

# Effective Code Review

*By Dhanya Kumar K.V.*

*19 November, 2007*

This article explains the purpose, benefits and symptoms of code review. It also shares some real time experiences and motivates the programmers towards writing good quality code. As you complete this article you will gain an appreciation and understanding of code review practices.

In any development team, you can find two kinds of programmer's namely, task-oriented programmer [TOP] and quality-oriented programmer [QoP].

Are you task or quality oriented?

If you answer NO to any of these:

1. Do you review your code to improve the quality, every time?
2. Do you respect your peers review comments?
3. Do you look for both Gold and Dirt in others code?
4. Do you strictly follow coding conventions?

You are not quality-oriented programmer!

Task-oriented programmers mainly aim to complete the task within the scheduled time. Codes written by these programmers are computer friendly. They maintain poor coding standards, writes complicated and redundant code. These programmers take pride when their code meets desired functionalities.

Quality-oriented programmers too have the same goal, but they apply best practices in their work. They review their code to improve the quality. They avoid sloppy code. They write code that's cleaner, more maintainable, more understandable, and less likely to be riddled with bugs. They do it right the first time, every time. These programmers take pride in doing best in their work.

"Any fool can write code that a computer can understand. ... Good programmers write code that humans can understand." -- Martin Fowler

"Most people forget how fast you did a job, but they remember how well it was done." -- Michelangelo

## Why projects fail?

Several decades ago, client's expectation is to get correctly working software. Today, client's expectation has changed. They expect 'correctly working software built with good quality code'. To find out why projects fail, we need only ask our clients. They'll gladly tell us: "You don't give us what we want!" One of the reasons to reject our efforts is "poor quality code".

Here is an example for poor quality code, developed by TOPs. Writing business logic with EJB 2.1 SessionBean.

```
public class MyBean implements SessionBean {
    //Business Logic
    // Life Cycle Methods of Session Bean
    ...
}
```

SessionBean interface provides life cycle methods namely `setSessionContext()`, `ejbCreate()`, `ejbRemove()`, `ejbPassivate()`, and `ejbActivate()`, which are abstract by default. So it's mandatory to provide definitions to all the methods in the bean class. Sometimes, no behavior but still you provides empty methods.

Some of the code smells I observed in this application are:

- Life cycle methods were repeated in all bean classes, say over 150 times in that application.
- Too many commented codes. Some classes have less executable code compared to commented code. When questioned about this, Top says:
  - I didn't understand old logic so I wrote a new method.
  - I forgotten or I'm too busy or it works leave it.
- Too complex logic accommodated in single method.
- Some source files have over 1K LOC

This application meets almost 98% of the business requirements. But the client may surprise the team by scrapping/rejecting the application with a statement "You don't give us what we want!"

Solution to repeated code:

```
public class GenericBean implements SessionBean {  
    //Only Life Cycle Methods of Session Bean  
}  
  
public class MyBean extends GenericBean {  
    //Only required Business Logic  
}
```

A clear separation of business logic and life cycle methods. Reusability- "write ones, use many times". Inherit behavior from GenericBean. More readable and manageable.

"Being ignorant is not so much a shame as being unwilling to learn to do things the right way".

--Benjamin Franklin

We noticed how it affects the code quality if we are task-oriented. Any team is a good combination of TOP and OoP. Unless there is bridge between these two programmers, you can't assure the released code is of good quality. Effective Code Review is one of the practices which build the bridge between these two programmers and certainly improves the code quality.

## Code Review?

- ✓ "Code Review is a practice to improve code quality".
- ✓ "Code Review is a pipeline between two programmers to exchange Coding best practices".

Code Review can be done in two ways:

- Using Code Analyzer Tools
- Manually reviewing the code

## Code Analyzer Tool [CAT]:

CAT quickly scans the source code and looks for potential smells like: unused variable, empty methods/catch/if blocks, duplicate imports and so on.

