Studying the Relationship between Exception Handling Anti-Patterns and Defects

*Concordia University*

**Department of Engineering and Computer Science**

Hina Masood Ahmed Anusha K.T Bethi Kiranmayie Shruthi K. V

40076287 40102962 40092284 40091427

***Abstract*— In this paper, we examine the relationship between software quality and exception flow characteristics and over catch, destructive wrapping, Log and return null exception handling anti-patterns**. **Software quality reflects how well it consents to or adjusts to a given design, based on functional requirements or specifications. It is the degree to which the**[**correct**](https://en.wikipedia.org/wiki/Software_verification_and_validation#Software_validation)**software was produced. Exception handling is the method of reacting to the event, during computation, of special cases – anomalous or exceptional conditions requiring special processing – regularly disrupting the ordinary flow of program execution. Exception handling aim to assist in the practice of software comprehension and maintenance. Post-release defects, Traditional product metrics, Process metrics, change metrics, Pre-release quality metrics, Exception handling metrics are considered in our case study and we have considered over catch, destructive wrapping, Log and return null antipatterns. We performed our case study on Elasticsearch which is an open source java project. We study whether exception flow characteristics and over catch, destructive wrapping, Log and return null anti-patterns have a statistically significant relationship with post-release defects. We built a model “Logistic Regression Model” and hereby find that exception handling anti patterns in Elasticsearch project have a significant relationship with the defects.**

***Keywords—anti-pattern, exception handling, over catch, destructive wrapping, Log and return null, post-release defects, pre-release defects, product metrics, process metrics, change metrics, quality metrics***

1. **INTRODUCTION**

Exception is the method of reacting to the event, during computation, of special cases – anomalous or exceptional conditions requiring special processing – regularly disrupting the ordinary flow of program execution. It is given by specialized programming language develops, computer hardware components like interrupts or operating framework IPC facilities like signals. In common, an exception breaks the typical stream of execution and executes a pre-registered exception handler. The details of how this can be done depends on whether it may be a hardware or software exception and how the program exception is executed. A few special cases, particularly hardware ones, may be handled so smoothly that execution can continue where it was hindered. An antipattern draws on real-world experience to distinguish a commonly happening programming mistake. It portrays the general form of the awful design, distinguishes its negative results, prescribes a remedy, and helps define a common vocabulary by giving each design a title. Software quality is characterized as a field of study and practice that describes the desirable properties of software products. There are two primary approaches to software quality: defect management and quality attributes. Software quality metrics are a subset of software metrics that center on the quality viewpoints of the product, process, and project. These are more closely related with process and product metrics than with project measurements. Software quality metrics can be further divided into three categories − Product quality metrics, In-process quality metrics, Maintenance quality metrics. We have considered prerelease defects and post release defects in our case study.

In this paper, we discuss three common exception handling antipatterns which are over catch, destructive wrapping, and Log and return null. We examine the relationship between software quality and exception flow characteristics and over catch, destructive wrapping, Log and return null exception handling anti-patterns. We construct an eclipse plugin in java to automatically distinguish these anti-patterns in java open source projects by syntactically examining the code. To test our plugin, we utilize the Elasticsearch java project. It is a search engine based on the Lucene library. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents. The main reason behind selecting this project is that it has a substantial amount of exception handling statements and prerelease defects, post release defects. We generated Exception handling anti-pattern metrics using this plugin and results were stored in CSV file. All other metrics were generated using Scitools understand software. This is a customizable integrated development environment (IDE) that enables static code analysis through an array of visuals, documentation, and metric tools.

In this paper we perform an empirical study of the relationship between exception handling practices and post-release defects using the results obtained from plugin and Understand tool. We are referring the research paper 1 for our study and answers on below questions from research paper 1.

We find that, in ElasticSearch, we do not observe any statistically significant relationship between exception flow characteristics and post-release defects. The majority of the anti-patterns do not have a statistically significant relationship with post-release defects except for very few of them.

