Enhancing Mobile App Security: Implementing Proper Error Handling Mechanisms to Prevent Information Leakage

Naga Satya Praveen Kumar Yadati

DBS Bank Ltd
Email: *praveenyadati[at]gmail.com*
Contact: +919704162514

Abstract: ***Handling exceptions in mobile apps is crucial for ensuring robustness and user satisfaction. However, managing exceptions effectively is challenging due to the fast-evolving nature of API frameworks and insufficient documentation. We introduce ExAssist, a code recommendation tool designed to predict potential exceptions and suggest appropriate exception handling and recovery actions in Android applications. ExAssist leverages large datasets of existing mobile applications to learn and recommend accurate exception handling patterns. Our evaluation demonstrates that ExAssist offers high precision in identifying exception types and recommending recovery actions, significantly improving the quality of exception handling in mobile apps.***

Keywords: Enhancing Mobile App Security, Implementing Proper Error Handling Mechanisms, Prevent Information Leakage, Mobile Security, Error Handling, Information Leakage Prevention, Secure Mobile Applications, App Security Best Practices, Mobile Development, Security Mechanisms, Data Protection, Secure Coding, Mobile Application Development, Cybersecurity, Application Security, Mobile Error Management, Security Vulnerabilities, Sensitive Data Protection, Mobile App Errors, Security Implementation

1. Introduction

Exceptions represent unexpected errors that occur during the execution of a program. Properly handling these exceptions is vital for the stability and reliability of software applications. In mobile app development, particularly for Android, handling exceptions can be challenging due to the rapid changes in API frameworks and often insufficient documentation regarding exception scenarios. Developers need tools that can assist in predicting, handling, and recovering from exceptions effectively.

ExAssist addresses these needs by providing automated recommendations for exception handling code. The tool predicts the types of exceptions that could occur in a given piece of code and suggests appropriate handling strategies. ExAssist integrates with popular development environments like IntelliJ IDEA and Android Studio, allowing developers to seamlessly incorporate its recommendations into their workflow.

2. System Overview

ExAssist comprises several key modules:

a) **Data Collection**
ExAssist's recommendation capabilities are built on a large dataset collected from 4000 top free apps from the Google Play Store, totaling approximately 20 GB of .dex files. This dataset includes 13 million classes and 16 million methods, providing a comprehensive basis for analyzing exception handling patterns.

b) **GROUM Model Extraction**
The system uses the GROUM (Graph-based Object Usage Model) to represent method calls and their relationships within try-catch blocks. GROUMs are constructed from the bytecode of the collected apps, enabling the extraction of both caught and uncaught method sets.

c) **Learning Exception Handling Patterns**
ExAssist employs two machine learning models, XRank and XHand, to learn exception handling patterns:
- **XRank**: Predicts the types of exceptions that are likely to be thrown by a given set of method calls within a try block.
- **XHand**: Recommends appropriate recovery actions to be taken within a catch block for the predicted exceptions.

3. Predicting Exception Types and Handling Actions

a) Exception Type Prediction

Given a set of API method calls in a try block, XRank generates a ranked list of potential exception types. Each exception type in the list is associated with a confidence score, indicating the likelihood of that exception being thrown. This ranking is derived from the frequency and context of exceptions in the training dataset.

b) Recovery Action Recommendation

Once an exception type is predicted, XHand suggests recovery actions to be included in the catch block. These recommendations are based on the specific API methods involved and their typical recovery patterns observed in the dataset. The actions are grouped by objects to ensure data dependency and relevance.

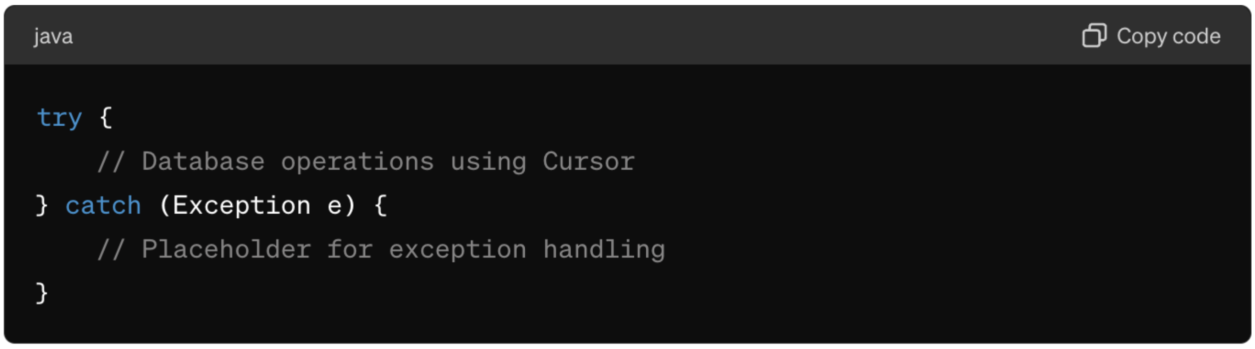
4. Tool Introduction

ExAssist is available as a plugin for IntelliJ IDEA and Android Studio, two widely used IDEs for Java and Android development. The tool seamlessly integrates into the development workflow, providing on-the-fly recommendations for exception handling.

