

REFACTORING

of a long and complex legacy class



Introduction

This presentation is about the refactoring of a class called **CompositeTransformer**, which has a single complex method **transform** whose intent is to interpret all the data about the clinical path of a patient during his stay in hospital (read from an external API) and produce a snapshot in a certain moment in time (exposed on a public API).

This is legacy code:

- no tests
- no specifications
- no documentation
- one single developer/maintainer
- difficult to understand
- god method (>400 lines)

This code is in **PROD** since 2017 and has no known bugs

Step1 – Setup the Oracle Pattern

I want to keep the original implementation and work on a duplication of it

Before refactoring I need all the tests to ensure that the behaviour is not changing

1. Break external dependencies
2. Stub source data to be transformed (trying to cover every possible data variation)
3. Stub external data from other sources
4. Extend classes to make them comparable (string conversion, sorting, etc.)
5. Write characterization tests
6. Coverage <100% is acceptable if not covering external calls

```
@Test
public void produceSameJsonOfTheOldMethod_SingleDEComposite() {
    CompositeTransformer compositeTransformer = new CompositeTransformer();
    List<BoardComposite> transformedOld = compositeTransformer.transform(composite3734832, wardId, from, to, mar, false, now);
    String jsonOld = Serializer.toJson(transformedOld);

    CompositeTransformerNew compositeTransformerNew = new CompositeTransformerNew();
    List<BoardComposite> transformedNew = compositeTransformerNew.transform(composite3734832, wardId, from, to, mar, false, now);
    String jsonNew = Serializer.toJson(transformedNew);

    assertEquals(jsonNew, jsonOld);
}
```

Step 2 - Mutation coverage

Check coverage quality using Mutation testing

Benefits:

- ensure test coverage is solid with respect to code mutations
- integrate tests with possible missing test inputs



```
185 1 if (part.getRoleType().equals(ROLETYPE_WARD)) {
186 2   if (wardId != null && part.getRoleCode().equals(wardId) && planned == null) {
187     planned = new BoardParticipation(PARTICIPATIONTYPE_PLANNED);
188     planneds.add(planned);
189     planned.participationId = part.getCode();
190     //if (active==null) active = "planned";
191     planned.extSys = part.getExtSys();
192     planned.extSysFrom = part.getBeginDate().toString();
193     planned.extSysTo = part.getEndDate().toString();
194     planned.ward = part.getRoleLabel();
195     planned.wardId = part.getRoleCode();
196     planned.wardDateTime = part.getBeginDate();
197     planned.status = PARTICIPATIONSTATUS_CURRENT;
198   }
199 2 } else if (part.getRoleType().equals(ROLETYPE_ROOM) && part.getRoleCode() != null) { // non ci dovrebbe essere mai
200 1   if (planned == null) {
201     planned = new BoardParticipation(PARTICIPATIONTYPE_PLANNED);
202     planneds.add(planned);
203   }
204   planned.room = part.getRoleLabel();
205   planned.roomId = part.getRoleCode();
206 2 } else if (part.getRoleType().equals(ROLETYPE_BED) && part.getRoleCode() != null) { // non ci dovrebbe essere mai
207 1   if (planned == null) {
208     planned = new BoardParticipation(PARTICIPATIONTYPE_PLANNED);
209     planneds.add(planned);
210   }
211   planned.bed = part.getRoleLabel();
212   planned.bedId = part.getRoleCode();
213   planned.bedDateTime = part.getBeginDate();
214 }
215 2 } else if (part.getParticipationType().equals(TYPE_MEDICALCATEGORY) && part.getRoleType().equals(ROLETYPE_SERMED)) {
216   medicalCategory = new BoardMedicalCategory();
217   medicalCategory.code = part.getRoleCode();
218   medicalCategory.shortName = part.getRoleLabel();
```

Step 3 – Code smells

Working on the new duplicated implementation:

