Bug Mining The Canadian Government's Websites: An Empirical Study

Sasha Vujisic
CMPT 479-D100
Simon Fraser University
sva39@sfu.ca

***Abstract*—This study investigates software bugs and performance issues on Canadian government websites across municipal, provincial, and federal levels. The research focuses on identifying and categorizing JavaScript and DOM-related bugs, assessing their severity, and evaluating differences in bugginess among government services and levels. Using a systematic classification framework informed by existing literature, bugs were categorized by type (e.g., permission denied, null exceptions) and severity (high, medium, low), with new categories introduced as needed. Data collection tools such as Screaming Frog SEO Spider, ChromeDevTools, and Chrome Lighthouse facilitated the comprehensive evaluation of fifteen representative government websites. The findings highlight variations in website quality and performance, identify recurring issues, and provide insights into public bug reporting and resolution procedures. This research contributes to improving government website quality, enhancing user experience, and informing best practices for bug detection and management.**

Keywords—Bug classification, Canadian government websites, software quality, JavaScript bugs, DOM issues, web performance analysis, Chrome Lighthouse, Screaming Frog, government services evaluation, user experience improvement.

I. INTRODUCTION

Since its inception, the web development field has been constantly growing and evolving, spearheaded by human ingenuity and effort. Yet web development remains ever prone to human error, manifesting itself in the form of software bugs. To better understand what mistakes software engineers make at large, empirical studies that systemically gather and analyze information about bugs from websites can be conducted- this is called bug mining. From a statistically representative sample of websites, these studies draw conclusions that should be applicable to the World Wide Web. The web itself is full of commercial or private websites, which rely user traffic to survive. Bugs can make a website uncompetitive or undesirable to use, resulting in revenue loss and potential bankruptcy. What this means for most sampled websites is that they are under "market pressure" to minimize bugs in their products- lest they concede users to a competitor. However, not all website are subject to this "market pressure." Governments around the world, like Canada's, are making the transition into the digital realm, and this means digitizing many of their services- often on the web. Conveniently, these websites enjoy the luxury of unconstrained funding (taxation) and fully dependent users (citizens.) These factors create a situation where government websites will always be up and in demand despite their quality or performance, meaning that government websites can afford to be buggy in ways that other sites can't. Therefore, bug mining studies on government websites may lead to findings that are not usually observed in a typical sampling of (commercial) websites. This study aims to explore that gap by bug mining the government of Canada's (GC) websites at the municipal, provincial, and federal level. More specifically, this study will count, categorize, and measure the severity of JS Runtime Exceptions, DOM-related issues, and security problems found across GC websites, whilst also assessing their overall performance. This research will inject more much needed data into the public sector side of bug mining research. Furthermore, this research will impact anyone who utilizes Canadian government e-services, as these findings may assist GC developers in improving the user experience.

II. RESEARCH QUESTIONS

To accomplish the aims of this study, six research questions were proposed to guide the research. The first three questions strictly pertained to bug mining, being **(1) What type of JavaScript bugs were found on GC websites, and how many? (2) What type of DOM-related issues were found on GC websites, and how many?** and **(3) How severe were the discovered JS bugs & DOM-related issues on GC websites?** Next, we tried to assess the impact of buggy behavior that the discovered bugs caused by asking **(4) What was the overall quality of GC websites?** Finally, we attempted to contextualize our data in terms of how it relates to GC, and in turn, how it impacts people by asking **(5) How did bugginess vary among the levels & services of GC?**

III. RELATED WORKS

A. Defining and Categorizing Bugs by Type

One of the challenges of bug mining was systemically defining categories for bug types and severity levels. [Ocariza, F. S. Jr., Pattabiraman, K., & Zorn, B., 2023] provided an empirical categorization of JS runtime exception types that

existed in popular websites. Their study found that JS errors tended to fall into four well-defined categories, being "Permission Denied", "Undefined Symbol", "Syntax Error", and "Null Exception." Their study also found that production level websites, on average, had four errors per webpage. Herein lies the issue of sampling bias that our study aimed to address, as their sampled sites were all commercial, and so their findings might not have been representative of public sector websites. Nonetheless, their categorizations served as a good starting point for our study's own categorization process.

