Politecnico di Milano

AA 2016-2017

Software Engineering 2 project: "*PowerEnJoy*"

Prof. Mottola Luca

CI

Code **I**nspection

05/02/17 - version 1.0

Edited by:

Giulia Pennati **878686**

Paola Sanfilippo **809689**

Marco Siracusa **878083**

Table of Contents

1. CLASSES ASSIGNED

Our group was assigned with two classes, both from Apache OFBiz:

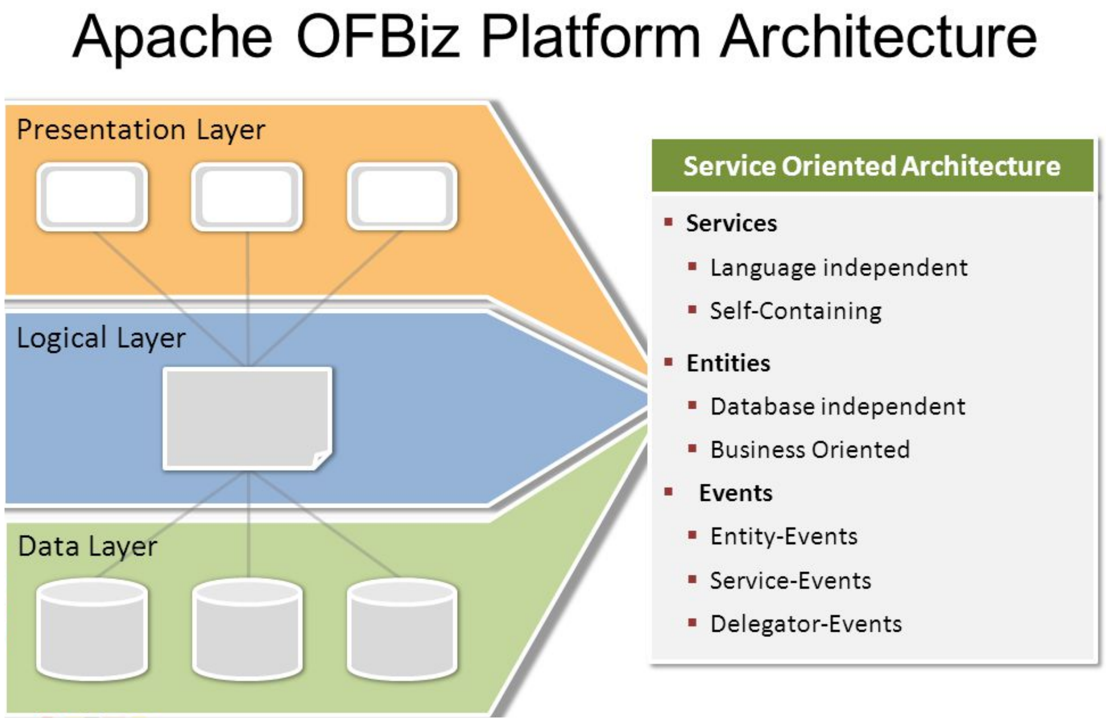
- OutputServices.java
  The path of OutputServices class is
  ../apache-OFBiz-16.11.01/applications/content/src/main/java/org/apache/OFBiz/content/output/OutputServices.java

- HtmlTreeRenderer.java
  The path of HtmlTreeRenderer class is ../apache-OFBiz-16.11.01/framework/widget/src/main/java/org/apache/OFBiz/widget/renderer/html/HtmlTreeRenderer.java

2. FUNCTIONAL ROLE OF ASSIGNED SET OF CLASSES

The following explanation of the functional role of our classes have been deduced by the available documentation and code reading. In particular, with the term documentation we don't mean only the Javadoc, as for our classes wasn't useful at all. To see in the detail the documents we referred to, please consult the Reference paragraph in this document.

Apache OFBiz is an open source product for the automation of enterprise processes. It is a 3-tiers framework (presentation layer, logical layer and data layer)



- OutputServices

This class deals with the output options available for the current page.

sendPrintFromScreen():
starting from the parameters we can find:

1. dctx of class DispatchContext (according to Javadoc: *This context contains the reference to each of the service definition files*)
2. A map called serviceContext

As for local variables, the "locale" variable identifies where the user is located throughout the world (according to the Javadoc *an operation that requires a Locale to perform its task is called locale-sensitive and uses the Locale to tailor information for the user*), and a series of string identifying better the location.
Then, in try-catch block, as said in the comment, the method generates the input and output streams, and then the generated print is ready to be printed.

