Code Review Defects

Authors: Mika V. Mäntylä and Casper Lassenius
Original version: 4 Sep, 2007
Made available online: 24 April, 2013

This document contains further details of the code review defects presented in [1]. Description of each defect is given as well as notes describing the most typical cases.

[1] Mäntylä, M. V. and Lassenius, C. "What Types of Defects Are Really Discovered in Code Reviews?" IEEE Transactions on Software Engineering, vol. 35, no 3, May/June 2009, pp. 430-448, Available online:
http://dx.doi.org/10.1109/TSE.2008.71
http://lib.tkk.fi/Diss/2009/isbn9789512298570/article5.pdf

1. Documentation - Textual Defects

Defect	Naming
Description	Problems relating to software element (=methods, classes, variables, etc) names
Notes Industrial	Most of the Naming defects were due either to not conforming to the company's naming policy or having uninformative names. In some cases, naming defects led to longer discussions on how to name certain elements as the company did not have policy for naming every code element. Some naming problems were inherited from legacy code that did not conform to the current naming policy. Discussions with the industrial reviewers made it clear that the company strongly believed self-descriptive naming over code commenting explaing the difference in Naming issue shares.
Notes Student	Most of the Naming defects found by students pointed out uninformative names. Often the names were too short, or too general, e.g. "arg1" and "arg2". In a few cases, the names were incorrect when compared with the behavior. Occasionally, the naming did not follow Java coding standard by SUN.
Defect	Comments
Description	Problems in code comments
Notes Industrial	Comments were needed, e.g., to explain complicated programming logic. Some comments were incorrect, which was often the result of old legacy comments or copy-paste-programming. Some comments lacked information required by the company's commenting policy.
Notes Student	Many Comments defects were related to the omission of JavaDoc elements, e.g., missing method descriptions. Some comments were requested to further explain the rules implemented. Finally, some defects pointed out incorrect comments or just simple spelling mistakes.
Defect	Debug Info
Description	Problems with debugging messages. Debug Info is placed in this sub-group because it improves programs static and runtime documentation.
Notes	All the Debug Info defects requested more information on the error condition
Defect	Other Textual Defects
Description	Other Textual Defects that could not be placed to other defect classes.
Notes	Should comments or method prototypes be presented in the header files or in the source files, general comments of language usage affecting all comments and names.

2. Documentation - Supported By Language Defects

Defect	Element Type

Description	The software element is with wrong type (only cases not causing runtime failure)
Notes Industrial	All Element Type defects were in Return value types, e.g., changing the return value from “int” to “boolean”, or returning a value instead of a “void”. Interestingly, the different teams of the company had somewhat different policies regarding return values. In one team, “void” was allowed as a return value type, while in many others it was not permitted.
Notes Student	In the student reviews, the Element Type defects were more diverse. In addition to defects related to return value types, there were excessively general description of exceptions (e.g. throws “Exception” versus throws “IOException”), missing an implemented interface in the class description, having the wrong type of variables, claiming that a method throws an exception when it does not, and unnecessarily extending the “java.lang.Object” class as it is extended by default.
Defect	Immutable
Description	Not declaring variable to be immutable when it should have been or declaring it immutable when it should have not been
Notes	This was present in both reviews as both languages Java and C support such mechanism.
Defect	Visibility
Description	Software element (e.g. method, variable, class) has too much or too restricted visibility
Notes	In the industrial reviews, all the Visibility defects addressed method visibility, and in the students reviews there were cases of both the methods and variables visibility
Defect	Void Parameter
Description	Using empty brackets instead of keyword “void” as parameter
Notes Industrial	This was only present in the industrial reviews as Java does not have such option.
Defect	Element Reference
Description	Referring to software element with incomplete name
Notes Student	This was only present in student reviews and all defects found indicated that keyword "this" should have been used when referring to instance variables or methods

3. Visual Representation

Defect	Bracket Usage
Description	Incorrect usage of brackets or braces
Both Reviews	In both reviews these issues mostly referred to cases in which the developer had omitted brackets when having only a single statement after a conditional branch. Occasionally, brackets were requested to clarify the precedence operations in complex comparisons or mathematical statements.
Defect	Indentation
Description	Incorrect indentation
Notes	Often the developers thought the indentation might be correct in the code, and it was wrongly indented only in the printing. However, since they from the point of view of the review sessions were defects, we cannot remove them from our analysis simply because the developer believes this is not a problem in the code.
Defect	The Blank Line Usage
Description	Blank Line Usage is incorrect
Notes	This occurred mostly due to having an excess of blank lines or too few blank lines. However, it also included cases in which lines were split at incorrect positions.
Defect	Long Line
Description	Long Line means defects where a long code statement was contained in an excessively long single line, often more than 80 characters.
Defect	Space Usage
Description	Space Usage defects referred to the usage of the blank space character in the code.
Defect	Grouping
Description	Grouping refers to the grouping of code elements, e.g., in header files, one should group functions so that closely related functions are close to each other in the file, introducing class variables at the

	beginning of the class.