**RQ1: Do exception handling flow characteristics contribute to a better explanation of the probability of post-release defects?**

**RQ2: Do exception handling anti-patterns contribute to a better explanation of the probability of post-release defects?**

The rest of the paper is organized as follows: Section II the three types of anti-patterns studied as part of this paper. Section III presents our eclipse plugin, the java code implementation to detect the anti-patterns. Finally, Section IV is the conclusion of the paper and talks about potential future investigate bearings based on our early inquiring about discoveries.

1. **ANTI-PATTERNS**
2. *Log and Return Null*

Rather than returning null, throw the exception, and let the caller deal with it. We should only return null in a non-exceptional use case. For example, this strategy returns null if the search string was not found.

Example:

**catch (NoSuchMethodException e) { LOG.error("Blah", e); return null; }**

1. *Destructive Wrapping*

This destroys the stack trace of the original exception.

Example:

**catch (NoSuchMethodException e)**

**{**

**throw new MyServiceException("Blah: " + e.getMessage());**

**}**

1. *Over Catch*

Where a higher-level exception is used to catch multiple different lower-level exceptions. We may not know which specific exception is caused when it is caught by a higher-level exception. Over-catch is one of the trivial mistakes which causes catastrophic failures when the error handler over-catches exceptions and aborts. Possible types of over-catch exceptions are as follows:

An Exception which is checked by the compiler for example IOException, FileNotFoundException are caught by the generic higher-level exception. When runtime exceptions are thrown, and the exceptions are caught by generic higher-level exceptions.

1. *GetCause Handler*

The problem with relying on the result of getCause is that it makes your code fragile. Now calling getCause may return aa wrapping exception, really is wanted is the reason of of getCause() Instead, we should unwrap the causes the problem of the exception is found

Example:

**catch (MyException e) { if (e.getCause() instanceof FooException) { …**

*E.* *Throws Kitchen Sink*

Throwing multiple checked exceptions from your method is fine, as long as there are different possible courses of action that the caller may want to take, depending on which exception was thrown.

Example:

**public void foo() throws MyException, AnotherException, SomeOtherException, YetAnotherException {**

*F. Log and throw*

Either log the exception, or throw it, but never do both. Logging and throwing results in multiple log messages for a single problem in the code, and makes the exception more less mitigated.

Example:

**catch (NoSuchMethodException e) { LOG.error("Blah", e); throw e; }**

or

**catch (NoSuchMethodException e) { LOG.error("Blah", e); throw new MyServiceException("Blah", e); }**

*G. Ignoring Interrupted exception*

The InterruptedException is thrown when a thread is waiting or sleeping and another thread interrupts it using the interrupt method in class Thread. So, if you catch this exception, it means that the thread has been interrupted

Example:

**while (true) { try { Thread.sleep(100000); } catch (InterruptedException e) {} doSomethingCool(); }**

1. **CASE STUDY**
2. *Subject System*

We have considered Elasticsearch 2.4.0 open source Java project. It has many version releases before 2.4.0 and after that. The release notes were also easily available online, hence it was easy for us to track the record of prerelease and post release bugs. We also found by manual inspection, there were many instances of exception handling in Elasticsearch 2.4.0. We used the anti-pattern detecting plug-in, which was designed earlier during one of our assignments, to detect the exception flows and the exception handling anti pattern. Table 1 is with the details we found during our study on our subject project.

|  |  |
| --- | --- |
| Project Characteristic | Elasticsearch |
| Language | Java |
| Purpose | search and analytics solution |
| Release Version (tag name) | 2.4.0 |
| Latest Post Release Version | 5.0.0 |
| Files | 13245 |
| SLOC (K) | 1666 |
| Pre-Release Defects | 47 |
| Post Release Defects | ~500 |
| Files with Post Release Defects | 219 |
| Catch | 1476 |
| Files with Catch | 484 |

1. *Research questions*

The purpose of this paper is to know that is there any relation between exception handling practices and post-release defects. From the research paper1 we have two research questions to obtain the relation between exception handling practices and post-release defects. Based on the previous studies from the research paper1, they exposed the suboptimal exception handling practices in two ways, one by quantifying the exception handling characteristics and other by defining exception handling. There could be chances that exception handling may not be affecting the software quality metrics. But mishandling exception handling could lead to errors. Hence the authors from the research paper has come up with two research questions and they are

***RQ1: Do exception handling flow characteristics contribute to a better explanation of the probability of post-release defects?***

***RQ2: Do exception handling anti-patterns contribute to a better explanation of the probability of post-release defects?***

1. *Metrics*

As per our research paper1, to obtain the relation between exception handling practices and post-release defects, we had to form metric based on several analysis of our subject project. We obtained the metrics based on the different studies on our subject project, development history of the version control system and issue tracking systems of the subject projects. There are four different categories of metrics for our study.