Then the method creates the lookup (according to PrintServiceLookup Javadoc i*mplementations of this class provide lookup services for print services (typically equivalent to printers) of a particular type*) and search for available print services. If at least one is found, it's taken and the methods controls if the object to print is compatible with the DocFlavor (*Class DocFlavor encapsulates an object that specifies the format in which print data is supplied to a DocPrintJob*) and return error in case of problems or if there're not available printers.
Then, if all goes fine, there's the specification of the settings to apply to the print through the class PrintRequestAttributeSet (*The client uses a PrintRequestAttributeSet to specify the settings to be applied to a whole print job and to all the docs in the print job*), and then the method prints the document.

createFileFromScreen():
starting from the parameters we can find:

1. dctx of class DispatchContext (according to Javadoc: *This context contains the reference to each of the service definition files*)
2. A map called serviceContext

As for local variables, the "locale" variable identifies where the user is located throughout the world (according to the Javadoc *an operation that requires a Locale to perform its task is called locale-sensitive and uses the Locale to tailor information for the user*), a delegator and a series indicating the fileName, filePath and the contentType. Then, in try-catch block, as said in the comment, the method generates the input and output streams, and the Fop instance. Then, according to the contentType of the file, the method adds the file extension to the fileName and generates the new file in the given path or in the default one.

- HtmlTreeRenderer

As stated by the class' name, the main purpose is to render html trees, by building them according to the contextual parameters.

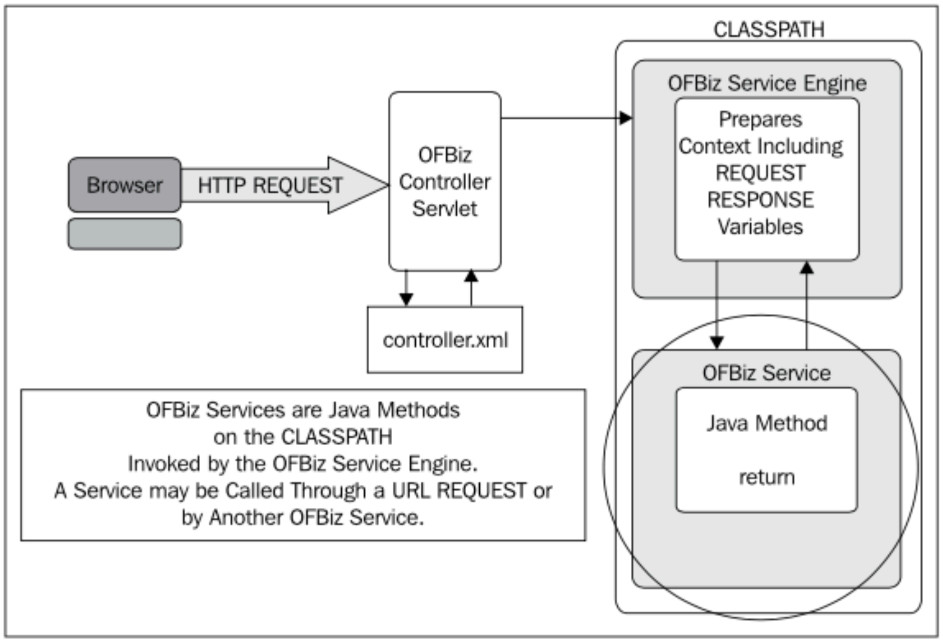
To analyse in the detail the HtmlTreeRenderer we need to start by defining some terms that are used frequently. At first let us say what do we mean by renderer: a **renderer** is what makes an object appear on the screen.
To define the term "**screen**" let us have a look at how is defined in external documentation:
"Screen widgets are part of the OFBiz Widget toolkit. They are the front line of the View element of the OFBiz VC architecture. Every View in OFBiz starts with or is contained in a screen, and every screen is defined by a screen widget".[1]
And also (from a different source):
"Users interact with OFBiz-that is the "User Interface"- through web pages often referred to within the project as "screens" or "screen views"."[2]

We want now to analyse the parameters of this class' methods.

