Poster: Recommending Exception Handling Patterns with ExAssist

Tam The Nguyen[1], Phong Minh Vu[1], Hung Viet Pham[2], Tung Thanh Nguyen[1]
[1]Auburn University, [2]University of Waterloo
tam@auburn.edu,lenniel@auburn.edu,hv.pham.2704@gmail.com,tung@auburn.edu

ABSTRACT

Exception handling is an advanced programming technique to prevent run-time errors or crashes for modern software systems. However, inexperienced programmers might fail to write proper exception handling code in their programs. In this paper, we introduce ExAssist, a code recommendation tool for exception handling. ExAssist can predict what types of exception could occur in a given piece of code and recommend proper exception handling code for such an exception. Preliminary evaluation of ExAssist suggests that it provides highly accurate recommendations.

1 INTRODUCTION

Exceptions are unexpected errors occurring at run-time of software systems like a resource is not found or a division by zero. Improperly handled exceptions can lead to severe system failures, such as crashes or hangs. To prevent such failures, popular programming languages and frameworks often include advanced exception handling mechanisms. For example, Java has the try catch code construct and an extensive collection of exception types (e.g. FileNotFoundException, NullPointerException, IllegalArgumentException etc) for handling exceptions. However, prior research suggests that inexperienced programmers still fail to include proper exception handling code in their programs [1], leading to severe consequences.

There are several tools for API usage recommendation, such as Grapacc [2], DroidAssist [3], have been developed, but most of them do not suggest exception handling usages when using objects. Other methods such as WN-miner[5] and CAR-miner[4] aims to mine exception-handling rules for bug detections not code suggestion.

To address that drawback, in this paper, we introduce ExAssist, a code recommendation tool for exception handling. ExAssist predicts what types of exception could occur in a given piece of code and recommends proper exception handling code for such an exception. When requested, it will add such code into the given piece of code.

ExAssist is released as a plugin of IntelliJ IDEA and Android Studio, two popular IDEs for Java and Android mobile apps. After installation, it is incorporated with the IDE and users can invoke it directly via shortcut key Ctrl + Alt + R or via the menu bar. Let us present ExAssist's main functionality via two usage scenarios.

Recommending Exception Types. Assume a developer is writing code to open and get data from a database (Figure 1). The developer is aware that the code is dealing with database and Cursor objects might throw unchecked exceptions at runtime, but she might be unsure whether to catch exceptions on the code and which type of exception to catch. The built-in exception checker in Android Studio only supports adding checked exceptions, thus, does not help her to make appropriate action in this case.

ExAssist aims to support the developer to make decisions whether or not to add a try-catch block and what type of exception to caught. The developer invokes ExAssist by first selecting the portion of code that she wants to check for exception then pressing Ctrl + Alt + R. Figure 1 shows a screenshot of Android Studio with ExAssist invoked for the portion of code that using the Cursor object for reading data from database. As seen, ExAssist suggests that the code is likely to throw an unchecked exception. It also displays a ranked list of unchecked exceptions that could be thrown from the current selecting code. Each unchecked exception in the ranked list has a confident score represents how likely the exception will be thrown from the code. The value for confident scores is between 0 and 1. The higher the value of the confident score, the higher likelihood the exception type is thrown. In this example, SQLiteException has the highest score of 0.80. If the developer chooses that exception type, the currently selected code will be wrapped in a try-catch block with SQLiteException in the catch expression.

ExAssist uses the context of current selecting code to infer whether or not adding exception handling code and the type of the exception. For example, in Figure 2, the context changes as the developer selects the portion of code for opening and querying on the SQLiteDatabase object. Thus, ExAssist updates the recommendation list with SQLException has the highest confident of 0.81, which is highest among all other exception types.

ExAssist could provide recommendations for a selected portion of code includes one or multiple method calls. Additionally, ExAssist could also recommend not to add try-catch block if it infers that the selected code is very unlikely to throw an unchecked exception. For example, if the developer selects the statement bookTitles.add(bookname); and queries ExAssist, the tool will return an empty list of exceptions as it is very unlikely the selected method throws exceptions when it is executed.

Recommending Exception Repairs. Handling exception situations and executing necessary recovery actions are important as it could help apps continue to run properly when an exception occurs. For example, when an app reuses resources such as database connections or files, the app should release the resources if an exception is thrown. ExAssist is also designed to recommend such repairing actions in the exception handling code based on the context in the try block.