4. Structure - Organization Defects

Defect	Move Functionality
Description	Move Functionality, refers to the need to move functions, part of functions, or other functional elements to a different class, file, or module.
Defect	Long Sub-routine
Description	Function, procedure or method is of excessive length and functionality.
Notes	Some of these recommendations suggested particular elements that should be extracted to a new method while others merely pointed out that method is too long and difficult to understand
Defect	Dead Code
Description	Code that is not executed or used in the software.
Notes Industrial	Industrial Dead Code defects were evenly distributed and they consisted of unnecessary variables, unnecessary headers, uncalled functions, and branches of code that are never executed.
Notes Student	The most prominent Dead Code defect in the student code reviews was an unnecessary header with 6 mentions. Other Dead Code defects in the student reviews were an unnecessary type cast, an unnecessary variable, an unnecessary method, and an unnecessary return statement.
Defect	Duplication
Description	Code that is duplicated
Notes Industrial	The Duplication issues included functions that were partially duplicated, completely duplicated functions, duplicated comparison operations, and duplicated variables.
Notes Student	They types were similar to the issues of the industrial review. In the student code review, many of the duplicated code segments were quite small three to five lines of code, but there were duplicates up to 20 lines of code.
Defect	Complex Code
Description	a piece of code that is difficult to comprehend
Both reviews	Some Complex Code defects were quite general, e.g. a method is messy and needs to be rewritten, pointer usage is wild, or the implementation is unnecessarily complex. Other defects were more specific, e.g., an if-statement has too many comparisons or unnecessary use of negative comparison when using a positive result would be more readily understood.
Defect	Statement Issues
Description	These require splitting, combining or otherwise reorganizing a statement inside a function.
Notes Industrial	identified once in industrial reviews, when a reviewer suggested combining a declaration and an initialization of a variable to one code line
Notes Student	Six Statement issues were recognized in the student code reviews. Three of them suggested splitting a long boolean return statement to several if-else statements. Three Statements issues suggested combining statement, two of them suggested combining variable introduction with initialization and one them suggested combining statements rather than using temporary values to store the results.
Defect	Consistency
Description	Means the need to keep code consistent in a sense that similar code elements operate in a similar fashion and are more or less symmetrical. For example, similar tasks in similar classes should have similar implementations
Notes Industrial	Examples the code should not alter between using pre-defined values and numeric values, functions performing similar task in similar classes should have similar naming and naming and implementation, and comparison statements should be consistent with each other or even extracted to their own functions to remove repetition.
Notes Student	In student reviews only one Consistency issues was found. This issue pointed out inconsistent variable initialization, as some variables were initialized at the variable declaration, others at the method's constructor and some of the variables where not initialized at all.
Defect	Other
Description	Other Organization defects

Notes Industrial	In the industrial reviews, these defects included splitting up a large file containing several classes and 2000 lines of code into several files; defects were a logical piece of code was unnecessarily split into several places; the need to remove wrong couplings between software elements; the need to split up functionality into several implementations; and reducing the number of different error handling mechanisms.
Notes Student	In the student reviews, they included having several return statements in a method; using loop variables outside of the loop structure; in-lining a method as part of an other method; starting indexing from zero instead of one; using negative return values instead of positive; having too many temporary variables; having variables in class scope instead of method scope; having a method with too many parameters, and commented code that should be deleted.