B. Defining Categories of Severity for Bugs

[Zhou, B., Neamtiu, I., & Gupta, R., 2014] examined the differences among bugs that were categorized by their severity class and bug type. They defined three levels of bug severity (low, medium, high) as well as the associated criteria (improvements, logic issues, crashes, etc.) The severity thresholds outlined in this paper provided adequate representation for most bugs discovered by our study. However, [Kashiwa, Y., Yoshiyuki, H., Kukita, Y., & Ohira, M., 2014] added an important layer of nuance to our severity scale, providing six categoric definitions of high-impact bugs, which are performance, dormant, surprise, blocking, security, and breakage to better understand what constituted a high impact bug. Still, the discussed bug severity research suffered from the sampling bias problem already previously mentioned in the earlier related works. Regardless, our study can still apply the categories that these works have established.

C. GC Web Development and Bug Reporting Practices

It is important to know how GC was trying to improve user experience. The [Canadian Digital Service, 2024] progress report outlined GC's steps toward this goal to show users that Canadian tax dollars were being spent well. [Aamir, T., Chhetri, M. B., Chamikara, M. A. P., & Grobler, M., 2023] demonstrated a high correlation between a government's digital transformation strategy and user feedback. [Obeidat, I., Alhayek, E., & Obeidat, A., 2024] discussed how security-related bug bounty infrastructure is lacking in government, which is true for GC as our study found out. [Eibl, G., & Thurnay, L., 2023] covers the pros and cons of government releasing open-source software on Github, something that GC has done. In terms of improving the user experience, the discussed works in this section shed insight into where and what the GC tried, but not *how much.* That is a question that our empirical study aimed to answer.

IV. METHODOLOGY

The methodology behind this study is built on a foundation of existing works, toolsets, and careful manual input. The first step of data collection is to sample a series of common-themed websites across GC's levels of government. The second step is to create bug type & bug severity categories for JS and DOM related issues by borrowing from and expanding upon existing works. The next step is

A. Website Sampling

Fifteen GC websites were selected for this study. As Canada has three levels of government (federal, provincial, and municipal), each tier was allotted five websites each. British Columbia and Burnaby were selected to represent the provincial and municipal levels of government. The next step was to select five websites from each level of government. To keep things consistent, the same type of website should exist across all three levels of government. There should also be as many types of websites represented as possible to capture the scope of GC. With this criteria in mind, five categories of websites were made, outlined below:

Info: The homepages of GC. They provided general information, usually on statically loaded pages that offered minimal user interaction.

Jobs: The job banks of GC that hosted job postings. Users could search or apply for jobs by querying a database, making Jobs more complex than Info.

Data: The online archives of GC, these sites were giant searchable data repositories. They had more advanced search queries than Jobs and surely presented a challenge with data warehousing.

Services: The service portals of GC. They offered a wide variety of services to users, some of which required user authentication. These sites were built around user interaction.

Unique: This was the quasi-random control group of websites that have no functional or thematic similarities to one another.

Links to all fifteen websites can be found on the data spreadsheet attached at the end of this paper.

B. JavaScript Bug Classification

As already defined by existing works, the initial four types of JavaScript exceptions that were examined in this study were be **Null Exceptions, Undefined Symbols, Permission Denied, and Syntax Errors.** The bug type could be determined by looking at the ErrorType in the web console. If a new bug type was encountered, a new category would've been created for it, so long as it did not overlap with pre-existing categories.

Chrome DevTools was primarily used to detect JS runtime bugs. Screaming Frog SEO Spider was used to crawl each website. All URLs flagged by the crawler to have detected issue were manually inspected for JS bugs using Chrome DevTools. The same URL was then inspected with Firefox DevTools to see if the same issues would have manifested. Furthermore, websites whose functionality was heavily dependent on user-input interaction were manually inspected.

Counting bugs was not straightforward nor simple. While most bugs were standalone, sometimes a very specific bug would recur on every related webpage. An example of this could be a faulty module that was used in multiple pages- so either the bug is counted once in the module or multiple times across the pages. In this study, we differentiated standalone JS bugs from

recurring JS bugs. The number of standalone and recurring JS of each bug type for each website was tracked.

