

Jack Ganssle's Seven step plan to better development.

1 Install and use a Version Control System.

Why?

- a) Insulates code from developers, it stops people fiddling with the code.
 - b) It enables you to track each change in the code.
 - c) It controls the number of people working on a module.
 - d) It allows you to create a single correct module from one inadvertently modified by two people.
- Check-in and check-out modules as needed.
 - Don't hoard checked-out modules.
 - Back up your VCS every night, machines can be replaced, data can't.
 - Include the development tools in the VCS at the end of the project. Embedded systems can have very long lives.

2 Institute a Firmware Standards Manual.

- Stick to it.
- Code has two important functions:
 - a) It must work.
 - b) It must clearly communicate how it works.
- Avoid holy wars: Make decisions.
 - Indentation styles are not worth fighting over.
 - Pick one and make everybody stick to it.

3 Use Code Inspections.

- Code inspection has been shown to be one of the best tools for removing bugs, even before verification. It can reduce debugging time by a factor of 10. IBM and HP found that 80% of bugs were found before code testing through the use of code inspections.
- The Inspection Team:
 - a) **KEEP MANAGEMENT OUT OF THE TEAM!!!!**
 - b) There are four formal roles in an inspection team.
 - Moderator
 - Technically competent, leads the process, manages the scheduling, meeting location, distribution of materials and follows up on reworks.
 - Reader
 - Takes the team through the code by paraphrasing its operation. This should NEVER be the Author as s/he may read what was intended not what was written.
 - Recorder
 - Notes each error on a standard form. Stays out of the deep thinking.
 - Author
 - Role is to understand the errors and to illuminate unclear areas. Inspections are non-confrontational and the Author should not be in a defensive position.
 - Trainee
 - How else to you create new developers?
 - c) Use four person teams, they are twice as effective as three person teams, but three is better than none so double up Author and Recorder if you only have three people.
- The Process
 - Planning
 - When code compiles cleanly, no warnings and lint free,

- i) The Author submits the code to the Moderator.
- ii) The Moderator distributes the code to each team member as well as any other relevant documentation. Use e-mail, respect time constraints of other members and avoid interruptions.

Overview

Optional; When team members are not familiar with the project, this meeting provides background information.

Preparation

Inspectors individually examine the code, Use the checklist to ensure all potential problem areas are covered. Inspectors mark up their copy of the code listing.

Inspection Meeting

The entire team meets to review the code. The meeting must be run tightly. The ONLY subject for discussion is the code being reviewed.

- i) The Reader presents the code by paraphrasing small sections of code at a time, typically two to three lines at a time. Every decision point, every case, every branch etc.
- ii) Use the checklist as a reminder of the important points.
- iii) Reject code that does not conform to the Firmware Standards.
- iv) Log and classify defects as Major (could be visible to the client), or Minor (spelling errors, non-compliance with standards, poor workmanship). Classification helps when the pressure is on to focus on the important bugs, and to show management that the inspections ARE important.
- v) Fill out both forms. The Checklist summarises the numbers of bugs found and their type. The Inspection list details "what and where" for items needing rework.
- vi) Only the code is under review, no criticising the Author. Get the Author to bring a pizza to the first few meetings.
- vii) Leave the code alone. It's up to the Author to rework as needed, later.
- viii) Inspect the comments as well as the code.
- ix) Limit meetings to a maximum of two hours.

Rework

The Author makes all suggested corrections. when it compiles with no warnings and is lint free, then send it back to the Moderator.

Follow-up

The Moderator checks the reworked code, when satisfied, the inspection is formally complete.

4 Create A Quiet Work Environment.

- The biggest single factor in code productivity is office environment. More so even than experience. The top 25% outperform the bottom 25% by a factor of 2.6. It takes at least 15 minutes to recover from an interruption. So 3 or 4 phone calls an hour and you are doing NO creative work. If you are stuck with an open plan cubicle then try the following.
 - a) Wear headphones and listen to music.
 - b) Turn the phone off or unplug it.
 - c) Use your most productive hours wisely. Are you a morning person or an afternoon person. Use the least creative hours for the low IQ stuff, meetings, phone calls etc.
 - d) Disable the "new e-mail alert". It's mostly spam or jokes anyway. Deal with the mail first thing and then leave it.
 - e) Put up a curtain across the opening. Educate people that "If it's shut then don't disturb".

5 **Measure Your Bug Rates.**

- He who makes no mistakes surely writes no code. There are three key reasons for bug measurement.
 - a) We 'find and fix' too quickly. Slow down, think before implementing a fix and then log it. Logging slows us down, fewer errors.
 - b) Some code is just junk, Measuring bugs helps identify these functions. Take appropriate action.
 - c) Defects are a measure of customer satisfaction, once a product ships then log the defects to understand how effective our processes are.
- Identify Bad Code. The 80:20 rule, 80% of bugs are in 20% of the code. Tracking bug fixes against functions makes for easy identification of bad code. Problem functions such as these can cost up to 4 times as much as others. Consider rewriting them from scratch, it may be more cost effective, even rewritten twice.

6 **Measure Your Code Production Rates.**

- Schedules collapse. Design the system, break it into blocks, then smaller chunks, until no part of the code is more than 2 pages. Now you have a grasp of the system complexity, but what about the time required?

$$\text{development time} = (\text{program size in Lines of Code}) \times (\text{time per LOC})$$

- Any one else's figure for code production rates is useless to you. Start keeping a measure, every day, forever. Now you can give an accurate estimate for the development time needed.

7 **Constantly Study.**

- Development is a dynamic environment. Learn new techniques, Experiment.

Project:		
Author:		
Function Name:		
Date:		
<i>Number of errors</i>		<i>Error type</i>
Major	Minor	
		Code does not meet firmware standards
		Function size and complexity unreasonable
		Unclear expression of ideas in the code
		Function prototypes not correctly used
		Uninitialised variables (function or loops)
		Name clash in Schematic/project
		Poor logic—won't function as needed
		Poor commenting
		Error condition not caught (eg return codes from malloc())?
		Switch statement without a default case? (or without default states in VHDL)
		Incorrect syntax—such as proper use of ==, =, &&, &, etc
		Incorrect sensitivity list in process
		Incorrect reset or initialisation in Process or state machine
		Incorrect register inference in VHDL
		Length mismatch in tests on vectors ("000" > "00" is true!)
		Incorrect range direction in vectors (TO v DOWNTO)
		Unnamed buses or signals in Schematic