- identify code smells and annotate the code
- make simple changes that won't affect code integrity (e.g.: replace primitives with constants, rename variables)

```
CompositeTransformer.java 103 items
(36, 6) // TODO Code smell: long method + long parameter list
(42, 72) BoardParticipation primary = new BoardParticipation("primary"); // TODO Code smell: primitive obsession
(43, 76) BoardParticipation secondary = new BoardParticipation("secondary"); // TODO Code smell: primitive obsession
(64, 71) continue; // elimino quelle troppo vecchie o troppo future // TODO Code smell: comments
(65, 68) if (part.getParticipationType().equals("CONSULTATION")) { // TODO Code smell: primitive obsession
(66, 53) if (part.getRoleType().equals("WARD")) { // TODO Code smell: primitive obsession
(75, 66) if (!part.getParticipationType().equals("LOCALISATION")) // TODO Code smell: primitive obsession
(76, 77) && !part.getParticipationType().equals("SECONDARY_LOCALISATION") // TODO Code smell: primitive obsession
(77, 75) && !part.getParticipationType().equals("PLANNED_LOCALISATION") // TODO Code smell: primitive obsession
(78, 70) && !part.getParticipationType().equals("MEDICALCATEGORY") // TODO Code smell: primitive obsession
(79, 80) && !part.getParticipationType().equals("SECONDARY_MEDICALCATEGORY") // TODO Code smell: primitive obsession
(80, 61) && !part.getParticipationType().equals("LEAVE") // TODO Code smell: primitive obsession
(83, 113) if (part.getParticipationType().equals("PLANNED_LOCALISATION") && part.getCheckedPartCode() != null) { // TODO Code smell: primitive obsession
(88, 10) // TODO Code smell: comments
(89, 76) if (part.getParticipationType().equals("PLANNED_LOCALISATION")) // TODO Code smell: primitive obsession
(112, 10) // TODO Code smell: switch statements
(113, 68) if (part.getParticipationType().equals("LOCALISATION")) { // TODO Code smell: primitive obsession
(114, 53) if (part.getRoleType().equals("WARD")) { // TODO Code smell: primitive obsession
(116, 36) active = "primary"; // TODO Code smell: primitive obsession
(118, 56) primary.participationId = part.getCode(); // TODO Code smell: inappropriate intimacy
(122, 72) primary.status = part.isTransit() ? "future" : "current"; // TODO Code smell: primitive obsession
(123, 80) } else if (part.getRoleType().equals("ROOM") && part.getRoleCode() != null) { // TODO Code smell: primitive obsession
(126, 89) } else if (part.getRoleType().equals("BED") && part.getRoleCode() != null) { // TODO Code smell: primitive obsession
(132, 85) } else if (part.getParticipationType().equals("SECONDARY_LOCALISATION")) { // TODO Code smell: primitive obsession
(133, 53) if (part.getRoleType().equals("WARD")) { // TODO Code smell: primitive obsession
(135, 38) active = "secondary"; // TODO Code smell: primitive obsession
(136, 14) // TODO Code smell: duplicated code
(137, 57) secondary.participationId = part.getCode(); // TODO Code smell: inappropriate intimacy
(138, 74) secondary.status = part.isTransit() ? "future" : "current"; // TODO Code smell: primitive obsession
(142, 90) } else if (part.getRoleType().equals("ROOM") && part.getRoleCode() != null) { // TODO Code smell: primitive obsession
(143, 14) // TODO Code smell: duplicated code
(144, 51) secondary.room = part.getRoleLabel(); // TODO Code smell: inappropriate intimacy
(146, 89) } else if (part.getRoleType().equals("BED") && part.getRoleCode() != null) { // TODO Code smell: primitive obsession
(147, 50) secondary.bed = part.getRoleLabel(); // TODO Code smell: inappropriate intimacy
(152, 83) } else if (part.getParticipationType().equals("PLANNED_LOCALISATION")) { // TODO Code smell: primitive obsession
(153, 12) // TODO Code smell: comments
```

Step 4 – Class refactoring

The behaviour is protected by the tests, is now time to start the refactoring

- isolate bounded responsibilities
- extract methods or classes (using IDE)
- keep classes small and significant
(single responsibility principle)
- use speaking names

Iterate on these steps if necessary

Step 5 – Write unit tests

After the split of the big method in smaller classes and methods we can start writing unit tests.

Goals of this step are:

- write unit tests that cover each class functionality
- tests should help documenting the behaviour
- target 100% coverage

Final considerations

- ❖ **High coverage level is not enough for stay safe with a refactoring**
- ❖ **Mutation testing is a real game-changer, confidence rises up**
- ❖ **With the back covered, craftman enjoys also an hard renovation!**

Thank you!

