Improving Exception Handling with Recommendations

Eiji Adachi Barbosa
Advisor: Alessandro Garcia
Exceptions occurrence

System that manages photo albums

```
View

PhotoScreen.handleEvent(event)
{
    Controller.performAction(event);
}

Controller

Controller.performAction(event)
{
    if(event == REMOVE_PHOTO)
    {
        Photo.remove(event.getObject());
        PhotoScreen.update(SUCCESSFUL);
    }
}

Model

Photo.remove(object)
{
    PhotoAccessor.delete(object);
}

PhotoAccessor.delete(object)
{
    RecordStore.deleteRecord(object);
}
```
Ignoring exceptions

Implementing an empty and generic catch block

```java
PhotoScreen.handleEvent(event) {
    Controller.performAction(event);
}

Controller.performAction(event) {
    if (event == REMOVE_PHOTO) {
        Photo.remove(event.getObject());
        PhotoScreen.update(SUCCESSFUL);
    }
}

Photo.remove(object) {
    PhotoAccessor.delete(object);
}

PhotoAccessor.delete(object) {
    try {
        RecordStore.deleteRecord(object);
    } catch (Exception e) { //ignore }
}
```
Poor quality of exception handling

- Excessive use of *generic* exception types and *empty* catch blocks [1]
- High number of *uncaught* exceptions [2]
- *Defect* density of exception handling code is three times higher than normal code [3]

Limitations of state-of-art

- Current solutions focus on computing *propagation paths of exceptions* [4, 5, 6]
  - No support to decide where and how to handle exceptions

---

### Detecting poor exception handling

<table>
<thead>
<tr>
<th>View</th>
<th>Controller</th>
<th>Model</th>
</tr>
</thead>
</table>
| PhotoScreen.handleEvent(event) {  
   Controller.performAction(event);  
} | Controller.performAction(event) {  
   if(event == REMOVE_PHOTO) {  
      Photo.remove(event.getObject());  
      PhotoScreen.update(SUCCESSFUL);  
   }  
} | Photo.remove(object) {  
   PhotoAccessor.delete(object);  
} |

PhotoAccessor.delete(object) {  
   try {  
      RecordStore.deleteRecord(object);  
   }  
   catch(Exception e) {//ignore}  
}
Limitations of state-of-art

- Lack of explicit exception handling policies
  - An exception handling policy regards to designer’s original intent regarding the use of exceptions
    - What exception types
    - Where exceptions are thrown, handled, propagated and remapped
  - There are no means to specify exception handling policies
Handling exceptions

Exceptions must be handled by the controller

View

```java
PhotoScreen.handleEvent( event ){
    Controller.performAction( event );
}
```

Controller

```java
Controller.performAction( event ){
    if(event == REMOVE_PHOTO){
        try{
            Photo.remove(event.getObject());
            PhotoScreen.update(SUCCESSFUL);
        }
        catch(Exception e){update(ERROR);}
    }
}
```

Model

```java
Photo.remove( object ){
    PhotoAccessor.delete(object);
}
```

```java
PhotoAccessor.delete( object ){
    RecordStore.deleteRecord(object);
}
```
Objective:
- Aid developers in producing better exception handling code

Key elements of proposed solution:
- **Domain-specific language** to specify exception handling policies
- **Verification tool** to check source code adherence
- **Recommender system** to assist developers in implementing adhering code
Usefulness and Applicability

Designers use the DSL to specify the exception handling policy.

Software designer uses Domain-Specific Language to specify Exception Handling Policy.
Usefulness and Applicability

Developers read the specification to comprehend expected use of exceptions.

Software designer

Domain-Specific Language

Exception Handling Policy
Usefulness and Applicability

Recommender aids developers to implement code adherent to the specification.

- Software designer
- Domain-Specific Language
- Exception Handling Policy
- Recommender System
- Source Code
- Developer
- Source Code
- Developer
Software designer

Domain-Specific Language

Verification tool

Recommender System

Exception Handling Policy

Designers use verification tool to check source code

Developer

Source Code

Developer

Source Code
Technical challenges

- Design and implement an expressive domain-specific language
- Improve current recommending techniques to incorporate information from exception handling policies
Evaluation

- Domain-specific language:
  - Case study
  - Qualitative study
- Recommender system:
  - Controlled experiments
  - Qualitative study
Progress and Planning

- **Done:**
  - DSL design, implementation and evaluation

- **To-Do (next 2 years):**
  - Expand recommending heuristics
  - Implement recommender system
Improving Exception Handling with Recommendations

Eiji Adachi Barbosa

Thank you