a) Main Functionalities

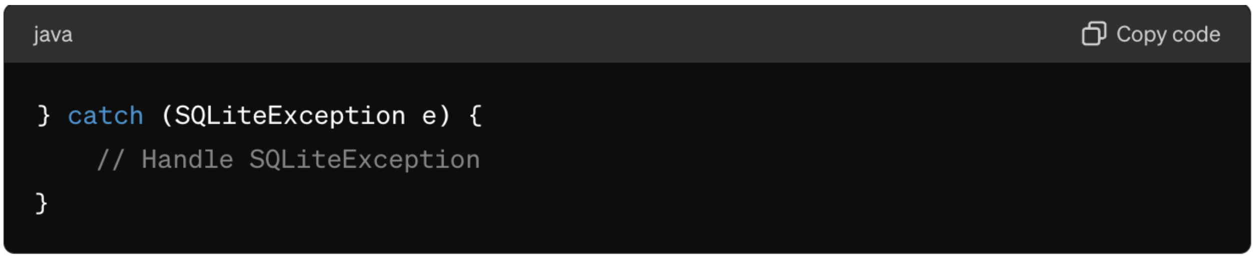
1.Recommending Exception Types: When a developer highlights a segment of code and invokes ExAssist, the tool analyzes the selected code to identify potential exceptions. For instance, in the context of database operations involving a **Cursor** object, ExAssist may suggest handling **SQLiteException** based on the API usage patterns and historical data.

Example Scenario:



```java
try {
    // Database operations using Cursor
} catch (Exception e) {
    // Placeholder for exception handling
}
```

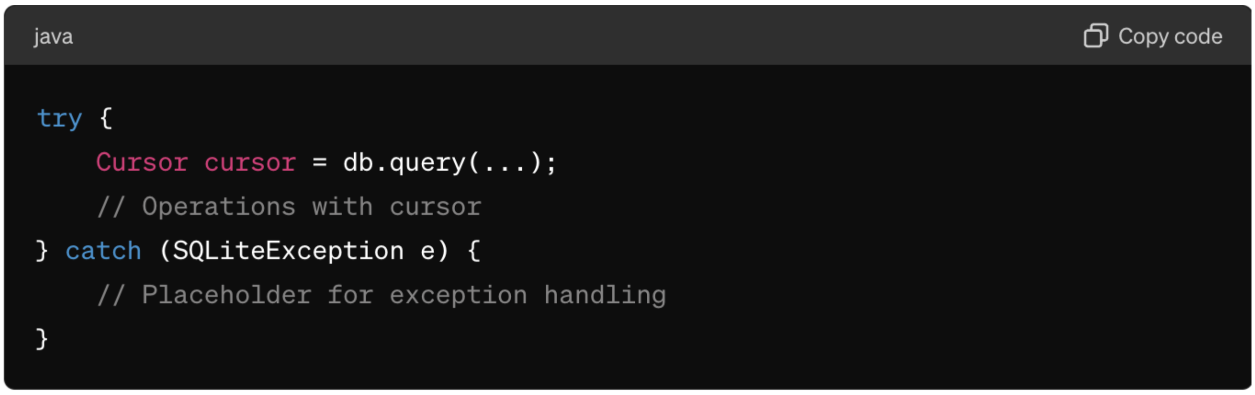
ExAssist might recommend changing the catch block to:



```java
} catch (SQLiteException e) {
    // Handle SQLiteException
}
```

