Tick-the-Code Inspection, a short introduction

Different and unconventional

The code inspection Qualiteers offer is probably different from anything the software designers are used to. Instead of looking at the code to find problems with the functionality, the checkers are supposed to follow strict rules and look for unnecessary complexity. Once the author of the code receives the feedback, he can use some Quality Time to improve the maintainability and understandability of the code by clarifying the checkers' findings.

There is no need for checkers to 'mentally debug' the code in a code inspection. Yet that is what generally happens in a desk-check, walkthrough or technical review. Using rules one by one, helps the checkers to focus. Functionality is at the fore front in all phases of software development: in specifying the requirements you think of what the software should do, in architecture and design phase you ponder the question how to build the software to do what it needs to do, in the implementation phase the software developers focus entirely on creating code to do what the specification says, and in testing the testers create their test cases on the requirement specification listing the functionality with the aim of breaking the functionality. *Why should code inspection perform essentially the same task for the fifth time?* In my opinion, that's mostly a waste of time.

Useful and practical

The participants learn the skill in the training and actually use it already in there. The trainer makes personally sure that all participants use the method successfully. The method is so practical that the findings the participants make during the training can be taken into account by the authors right after the training. They can improve their code right away. If making your code simpler and clearer saves money by saving maintenance time, then the training pays for itself possibly during the same day.

The training

The maximum group size is 10 people. The participants get practical experience with several rules by checking code they bring. The checkers get four rule cards (24 rules in all!) to keep. They also receive a card summarizing the whole training. A certain kind of quality mentality is emphasized in the course. Other topics include busyness, bear hunting, and feelings - especially happiness at work. People who create or maintain software source code every day benefit the most from the training. The rules work definitely on C, C++, Java and C# programming languages.

You can find more information on the training course at http://www.qualiteers.com/thecourse.php.

A few good rules

MAGIC: “Do not hardcode values”
INTENT: “A comment must either describe the intent of the code or summarize it.”
FOCUS: “A routine shall do one and only one thing.”

A round of inspection

Jesse has been implementing new functionality for a week. The module isn’t working quite yet, as a matter of fact it doesn’t even compile yet. Jesse hasn’t been able to test his code for that reason. He has worked as best he could considering the time pressure, but he suspects there are all kinds of things unclear in his code. He prints out the code, copies it three times and searches for checkers. Jesse’s closest colleague, Gary, agrees immediately and chooses the blue card out of the four rule cards in Jesse’s hand. Gary decides to inspect the code first thing the next morning. Jesse finds two other checkers; Anne gets the red card and John gets the green card. All of them get the same assignment: The thousand lines of code Jesse has written and printed need to be checked against all six rules on each rule card. It should take about an hour and everybody should return the material within a week with markings Jesse can read. The hour of checking is up to the checkers themselves.

Jesse is happy with the checkers he found and retires into a meeting room he reserved for himself. He takes with him the printed code, the purple rule card and a colorful pen. He has turned off his cellphone and is taking every precaution in order not to be disturbed during the checking. Then he starts to check using one of the simple rules to get into a more concentrated state of mind, into flow.

On the card there is the simple rule **MAGIC**. Jesse goes through the code line by line looking for hardcoded, i.e. literal numbers. He marks each one he finds and is surprised of how many there were. Jesse knows they hamper the understanding of the code and coming up with good names will help. He’s not trying to come up with the names now, he’s completely concentrated on marking any rule violation he discovers. Once he reaches the last code page with **MAGIC** in his mind, Jesse returns to the first page.

The next rule is called **INTENT**, and Jesse gets a whole different point of view into the code by looking first at the comments and then at the code. Many a comment makes Jesse wonder; these kinds of self-evident code lines need no comments. Only if a comment summarizes several lines of code or it reveals the intent of the lines of code, is a comment left unmarked. Once again, after concentrating fully only on **INTENT** for the whole one thousand lines of code, Jesse turns back to the first page.

After a few other rules, Jesse turns his attention to **FOCUS** and starts to look at his functions as a whole. The functions that clearly do many things, receive a rule violation marking without mercy.

Finally Jesse is ready. He has gone through his thousand lines six times, every time with a different rule in his mind. He has made 42 markings, which he needs to go through later as author. Some of the markings won’t cause anything, but changing most of them will make the code clearer and ease its maintenance, and Jesse is already looking forward to going through the markings.

And if somebody asks, he’ll volunteer and check somebody else’s code next week.

Advantages

The example above tells roughly how a normal **Tick-the-Code** Inspection proceeds in an organization in which the whole programming staff has received the checker training. Jesse can rely on getting excellent markings from all three checkers, and what's even more important, they all have gone through the same code using different rules. With around four hours of effort, the thousand lines of code has been checked with 24 different rules. Communication between participants is effective, really only the markings matter. There is no need for expensive discussion meetings. The focus of the checking isn't in functionality (the incompleteness of which is preventing testing) but in the complexity of the code. The method has clear roles for the checkers and the author, which must never be mixed (Jesse is not thinking about improving or fixing the code while checking).

The main advantage of **Tick-the-Code** Inspection is **learning**. You learn from your mistakes and from the mistakes and successes of others.

A case study

A sample of 39 participants from the same organization but different teams checked code for fifteen minutes before the training on their own. During the training they checked the same code again but for a whole hour. Comparing the extrapolated number of findings left to their own devices to the actual number of ticks during an hour showed a **5-fold improvement on average**! In other words, with regular **Tick-the-Code** Inspections every developer gets five times as many opportunities to improve his code than without.

39 samples	Findings in 15 min (on their own)	Findings in 1h (extrapolation)	Ticks in 1h	Improvement
Min	0	2	20	0.5x*
Average	8	32	82	5x
Max	26	100	250	20x

* These checkers were excellent to begin with. They found about 1 item / min on their own. They didn't get worse.

Summary

Tick-the-Code Inspection is a valuable new skill to any individual software developer as well as his team and the organization the team belongs to. The skill enables effective and efficient checking of code in almost any application area. Regularly used, the method improves the average quality of the produced software over time. The checkers improve every time, too.