Towards Variability-aware Smells

Unnecessary Annotation

DEFINITION:

`Unnecessary Annotation` smell occurs when a mandatory feature is annotated in the source code. Using conditional compilation mechanism only optional feature should be annotated in source code.

EXAMPLE:

Considering that Figure 7.1 shows a valid feature model, the annotation for the features *AStorage* (lines 1 and 3), *LLStorage* (lines 1 and 6), and *Locking* (lines 1 and 9) in Listing 7.2 represent `Unnecessary Annotation` because these three features are mandatory features in the Stack feature model shown in Figure 7.1.

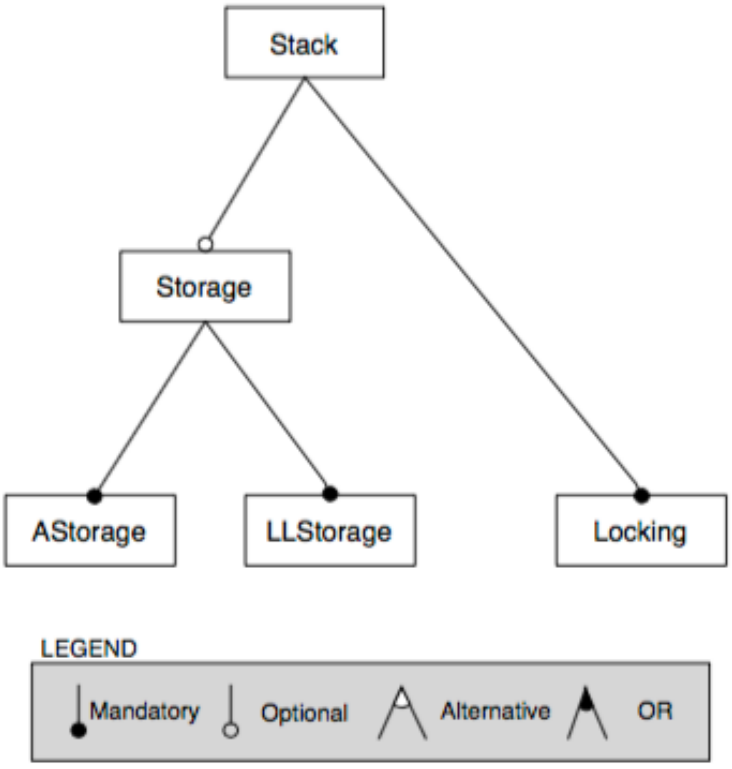

Figure 7.1 Bad mandatory child-feature example.

Listing 7.2 Redundant annotation example.

```
1  //#if ${AStorage} == "T" or ${LLStorage} == "T" or ${Locking} == "T"
2  class Stack <E> {
3   //#if ${AStorage} == "T"
4   List<E> store = new ArrayList<E>();
5  //#endif
6  //#if ${LLStorage} == "T"
7     List<E> store = new LinkedList<E>();
8  //#endif
9  //#if ${Locking} == "T"
10    public void push(E e, Lock lock) {
11     lock.lock();
12     store.add(e);
13     lock.unlock();
14   }
15    E pop(Lock lock) {
16     lock.lock();
17     try { return store.remove(store.size()−1); }
18      finally { lock.unlock(); }
19    }
20 //#endif
21 }
22 //#endif
```

PROBLEM:

`Unnecessary Annotation` smell can lead to exclude mandatory features from a product configuration and making this product configuration invalid. In annotation-based variability implementation through *`#ifdef directives`*, feature implementation annotation is a Boolean flag clause where it is possible to set true or false, respectively including or excluding features in product configurations. This smell affects the program comprehension and product derivation process.