[1] *J. Wong, R.Howell, "Apache OFBiz Development – The beginner's tutorial"*
[2] R. Hoffman, "Apache Ofbiz Cookbook"

The **renderNodeBegin**, **renderNodeEnd**, **renderLastElement**, **renderLabel**, **renderLink** and **renderImage** methods have in common some of the parameters:

- **Appendable writer:** writer is a parameter of Append datatype, i.e. an object to which char sequences and values can be appended
- **Map<String, Object> context**: context maps are initialized in the screen definition XML files and are passed to OFBiz Services:
  "*Services are independent pieces of logic which when placed together process many different types of business requirements. Services can be of many different types: Workflow, Rules, Java, SOAP, BeanShell, etc. A service with the type Java is much like an event where it is a static method, however with the Services Framework we do not limit to web based applications. Services require input parameters to be in a Map and the results are returned in a Map as well. This is nice since a Map can be serialized and stored or passed via HTTP (SOAP).*"[3]
  And
  "*Services are Java methods that are invoked by the OFBiz Service Engine as shown. Consumers of OFBiz Services may be web browsers, other OFBiz Events or Services, and even remote systems (for example SOAP-based messaging requests).*"[4]


CLASSPATH
OFBiz Service Engine
Prepares Context Including REQUEST RESPONSE Variables
Browser
HTTP REQUEST
OFBiz Controller Servlet
controller.xml
OFBiz Service
Java Method
return
OFBiz Services are Java Methods on the CLASSPATH Invoked by the OFBiz Service Engine. A Service may be Called Through a URL REQUEST or by Another OFBiz Service.

- And another parameter that changes according to the method's role (it may be a node, label, link or image)

Consider now all the methods:

- An empty constructor, **HtmlTreeRenderer**
- **renderNodeBegin**, **renderNodeEnd** and **renderLastElement** are methods used to build up the correct hierarchy of the tree, by managing the various initial nodes in the renderNodeBegin

[3] https://cwiki.apache.org/confluence/display/OFBIZ/Service+Engine+Guide
[4] R. Hoffman, "Apache Ofbiz Cookbook"

(also consider the expansions and collision of the tree and the possible presence of children nodes); on the other hand the renderNodeEnd takes care of ending correctly the html lists.

- **getScreenStringRenderer**, that, taken as parameter the context map, returns a ScreenStringRenderer according to the ScreenRenderer

Then we have 3 render

- **renderLabel**, used to render labels
- **renderLink**, used to compose and render hyperlinks
- **renderImage**, used to render images. At the beginning, the method appends the opening image tag through the writer in order to *decorate the screen*, and then appends all the image attribute (as id, width…) checking first if they're not null, creating in this way the HTML complete tag for the image.

3. LIST OF ISSUES FOUND

3.1. Naming conventions

1. **All class names, interface names, method names, class variables, method variables, and constants used should have meaningful names and do what the name suggests.**

Checking for meaningful names (also according to their role and goal):

-class names	OK
-interface names	OK
-method names	OK
-class variables	OK
-method variables	OK (sometimes a little bit too short but limited in a small scope then no source of issues)
-constants	OK

2. **If one-character variables are used, they are used only for temporary "throwaway" variables, such as those used in for loops.**

No one-char variables have been used but short variables (2/3 chars) do, anyway these are related to a small scope or a well define context therefore not source of problems

3. **Class names are nouns, in mixed case, with the first letter of each word in capitalized.**

Class names are nouns, in mixed case, with the first letter of each word in capitalized

4. **Interface names should be capitalized like classes**

No interfaces have been implemented, only one used for the htmlTreeRenderer class that has been named properly

5. **Method names should be verbs, with the first letter of each addition word capitalized.**

Method are verbs, with the first letter of each additional word capitalized

6. **Class variables, also called attributes, are mixed case, but might begin with an underscore (' ') followed by a lowercase first letter. All the remaining words in the variable name have their first letter capitalized.**

Class variables have been named properly

7. **Constants are declared using all uppercase with words separated by an underscore**

Both classes do not have the constant variables (considered as the static final ones) properly declared (supposed to be written in uppercase chars with words separated by underscores); in particular:

In OutputServices:
- module, line 76
- foFormRenderer, line 78
- resource, line 79