**Post-release defects**. For Elasticsearch 2.4.0, to get the post release defects. We searched the defects and changes which were fixed after the release of our subject project. The versions which were almost near to the date of our subject project release date were versions 2.4.1 - 2.4.6 and version 5.0. We referred the release notes of each version for bug fixes and defect changes. We later tracked each java file related to the bug fixes and defect changes.

**Traditional product metrics**. The product metric which involves metric like size (e.g., lines of code) and complexity (e.g., cyclomatic complexity), which are good indicators of product metrics. We took Understand tool and gave our subject project as an input to get the product related metrics.

**Traditional process metrics**. As per the research paper1, it is found that Process metrics to be more powerful than the usual product metrics. The process metrics includes prior release changes to our subject project.

**Change metrics**. The prerelease changes are Change metrics. We have taken the changes done before to the release date of our subject project. We calculate the total number of changes and total code churn as two change metrics.

"*git diff commit1 commit2 --numstat grep java"*, returns the added and deleted lines for each java file between commit1 and commit2.

**Human factors**. As per the research paper1, it is said to be have software defects related to the code ownership. The number of unique authors and their unique email addresses are noted to relate the number of changes done. We have not followed this concept as the information related to authors was not readily available and as per our studies there will not be much significance in knowing the authors behind the changes.

we used the following command to get the list of authors for each file,

**Git Command:** **git shortlog -e -n -s** **modules/reindex/src/main/java/org/elasticsearch/index/reindex/RestReindexAction.java**

**Pre-release quality metrics**. We have taken the prior releases in consideration to count the prerelease changes and prerelease bugs. From the release notes of versions 2.3.0 – 2.3.5, we were able to get the bugs fixed and changes made to the java files.

**Exception handling metrics**. As per the research paper1, there are two sets of sets of the exception handling metrics and they are as follows. Exception flow characteristics metrics. In these metrics it is all about the characteristics of exception flow. These metrics show the suboptimal exception handling practices followed by our subject project. To find these metrics we have designed a Java plug-in tool with visitor pattern, to visit all the classes and its elements in our subject project and meanwhile it will search for the characteristics of exception flow. Table 2 shows the metrics related to characteristics of exception flow.

**Exception handling anti-pattern metrics**. In this section it is finding the possible anti patterns. Detecting major anti patterns like log and return null, destructive wrapping, over catches etc. These anti patterns are detected along with the exception flow metrics. Using the Java plug-in tool, by serving the source project as the input to the plug-in we derive the required anti patterns.

We have used Eclipse JDT to parse our example Java Project-Elasticsearch to extract anti-patterns of the exception handling practices. Along with the parsing of exception handling patterns, we also analyze and extract flow metrics.

1. *Model Construction*

We construct logistic regression models to assess the explanatory energy of the exception coping with practices on post-launch defects We construct a base model with most effective conventional software metrics and without the metrics which might be associated with exception managing practices. We mix a model known as BSFC by adding code metrics that are associated with quantified exception flow characteristics from previous studies into the bottom model. we have a tendency to conjointly add code metrics that are related to the exception handling anti-patterns from previous studies into the bottom model to construct a second combined model known as BSAP. By examining the importance and also the informative power of the metrics in BSFC and BSAP, we have a tendency to answer.

*4.1 Missing data analysis*

After extracting metrics from the data, we’d still have missing knowledge. we have a tendency to manually examine the files with missing knowledge. we discover that the explanations might because of the cases wherever the file isn't compliable or cases within which the ways of a strive block really doesn’t throw any exception (e.g., forgotten try blocks throughout code evolution).

