|  |  |
| --- | --- |
| **Item(s) to Review** | Administrator |
| **Description of Item(s), Background and Context** | The Administrator module of the project Online Student Attendance System is the important module which implements the basic functions of Deleting existing user, modifying User Details, add new user, send email notifications uploading the various study materials. |
| **Objectives of Review** | The sole objective of the review is to correct the incorrect functionality and if any of functions are left to be completed must be completed. However code reviews have the following objectives:   * Quality, defect-free software * Systems that are appropriate and complete meeting requirements * Software that complies with enterprise coding standards |
| **Owner(s) of Item(s)** | Administrator (Tanushree Upreti, Manu Rastogi, Mohit Singh) |
| **Who is Impacted? Who Depends on these Items?** | Faculty, Student and Parent modules are directly impacted by the Administrator, as the Administrator has the privilege to remove any of the above stated user and modify their details. |
| **Reviewed by** | Tanushree Upreti, Manu Rastogi, Mohit Singh |
| **Next Steps** | The next step to be taken after review is to enter into maintenance phase. |

# **Logical Design and Layering Approach**

The existing business system although provides online support but fails to provide online study material to the students. Moreover, the materials often provided doesn’t contain total solutions to the problems of the students. There is no provision through which teachers can upload assignments that can be downloaded by the students. There is no process for parents' to view their child's attendance and to check the student feedback given by the teachers. The existing system does not provide an interface for parents/guardians and faculty to communicate. It also fails to provide email notifications whenever an important announcement is made.

These limitations have been improved in the developed system **Online Student Attendance System** and it has more features to improve on these problems, providing a one stop solution.

# **Maintainability, Adaptability**

*Assess factors that will impact the ease with which the software may be maintained or altered over time.*

|  |  |  |
| --- | --- | --- |
| **Area of Review** | **Observations** | **Suggestions** |
| Duplicated Code | The code was found duplicated in Admin class where it is reused for implementing various functions such as deleteUser(), login(), uploadMaterials() and updateDetails().  The code is reused as it allows us to save effort and to redesign the solution again and again. | *Whenever you see code that looks like it was copied and pasted, even with a few minor changes, explore opportunities for reuse. You might consider implementing classes that encapsulate the logic, or creating helper classes.* |
| Use of Short Methods | Short methods are used in Admin class for implementing functions such as update(), updateLatesttutor(), updateJobboard(), updateNews(), etc. This allows us to easily maintain these codes. | *Try to keep all methods short. Long methods become very hard to maintain over time* |
| Variable Scoping | Since using Advanced Java as the platform for development, thus no global variables had been used. All the variables declared are local to the function where they are declared. | *Try to keep variables scoped to the lowest level possible. Global variables should be avoided.* |
| [Cohesion](http://en.wikipedia.org/wiki/Cohesion_%28computer_science%29) of Logic in Classes | The functions used in the Admin class have their functionality purely independent of the other and each uses unique parameters to implement the functionality, which are to be supplied by the human via the interface, and not by any other function. | *Do everything you can to encourage high cohesion in all classes.* |
| Coupling: Long Parameter Lists | Short parameter lists had been used to implement the designed functionality. This makes the code easy to maintain in future. | *Generally, methods with long parameter lists create higher degrees of coupling and therefore decrease maintainability* |
| Coupling: Control Coupling | Control coupling has been implemented in the form of flags used which return the assigned value when the function is called. Based upon these values the decision is taken whether the function has executed or not. It has been implemented in Admin class in functions such as deleteUser(), login(), modifyHome(), updateUser(), updateLatesttutor(), updateJobboard(), updateNews(), uploadMaterials(). | *If you use “control flags” to drive the internal behavior of a method, explore opportunities for specialized classes or overloaded methods* |
| Coupling: Global Data Coupling | The session variable is used as a type of global variable which stores the username of the Administrator. The username is displayed on the home page of the Administrator. | *This typically occurs when global variables are used to drive the behavior of a group of classes* |
| Coupling: Solution Sprawl Across Classes | The Administrator can delete the user from the system thus this will affect the modules of Tutor, Parent and Student. Also no functionality has been sprawled across other classes and is limited to a single class where the function is implemented. | *When you need to make changes across a large number of classes in order to implement a change in the application’s behavior, you’ve got solution sprawl.* |
| Coupling: Inter-Layer Dependencies | The Admin class has a large number of Inter-Layer Dependencies as initially the values are received from the interface to the AdminPage class which is a .jsp page and then these values are used to call the functions in the Admin class. | *The more method calls you have from one class in a given assembly or “logical layer” to other classes in a different assembly or “logical layer”, the tighter the coupling, and the harder it will be to maintain over time.* |
| Conditional Complexity, Level of Nesting, Use of Flags, use of switch statements | No deeply nested “If” statements have been used in implementing the logic. The Logic is directly implemented by defining each in a separate function for independent functionality. | *Consider applying the “*[*Extract Method*](http://refactoring.com/catalog/extractMethod.html)*” refactoring to move code from within an “If Block” to a method that describes what that code does.*  *Whenever you see switch statements or “If” statements in a class, you might have an opportunity to use class specializations instead.*  *If a method has deeply nested “If” statements, uses flags (e.g. Booleans, etc.) to drive logic, it can become very difficult to read and maintain* |
| [Encapsulation](http://en.wikipedia.org/wiki/Information_hiding), Information Hiding, Inappropriate Intimacy between Classes | Since there is only a single class Admin which is used to drive the logic of the Administrator functionality and a .jsp page comprising of a AdminPage class which is used to call the functions of the Admin class, however it doesn’t knows much about the Admin class. | *When classes know too much about the internals of each other, they become very tightly coupled and hard to maintain.* |
| Magic Numbers and Literals | Not Applicable. | *Try to replace magic numbers and literals with constants that have meaningful names. Magic Numbers and literals are numeric or alpha-numeric values in the code whose meaning may not be self-explanatory.* |
| Speculative Generality | Every functionality designed has been used in the solution. | *A.K.A.* [*You Aren’t Going to Need It*](http://www.extremeprogramming.org/stories/simple2.html) |
| Versioning Approach | The logic of the module has been programmed by using Servlet and JSP page, further versioning can be implemented by using Struts, as a logic of implementation leading to advanced version. | *Has the developer produced an approach that may be easily versioned over time?* |
| Use of Interfaces | The interfaces are reused leading to high reusability of the code. The interfaces are simple, user-friendly and full of instructions of how the operation is done. | *Are interfaces used appropriately?* |
| Simplicity of Solution | The designed solution is simple and user-friendly. |  |
| General readability and intuitive naming of fields, properties, variables, methods, etc. | The name of variables, methods used in the module have an appropriate name applying the same logic as to be used in the application of module. |  |
| [Appropriate Use of Comments](http://www.designpatternsfor.net/default.aspx?pid=24) | The inline comments have been used in the module. |  |
| General adherence to Microsoft Coding Standards |  |  |
| Unit-Tests were created to support [regression testing](http://en.wikipedia.org/wiki/Regression_testing) | The various unit tests for the Administrator module are implemented in the test case document and tested and is found working correct against these tests. |  |