C. DOM-Related Issue Classification

A DOM-related issue is a runtime bug that has to do with an error or bad coding practice related to DOM-JS interactions (e.g., such as getDocumentId(badId) returning null.) The Screaming Frog SEO Spider automatically detected DOM-related issues and then categorized them by type on its own. DOM-related bugs were tracked for each website, and no distinction was made between standalone DOM-related issues and recurring ones, unlike JS bugs.

Screaming Frog SEO Spider is a powerful WebCrawler that can find broken links, errors and redirects, analyze page details and metadata, review meta robots and directives, audit hreflag attributes, discover exact duplicates, generate XML sitemaps, and site visualizations. This crawler's main limiting feature is its web crawling limit of 500 URLs per website, which limits potential coverage.

D. Severity Classification

Classifying bugs by type would not have been an informative enough metric on its own to determine a website's bugginess. To evaluate bug impact, we then classified bug types by severity- low, medium, and high.

Low severity bugs were issues that would be considered nuisances or improvement requisitions. These were usually just poor coding practices with little to no consequence. Warnings mostly fall into this category.

Medium severity bugs were issues that did impact the website, such as a fault or flaw in application logic, an occasional crash, or something not loading properly.

High severity bugs were showstoppers- crashes, memory leaks, vulnerabilities, etc. These bugs either derailed the user experience or compromised the website's functionality. High impact bugs were further diagnosed as either dormant, surprise, security, performance, blocking, or breakage. Revisit [Kashiwa, 2014] for details on this specifically and Related Works Section B for the severity classification literature that we derived our classifiers from.

E. Overall Site Quality

To empirically and objectively measure the quality of a site, we used the tool Chrome Lighthouse to automatically analyze each website, and then algorithmically generate a score out of 100 for Performance, Accessibility, Best Practices and SEO. Chrome Lighthouse also flagged issues that contributed to lower scores, which could be matched to pre-existing bug categories. This heuristic evaluation was essential to correlating bug presence with bug impact and website comparison.

V. RESULTS

Table I. JS Runtime Exceptions

Type	Null Error	Undefined Symbol	Script Error	Uncaught Promise	Missing Dependency
Count	2, 1*	4, 1*	2**	1***	1
Severity	Medium	Medium	Medium	High, Blocking	Medium

* BC Data, this error occurs every time a study document is accessed
** Burnaby Data, this error occurs on each eco-sculpture page (6 of them)
*** Burnaby Data, this error occurs every time a dataset is accessed

Table II. JS Warnings

Type	Preload Misuse	Deprecation	Cookies
Count	102	25*	1000+**
Severity	Low	Low	N/A

* 2 Deprecations are jQuery
** 3rd party cookies were slated for deprecation, but the decision was reversed. Due to this development, cookie warnings were not formally counted/tracked.

For tables III, IV, ***Count*** refers to the number of affected URLs and ***Prevalence*** refers to the Count as a % of total URLS (4150.)

Table III. DOM-related Issues

Bug type	**Count**	**Prevalence**	**Severity**
H1 Missing	101	2.43%	Medium
Missing Alt Text	331	7.98%	Low
Missing Alt Attribute	117	2.81%	Low
Content: Exact Duplicates	99	2.38%	High, Performance
HTTP URL	2	0.05%	High, Security
Missing <head> tag	6	0.14%	High, Performance
Missing Page titles	4	0.10%	High, Performance
Internal Client Error 4XX	62	1.49%	High, Breakage
Multiple <head> tags	24	0.58%	High, Performance
Multiple <body> tags	24	0.58%	High, Performance
Page Titles outside <head>	6	0.14%	High, Performance
Multiple Page Titles	6	0.14%	High, Performance
Multiple Meta Descriptions	4	0.10%	Medium
Hreflag: No index return links	2	0.05%	High, Blocking

Table IV. Security Warnings

Type	**Count**	**Prevalence**	**Severity**
HTTP URL	2	0.05%	High, Security
Missing HSTS Header	1120	26.99%	Low
Missing X-Frame Options Header	1420	34.21%	Low
Missing Secure Referrer Policy Header	1840	44.34%	Low
Unsafe Cross-Link Origins	341	8.21%	Low
Protocol Relative Resource Links	636	15.33%	Low
Missing Content Security Policy Header	1730	41.69%	Low

Table V (below). Chrome Lighthouse Scores for Burnaby, BC, & Canada. The scores are ordered {Burnaby, BC, Canada} in each cell.