*4.2 Normality adjustment*

To achieve normality within the logistic regression model, within the outcome we need Standard deviation as an instance, since post-release defects exists only a relatable amount of files, we apply a log transformation. This results on the data to provide better assumptions and lessen skewness.

*4.3 Correlation analysis*

High correlation can be found between software metrics. This high correlation. But clustered data might be represented using a single predictor. So, we used Spearman pairwise rank correlation to correct the normality of data. The findCorrelation() of Caret R package is used for this and used to separate the highly correlated values.

*4.4 Redundancy analysis*

Other than pairwise correlations, we even take one predictor and reduce it more down to form our model. We execute the steps in iterative manner, in which predictors are dropped with an R2 until R2 dropped until 0.9. This is achieved using the redun() of R-HMisc package. As a result, we get the final number of useful predictors for our prediction. Table1

*4.5 Fit regression model*

We use logistic regression to find and model the probability of post release defects.When we have significant predictors to build our model, we use the glm() from the R package. The probability of having post-release defects in a given file is predicted accordingly.

*5. Model Analysis*

*5.1 Model Stability assessment*

We assess the model using the R2 which is generated using the glm method. Since we build our model with features obtained from static code analysis and historical data put together, the validate() from the R package. This helps in predicting more bugs with our different data samples. This R2 helps in dealing with the noise among the predictors and avoid overfitting.

*5.2 Predictors in the outcome measurement*

In this step, we calculate the model outcome by setting all predictors at their mean value. For each significant predictor, we increase its value by 10% while keeping all other significant predictors at their mean values. We measure the differences of the model outcome as the effect of the predictor. We use the predict method to arrive to the predictors which make a higher contribution to model fit

Table 1:

|  |  |
| --- | --- |
| Project | Significant Metrics |
| Elastic Search 2.4.0 | Changes and human factors |
| Complexity |
| Pre-release bugs |
| Action of Log |
| Number of throws generic AP |
| Number of relying on GetCause AP |

*5.3 Prelim results*

The preliminary results caused a lot of stand deviation turning to zero with few NA. column results. Those results contributed

to less than 5% of the data, so they are found to be irrelevant and removed to get precise modelling results.

*5.4 Case Study Results and Discussion*

The following section discusses the case study results according to the research question. We analyze about each part of our sample data -base flow metrics, exception metrics and exception flow metrics.

***RQ1: Do exception handling flow characteristics contribute to a better explanation of the probability of post-release defects?***

Understanding the exception handling flow metrics effect on probability of post release bugs, can be interpreted using the significant predictors of the flow metrics. We compare the model of BASE and BSFC to deduce the case results. We observe the flow characteristics have a lesser effect on the post-release defects. These values tend to be very less significant compared to other values, 0.2. Even though we omit few data with higher power, **the prevalence of negative relationship on the probability of post release defects**. The most significant values tend to be human factors with file changes and pre-release bugs.

Moving on to the exception blocks - log actions inside the catch blocks contribute to probability of post release defects. This accounts to the action of throw has been an cause of creating an exception anti pattern among the files on elastic search. For example, the developers of ElasticSearch project, have handled all kinds of exception antipattern like destructive wrapping and over-catch. It also implies, log action should be handled better than any other exceptions.

**The Log actions have a significant effect on the probability of the post release defects.**

Anti-patterns like destructive wrapping, over -catch, were found to be less relative affecting the post release bugs. Also, dummy handlers were less prevalent and too low in affecting the probability. The “Multiline log” and “Log and throw” were the primary ones among exceptions.

The Log action is an indicator that the exception is not handled, but, instead, the exception is recorded by logging and propagated with throw. Therefore, logging the exception is often required to later (i.e., in the case of a runtime event) examine such exceptions. Research also suggests that more logs may indicate that developers have uncertainties about the source code, leading to a positive relationship with the post-release defects

The number of changes added and deleted contribute a significant value to post release bugs probability. These are primary human factors and changes in file take the highest significance and portray that bugs arise in the changed files. Changes in one part of the file might escalate to other dependent files and few additions, deletion become significant enough to cause a post release bug.

The characteristics of changes - additions and deletion contribute a significant amount of contribution to post release bugs.