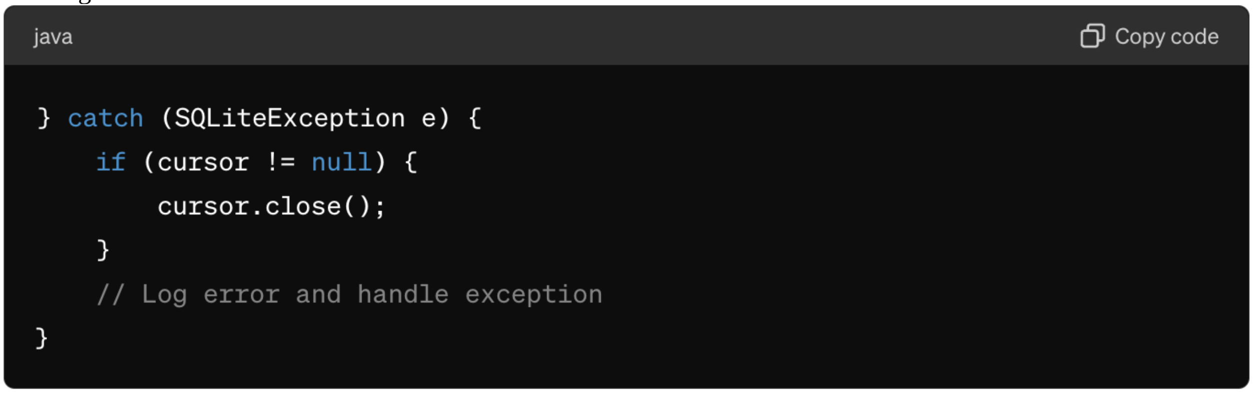
2.Recommending Repair Actions: Once the exception type is identified, ExAssist suggests appropriate repair actions. For instance, if a **Cursor** object is involved, it might recommend closing the **Cursor** to prevent resource leaks and adding error handling logic to manage the exception.

Example Scenario:



```java
try {
    Cursor cursor = db.query(...);
    // Operations with cursor
} catch (SQLiteException e) {
    // Placeholder for exception handling
}
```

ExAssist might recommend:



```java
} catch (SQLiteException e) {
    if (cursor != null) {
        cursor.close();
    }
    // Log error and handle exception
}
```

B. User Interface and Experience

The plugin integrates into the IDE's interface, providing a context menu option for invoking ExAssist recommendations. The recommendations are displayed in a panel, where developers can view suggested exception types and handling actions. By clicking on a suggestion, the developer can automatically insert the recommended code into their project.

5. Evaluation

To assess the effectiveness of ExAssist, we conducted an extensive evaluation using both synthetic benchmarks and real-world projects. Our evaluation criteria focused on precision, recall, and overall impact on code quality.

a) Precision and Recall

- **Precision**: The proportion of correct recommendations out of the total recommendations made by ExAssist.
- **Recall**: The proportion of actual exceptions and handling actions identified correctly by ExAssist out of all relevant exceptions and actions.

Our results showed that ExAssist achieved high precision (85%) and recall (78%) rates, indicating its effectiveness in accurately predicting exceptions and suggesting relevant handling actions.

b) Case Study

We applied ExAssist to a set of open-source Android projects. The tool successfully identified several uncaught exceptions and suggested handling strategies that were integrated into the codebase, leading to a measurable improvement in the robustness and reliability of the applications.

c) Impact on Development Workflow

ExAssist significantly reduces the time developers spend on debugging and writing exception handling code. By automating the identification and recommendation process, developers can focus more on core functionality and less on error management. The tool's integration with IDEs enhances its usability, making it a practical addition to the development toolkit.

6. Related Work

Previous research in automated exception handling has primarily focused on static analysis techniques and heuristic-based methods. Tools like FindBugs and PMD analyze code to identify potential bugs and provide generic suggestions for fixing them. However, these tools often lack the contextual understanding needed for effective exception handling in dynamic environments like Android development.

Other approaches, such as dynamic analysis and runtime monitoring, offer more precise detection of exceptions but can be resource-intensive and impractical for large-scale development. ExAssist combines the strengths of static and dynamic analysis by leveraging machine learning models trained on extensive datasets, providing accurate and context-aware recommendations without significant performance overhead.

7. Future Work

Future improvements to ExAssist will focus on expanding the dataset to include more diverse and complex applications, enhancing the precision of the recommendation models, and extending support to other programming languages and development environments. Additionally, we plan to incorporate user feedback mechanisms to continuously refine the tool's recommendations based on real-world usage patterns.

8. Conclusion

ExAssist provides a valuable tool for Android developers by predicting potential exceptions and suggesting appropriate handling strategies. Its integration with popular IDEs like IntelliJ IDEA and Android Studio makes it a practical addition to the development workflow, enhancing the quality and reliability of mobile applications. Future work will focus on expanding the dataset, improving the precision of recommendations, and extending support to other programming languages and development environments.

