Code review guide

Author: Jameel Nabbo
Website: www.jameelnabbo.com

Table of contents

Introduction:
Preforming a code review job needs some concentration and focus on the written environment and understanding of its IO (input/output) and processing system, and before preforming a code review task we should make some checks and ensure that the application/system we're testing is up and running.

Notice: Read about the language that you will test its code, if you don't have an idea about the language this will be difficult.

Code review Checklist:
In some situations, with our clients, like preforming code review for banks, you may not use your personal/work computer instead they may give you a private computer to perform the task. In this situation we have to check and ask the developers or responsible persons to run the code in front of our eyes within the written IDE, so we can see its up and running,

Notice: some clients give you the code and tell you to run it, also they may not have the knowledge of how its developed, they may be uses outsource company to write the code, in this situation, you have to build the application using the written IDE and report the build errors and bugs to the project manager.

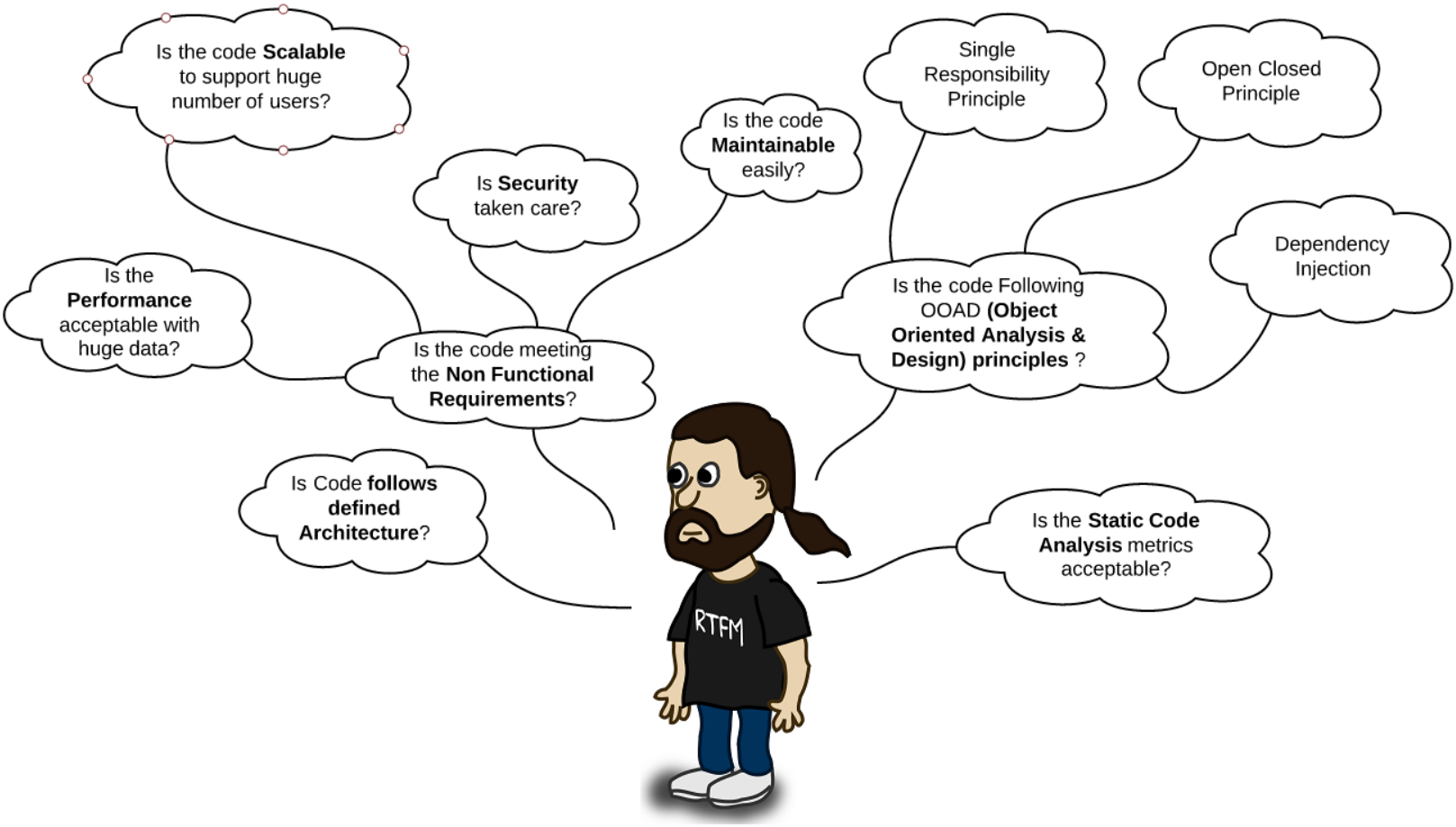
Never preform a code review job without having a working code, this is very dangerous, because clients sometimes give us the wrong project and because for some reason we didn't tested the code in the IDE, and directly jumped into the code analysis we may have a problem latter on.