***RQ2- Do exception handling antipatterns contribute to a better explanation of the probability of post release defects***.

As mentioned in paper 1- exception handling patterns complement traditional metrics in explaining post release defects, but most of the anti-patterns were not proved to be statically significant to the cause of post release defects.

With all the metrics of exception handling ant patterns, the log and throw antipattern found to be significant, but the majority of them were not statically significant.

The total number of catch blocks affected by Log and Throw has a positive relationship with the probability of post-release defects in Hadoop. The Log and Throw antipattern has been advocated to cause adverse effects on the code. As mentioned in the paper [1] Log and throw in a file can make it harder for developers to understand where an exception comes from. This anti-pattern could affect software executions since only records are the logs present in files and we lose the flow of an exception. This anti-pattern could also affect debugging by preventing developers from finding the errors.

The getcause handler is also bringing significant change in post release bugs. This could be the reason, the developers focused too much on checked exceptions and handling the exception using the generic clause. The anti-pattern explaining the irrelevant handling of the exceptions and the developers need to lessen its effect in handling post release bugs.

**IV. THREATS TO VALIDITY**

**External Validity:**

Our study is based on only 1 Java project -elastic search which gave us profound results on all aspects of feature collection. Replication of the same project on other java systems might give us project specific results along with other results. Using multiple projects and building models with those system could contribute a much accurate information

**Internal Validity:**

We aim to include all possible sources of information in our automated exception flow analysis. However, our data collection and our modelling might still leave out possible exceptions, when there is lack of information or file detection issues. Also, the documentation of the exception may be incorrect or outdated. In our analysis, we trust the content of documentation. Therefore, we cannot claim that our analysis fully recovers all possible exceptions nor that the recovered information is impeccable. We also include these effects of pre-release, post -release and file changes with manual intervention of git commands. Manual error might also tend to few losses and discrepancies in the data.

Moreover, we aim to understand the relationship between exception handling practices and post-release defects, and we do not aim to predict post-release defects. We don't advocate the effects of post release bugs on the software and its improvement with this paper.

**Construct validity:**

Our study focused on the paper [1] metrics and patterns. . We selected actions based on the previous research in the subject. We extracted exception handling patterns, their flow with Understand tool results to find the significance. However, we might specifically no taken Object oriented based exception handling methods in the files. Few columns in the bas and flow metrics given by the understand tool tended to be zeros. Their significance did not contribute much to our model. We entirely trust the tool results to keep up to our data model. Adding more metrics may provide a further understanding of its relationship with post-release defects. Some metrics are even removed through high correlation, we do not include those ones and test our metrics to build the model. Another approach to resolving this issue is using expert knowledge Expert knowledge would indicate which predictor should not be considered. We do not opt to leverage expert knowledge since we want to avoid subjective bias in the results.

**REFERENCES**

1. G. B. de Pádua and W. Shang, "Studying the Relationship between Exception Handling Practices and Post-Release Defects," *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, Gothenburg, 2018, pp. 564-575..
2. <https://community.oracle.com/docs/DOC-983543>.
3. <https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/glm>
4. <https://cran.r-project.org/web/packages/Hmisc/index.html>
5. <https://www.rdocumentation.org/packages/Hmisc/versions/4.4-0/topics/redun>
6. <https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/cor>
7. G. B. D. Pádua and W. Shang, "Studying the Prevalence of Exception Handling Anti-Patterns," 2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC), Buenos Aires, 2017, pp. 328-331.

[8] https://github.com/Kiranmayie/SOEN691Project.git

**V. CONCLUSION**

Exception handling is an important feature in modern programming languages. It helps the developers to mitigate errors in their system in a smoother way. The effects of anti -patterns are widely analyzed in this paper. .In this study and implementation, the exception handling practices, including the characteristics of exception flow and the exception handling anti-patterns, have a statistically significant relationship with post-release defects. We find exception flow characteristics in Elastic search projects have a significant explanatory power when complementing traditional software metrics in modeling post-release defects. Such results implore the importance of properly handling exceptions. In addition, although the majority of the exception handling anti-patterns are not significant in explaining post-release defects, there exist some anti-patterns that indeed have a positive relationship with post-release defects.