Some tools are freely available, please visit the links:

Java Code Analyzer:

<http://java-source.net/open-source/code-analyzers>

<http://jalopy.sourceforge.net/>

Links to other code analyzer:

[http://www.laatuk.com/tools/review\\_tools.html](http://www.laatuk.com/tools/review_tools.html)

## Manually reviewing the code:

Manually reviewing the code can be done in three ways:

- ✓ Self Code Review
- ✓ Face-to-Face Code Review
- ✓ Off-line code review



Self code review helps to catch your own bugs. Do it before you commit the files to Concurrent Version System, CVS or some code repositories.

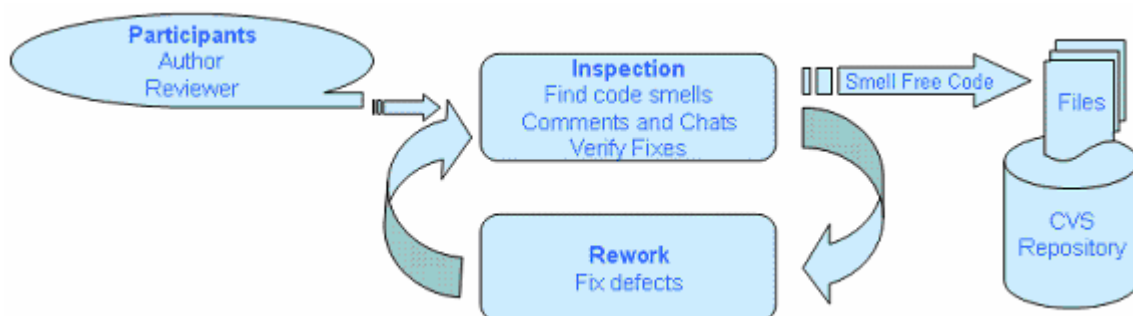
Face-to-Face code review can happen while pair programming or when you ask your peer developer to review the code. It's a good practice to explain the flow to the reviewer. So that he/she can suggest if any better approach to the problem. Avoid asking your peer programmer to review very simple logic.