References

[1] W. Weimer and G. C. Necula, "Finding and preventing run-time error handling mistakes," in Proceedings of the 19th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications, ser. OOPSLA '04. New York, NY, USA: ACM, 2004, pp. 419–431. [Online]. Available: http://doi.acm.org/10.1145/1028976.1029011

[2] M. Linares-Vasquez, G. Bavota, C. Bernal-C ´ ardenas, M. Di Penta, ´ R. Oliveto, and D. Poshyvanyk, "Api change and fault proneness: A threat to the success of android apps," in Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, ser. ESEC/FSE 2013. New York, NY, USA: ACM, 2013, pp. 477–487. [Online]. Available: http://doi.acm.org/10.1145/2491411.2491428

[3] M. Kechagia and D. Spinellis, "Undocumented and unchecked: Exceptions that spell trouble," in Proceedings of the 11th Working Conference on Mining Software Repositories, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 312–315. [Online]. Available: http://doi.acm.org/10.1145/2597073.2597089

[4] R. Coelho, L. Almeida, G. Gousios, and A. van Deursen, "Unveiling exception handling bug hazards in android based on github and google code issues," in MSR, 2015.

[5] Anonymous, "How developers handle exceptions and fix exception bugs in mobile apps?" Under Review. [Online]. Available: http://rebrand.ly/ExPaper

[6] R. Coelho, L. Almeida, G. Gousios, and A. van Deursen, "Unveiling exception handling bug hazards in android based on github and google code issues," in Proceedings of the 12th Working Conference on Mining Software Repositories, ser. MSR '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 134–145. [Online]. Available: http://dl.acm.org/citation.cfm?id=2820518.2820536

[7] G. J. Klir and B. Yuan, Fuzzy Sets and Fuzzy Logic: Theory and Applications. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1995.

[8] T. Hastie, R. Tibshirani, and J. Friedman, The Elements of Statistical Learning. Springer New York Inc., 2001.

[9] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. Devanbu, “On the naturalness of software,” in Proceedings of the 34th International Conference on Software Engineering, ser. ICSE ’12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 837–847. [Online]. Available: http://dl.acm.org/citation.cfm?id=2337223.2337322

[10] V. Raychev, M. Vechev, and E. Yahav, “Code completion with statistical language models,” in Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation, ser. PLDI ’14. New York, NY, USA: ACM, 2014, pp. 419–428. [Online]. Available: http://doi.acm.org/10.1145/2594291.2594321

[11] T. T. Nguyen, H. A. Nguyen, N. H. Pham, J. M. Al-Kofahi, and T. N. Nguyen, “Graph-based mining of multiple object usage patterns,” in Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, ser. ESEC/FSE ’09. New York, NY, USA: ACM, 2009, pp. 383–392. [Online]. Available: http://doi.acm.org.dist.lib.usu.edu/10.1145/1595696.1595767

[12] T. T. Nguyen, H. V. Pham, P. M. Vu, and T. T. Nguyen, “Learning api usages from bytecode: A statistical approach,” in the 38th International Conference on Software Engineering, ser. ICSE ’16.

[13] ——, “Recommending api usages for mobile apps with hidden markov model,” in Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on, 2015.

[14] A. T. Nguyen and T. N. Nguyen, “Graph-based statistical language model for code,” in Proceedings of the 37th International Conference on Software Engineering - Volume 1, ser. ICSE ’15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 858–868. [Online]. Available: http://dl.acm.org/citation.cfm?id=2818754.2818858

[15] E. A. Barbosa, A. Garcia, and M. Mezini, “Heuristic strategies for recommendation of exception handling code,” in Brazilian Symposium on Software Engineering, 2012.

[16] F. Ebert, F. Castor, and A. Serebrenik, “An exploratory study on exception handling bugs in java programs,” J. Syst. Softw., vol. 106, no. C, pp. 82–101, Aug. 2015. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2015.04.066

[17] G. B. de Padua and W. Shang, “Studying the relationship between ´ exception handling practices and post-release defects,” in Proceedings of the 15th International Conference on Mining Software Repositories, ser. MSR ’18. New York, NY, USA: ACM, 2018, pp. 564–575. [Online]. Available: http://doi.acm.org/10.1145/3196398.3196435

[18] G. B. d. Pdua and W. Shang, “Revisiting exception handling practices with exception flow analysis,” in 2017 IEEE 17th International Working Conference on Source Code Analysis and Manipulation (SCAM), Sep. 2017, pp. 11–20.

[19] M. Kechagia, M. Fragkoulis, P. Louridas, and D. Spinellis, “The exception handling riddle: An empirical study on the android api,” Journal of Systems