Website	Performance	Accessibility	Best Practices	SEO
Info	78, 93, 95	79, 89, 100	74, 96, 59	85, 92, 100
Jobs	88, 73, 87	85, 100, 93	78, 78, 78	82, 83, 83
Data	23, 59, 90	90, 93, 91	78, 100, 78	100, 83, 92
Services	88, 91, 85	61, 100, 100	70, 100, 56	64, 100, 92
Property	97, 73, 94	90, 87, 92	78, 78, 78	91, 100, 100
Average	74.8, 77.8, 90.2	81, 93.8, 95.2	75.6, 90.4, 69.8	84.4, 91.6, 93.4

Table VI. Bugs Per URL

Bugs	Burnaby (1013)	BC (1375)	Canada (1999)
JS Exceptions	0.015	0.003	0.002
JS Warnings	0.015	0.002	0.053
DOM Issues	5.938	4.579	4.159

Table VII. Chrome Lighthouse Averages

Level of GC	Performance	Accessibility	Best Practices	SEO
Burnaby	74.8	81	75.6	84.4
BC	77.8	93.8	90.4	91.6
Canada	90.2	95.2	69.8	93.4

Table VIII. Average Lighthouse Scores for each GC service

	Info	Jobs	Data	Services	Unique
Performance	85.3	82.6	57.3	88	88
Accessibility	89.3	92.6	91.3	87	89.6
Best Practices	76.3	78	85.3	75.3	78
SEO	92.3	82.6	91.6	85.3	97

Table IX . Bugs Per URL (BPU) for each GC service

	Info (980)	Jobs (1009)	Data (588)	Services (843)	Unique (962)
JS Exceptions	0.011	0.002	0.009	0.002	0
JS Warnings	0.013	0.003	0.002	0.002	0.110
DOM Issues	4.781	7.119	2.476	7.7870	3.358

VI. DISCUSSION

The results from the data collection raised a number of interesting points. Tables I & II showed that runtime JS exceptions and warnings were rare overall, but the majority of bugs found were Null or Undefined Symbol. This should answer RQ1. All of the recurring JS bugs required user-driven queries to trigger. Another interesting finding was that JS exceptions and warnings that were found on one webpage tended to exist on all related webpages. Lastly, although preload warnings outnumbered deprecation warnings, the latter's root causes were more diverse whereas the former came from one site repeating the same bug on its webpages.

Table III showed that the bulk of DOM related issues had to do with missing alternate information or images, which negatively impacted accessibility. Additionally, although DOM issues were relatively infrequent overall, some were highly prevalent on specific websites. This should answer RQ2. One major roadblock encountered in our data collection was that the BC General Info website was not scrapeable despite having no apparent web-crawl blockers. We also found that webpages with DOM issues were not a useful predicator for finding webpages with JS exceptions, as the recurring JS exceptions were only uncovered by manual inspection. This is not surprising given Screaming Frog's URL crawling limits and the fact that some errors required user-input queries, which web crawlers aren't capable of. Lastly, as shown in Table IV, it was evident that many pages did not use available security protocols to safeguard themselves. This is a questionable design choice given that these pages are not performance-minded, meaning the additional overhead would be of little consequence.

Regarding severity, for JS, most bugs were medium severity. One bug was high severity, as it was an uncaught promise exception that resulted in visual data not being returned to a UI. For DOM-related issues, many types of detected issues were high severity, but the majority of the counted bugs overall were low severity missing alternate attributes. Most of the DOM-related high impact bugs were performance related in terms of severely impacting SEO. The breakage bugs were attributed to broken links and the security bugs were due to HTTP URLs. This should answer RQ3.