In HtmlTreeRenderer:
- module, line 52

3.2. Indention

8. **Three or four spaces are used for indentation and done so consistently.**

- OutputServices, 107: any indentation style (convention) has not been defined for the carriage return
- OutputServices, 219: any indentation style (convention) has not been defined for the carriage return
- htmlTreeRenderer, 228: any indentation style (convention) has not been defined for the carriage return
- htmlTreeRenderer, 343: any indentation style (convention) has not been defined for the carriage return

but coherent in the end, always 4 spaces as indentation

9. **No tabs are used to indent.**

No tabs found

3.3. Braces

10. **Consistent bracing style is used, either the preferred "Allman" style (first brace goes underneath the opening block) or the "Kernighan and Ritchie" style (first brace is on the same line of the instruction that opens the new block).**

Only the Kernighan and Ritchie braces convention has been used

11. **All if, while, do-while, try-catch, and for statements that have only one statement to execute are surrounded by curly braces.**

All "if, while, for ..." with only a statement have open and close braces surrounding that.

3.4. File Organization

12. Blank lines and optional comments are used to separate sections (be- ginning comments, package/import statements, class/interface declarations which include class variable/attributes declarations, constructors, and methods).

Blank lines and optional comments are used to separate sections:

-beginning comments	not spaced from the package statement in both classes
-package/import statements	not spaced from the beginning comment in both classes
-class/interface declarations which include:	well spaced with the rest
- class variable/attributes declarations	well spaced with the rest but not among them
- constructors	well spaced with the rest
-methods	well spaced with the rest

13. Where practical, line length does not exceed 80 characters.

Not respected (many lines over 80 chars, even where not so necessary)

14. When line length must exceed 80 characters, it does NOT exceed 120 characters.

Not respected (many lines over 120 chars)

3.5. Wrapping Lines

15. Line break occurs after a comma or an operator.

Line breaks are always placed after the "{ } , ;" symbols but in the HtmlTreeRenderer (342nd row) there is a comma followed by a space and a line break

16. Higher-level breaks are used.

In OutputServices, 218th row, the break splits the list of parameters of a function invocation.
In HtmlTreeRenderer, 342nd row, the break splits the list of parameters of a function invocation.

17. A new statement is aligned with the beginning of the expression at the same level as the previous line.

The two broken lines are properly indented, their second lines cannot be aligned (as the convention says) to the beginning of the first expression because it would be too at the far right therefore an 8-space indentation is applied
(See http://www.oracle.com/technetwork/java/javase/documentation/codeconventions-136091.html)

3.6. Comments

18. Comments are used to adequately explain what the class, interface, methods, and blocks of code are doing.

OutputServices
There's no explanation about the function of the class, there's only a line with the class' name.
The same for the methods, no lines are found about method functionality.
As for blocks of code we can find various explanation, but they're very short

HtmlTreeRenderer
There's a no explanation about the functionality of the class.
There's not any line found before methods' declaration, and also the codeblocks are explained in a careless way.
Issues found in lines 85, 92, 106, 121.

In the end, the classes are commented not that much through the code, therefore this point could not be consider properly fulfilled; furthermore, some comments are not so understandable (on whether they are TO-DO or else)

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.

Only the initial license agreement is an out of code comment (in both classes)

3.7. Java Source Files

20. Each Java source file contains a single public class or interface.

Each Java source file contains a single public class

21. The public class is the first class or interface in the file.

The public classes are the first classes in the files

22. Check that the external program interfaces are implemented consistently with what is described in the javadoc.

In both the classes the Javadoc is missing and the documentation on the web is absent of comments (only the class structure, fields, methods, interfaces, and tree are specified).
Furthermore:

- In the OutputServices class the declarations of the class fields are more specified than the documentation on the web, however the methods are fine despite of the constructor that is missing in the class

- In the HtmlTreeRenderer class the methods are declared coherently with the documentation on the web

23. Check that the javadoc is complete

The Javadoc is missing

3.8. Package and Import Statements

24. If any package statements are needed, they should be the first non- comment statements. Import statements follow.

The package statements are the first non-comment statements and the import statements follow.

3.9. Class and Interface Declarations

25. The class or interface declarations shall be in the following order:

a. class/interface documentation comment;
b. class or interface statement;
c. class/interface implementation comment, if necessary;
d. class (static) variables;
  i. first public class variables;
  ii. next protected class variables;
  iii. next package level (no access modifier);
  iv. last private class variables.
e. instance variables;
  i. first public instance variables;
  ii. next protected instance variables;
  iii. next package level (no access modifier);
  iv. last private instance variables.
f. constructors;
g. methods.