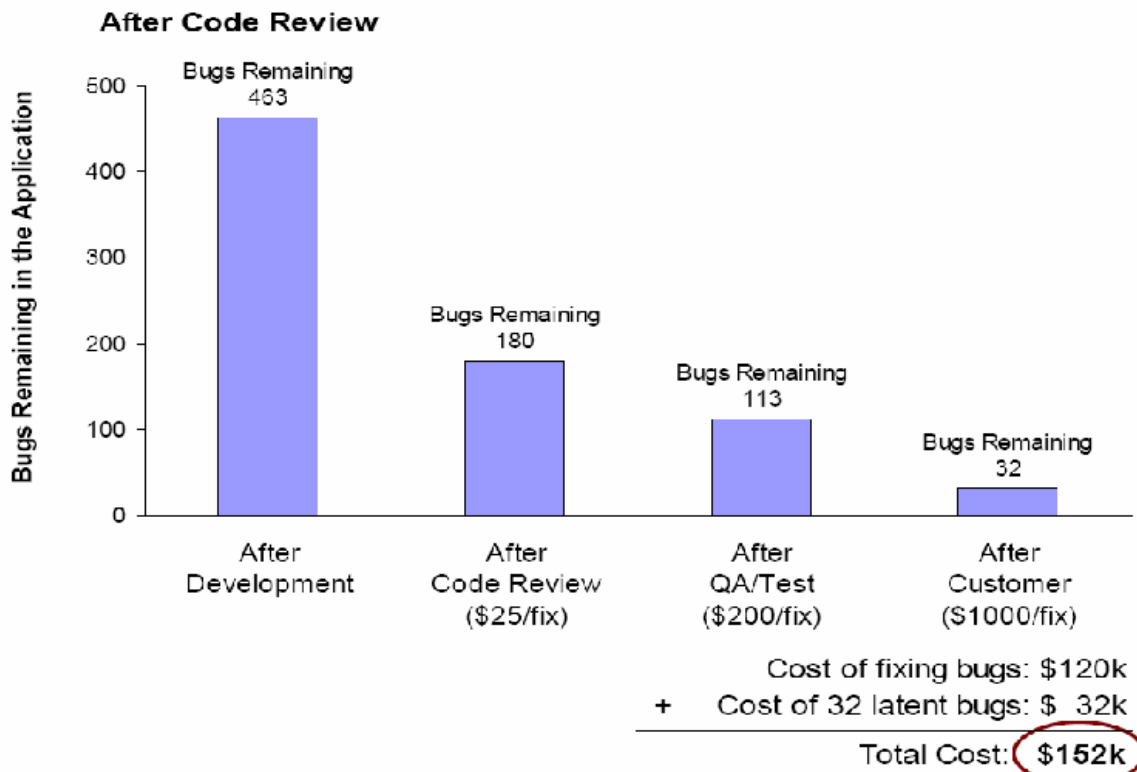
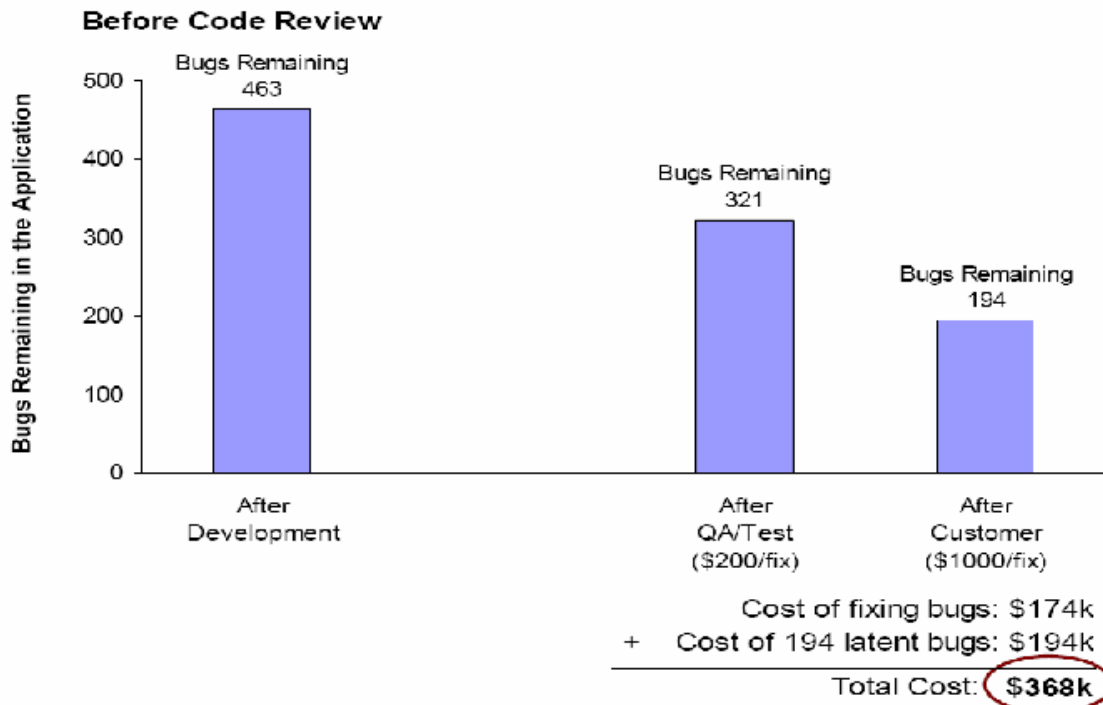
Off-line code review: CVS or Code Repository becomes the mediator between two programmers. Other programmer reviews your code and puts some comments related to code smells in the source file and re-commits to the CVS/repository. It has a drawback if review comments went unnoticed. I.e. when there is no sync between author and reviewer.

“ An effective code review makes best use of CATs and Reviewers.”

Lifecycle of a code review



The \$1 billion bug and why no one talks about peer code review  
Saving \$150k: A real-world case study



## LOOKING FOR THE GOLD AND DIRT

Effective Code Review is not only about finding code smells but also noticing good practices applied by others.

### LOOKING FOR THE GOLD

As a young Scots boy, Andrew Carnegie came to America and started doing odd jobs. He ended up as one of the largest steel manufacturers in the United States.

At one time he had 43 millionaires working for him. Several decades ago, a million dollars used to be a lot of money; even today it is a lot of money. Someone asked Mr. Carnegie how he dealt with people? Andrew Carnegie replied, "Dealing with people is like digging gold: When you go digging for an ounce of gold, you have to move tons of dirt to get an ounce of gold. But when you go digging, you don't go looking for the dirt, you go looking for the gold."

