequent. In addition, we investigate the frequency of rules at different levels of criticality (e.g., warning, error, critical error, etc.)

RQ_2: What is the severity of SharePoint rule violations?
SharePoint quality rules can be classified based on several characteristics (e.g., some rules are related to correctness). Before specific defects can be detected, it is important to determine what kinds of rules are more severe, either due to the difficulty of their manual identification or due to their criticality. Therefore, this research question is focused on the orientation of preventive maintenance activities (i.e., the ones that aim at identifying and correcting latent faults in the software product before they become effective faults [11]).

Case Selection: This study involves different cases for each research question. For RQ_1, as cases we used three projects developed at Capgemini. The three projects have been selected so as to belong to a different production stage (ranging from 'under development' to 'production-ready' versions). The projects are referenced as Project-A, Project-B, and Project-C for confidentiality reasons. Project-A was developed internally by Capgemini. At the time of the analysis, the project was still in its early stages of development. Project-B was developed by an external organization and Capgemini was managing the code-base. At the time of the analysis, this code-base was ready for use in a production environment according to the external organization that was responsible for the development. Finally, Project-C had been developed internally by Capgemini. At the time of the analysis, the code-base was being used in a production environment. Concerning RQ_2, we conducted a supervised survey with SharePoint experts, so as to investigate the relationship of quality rules and defects. The term supervised survey [5] refers to the process during which an interview takes place, but with a specific data collection instrument (questionnaire) with mostly closed questions that the responded would be able to complete without any guidance. Nevertheless, in supervised surveys the researcher is in the same place as the subjects to provide possible clarifications. The questionnaire has been given to 10 SharePoint experts of Capgemini Netherlands.

Data Collection and Analysis: The case study has been performed within *Capgemini*, which is an international corporation primarily focused on providing IT ser-

vices, which is present in over 40 countries with more than 180,000 employees. At this point, concerning SharePoint solutions, the quality assurance process is in a research stage. Therefore, this project was of great interest to the company.

Used Tool—Code quality analysis for SharePoint is still in its infancy. An initial research on the state-of-practice on this topic, unveiled that there is only one tool that is in front of competition, namely SPCAF[1]. SharePoint Code Analysis Framework (SPCAF) is a commercial set of four tools, specialized for SharePoint applications, each one focused on different aspects of code quality in SharePoint. From these tools, since we are interested in identifying rule violations, we used only SPCop, which validates both the imperative and declarative code. The tool is offered with several predefined rule sets, each set consisting of a certain group of rules (e.g., the "All Rules" set includes all of the rules[2]). If a violation of one of the rules is encountered, the occurrence is added to the rule violations report. The rule violations are listed using the title of the rule that is violated and the location (file and line number), where the violation was discovered. For the purpose of our study, we used the predefined set of analysis rules (i.e., those that would be selected without any tool customization), to increase replicability of the study, and mitigate researcher bias (which would be raised if any selection was made by the authors). The rules that we used in our case study identify quality rule violations of the following categories: correctness (cor), supportability (sup), deployment (dep), security (sec), design (des), best practices (pra), memory (mem), naming (nam), localization (loc), and JavaScript (jsh).

Data Collection Instruments from Experts—The outcome of using certain constructs in SharePoint can be difficult to predict, because some aspects are hard to assess, unless they have been previously encountered. Therefore, we decided to collect knowledge from experts to fill in these uncertainties, and in addition to validate the information and conclusions. We elicit data from experts using one questionnaire, aiming at answering RQ_2. The *questionnaire* consisted of five questions, each inquiring about the severity of a certain SharePoint rule type, and an initial one aiming at understanding which part of SharePoint code is more defect prone.

[q1] SharePoint projects are built using two types of code. The first type being the imperative code, which is the C# code. The second type being the declarative code, which is the XML code. Which of these types of code do you consider more prone to produce defects (*Imperative code*, *Declarative code*, or *Equal*)?

[q2] The absence of referenced resources will most probably be followed by rule violations. A missing resource should be easily detectable. From this perspective, how severe do you consider this kind of rule violations ("*Not severe at all considering it is easily detectable*", "*Moderately severe, it may be easy to detect, but that does not make the rule less important*", or "*Highly severe, it is important such a rule violation is detected and solved as soon as possible*")?

---

1 http://www.spcaf.com

2 Rules are available in https://docs.spcaf.com/v6/ SPCAF_PAGE_QUALITY.html

[q3] On the contrary, having too many resources might also result in rule violations. In some cases SharePoint deploys prohibited assemblies, e.g. "ssocli.dll", or includes the same assembly in different configuration files. How severe do you consider this (*the options are the same as q2*)?

[q4] In the XML configuration files, a lot of required attributes have to be defined. Not filling in these required attributes might result in rule violations. How would you value assistance on this kind of violations ("*Very important, I consider these rules to be severe*", "*Moderately important, assistance would help me solve these rule violations faster*", or "*Do not need assistance, these rule violations are easy enough to solve on my own*")?