Here a list of the required points with the needed comments:

a. Follows the order
b. Follows the order
c. Follows the order
d. The OutputServices class does not follow this ordering convention and neither does the HtmlTreeRenderer
e. The OutputServices class does not have any instance variable and the HtmlTreeRenderer class does not follow the order (instance class declared first)

f. The OutputServices class does not have any constructor and the HtmlTreeRenderer have the constructor first
g. The methods are declared at last

26. Methods are grouped by functionality rather than by scope or accessibility.

The methods of both classes are actually grouped by functionality

27. Check that the code is free of duplicates, long methods, big classes, breaking encapsulation, as well as if coupling and cohesion are adequate.

The code happens to be free of:

-duplicates	Yes
-long methods	Yes
-big classes	Yes
-breaking encapsulation	Yes
-inadequate coupling and cohesion	Yes

3.10. Initialization and Declarations

28. Check that variables and class members are of the correct type. Check that they have the right visibility (public/private/protected).

Variables and class members are of the correct type, visibility included (coherent with the JavaDoc too)

29. Check that variables are declared in the proper scope.

It would be in contrast with the 33rd point of the checklist but:
- OutputServices, 83: could have been declared in a deeper scope
- OutputServices, 198: could have been declared in a deeper scope
- OutputServices, 201: could have been declared in a deeper scope
- OutputServices, 202: could have been declared in a deeper scope
- HtmlTreeRenderer, 57: could have been declared in a deeper scope
- HtmlTreeRenderer, 58: could have been declared in a deeper scope

30. Check that constructors are called when a new object is desired.

Beside few lines concerning some framework's elements, constructor is always called when an object is instantiated (See line 142 OutputServices, PrintService printer = null;)

31. Check that all object references are initialized before use.

All object references are initialized before used (See: line 142 in OutputServices, PrintService printer = null; not so straightforward since are usually retrieved from the "context")

32. Variables are initialized where they are declared, unless dependent upon a computation.

Variables are always initialized where declared.

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.

Both OutputServices class and HtmlTreeRenderer class do not follow this convention (e.g. OutputServices, 140, and HtmlTreeRenderer, 102)

3.11. Method Calls

34. Check that parameters are presented in the correct order.

According to the documentation on the web (not the JavaDoc), for each method call all the parameters are presented in the correct order.

35. Check that the correct method is being called, or should it be a different method with a similar name.

According to the documentation on the web (not the JavaDoc), for each method call the correct method is being called

36. Check that method returned values are used properly.

According to the documentation on the web (not the JavaDoc), for each method call the returned values are used properly.

3.12. Arrays

37. Check that there are no off-by-one errors in array indexing (that is, all required array elements are correctly accessed through the index).

The read collections are

- List that are accessed in a safe way everywhere (iterating through all the elements or using the get method)
- Map that are accessed by the string element identifier

38. Check that all array (or other collection) indexes have been prevented from going out-of-bounds.

In HtmlTreeRenderer, 107th row, no checks are done on whether the length of the list is greater than zero since the last element would be removed

39. Check that constructors are called when a new array item is desired.

The list elements are pushed implicitly

3.13. Object Comparison

40. Check that all objects (including Strings) are compared with equals and not with ==.

In the OutputServices class:

- the == operator is used in the lines 157, 172
- the != operator is used in the line 167

In the HtmlTreeRenderer class:

- the == operator is used in the lines 88, 109, 259, 340
- the != operator is used in the lines 124, 225, 226, 229, 234, 239, 310, 311, 319, 320, 337

3.14. Output Format

41. Check that displayed output is free of spelling and grammatical errors.

In the OutputServices class the only textual output is done by a Debug class used for info or messages (in the lines 135, 154, 181, 188, 256).
In the HtmlTreeRenderer class there isn't any output beside the "Appendable writer" that parses the HTML language where it is fine.

42. Check that error messages are comprehensive and provide guidance as to how to correct the problem.

In the OutputServices class, the errors in line 188 and 256 have been attached only with the exception caught and not there's not how to overcome that

43. Check that the output is formatted correctly in terms of line stepping and spacing.

The output is formatted correctly in terms of line stepping and spacing.