Moving on, we need to make a cheat-sheet that contain the following information, and this information we'll put them into the Code review Header area, so we can know what we did and also the client as well:

Question	Example answer
What is the application language	Java,C#,C++,C,VB,PHP,Python etc,
How much lines in it	100,000K
What is the development IDE	Eclipse , Visual studio, Net beans, Xcode ,Android Studio , Zend PHP…
What is the development environment	.Net 4.5 , Java SE , Java EE
Are there any dependencies in the application written in a deferent language?	Assembly DLL, C++ libraries etc..

Answering the above questions will help us understand the project and have an idea about what tasks can be performed to well done this job.

Then we have to make a clone of the project(s) and build it, by us or by the responsible person in the company.



Expert Code Reviewer

Steps to perform on code review task:

1- Build the project
2- Understand the environment
3- Create a cheat sheet of the external libraries used
4- Run the code within its IDE
5- Insert break points on the main functions (if possible) to understand the data submitted and parameters in the run time.
6- Create a new folder for the project, and copy all code files into it as the following:
   6.A: Remove all dependencies in the project
   6.B: Remove all DLL files and exclude all files that doesn't have codes in it.
   6.C Make sure the code files you copied have same structure as the original project.

Summary of what we did:

We built the project and confirmed its working on the run time, also we created a new folder and excluded all other dependencies that is not code with the same project structure.

Now we are good to go for the code analysis using manual and automated ways, but there's things to understand before jumping to the stage:

1- Automated tools generate lots of false positive/negatives alerts.
2- The Manual code review first is **a must**.
3- Don't depend on the what automated tools founds.

Tips for code review

Tip	Description
Review fewer than 400 lines of code at a time	Developers should review no more than 200 to 400 lines of code (LOC) at a time. The brain can only effectively process so much information at a time; beyond 400 LOC, the ability to find defects diminishes. In practice, a review of 200-400 LOC over 60 to 90 minutes should yield 70-90% defect discovery. So, if 10 defects existed in the code, a properly conducted review would find between seven and nine of them.
Take your time. Inspection rates should under 500 LOC per hour	It can be tempting to tear through a review, assuming that someone else will catch the errors that you don´t find. However, a research shows a significant drop in defect density at rates faster than 500 LOC per hour. Code reviews in reasonable quantity, at a slower pace for a limited amount of time results in the most effective code review.
Do not review for more than 60 minutes at a time	Just as you shouldn´t review code too quickly, you also should not review for too long in one sitting. When people engage in any activity requiring concentrated effort over a period of time, performance starts dropping off after about 60 minutes. Studies show that taking breaks from a task over a period of time can greatly improve quality of work. Conducting more frequent reviews should reduce the need to ever have to conduct a review of this length.
Set goals and capture metrics and screenshots	Using SMART criteria, start with external metrics. For example, "reduce support calls by 15%," or "cut the percentage of defects injected by development in half." This information should give you a quantifiable

	picture of how your code is improving. "Fix more bugs" is not an effective goal. It´s also useful to watch internal process metrics, including: Inspection rate: the speed with which a review is performed Defect rate: the number of bugs found per hour of review Defect density: the average number of bugs found per line of code Realistically, only automated or strictly controlled processes can provide repeatable metrics. A metrics-driven code review tool gathers data automatically so that your information is accurate and without human bias. To get a better sense of effective code review reporting, you can see how our code review tool, Collaborator, does it.