[q5] A possible aspect of security related rule violations is that they may not be as detectable as other violations, since security-related rules do not have to cause crashes, but result in unwanted behavior that is more difficult to discover. These rule violations might pose a big threat. How do you value security related rule violations that produce this unwanted behavior ("*Highly severe, any assistance in this field would be highly helpful since these rule violations are hard to detect*", "*Moderately severe, this kind of rule violations are uncommon*", or "*Not severe, these kind of rule violations are no problem at all*")?

[q6] The previous questions have covered the following kinds of rule violations: (a) Missing resources, (b) Having too many resources, (c) Missing attributes in XML, and (d) Security issues. Are there categories of rule violations that have not been covered by the previous questions? If so, what types of rules do you feel are not represented (*This is an open-ended question*)?

Data Analysis: The research questions have been answered by using descriptive statistics. In particular we have used frequency tables, and bar charts for all questions. For RQ_2, in case subjects provided some qualitative data, we tried to analyze and synthesize their answers using semantic analysis. Nevertheless, this process was very simple due to the low number of responses and the similarity of answers.

4 Results

In this section we present the results that have been obtained from data analysis, organized by research question. Implications to researchers and practitioners are provided in the discussions section (see Section 5). Figure 1 presents the number of violations identified in each project. From the figure, we can observe that the three studied projects are different, providing broad and representative data. Additionally, regardless of the maturity of the code-base, the tool was able, in all three cases, to provide valuable data on how to improve the source code quality.

Rule Violations Frequency: In Table 1, we present the most frequent rule violations in the three examined projects. In particular, in Table 1 we present the category, the criticality, the name, and the occurrence frequency of the most recurrent rule violations. We note that each rule has a single criticality, regardless of the context in which it is used.

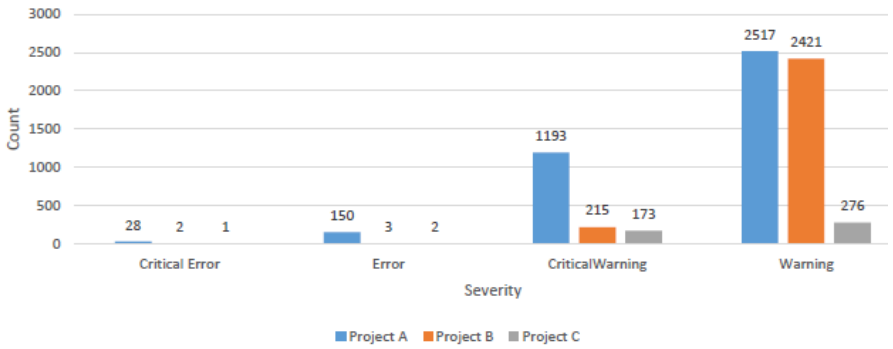

Fig. 1. Demographics on Rule Violations Criticality

From Table 1, two observations can be made: (a) *jsh* is the category, in which the most frequent rule violations can be identified, and (b) the top-10 most frequent rule violations are *Warnings or Critical Warnings*. The first column of the table shows that six out of the ten most recurring rule violations are found in the JavaScript (jsh) category. This means that *an important percentage of the identified rule violations is found in the JavaScript code*. However, this does not necessarily mean that violating these rules has a negative impact on the external behavior of the system, since some of these rules focus only on code conventions. Nevertheless, it still is interesting to take such an observation into account, when optimizing and improving code quality. Additionally, two out of the ten most frequent rule violations are critical warnings, while the remaining eight are warnings. Therefore, *there are no errors in the first positions of the rule violation frequency table*. By inspecting the complete list of rule violations (omitted from this manuscript due to space limitations), it becomes apparent that the first twenty-two rule violations are either critical warnings or warnings.

Table 1. Most Frequent Rule Violations.

Category	Criticality	Name	Frequency
loc	W	Use resources for localizable attributes	1642
jsh	W	Use curly braces around blocks	745
jsh	W	Use correct === and !==	700
jsh	W	Declare variable before it is used	430
jsh	W	Avoid trailing whitespaces	379
jsh	W	Do not exceed max length of a line in code	324
sec	CW	Avoid usage of “RunWithElevatedPrivileges”	196
jsh	W	Remove unused variables	189
nam	W	Files / Folders should contain the name of the parent solution	146
sec	CW	Avoid setting “AllowUnsafe Updates” on SPWeb	142

By inspecting the top-10 most encountered errors (see Table 2), one can highlight the following:

- **The number of occurrences of the ten most occurring critical errors and errors is relatively low**, especially when compared to the amount of occurrences of the ten most frequently occurring rule violations. However, each of these errors has a higher potential to cause system crashes or unexpected behavior. Therefore, the information that is provided by these reports can be considered valuable, since they directly provide information on aspects that require immediate attention.
- The other aspect that is different between the ten most frequently occurring rule violations and the ten most recurring errors are the categories of the violations. The list of top-10 most frequently occurring rule violations was dominated by the JavaScript (front-end) category, whereas **the top-10 most recurring errors is identified in the SharePoint backbone categories** (i.e., correctness and deployment). A possible explanation for this is that rules of the JavaScript category rule violations cannot have a large negative influence on the system, but rather result in user interface problems, while SharePoint backbone categories, can potentially have a huge impact on the ability of the system to function as intended.