3.15. Computation, Comparisons and Assignments

44. Check that the implementation avoids "brutish programming".

The code avoids "brutish programming" as much as possible.
Therefore, it could be said that the code is:

- rather readable
- efficiency not decreased by any code fragment
- as thin as possible
- easy to modify (as far as the framework allows)
- enough reusable where possible

45. Check order of computation/evaluation, operator precedence and parenthesizing.

The arithmetical method invocation outlines are fulfilled since no particular computations have been coded and the evaluations have been written correctly.

46. Check the liberal use of parenthesis is used to avoid operator precedence problems.

There is no need to impose an order over some operations

47. Check that all denominators of a division are prevented from being zero.

No explicit divisions have been done

48. Check that integer arithmetic, especially division, are used appropriately to avoid causing unexpected truncation/rounding.

Since no divisions have been carried out and no numbers with decimal part have been defined this point can be considered as fulfilled

49. Check that the comparison and Boolean operators are correct.

The comparisons and booleans usage is correct (booleans are checked directly into the "if" or passed to methods)

50. Check throw-catch expressions, and check that the error condition is actually legitimate.

The OutputServices class catches the general "Exception e" exception only
The HtmlTreeRenderer class catches the "IOException" that being pretty general includes all the main possible failing cases (actually it also catches the "TemplateException")

51. Check that the code is free of any implicit type conversions.

A great amount of variable casting is used but always well managed

3.16. Exceptions

52. Check that the relevant exceptions are caught.

In the OutputServices class only the generic "Exception e" is caught
In the HtmlTreeRenderer class TemplateException and IOException are caught

53. Check that the appropriate action are taken for each catch block.

In the OutputServices class, when the "Exception e" is caught, an error is notified
In the HtmlTreeRenderer class, when the exceptions are caught, the stack trace is printed
This handling could be adequate considering the goal of the classes

3.17. Flow of Control

54. In a switch statement, check that all cases are addressed by break or return.

There are no switch statements

55. Check that all switch statements have a default branch.

There are no switch statements

56. Check that all loops are correctly formed, with the appropriate initialization, increment and termination expressions.

Loops:

-for	Ok	(two for loops in the OutputServices class)
-while	Ok	(no cycles of this kind)
-do-while	Ok	(no cycles of this kind)

3.18. Files

57. Check that all files are properly declared and opened.

The OutputServices class is the only one with File objects (lines from 249 to 253) where this point is fulfilled

58. Check that all files are closed properly, even in the case of an error.

The OutputServices class is the only one with File objects (lines from 249 to 253) where the file stream is properly close in the normal flow of the program but anything is done in case of error

59. Check that EOF conditions are detected and handled correctly.

The OutputServices class is the only one with File objects (lines from 249 to 253) but no EOF should be managed

60. Check that all file exceptions are caught and dealt with accordingly.

A general "Exception e" is caught and notified in both classes but nothing is done to recover the issue. The HtmlTreeRenderer class is a little better managed since each method has the properly IOException catch and also some custom one.

4. OTHER PROBLEMS FOUND

4.1. HtmlTreeRenderer problems

- Useless if (line 90): expandeCollapseLink cannot be null because it is initialized in every case (in the if statement beginning at line 102 it is initialized at line 102, and in the else statement(104) it is initialized at line 120.

- Parameters not checked: in every method there isn't any type of check about null parameters, and the javadoc doesn't provide any type of requirement, so null parameters are allowed, and they can cause bugs.

-Useless variables(lines 307, 308, 309): local variables declared at these lines are never modified by the method execution, so, in the method call at line 316, the developer may give directly three "false" as parameters.

4.2. OutputServices problems

- Parameters not checked: in every method there isn't any type of check about null parameters, and the javadoc doesn't provide any type of requirement, so null parameters are allowed, and they can cause bugs.

5. REFERENCES

- https://OFBiz.apache.org/documentation.html
- J. Wong, R.Howell, “Apache OFBiz Development – The beginner’s tutorial”
- R. Hoffman, “Apache Ofbiz Cookbook”
- https://cwiki.apache.org/confluence/display/OFBIZ/Service+Engine+Guide

6. TIME EFFORT

Each team member spent about 19 hours on the completion of this assignment.