Figure 3 demonstrates an usage of ExAssist in the task. After adding a try-catch block with SQLiteException for the code in the

ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5663-3/18/05.
https://doi.org/10.1145/3183440.3194971

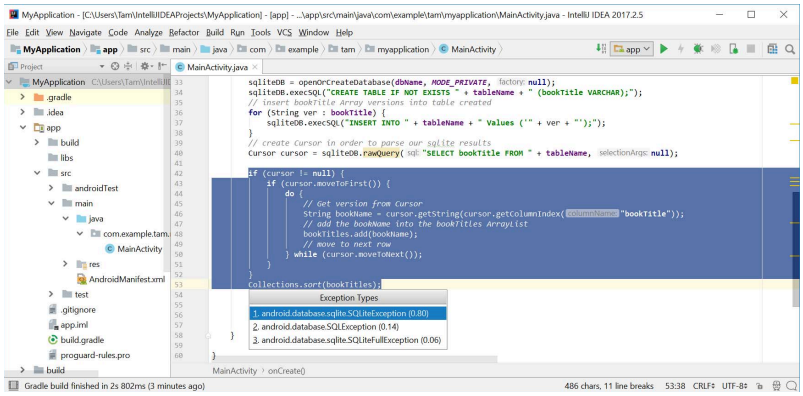

Figure 1: **Recommending Exception Types by ExAssist**

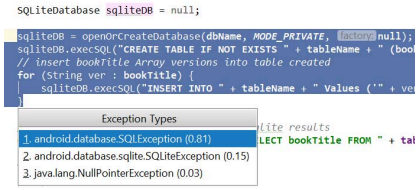

Figure 2: **Recommendation for the usage of object sqliteDB**

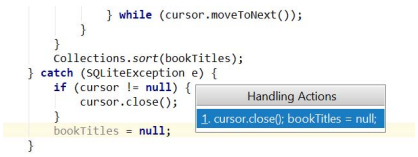

Figure 3: **Recommending repairing actions**

previous scenario, the developer wants to perform recovery actions. To invoke ExAssist, she moves the cursor to the first line of the catch and presses Ctrl + Alt + R. ExAssist then will analyze the context of the code and provide repairing actions in the recommendation windows. In the example, ExAssist detects that the Cursor object should be closed to release all of its resources and making it invalid for further usages. It also suggests to set bookTitles equals null to indicate the error while collecting data from cursor. If the developer chooses the recommended actions, ExAssist will generate the code in the catch block as in the Figure 2.

2 PRELIMINARY EVALUATION

2.1 Detecting Exception Related Bugs

With the usage described above, we can see that ExAssist could be applied to detect real exception related bugs. In particular, we focus on bugs that are caused by not catching proper unchecked exceptions, which often lead to crashes and/or unexpected behaviors of apps. For convenience, we define those bugs as exception bugs.

To evaluate ExAssist, we manually collected 128 exception bug fixes from 10 open-source Android projects. Each exception bug fix is a fix for an exception bug in which developers add a try-catch block to handle exceptions. For each exception bug in the dataset, we used ExAssist to recommend whether or not adding a try-catch block on the code that causes the exception and what type of exception that should be caught. We then compared recommendations of ExAssist with the fixes provided by developer.

Overall, ExAssist achieves a high level of accuracy in detecting exception bugs. Over 128 bugs, ExAssist recommends adding try-catch blocks for 116 cases (90.62%). It recommends the exception types in top-1 recommendation for 86 cases (74.13%), top-2 recommendation for 96 cases (82.75%), and top-3 for 104 cases (89.65%).

2.2 Handling Exception Bugs

In the section, we evaluation ExAssist in recommending repairing actions with real exception handling bug fixes. We collected a dataset contains 82 exception bug fixes from the same 10 Android open-source projects as in the previous section. In each bug fix of the dataset, developers performed at least one repairing action (i.e. a method call, an assignment, etc.) in the exception handling code.

For each bug fix in the dataset, we invoked ExAssist to recommend repairing actions and compare the recommendation result with the corresponding fix of the developer. If the recovery actions recommended by ExAssist exactly match with the fix, we count it as a match. If the recommended actions of ExAssist contain the code in the fix of the developer, we count it as a partial match. Otherwise, we consider the case as a miss.

In the total of 82 exception bug fixes, the recommendations of ExAssist match the fixes of developers in 40 cases. There are 20 cases in which the recommended actions of ExAssist contain the fix code. Overall, ExAssist could provide meaningful recommendations in roughly 75% of the bug fixes.

REFERENCES

[1] R. Coelho, L. Almeida, G. Gousios, and A. v. Deursen. 2015. Unveiling Exception Handling Bug Hazards in Android Based on GitHub and Google Code. In *MSR*.
[2] A. Nguyen, H. Nguyen, T. Nguyen, and T. Nguyen. 2012. GraPacc: A Graph-based Pattern-oriented, Context-sensitive Code Completion Tool. In *ICSE '12*.
[3] T. Nguyen, H. Pham, P. Vu, and T. Nguyen. 2015. Recommending API Usages for Mobile Apps with Hidden Markov Model. In *ASE*.
[4] S. Thummalapenta and T. Xie. 2009. Mining Exception-handling Rules As Sequence Association Rules. In *ICSE*.
[5] W. Weimer and G. Necula. 2005. Mining Temporal Specifications for Error Detection. In *TACAS*.