Table 2. Most Frequent Errors caused by Rule Violations

Category	Criticality	Name	Frequency
cor	E	Define attribute 'ID' in FieldRef in correct casing	50
dep	E	Do not deploy assembly multiple times	28
cor	E	Declare required attributes in schema of ListTemplate	27
dep	E	Do not deploy assembly with DEBUG mode	24
cor	CE	Define unique value for 'Id' in CustomAction	22
dep	E	Do not deploy TemplateFile multiple times	10
sup	CE	Do not access SharePoint API via reflection	6
sup	E	Do not read ConnectionString from SPContent-Database	6
cor	E	Declare required attribute in CustomAction	4
cor	E	Declare required attributes in SiteDefinition	4

Severity of Rule Violations: In this section we discuss the level of severity of rule violations, organized by the six questions included in our questionnaire. The first question was designed to determine what type of code, (i.e., imperative or declarative), *was considered most prone to result in defects* (related to Correctness). The results obtained based on experts opinion suggest that 10% of the participants considers imperative code and declarative code equally prone to defects. The remaining 90% chose the imperative or the declarative code options nearly the same amount of times, consisting an indecisive difference. Therefore, both types of code

are considered equally prone to result in defects. In addition to the quantitative results, we have encountered some qualitative results as well: One developer stated that coding in C#, and XML was mostly used to facilitate communication between websites. A second developer pointed to code written with XSL, which is a form of declarative language, as the most defect prone parts of the code. The third comment stated that XML is more sensitive to syntax related mistakes, and that these small mistakes may have big consequences. However, these are supposed to be easier to fix, leaving choice on the multiple choice part to the imperative option. Finally, the fourth comment stated that XML is more error prone, but the impact on the systems' defects and performance is significantly less.

The second question was designed to ***determine the severity of leaving out referenced resources***, the emphasis being on the influence, since this rule violation should be relatively easy to detect (related to Deployment). Only 20% of the participants considered the potential to result in defects to be highly severe. On top of that, 60% considered the potential to result in defects to be moderately severe. Overall, this means that a total of 80% considered the potential for defects at least moderately severe. This is a good indication that this aspect of SharePoint has to be monitored when analyzing the code quality. The third question was designed to ***determine the negative influence of adding too many assemblies***, i.e. prohibited assemblies or including the same assembly twice (related to Deployment). Out of the 80% of the participants that chose one of the multiple choice answers, only 10% considered the defect not severe. Therefore, it was concluded that this kind of rule violation in SharePoint will be considered moderately severe meaning this kind of rule violations will be monitored when analyzing the code quality. After initial research into the SPCAF tool, it soon became clear that it offered good detection and assistance on ***missing required attributes in the XML configuration files***. The fourth question was designed to explore ***the severity of this kind of defects***, to determine its impact on the number of times the system present unexpected behavior, and to ascertain the importance of this rule. 40% of the participants considered the assistance very important, since they considered potential defects that can result from this rule violation. 30% considered the assistance moderately important, mainly because it allowed them to solve the problematic code faster. Only 20% of the participants stated that they did not find the assistance valuable. In conclusion, the severity of this type of rules will be regarded as highly severe since 70% considered it at least moderately important and 40% considered it as very important.

Even though Security related vulnerabilities might not have a huge impact on the behavior of the software, possible rule violations may have an even bigger impact on the system. Software carrying security related vulnerabilities may appear to function as intended, but would malfunction in the security area, e.g., it may provide entrance to users to parts that should not be accessible. The fifth question is designed to ***determine how valuable the experts consider detection and assistance in the security area***. The results suggested that 70% of the participants considered the potential of rule violations to result in defects, highly severe, and that assistance on this type of defects is highly appreciated. The remaining 30% of the participants consider the rule violations moderately severe (since they are uncommon). Finally,

the sixth question aimed to ***provide the experts with the ability to name types of rules that were not discussed*** in the first five questions. This way, the types of rules that were not yet represented, could still be brought forward. The additional types of vulnerabilities were: (a) Memory violations, e.g. disposing all sorts of instances or memory leaks; (b) Performance related violations, e.g. endless loops or other inefficient code; (c) Common coding mistakes, e.g. wrong syntax or improper use of variables; (d) Not using unique identification of components; and (e) Inconsistency of developed code. This new insight posed a valuable addition to the types of rules that were already considered in the code quality analysis.