Well, that's true when you deal with people. But when you deal with code, look for both gold and dirt.

What does dirt represents?

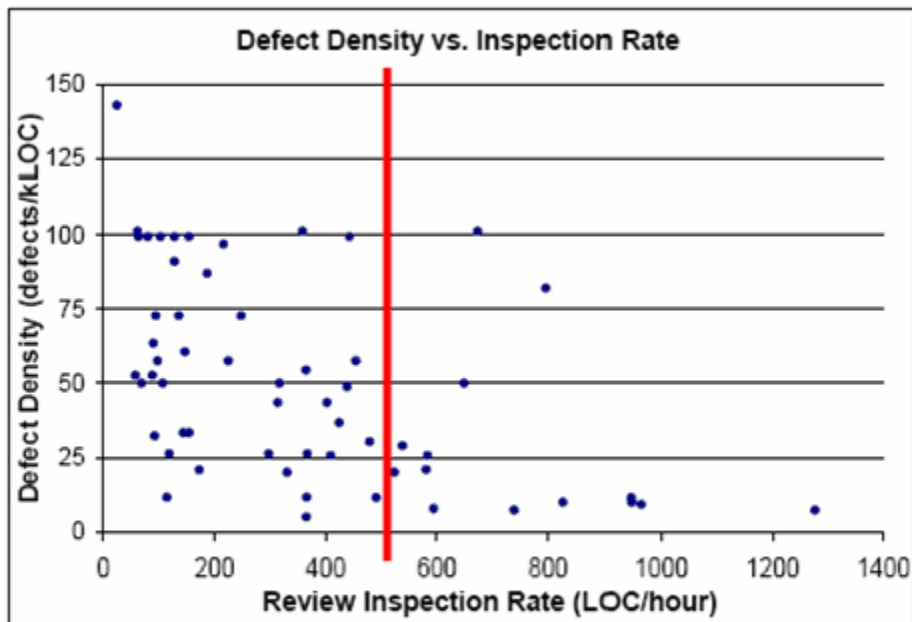
Dirt represents code smells. Code smells are warning signs about potential problems in code. Some of the measured code smells are: duplicate code, long methods, large class, long parameter list, unnecessary comments, abbreviated identifiers, code formatting, dead code/unused code and so on. Please see related resource on code smells.

Benefits of Code Review

1. Improves the code quality
2. Fosters technical skills of a programmer
3. Improves collaboration between programmers.

Some of the best practices for peer code review:

1. Baby steps during code walk-through/review.



**Inspection effectiveness falls off when greater than 500 lines of code are under review.**

2. Track code review defects & comments

Code Review Tracker (CRT) can be used to measure the effectiveness of code review. There are many tools available for this purpose, namely Code Collaborator or Jira. Jira can also be used for tracking code review defects/comments. One of the good features of this tool is E-mail notification. This feature solves the drawback of offline code review by notifying the reviewer /author, as and when, the code smells are identified/fixed. Don't merge Code Review Defects with app. defects.

3. Make a habit of do it now

programmer who works on multiple tasks, may procrastinate fixing reviewer comments. Delayed fixes may lead to major refactoring when too many new features were added by others to the same file.

"Never leave till tomorrow which you can do today."  
--Benjamin Franklin

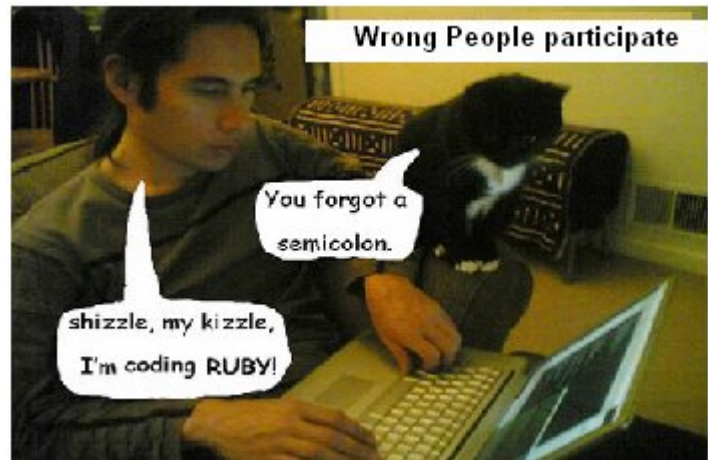
4. Dedicated team for code review task  
good to have a dedicated team for code review. One reviewer for a team size of five. Reviewers must encourage individuals to do peer code review. They must groom the people towards writing quality code. Team needs a qualified reviewer.
5. Having checklist for coding conventions.