# **Robustness**

*Assess the primary factors that affect how well the software handles incorrect data or unforeseen scenarios.*

|  |  |  |
| --- | --- | --- |
| **Area of Review** | **Observations** | **Suggestions** |
| Defensive Programming | The values received through the interface of the module are checked that they may be entered into the table for insertion, and for modification that the value for the modification already exist in the module. | *Performs early parameter checking (e.g. boundary checks, type checks, assertions, etc.) before executing main body of logic*  *Checks return values received from service or method calls*  *Checks for nulls when appropriate*  *Avoids “Apocalypse Ready” designs; These are designs that handle exceptions that will probably never happen or “fringe case” issues* |
| Proper use of Exception Handling | The try-catch statement has been used to implement the Exception Handling. In order to avoid errors t be handled by the programmer the exceptions are thrown to Java which handles these exceptions and displays the appropriate error message.  The use of try-catch enables the system to keep working even when one of its methods do not function correctly. | *All exceptions are caught and handled at the “top of call-stack”*  *Lower in the call stack, exceptions are only caught to log or gather information, add information to the exception, perform cleanup, or attempt to recover*  *Prefer the use of standard framework-defined exceptions when possible*  *Exceptions are thrown only for clearly abnormal cases; Exceptions aren’t used to control application flow.* |
| Exceptions are Logged to Facilitate Debugging | The thrown exceptions may be used by the developer to repair and maintain the code accordingly. |  |
| Parameters are strongly typed | The parameters used in the Student module had been declared beforehand and are all defined data type of Java and we know Java is strongly typed. |  |

# **Performance**

*Assess the areas that will typically have the greatest impact on application performance.*