Table V showed the scores for Burnaby, BC, and Canada for each website type. Table VI showed the proportionality of JS Exceptions, Warnings, and DOM-issues for each level of government. Canada outperformed BC and Burnaby for JS Exceptions and DOM Issues, whereas Burnaby performed the worst for all three categories. Table VII broke down the average category score for Burnaby, BC, and Canada. Canada outperformed BC and Burnaby in three out of four categories, whereas Burnaby was the worst in three out of four categories. Both tables suggested that overall, Canada-level websites had the lowest BPU scores and the highest Lighthouse scores, whereas Burnaby was the opposite, and BC was somewhere inbetween. This should answer RQ4. In general, lower BPU statistics seemed to correlate with higher Lighthouse scores. Table VIII shows how website services compare in quality by taking the average score for a website type. After ranking the entries in each row by score (highest first, lowest last), we found that Unique ranked first overall, Services third, and the rest tied for second place. Table XI measured BPU per service. Ranking the scores in the same way as Table VIII showed the following results: 1) Data, 2) Unique, 3) Jobs, Services, and 4) Info. Comparing the rankings from Table VIII and IX yielded an overall site quality ranking that had Unique sites as being the best, followed by Info, Jobs, Data, and lastly Services. This should answer RQ5.

VII. CONCLUSION

After having discussed our findings, we found that JS exceptions were rare occurances. DOM-related issues were comparatively more frequent. JS exceptions tended to be medium severity in contrast to DOM-related issues which were often high severity. Many security warnings were discovered that indicate that security best practices were not being followed. Our analysis indicates that federal that Canada websites are the least buggy, followed by BC and then Burnaby. Lastly, our analysis also showed that unique/level-specific websites performed the best, followed by Info, Jobs, Data, and Service websites.

VIII. LIMITATIONS & FUTURE WORK

The two main limitations of this study were its limited sample size of fifteen GC websites and its limited coverage of those sampled websites. However, the study still collected enough data to answer the RQs with decent accuracy. Moreover, the study's limited sample size was made up for the fact that its scope targeted the SFU locality, making it relevant to the institution and community. Wider and more comprehensive studies are needed to see if the trends observed here hold true.

IX. References

Ocariza, F. S. Jr., Pattabiraman, K., & Zorn, B. (2023). JavaScript errors in the wild: An empirical study. Proceedings of the 2023 IEEE/ACM International Conference on Automated Software Engineering (ASE), 1–8. IEEE. https://doi.org/10.1109/ASE56229.2023.00190

Zhou, B., Neamtiu, I., & Gupta, R. (2014). Experience report: How do bug characteristics differ across severity classes: A multi-platform study. Proceedings of the 2014 IEEE International Conference on Software Maintenance and Evolution (ICSME), 1–8. IEEE. https://doi.org/10.1109/ICSME.2014.89

Kashiwa, Y., Yoshiyuki, H., Kukita, Y., & Ohira, M. (2014). A pilot study of diversity in high impact bugs. Proceedings of the 2014 IEEE International Conference on Software Maintenance and Evolution (ICSME), 1–8. IEEE. https://doi.org/10.1109/ICSME.2014.89

Canadian Digital Service. (2024). Aligning our efforts to serve people better: Canadian Digital Service annual report (Fiscal Year 2023-2024). Canadian Digital Service. https://digital.canada.ca (ISSN: 2818-4122, CAT: SG2-11E-PDF)

Aamir, T., Chhetri, M. B., Chamikara, M. A. P., & Grobler, M. (2023). Government mobile apps: Analysing citizen feedback via app reviews. Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering (ASE), 1–8. IEEE. https://doi.org/10.1109/ASE56229.2023.00190

Obeidat, I., Alhayek, E., & Obeidat, A. (2024). A model for adaptive bug bounty programs and responsible disclosure in e-government vulnerability management. Proceedings of the 2024 International Conference on Multimedia Computing, Networking and Applications (MCNA), 1–7. IEEE. https://doi.org/10.1109/MCNA63144.2024.10703931

Eibl, G., & Thurnay, L. (2023). The promises and perils of open source software release and usage by government—Evidence from GitHub and literature. In 24th Annual International Conference on Digital Government Research—Together in the unstable world: Digital government and solidarity (pp. 1-11). ACM. https://doi.org/10.1145/3598469.3598489

X. Project Links

Project Github
https://github.com/sashavujisic/bug_mining_gc/tree/main

Project Data Spreadsheet (Primary)
https://docs.google.com/spreadsheets/d/1P4IHbUTfmlnMEB_Kk_wWfpw0k5zaCYJnoJwy-0bu4Ow/edit?usp=sharing

Project Data Document (Secondary)
https://docs.google.com/document/d/1gRpIVtfKTpCnl1RSeIHTgC7B4yZY0SorP1noQR2yLMs/edit?usp=sharing