A Guide to Code Inspections

Jack G. Ganssle
jack@ganssle.com

The Ganssle Group
PO Box 38346
Baltimore, MD 21231

There *is* a silver bullet that can drastically improve the rate you develop code while also reducing delivered bugs.

Though this bit of magic can reduce debugging time by an easy factor of 10 or more, despite the fact that it's a technique well known since 1976, and even though neither tools nor expensive new resources are needed, few embedded folks use it.

Formal code inspections are probably the most important tool you can use to get your code out faster with fewer bugs. The inspection plays on the well-known fact that "two heads are better than one". The goal is to identify and remove bugs *before* testing the code.

Effectiveness

Those that are aware of the method often reject it because of the assumed "hassle factor". Usually few developers are aware of the benefits that have been so carefully quantified over time. Let's look at some of the data.

The very best of inspection practices yield stunning results. For example, IBM manages to remove 82% of all defects *before* testing even starts!

One study showed that, as a rule of thumb, each defect identified during inspection saves around 9 hours of time downstream.

AT&T found inspections led to 14% increase in productivity and tenfold increase in quality.

HP found 80% of the errors detected during inspections were unlikely to be caught by testing.

HP, Shell Research, Bell Northern, and AT&T all found inspections 20 to 30 times more efficient than testing in detecting errors.

IBM found inspections gave a 23% increase in productivity and a 38% reduction in bugs detected after unit test.

So, though the inspection may cost you 20% extra time during coding, debug times can shrink by an order of magnitude or more. The reduced number of bugs in the final product means you'll spend less time in the mind-numbing weariness of maintenance as well.

The Inspection Team

The best inspections come about from properly organized teams. Keep management off the team! Experience indicates that when a manager is involved usually only the most superficial bugs are caught, since no one wishes to show the author to be the cause of major program defects.

Four formal roles exist: the Moderator, Reader, Recorder, and Author.

The Moderator, who must be very competent technically, leads the inspection process. He or she paces the meeting, coaches other team members, deals with scheduling a meeting place and disseminating materials before the meeting, and follows up on rework (if any).

The Reader takes the team through the code by paraphrasing its operation. Never let the Author take this role, since he may read what he meant instead of what he implemented.

A Recorder notes each error on a standard form. This frees the other team members to focus on thinking deeply about the code.

The Author’s role is to understand the errors found and to illuminate unclear areas. As code inspections are never confrontational, the Author should never be in a position of defending the code.

An additional role is that of Trainee. No one seems to have a clear idea how to create embedded developers. One technique is to include new folks (only one or two per team) into the code inspection. The Trainee(s) then get a deep look inside of the company’s code, and an understanding of how the code operates.

It’s tempting to reduce the team size by sharing roles. Bear in mind that Bull HN found four person inspection teams are twice as efficient and twice as effective as three person teams. A code inspection with three people (perhaps using the Author as the Recorder) surely beats none at all, but do try to fill each role separately.

The Process

Code inspections are a *process* consisting of several steps; all are required for optimal results. The steps are:

Planning - When the code compiles cleanly (no errors or warning messages), and after it passes through Lint (if used) the Author submits listings to the Moderator, who forms an inspection team. The moderator distributes listings to each team member, as well as other related documents such as design requirements and documentation. The bulk of the Planning process is done by the Moderator, who can use email to coordinate with team members. An effective Moderator respects the time constraints of his colleagues and avoids interrupting them.

Overview - This optional step is a meeting for cases where the inspection team members are not familiar with the development project. The Author provides enough background to team members to facilitate their understanding of the code.

Preparation - Inspectors individually examine the code and related materials. They use a checklist to ensure they check all potential problem areas. Each inspector marks up his or her copy of the code listing with suspected problem areas.

Inspection Meeting - The entire team meets to review the code. The Moderator runs the meeting tightly. The only subject for discussion is the code under review; any other subject is simply not appropriate and not allowed.

The person designated as Reader presents the code by paraphrasing the meaning of small sections of code in a context higher than that of the code itself. In other words, the Reader is translating short code snippets from computer-lingo to English to ensure the code's implementation has the correct meaning.

The Reader continuously decides how many lines of code to paraphrase, picking a number that allows reasonable extraction of meaning. Typically he's paraphrasing 2-3 lines at a time. He paraphrases every decision point, every branch, case, etc. One study concluded that only 50% of the code gets executed during typical tests, so be sure the inspection looks at *everything*.

Use a checklist to be sure you're looking at all important items. See the "Code Inspection Checklist" for details.

Record all errors and classify them as Major or Minor. A Major bug is one that if not removed could result in a problem that the customer will see. Minor bugs are those that include spelling errors, non-compliance with the firmware standards, and poor workmanship that does not lead to a major error.