5. Structure - Solution Approach Defects

Defect	Semantic Duplication
Description	Means syntactically different code blocks with equal intent, e.g., different sorting algorithms such as quicksort and heapsort have equal intent but they are not identical at the code level.
Notes Industrial	In the industrial case, we had no defects where the code under review had Semantic Duplication with itself. Instead, all the Semantic Duplication defects were identified based on the reviewers' knowledge of existing functionality located elsewhere in the software system. E.g. functionality is already implemented in some other place and there is no need to duplicate this functionality in the code under review.
Notes Student	All Semantic Duplication defects in the student reviews suggested replacing code with pre-built functionality included in the Java class library.
Defect	Semantic Dead Code
Description	Code fragments that are executed, but they do not serve any meaningful purpose and/or have no effect on the result
Notes Industrial	In the industrial reviews, two code review sessions identified nearly half of the Dead Code and Semantic Dead Code defects. This was caused by code templates that were re-used and modified.
Notes Student	All the Semantic Dead Code defects of the student reviews were unnecessary checks performed in conditional statements while in the industrial review the reasons were more varying
Defect	Change Function
Description	Need to change a certain function call to another when the program used old or deprecated functions
Notes Industrial	Often the reason for changing a function was that a better alternative was available, for example, the company had provided wrapper functions on top of many basic C-functions that added extra features or made using the functions easier.
Defect	Use Standard Method
Description	Use Standard Method contains defects where a standardized way of working should be used.
Notes	In the industry reviews these often meant using predefined constants rather than magic numbers. In the student the use exceptions for error messaging instead of return values was mentioned often. Other issue in this category were use of enumeration instead of integers; and use accessor-methods to access a class.
Defect	New Functionality
Description	Need of new functionality to ensure evolvability
Notes	In the industrial reviews, we saw defects where the creation of new functionality or classes was required to make the software more evolvable. We categorized these under the heading of New Functionality. No such defects were identified in the student reviews.
Defect	Other
Description	Other contains a wide range of defects that truly represent the Alternative Approach in its most fruitful form
Notes Industrial	In the industrial reviews, this type contained implementation changes such as using arrays instead of other more complex memory management structures, changing the code to enable an easier removal of several data items from the database, using dedicated arrays for each data element instead of a

	shared array, and using a simpler and more efficient way of keeping records in a database.
Notes Student	In the student reviews Other defects included suggesting a simpler way of performing computing and comparison operations, using Java's Generics data structures, and caching numeric values rather than recomputing them.
Defect	Minor
Description	Minor gathers implementation changes, but the defects are easier to fix and seemed less important than those categorized under Other
Notes Industrial	These defects were only identified in the industrial review. Examples of such defects are, having a default branch in a switch block, changing an if-else block to a switch-block, changing comparison element from a class name to a class id.

6. Resource Defects

Defect	Variable Initialization
Description	Means defects where variables are left uninitialized prior to use. Uninitialized variables may contain any value and using such variable for comparison or calculation produces arbitrary results.
Notes	Only present in industrial reviews because students used Java language that forces primitive variable initialization
Defect	Memory Management
Description	Means a defect where a mistake is made in handling the system memory.
Notes	In industrial reviews these include allocating too little memory, not freeing memory after its use, and freeing the same memory more than once. In the Java language used by the students, memory management is automatic, thus, the students identified no memory allocation defects.
Defect	Data & Resource Manipulation
Description	Defects related to manipulating or releasing data or other resources,
Notes	For example not releasing database cursor, and mistakes in data modification. There was only single Data & Resource manipulation defect in the student code reviews, and it is likely due to the application, which did not require many data structures.

7. Check Defects

Defect	Check Function
Description	Means that when a function is called there is also a need to check that the value returned is valid and that no error occurred
Notes	In industrial reviews 4 instances were needed to check the result of memory allocation operation
Defect	Check Variable
Description	Means that there is a need to check variable
Notes	Often such variables were function parameters, or loop control variables. In industrial reviews pointer checking was found 6 instances
Defect	Check User Input
Description	Check User Input means the need to validate user input
Notes	No additional description

8. Interface Defects

Defect Description	Function Call Means that a function call to another part of the system or class library is incorrect or missing.
Defect Description	Parameter Means that a function call or other interaction mechanism has an incorrect or missing parameter

9. Logic Defects

Defect Description	Compare means a mistake in a comparison statement
Notes	Examples: the wrong element is compared, a required comparison is missing, or the wrong type of comparison is made
Defect Description	Compute Such defects are made when computations produce incorrect results.
Defect Description	Wrong Location means that a correct operation is performed, but it is done too soon or too late.
Defect Description	Algorithm/Performance means that an inefficient algorithm is used.
Notes	Examples: performing unnecessary searches, passing large data arrays by value, and re-calculating values rather than storing them in variables
Defect Description	Other All other defect in this catogory
Notes	Examples: the need to use a loop instead of a single function call, missing an entire comparison statement and the required logic to handle the particular case, and using if-else blocks with two logically unrelated if blocks.

10. Larger Defects

Defect Description	Completeness A feature is partially implemented
Defect Description	GUI Defects in the user interface code relating to the consistency of the user-interface, and to the options made possible to the user in each situation.
Defect Description	Check outside code Defects that required that part of the application code that was not under review to be checked, as it was likely to contain incorrect code based on the current review.