|  |  |  |
| --- | --- | --- |
| **Area of Review** | **Observations** | **Suggestions** |
| Style of Communication | The query or object retrieves only that much information from the database which is asked in the logic or requested by the user; the view records function of the interface displays only the needed records of the user to the Administrator hiding the confidential information such as username and password of the user.  The code also has minimal cross-process class and finishes a logic in one function only. | *Code has minimal cross-machine calls*  *Code has minimal cross-process calls*  *Code favors chunky vs. chatty communications to services*  *Avoidance of “Data Buffet” anti-pattern; This occurs when a query or object retrieves more data than it’s consumer will probably use* |
| Evaluate use of [Boxing / Unboxing](http://msdn.microsoft.com/msdnmag/issues/1200/dotnet/) | Conversion from value type to object type and then again back to value type has been not implemented in the module. | *Avoid conversion to/from value and reference types where possible* |
| Loop considerations | The loops terminate as soon as the logic has been met, the expression are evaluated after the loop has terminated to store the final value only.  Logic getting the same results are not used within the body of the loop. | *Loops are exited as soon as conditions met*  *Expressions are not re-evaluated from within the loop controller statements*  *Logic that always gets same results does not occur within the body of loops*  *Use “For” instead of “For Each” when appropriate* |
| String Handling | Not string concatenation and StringBuilder approaches are used as no appending i.e to join or add on to the end of something is done. | *Assess string concatenation approaches*  *Assess use of StringBuilder* |
| Resource Cleanup | The imports used in the module are essential for implementing the logic and are acquired late and released early.  The finalizers are not used in the module to avoid function overlapping i.e., whether the logic has executed or not the statements in the final block are implemented. | *Resources are acquired late and released early*  *Assess potential for* [*Generation 2 garbage collection*](http://msdn2.microsoft.com/en-us/library/ms973837.aspx)  *Evaluate appropriate use of “using” statement*  *The* [*Dispose pattern*](http://msdn2.microsoft.com/en-us/library/b1yfkh5e(VS.71).aspx) *is implemented for managed resources*  *Finalizers are avoided* |
| Appropriate Use of Caching | Since in Admin module each value is a different one thus nothing is cached and is not used frequently.  Every value used in the function is unique and is implemented only when the values are supplied via the interface. | *Items that change frequently are not cached*  *Know your cache-hit ratio; don’t bother caching items that aren’t retrieved frequently* |
| Appropriate use of ViewState and Postback checking | We have not stored the ViewState on the server side in a session and then passing the viewstate id to the client side via a hidden field. However when a user signs in,his session is maintained and when he logs out the session expires and he cannot roll back. |  |
| Consider opportunities for Asynchronous or Queued Operations | No need was found to perform the Queued operations. |  |
| Solution makes minimal or no use of Reflection | No use of reflection is done. |  |
| Use of Code Instrumentation | No use of Code Instrumentation. |  |
| [Premature Optimization](http://en.wikipedia.org/wiki/Optimization_(computer_science))? | Premature optimization means if the value of core product is weak, doubling the percentage of users will not help much and it will hurt as every unit of effort put into optimization is one less unit that can be put into improving core product. We will not begin optimizing until users say that they are very disappointed with the product. Hence there is no premature optimization. |  |

# **Supporting Documentation**

*Assess the design solution’s need for supporting documentation*

|  |  |  |
| --- | --- | --- |
| **Area of Review** | **Observations** | **Suggestions** |
| Object Models | The Object Models are not implemented in this module. |  |
| Sequence Diagrams | The sequence diagram has not been implemented as the developed solution is a web-application and to define the timeline for a particular user is very difficult. |  |
| Entity-Relationship Diagrams | The ER diagram used to depict the Administrator module proved very useful as most of the functioning has been clearly stated in it and during coding proved very helpful in deciding the logic of the code. |  |
| Other Diagram | The class diagram also proved to be very useful in implementing the logic as all the name of functions, their return type, their variable names had been decided earlier; however the variables used to hold the values supplied through the interface has not been declared and defined in this diagram |  |
| Use of [nDoc](http://sourceforge.net/projects/ndoc/) or [XML Comments](http://msdn.microsoft.com/msdnmag/issues/02/06/XMLC/) | No use of XML comments. But comments are provided to make a clear understanding in Student module. |  |
| Where will Documentation be Stored? How will it be Maintained? | The various templates holds the documentation and can be maintained easily. |  |

# **Migration Considerations**

*Assess how current or future users (i.e. developers) dependent on this solution will migrate to the proposed approach. How will the risk of breaking changes be minimized?*

The sole aim of Academicbuddy.com is to find a good and reliable tutor for the students and applied to parents for their wards as to find a good and reliable tutor in contemporary world is a cumbersome task. It also helps the tutors to find good money and students. The solution can be used by the Coaching Institutes, users from remote locations etc. The solutions originally developed under Advanced Java framework provides a base and can be migrated to other platforms by breaking this project module-wise.

* The individual source code files which were written in pure Java have to be rewritten in the proposed environment language.
* The ojdbc Jar files has to be suitably replaced by the libraries of the proposed new environment.
* The HTML, CSS and JavaScript files hardly will need any changes because they are independent of the platform in which they are used and are dependent on the browsers interpretation of them.
* If the migration is being done to a environment which supports Object Oriented Programming paradigms then classes which are defined in this project need not be redefined. Only certain names and appropriate statements need to be written.
* The pre requirement for portability is the generalized abstraction between the application logic and system interfaces.

# **Other Areas to Consider**

*Provide commentary on any other aspect of the design solution.*

Not applicable.