A Tool-Supported Quality Smell Catalogue For Android Developers

Jan Reimann, Martin Brylski, Uwe Aßmann
Software Technology Group
Technische Universität Dresden
Dresden, Germany
jan.reimann|uwe.assmann@tu-dresden.de, martin.brylski@gmail.com

Usual software development processes apply optimisation phases in late iterations. The developed artefacts are optimised regarding particular qualities. In this sense *refactorings* are executed since the existing behaviour is preserved while the artefact improves its quality properties. The problem is that it is hard for developers to detect model structures dissatisfying a certain quality requirement manually. Without a set of potential problems and their explicit relation to particular qualities it is not possible to detect quality-specific deficiencies automatically. Furthermore, without potential solutions the identified poor structures cannot be resolved by tools. To overcome these problems we introduce a new quality smell catalogue focussing the Android platform.

1 Motivation

The *quality requirements* of applications may be specified explicitly in the requirements document, or become present in optimisation phases when deficiencies regarding the qualities are noticed, as e.g. that the battery of a mobile device drains too fast. What developers do when optimising w.r.t such qualities is refactoring [3]. They manually detect relevant artefacts containing structures being responsible for not satisfying the particular quality requirements. Fowler calls such structures *bad smells* which indicate candidates for applying refactorings to improve qualities while preserving the behaviour.

Due to this background we correlated the concepts *bad smell*, *quality* and *refactoring*, and introduced the term *quality smell*. A quality smell is a certain structure in a model, indicating that it negatively influences specific quality requirements, which can be resolved by particular model refactorings [4].

In contrast to Fowler's bad smells and refactorings (being universally applicable), quality smells are very concrete because satisfying a particular quality requirement has different meanings or granularities in distinct contexts. Thus, the principle of a quality smell is universal in a particular domain but the concrete instance is specific because it refers to a precise setting, as e.g. the use of a concrete framework. Because of this we decided to focus the context of mobile development since quality requirements play an essential role in this area. We chose Android since it is publicly available. The problem in mobile development is that developers are aware of quality smells only indirectly because their definitions are informal (best-practices, bug tracker issues, forum discussions etc.) and resources where to find them are distributed over the web. It is hard to collect and analyse all these sources under a common viewpoint and to provide tool support for developers.

To overcome these limitations we compiled a catalogue for Android. It contains 30 possible quality smells, explaining which qualities they influence, and potential refactorings to resolve them.

We implemented the concept of quality smells and the catalogue within our generic model refactoring framework Refactory [5].[1] Our tool is based on the Eclipse Modeling Framework (EMF) and seamlessly integrates into existing model-driven setups.

2 Quality Smell Catalogue

As already mentioned the catalogue contains 30 quality smells. Based on the catalogues from Brown et al. [2] and Fowler [3] we derived a similar scheme each quality smell conforms to which can be recognized in the example explained in Sect. 3. The whole catalogue can be found here:

http://www.modelrefactoring.org/smell_catalog/

In the following we show how this catalogue relates to the model-based context and discuss important elements of the quality smell concept.

As explained in detail in [4] an explicit relation between qualities and model structures dissatisfying these qualities is needed to provide tool support for being able to focus a particular quality. Once such

[1] http://www.modelrefactoring.org/

a relation is established developers are able to detect quality smells in each point in time automatically, instead of considering qualities after development phases and searching poor structures manually. Furthermore, our concept of quality smells correlates the aforementioned constituents with a set of refactorings to support the suggestion of resolving refactorings in case poor structures regarding a quality are detected.

As a consequence this concept consists of the three phases *detection*, *suggestion* and *resolution* whereas each of them follows a model-based approach. On the one hand side we use JaMoPP[2] to allow Java code to be handled in terms of a model. For the detection of poor structures we integrated IncQuery [1] into our architecture which is able to query EMF-based models for patterns. Thus, we defined patterns for each constituent in our catalogue. For the suggestion we automatically generate quick fixes each being able to execute a particular refactoring. The refactorings are specified with our role-based approach explained in [5]. Finally, the tool Refactory is able to resolve a quality smell by interpreting the refactoring specification.