Starting the Code review:

Starting the task by analyzing the main entry points for the application for example:

Java SE: we start looking at the main function
Java EE: we start looking at the config functions
C#: we look at the startup class
Python: we look at main function etc.

After we understand the application's main entry point we start analyzing the calls that made to initiate the application startup and analyze them one by one.

Then we start move into the first calls that made up by functions and procedures and what is the return points, notice that some applications made up using MVC logics which is stand for

Model	View	Controller

Model: database tier or business logic
View: user's tier or business logic
Controller: processing tier, and act as interface between View and Model.

So, in the MVC app's also catching the main entry point is deferent than other logic, in MVC we start look at the main config area that links all controllers in the application, also the MVC logic is not only made up for web applications, it also can be Desktop/mobile/ or IOT application.

For example, in the same .Net Framework we have also deferent environment and this can be categorized as the project type:
Mobile applications / Web applications / IOT applications / Desktop applications /

When looking to the functions we track the return points and data presentation points, here's an example of an XSS in sample code written in PHP,

Index.HTML -> have a form that submit the username to another PHP page
Hello.PHP -> prints the input that coming from index.html page using post request,

In the PHP page:

```
<?php
$username = $_POST[‘username’];
Echo $username;
?>
```

As we can see in the code above its 100% vulnerable to XSS at least, here's how:

If the user puts for example <script>alert (1) </script> this string will go to the PHP page and then printed on the page without any filtration on it. And as we can see there's no filtration or protection made in the backend side before presenting this information.

So, the main idea of analyzing functions is to understand their behaviors and the way they written, and what is their input, output and how this output made.

Here's a summary about a functions and procedures checklist:

Question	Description
What is this function doing?	Understand how this functions works, for example is it responsible for resetting user password?, or selecting user's data from the database and pass it to another function who process these data in a readable format!
How this function executed?	We should understand how this function have been executed, for example: this function is executed after providing the user a valid session that have expiry time.
What is the inputs or parameters for this function.	This step is very important since lots of the vulnerabilities show up in this point, for example this function is accepting two parameters and then pass a database select statement.
Is there any validation made on the input	When executing the code and taking the Params, we should track what king of validation made on these functions and start think about a way to bypass them,
What is the returned values after running this function	After tracking how the function works and understanding its behavior, we should track the returned (Processed input) after complete execution of this function.

As we can see in the checklist above, we should make these steps on all the functions that the code have manually and then write them into our report in case we found a vulnerability within it.

Another important point is to also see how the code preforming data manipulation inside of the memory, for example
Int number = ReadUserInput ();

The (number) variable has been defined as an integer, but what if we pass a number 9999999999999999 that's more than the int32 in memory what will happen? also what if we pass strings instead of passing numbers to the (number) variable also what will happen?

Asking the mentioned questions for the scenario above will let us check if there's any restrictions made before assigning the return value to (number) variable like if there's some if statements or Try Catch blocks that handle the malicious user input in the real time.

Now preforming using the automated tools:

Language	Tool
NodeJs	https://github.com/ajinabraham/NodeJsScan
PHP	https://sourceforge.net/projects/rips-scanner
Java	http://findbugs.sourceforge.net/
.Net	https://msdn.microsoft.com/en-us/library/bb429476(v=vs.80).aspx
JavaScript	http://jshint.com
C#	https://codecrawler.codeplex.com/
.Net, Java, C/C++, HTML, JavaScript, ASP, ColdFusion, PHP, COBOL	http://www.scovetta.com/yasca.html
ASP, JSP, Perl, PHP, Python	https://github.com/wireghoul/graudit
C, C#, PHP, Java, Ruby, ASP, JavaScript	https://github.com/CoolerVoid/codewarrior

Writing the vulnerability:

When you find a vulnerability, we should write the following in our report:

1- CWE
2- Vulnerability standard name
3- Vulnerability description
4- Vulnerability effect and risk
5- Line number and path for the Vulnerable code (screenshot is also fine).
6- How it can be fixed
7- Vulnerability type (Local/Remote).

Author: Jameel Nabbo
Website: www.jameelnabbo.com