**CODE INSPECTION**

**Slide 1:** *[image = waterfall model]*

* Introductory explanation - “This phase was carried out for a real open-source system (which is completely different the one we are describing here), but still was part of the same project assignment.”

**Slide 2:** *[image = OFBiz logo? Not necessary but...]*

* Target of analysis - “The inspection covered a single relevant class within a module of the complex system offered by *Apache OFBiz,* an open-source ERP + CRM.”
* Details of target class - “In particular, the target of the inspection process was the **'ModelDataFileReader.java'** class within the **'datafile'** package. The class is in charge of taking care of the conversion of data files represented in XML format into an object-based representation, in order for them to be used in the Java environment provided by OFBiz.”

**Slide 3:**

* Analysis procedure - “The process for accomplishing the task was split in two phases: analysis and understanding of the *role and functioning of the class* and analysis of the real code *in order to find issues based on a given checklist.*”
* Main issues - “Most of the issues in the code are convention violations for code structure (parenthesizing, line length, capitalization...).”
* Secondary issues - “Many of them involve incompleteness or total lackness of proper documentation. A minority of the issues is related to code and naming cleanness and clarity.”
* External potential issues - “The analysis of the class exposes some potential issues with other classes of the package, that were listed at the end for completeness.”

**TESTING**

**Slide 1:** *[image = waterfall model]*

* Establishing order of integration for components and subsystems.
* Defining necessary structures for each test case.
* Defining suitable procedures to perform efficient tests.

**Slide 2:**

* Entry criteria - “Some preconditions for this phase:

*Documentation* – Classes and methods must be clearly described wrt what they should do and how they should do so

*Unit tests* – Individual components must work properly wrt their internal functions

*Code inspection/Analysis* – Code structure must be compliant to the standard conventions, program flows and data flows must not contain useless or unreachable segments

*RASD and DD* – Clear indications about requirements and architectural design must be given, minor changes must be immediately reflected if they affect integration

* Approach - “The chosen approach is the *bottom-up integration* with focus on *critical modules first;* the criteria correspond to the natural order induced by the architectural choices described in the DD: modularity of components, hierarchical structure of interactions...”

**Slide 3:** *[image = integration of DB]*

* The integration starts from the database components and their interfaces: these represent the most independent components of the system, upon which most logical components rely.
* The interface exposed by the DMBS must be tested by using drivers that simulate access to every resource.
* The logical representation of DB entities must be tested in turn, by calling every offered method (by means of an appropriate set of drivers)

**Slide 4:** *[image = integration of user/utilities]*

* Similarly, we start from the most independent component within each logical group (container).
* In this case, the integration starts from the NotificationManager, whose exposed interface methods are called by one or more drivers to simulate the components “using” it.
* As decided in the architecture, the set of drivers will be substituted gradually with real components, that will need other (existing or new) drivers to be integrated as well.

**Slide 5:** *[images = integration of payment management + ride&reservation]*

* Same for the other logical groups.
* Order of integration always depends on the natural dependencies induced by architectural choices.

**Slide 6:** *[image = integration of upper levels]*

* In a simpler way, all containers must be integrated last, using drivers that simulate requests for bean instances.

**Slide 7:** *[images = integration of clients]*

* Since all clients have a similar “controller-centric” structure, integration must be done individually for each component, since the controller manages the interaction internally.

**Slide 8:** *[image =subsystems]*

* Subsystems must be integrated starting from the most critical part; the Application Server, which represents the logical coordination of the supersystem, naturally occupies a central position.
* Note the explicit need to integrate with the subsystems.

**Slide 9:** *[image = example table???]*

* All test cases follow the same structure:
  + *Identifier*
  + *Test items –* Involved components
  + *Input specification –* Procedure to be performed to test the integration
  + *Output specification –* Expected outcome of the test
  + *Environmental needs –* How the context in which the test is performed should be
  + *Test description –* Purpose and content of the test case
  + *Testing method –* Used tools/methodologies