Software Engineering 2: Ofbiz Code Inspection

Andrea Pace, Lorenzo Petrangeli, Tommaso Paulon

February 5, 2017

Contents

1 Introduction

The purpose of this document is to highlight all the errors, the imprecisions overlooked during the implementation phase by sistematically inspect the code and verifying that the code is compliant with the checklist provided fo this phase of the project.

2 Assigned class

We have to perform code inspection on the EntityAutoEngine class, whose path is /apache-ofbiz-16.11.01/framework/service/src/main/java/org/apache/ofbiz/service/engine/EntityAutoE

We refer to the Code Inspection Assignment Task Description.pdf file for what concerns the issues to be found.

2.1 Functional role of the class

The EntityAutoEngine implements Create, Update, Delete and Expire operation depending on what is present in the parameters of the runSync method. According to the discussion found at `http://ofbiz.135035.n4.nabble.com/The-fancy-new-entity-auto-service-execution-engine-td197201.html`, this class is mainly an entity manager, these four operations check if the right conditions are met before they can actually create/delete/update the entity. No direct usages of this class can be found by our Java IDE, so we suppose that this class

is part of some kind of pattern and objects are generated at runtime. The lack of documentation/javadoc of this class allows for this kind of hypothesis, since the documentation is largely missing not only in this class but also in the class that is extended by our class. The only javadoc found is in the GenericEngine interface, at the top of the extend/implement chain, but it's not sufficient to clearly understand the role of our class. There are some comments within our class with some example of XML code: according to the discussion found at `http://ofbiz.135035.n4.nabble.com/entity-auto-td157361.html`, these are example of correct XML example to be used with respect to the different requests(create/delete/update/expire).

3 List of issues found by applying the checklist

We are going to report the issues found in the code following the checklist.

- *11. All if, while, do-while, try-catch, and for statements that have only one statement to execute are surrounded by curly braces*: lines 549, 552, 556, 558, 562, 567
- *13. Where practical, line length does not exceed 80 characters:* lines 55, 96, 97, 160, 162, 181, 226, 270, 276, 284, 293, 294, 295, 296, 381, 393, 462, 488, 495, 509, 526, 554, 556.
- *14. When line length must exceed 80 characters, it does NOT exceed 120 characters:* lines 53, 60(javadoc), 63, 68(javadoc), 71, 72, 79, 83, 88, 102, 115, 119, 123, 130, 141, 142, 148, 192, 231, 236, 238, 245, 249, 271, 273, 290, 291, 300, 335, 343, 345, 351, 367, 383, 386, 403, 413, 419, 426, 429, 430, 433, 437, 463, 469, 478, 490, 496, 500, 502, 519, 528, 530, 550, 558, 562.
- *18. Comments are used to adequately explain what the class, interface, methods, and blocks of code are doing:* the comments are unsufficient to explain what the class and the methods do. Line 47: not a complete or detailed description of the class. Line 181: comment used to highlight that the else branch is useless and should be deleted. Every method lacks proper documentation, while some block of code has little comments.
- *19. Commented out code contains a reason for being commented out and a date it can be removed from the source file if determined it is no longer needed:* line 391. Some commented XML code is present but it's not meant to be executed, it can be considered as a standard comment
- *23. Check that the javadoc is complete (i.e., it covers all classes and files part of the set of classes assigned to you)*: the methods on lines 148,367,478 haven't got any javadoc.
- *27. Check that the code is free of duplicates, long methods, big classes, breaking encapsulation, as well as if coupling and cohesion are adequate*:

invokeCreate (lines 148-375) is a very long method based only on the if/else construction. Even if its functionality is guaranteed, this method could have been written in a more elegant and clear way.

- *33. Declarations appear at the beginning of blocks (A block is any code surrounded by curly braces '{' and '}'). The exception is a variable can be declared in a for loop*:
  - **crudValue** (line 135) is not declared at the beginning of his block
  - **lookedUpValue** (line 236) is not declared at the beginning of his block
  - **fromDateField** (line 301) is not declared at the beginning of his block previous
  - **previousStatus** (line 351) is not declared at the beginning of his block
  - **lookedUpValue** (line 386) is not declared at the beginning of his block
  - variables declared on lines 400-402 are not at the beginning of the block
  - **lookedUpValue** (line 500) is not declared at the beginning of his block
- *41. Check that displayed output is free of spelling and grammatical errors:* **party** on line 296 probably is a spelling error and stands for **part**.
- *44. Check that the implementation avoids "brutish programming":* **invokeCreate** (lines 148-375) as written above is very long and the if/else construction is abused. The term *"brutish programming"* refers to "solutions where someone applies an obvious but excessive technique when more refined and effective alternatives are available"[1] and surely can be used to describe the implementation of this method.

4 Hours of work

- Andrea Pace: 5 hours
- Lorenzo Petrangeli: 3 hours
- Tommaso Paulon: 4 hours

References

[1] http://users.csc.calpoly.edu/~jdalbey/SWE/CodeSmells/bonehead.html