3 Quality Smell Example

In the following one quality smell from the catalogue is presented.

Name Interruption From Background

Context UI

Affected Qualities User Expectation, User Experience, User Conformance

Description According to the Android developer guide[3] users really should not be interrupted when working with a mobile device since interruption does not conform to the user's expectations. In the worst case supplied data could get lost.

```
1 public class InterruptingService extends Service {
2   public IBinder onBind(Intent intent) {
3     return new Binder();
4   }
5   public void onCreate() {
6     super.onCreate();
7     Toast.makeText(this, "Hello World!",1000).show();
8   }
9 }
```

Listing 1: Interrupting service.

Listing 1 shows an interrupting background service creating a `Toast` (line 7) which pops up a dialogue and disturbs the user. This is to be avoided.

Refactoring The refactoring *Introduce Notification* replaces the `Toast` by a `Notification`. The result (only the refactored method) is depicted in Listing 2.

```
1   public void onCreate() {
2     super.onCreate();
3     Notification notification = new Notification.Builder(this).
          setContentText("Hello World!").build();
4     NotificationManager manager = (NotificationManager)
          getSystemService(Context.NOTIFICATION_SERVICE);
5     manager.notify(123, notification);
6   }
```

Listing 2: Notifying service.

With this resolution users can continue working without interruption since the message is only displayed in the notification area of the device.

We consider this modification a refactoring since the information to be displayed remains the same. Furthermore, Listing 3 illustrates the according IncQuery pattern as we implemented this quality smell structure. The green strings in dot notation denote concepts from our Java metamodel.

```
1  pattern interruptionFromBackground(ex:ExpressionStatement){
2    Class.~extends(actualClass, superClassRef);
3    NamespaceClassifierReference.classifierReferences(
         superClassRef, classifierReference);
4    ClassifierReference.target(classifierReference, superClass);
5    Class.name(superClass, "Service");
6    find startsToast(ex);
7    Class.members(actualClass, method);
8    find parentContainsChild+(method, ex);
9  }
10 private pattern startsToast(ex) {
11   ExpressionStatement.expression(ex, startToastMethod);
12   IdentifierReference.target.name(startToastMethod, "Toast");
13   IdentifierReference.next(startToastMethod, toastExpression);
14   MethodCall.target.name(toastExpression, "makeText");
15   MethodCall.next(toastExpression, showToastExpression);
16   MethodCall.target.name(showToastExpression, "show");
17 }
```

Listing 3: Interruption from background pattern.

4 Conclusion

In this paper we motivated the need for a quality smell catalogue. As a representative the quality smell *Interruption From Background* is presented. Our tool Refactory can be used to detect quality smells by either using IncQuery patterns to query models for certain structures, or by using metrics-based calculations. Furthermore it generates quick fixes for possible refactorings which then can be executed to resolve particular quality smells in each development phase.

References

[1] G. Bergmann, A. Hegedüs, A. Horváth, I. Ráth, Z. Ujhelyi, and D. Varró. Integrating Efficient Model Queries in State-of-the-Art EMF Tools. In *Objects, Models, Components, Patterns*. Springer, 2012.

[2] W. H. Brown, R. C. Malveau, and T. J. Mowbray. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*. Wiley, 1st edition, 1998.

[3] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.

[4] J. Reimann and U. Aßmann. Quality-Aware Refactoring For Early Detection And Resolution Of Energy Deficiencies. In *4th International Workshop on Green and Cloud Computing Management*, 2013.

[5] J. Reimann, M. Seifert, and U. Aßmann. On the reuse and recommendation of model refactoring specifications. *Software & Systems Modeling*, 12(3):579–596, 2013.

[2] http://www.jamopp.org

[3] http://developer.android.com/guide/practices/seamlessness.html#interrupt