5 Discussion

The results of this study can be considered as a starting point for code quality analysis in SharePoint applications, which is a rather understudied research field. The obtained results can be useful to both researchers and practitioners:

- (researchers) The relation of some SPCAF rules to defects remains uncertain. These rules require further investigation.
- (researchers) This research effort was exploratory since it was based on expert opinion and descriptive statistics. More explanatory research is required.
- (researchers) Evolution analysis with data analytics can be performed, to confirm the relationship between the existence of defects and specific rule violations.
- (practitioners) The majority (approx. 67%) of the predefined rules offered by SPCAF is associated to defects. Therefore SPCAF can consist a good starting point for tool-assisted code reviews.
- (practitioners) Most rule violations are related to the client-side of the application, but these rules are not that sever. Correctness, deployment, and security rule violations should be prioritized.

6 Threats to Validity

In this section we present potential threats to validity for our study following the guidelines proposed by Runeson et al. [9]. According to Runeson et al., there are four types of threats to validity: construct, reliability, external and internal validity threats. In this study internal validity will not be considered, since causal relations are not in the scope of this study. Concerning *construct validity*, we have identified one possible threat, i.e., the fact that we assessed the quality of the code, based on the suggestions of a single tool. Although this threat is important, a discussion with practitioners suggested that 2 out of 3 rule violations, identified by the tool, are considered vital by practitioners. In addition to that, no other tools for SharePoint applications quality assurance exist, to the best of our knowledge. To mitigate threats to *reliability*, we presented in detail the case study design, and we have not parameterized the used tools, to ensure that our results are reproducible and comparable to future replications. Concerning *external validity*, we need to underline that

the obtained results cannot be generalized to all SharePoint projects and that the use of a different tool for code reviews, might have led to different results. Nevertheless, the diversity of the examined projects ensures some heterogeneity in the cases.

7 Conclusions

This study aimed at exploring the quality assessment processes in SharePoint application development, through tool-assisted code reviews. The results of the study suggested that the majority of the rule violations can potentially lead to defects, and that they exist in all stages of software, regardless of their positioning in the software development lifecycle. As expected, the number of critical errors and errors are eliminated in production ready software. In addition, the most frequently occurring rule violations are warnings that exist in the JavaScript part of the applications, whereas the more severe errors (i.e., correctness, deployment, and security) are more probable to appear in the imperative parts of SharePoint applications.

References

[1] Ampatzoglou A., Ampatzoglou A., Chatzigeorgiou A., and Avgeriou P., "The Financial Aspect of Managing Technical Debt: A Systematic Literature Review", Information and Software Technology, Elsevier, 64 (8), pp. 52-73, August 2015.

[2] Charalampidou S., Ampatzoglou A., and Avgeriou P., "Size and cohesion metrics as indicators of the long method bad smell: An empirical study", 11th International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE '15), ACM, Beijing, China, October 2015.

[3] Feitosa D., Ampatzoglou A., Avgeriou A., Nakagawa E. Y., "Investigating Quality Trade-offs in Open Source Critical Embedded Systems", 11th International Conference on the Quality of Software Architectures (QoSA' 15), ACM, Canada, May 2015.

[4] Hovemeyer, D. and Pugh, W. "Finding bugs is easy", SIGPLAN Notices, ACM, 39 (12), pp. 92–106, 2004.

[5] Kitchenham B. and Pfleeger S. L. "Principles of Survey Research Part 2: Designing a survey", Special Interest Group on Software, ACM, 27 (1), pp. 18-20, January 2002.

[6] McConnell S. C., "Code Complete: A Practical Handbook of Software Construction", Microsoft Press, 2004.

[7] Meyer B., "Design and code reviews in the age of the internet", Communications, ACM, 51 (9), pp. 66-71, September 2008.

[8] Misra, S.C. and Bhavsar, V.C., "Relationships Between Selected Software Measures and Latent Bug-Density: Guidelines for Improving Quality", 1st International Conference on Computational Science and Its Applications (ICCSA' 2003), 2003.

[9] Runeson P., Höst M., Rainer A., and Regnell B., "Case Study Research in Software Engineering: Guidelines and Examples", John Wiley and Sons, Inc., 2012.

[10] Vokac M., "Defect frequency and design patterns: an empirical study of industrial code", Transactions on Software Engineering, IEEE, 30 (12), pp. 904–917, 2004.

[11] Van Vliet H., "Software Engineering: Principles and Practice", Wiley & Sons, 2008.

[12] Zaman S., Adams B., Hassan A. E., "Security versus performance bugs", 8th Working Conference on Mining Software Repositories (MSR' 11), pp. 93–102, 2011.