#### Symptoms of Code Review:

These points may undermine the effectiveness of code review:

1. What's your focus? Author or Code?  
A review is not an opportunity for a reviewer to show how much smarter he is than the author, but rather a way to use the collective wisdom, insights, and experience of a group of peers to improve the quality of the group's products. Try directing your comments and criticisms to the product itself, rather than pointing out places the author made an error. Practice using the passive voice: "I don't see where these variables were initialized, "not "You forgot to initialize these variables". When you focus the author, you will ruin team morale.
2. Reviewers Focus on Style, Not Substance  
a poor reviewer makes a big defect list related to indentation, brace positioning, variable scoping, and commenting rather than finding logic errors and missing functionality. Prepare a checklist for coding conventions and follow it, so that the team maintains unique style.

3. The Wrong People participate if the participants in a review do not have appropriate skills and knowledge to find defects, their review contributions are minimal. Participants who are there only to learn may benefit, but they aren't likely to improve the quality of the product. Management participation in reviews may (but doesn't always) also lead to poor review results.



What happens when a TOP Joins OoPs team?

Initial attempts to hold reviews sometimes lead to personal assaults on the skills and style of the author. A confrontational style of raising issues exacerbates the problem. Not surprisingly, this makes the author feel beaten down, defensive and resistant to legitimate suggestions that are raised or defects that are found. When authors feel personally attacked by other review participants, they will be reluctant to submit their future products for review. They may also look forward to reviewing the work of their antagonists as an opportunity for revenge.

It's good to explain why the team follows code review practices before making any review comments. This will help the new comer to take this practice in positive way. This will avoid personal conflicts between the two programmers.

"In a positive environment, a marginal performer's output goes up".

What happens when a QoP Joins TOPs team?

One of my friends always says "I'm frustrated with the way people work. All junk code, they don't follow coding standards, too many unused code and much more. Even Sr. developers too commit the same things. Feels no opportunity to grow" He works half-heartedly in that team.

Well, "in a negative environment, a good performer's output goes down". If you are working in such environment, you need to do two things:

Firstly, you should take the initiative to bring the changes you like to see. You must take this as an opportunity to grow not to curse.

"Better to light a candle than to curse the darkness"

-- Anonymous

Secondly, you must try all possible ways to educate/motivate the programmers. You need to accept calculated risks because risk takes you to the destination.

When you take initiative, sometimes, others may show appreciation, but that's a bonus: For the major satisfaction comes from within.

Friend, when nothing works, just forward this article to all your teammates 😊

Summary

So now you're armed with an arsenal of code review practice to ensure that you get the most of out your time spent in code reviews. Of course you have to actually do code reviews to realize the benefits. With the right tools and best-practices, your team can peer-review all of its code, and find costly bugs before your software reaches even QA - so your customers get top-quality products every time!

Related Resources:

Articles & Whitepapers from SmartBearSoftware

<http://smartbearsoftware.com/codecollab-white-paper.php>

Best Kept Secrets of Peer Code Review

<http://smartbearsoftware.com/codecollab-code-review-book.php#samples>

Code Smells

<http://c2.com/xp/CodeSmell.html>

<http://wiki.java.net/bin/view/People/SmellsToRefactorings>

Java/J2EE coding best practices

<http://www.dhanyakumar.com/Links.html>

Queries/Feedback

Please post your queries or feedback to [dhanya@dhanyakumar.com](mailto:dhanya@dhanyakumar.com) or [info@dhanyakumar.com](mailto:info@dhanyakumar.com)

PS: the motivational quotes/examples are adapted from the book, You Can Win by Shiv Khera. Some content for code review analysis is compiled from various websites of the net.