Why the classification? Because when the pressure is on, when the deadline looms near, management will demand that you drop inspections as they don't seem like "real work." A list of classified bugs gives you the ammunition needed to make it clear that dropping inspections will yield more errors and slower delivery.

Two forms get filled out. The "Code Inspection Checklist" is a summary of the number of errors of each type that's found. It's used for understanding how effective the inspection process is.. The "Inspection Error List" is the details of each error requiring rework.

The code itself is the only thing under review; the author may not be criticized. One effective way of defusing the tension in starting up new inspection processes (before the team members are truly comfortable with it) is to have the Author supply a pizza for the meeting. Then he seems like the good guy.

At this meeting no attempt is made to rework the code, or to come up with alternative approaches. Find errors and log them; let the Author deal with implementing solutions. The Moderator must keep the meeting fast-paced and efficient. In fact, a reasonable review rate is between 150 and 200 non-blank lines per hour.

Note that comment lines require as much review as code lines. Misspellings, lousy grammar, and poor communication of ideas are as deadly in comments as outright bugs in code. Firmware must do two things to be acceptable: it must work, and it must communicate its meaning to a future version of yourself - and to others. The comments are a critical part of this and deserve as much attention as the code itself.

It's worthwhile to compare the size of the code to the estimate originally produced (if any!) when the project was scheduled. If it varies significantly from the estimate figure out why, so you can learn from your estimation process.

Limit inspection meetings to a maximum of two hours. At the conclusion of the review of each function decide whether the code should be accepted as is or sent back for rework.

Rework - The Author makes all suggested corrections, gets a clean compile (and Lint if used) and sends it back to the Moderator.

Follow-up - The Moderator checks the reworked code. If the Moderator is satisfied the inspection is formally complete and the code may be tested.

Other Points

One hidden benefit of code inspections is their intrinsic advertising value. We talk about software reuse, while all too often failing spectacularly at it. Reuse is certainly tough, requiring a lot of discipline and work. One reason it fails, though, is simply because people are not aware of the code. If you don't know there's a function on the shelf, ready to rock 'n roll, then there's no chance you'll reuse it. The inspection makes more people aware of what code exists.

The literature is full of the pros and cons of inspecting code before you get a clean compile. My feeling is that the compiler is nothing more than a tool, one that very cheaply and quickly picks up the stupid silly errors we all make. Compile first, and let the tool rather than expensive people pick up the simple mistakes.

Along those same lines, I also believe that the only good compile is a clean compile. No error messages. No warning messages. Warnings are deadly when some other programmer, maybe years from now, tries to change a line. When presented with a screenful of warnings he'll have no idea if these are normal or a symptom of a problem.

Conclusion

Inspections break the dysfunctional code-compile-debug cycle. We know firmware is hideously complex and awfully prone to failure. It's crystal clear from data, both quantitative and anecdotal, that code inspections are the cheapest and most effective bug beaters in the known universe. Yet few organizations, especially smaller ones, use them on their firmware.

Inspect *all* of your code. Make this a habit. Resist the temptation to abandon inspections when the pressure heats up. Being a software professional means we do the right things, all of the time. The alternative is to be a hacker - cranking the code out at will with no formal discipline.

Inspection shouldn't be limited to code; all specification and design documents benefit from a similar process.

For those interested in more information, check out the following two books (both very highly recommended). Both are available from the Computer Literacy Bookstore and amazon.com:

Software Inspection, Tom Gilb and Dorothy Graham, 1993, TJ Press (London). ISBN 0-201-63181-4.

Software Inspection - An Industry Best Practice, David Wheeler, Bill Brykczynski and Reginald Meeson, 1996 by IEEE Computer Society Press (CA), ISBN 0-8186-7340-0.

Code Inspection Checklist

Project:	______________________
Author:	______________________
Function Name:	______________________
Date:	______________________

Number of errors		*Error Type*
Major	Minor	
		Code does not meet firmware standards
		Function size and complexity unreasonable
		Unclear expression of ideas in the code
		Poor encapsulation
		Function prototypes not correctly used
		Data types do not match
		Uninitialized variables at start of function
		Uninitialized variables going into loops
		Poor logic - won't function as needed
		Poor commenting
		Error condition not caught (e.g., return codes from malloc())?
		Switch statement without a default case (if only a subset of the possible conditions used)?
		Incorrect syntax - such as proper use of ==, =, &&, &, etc.
		Non reentrant code in dangerous places
		Slow code in an area where speed is important
		Other
		Other

A Major bug is one that if not removed could result in a problem that the customer will see. Minor bugs are those that include spelling errors, non-compliance with the firmware standards, and poor workmanship that does not lead to a major error.

Inspection Error List

Project:	
Author:	
Function Name:	
Date:	
Rework required?	

Location	*Error Description*	*Major*	*Minor*