[**How to be a zero-bug programmer? [closed]**](http://programmers.stackexchange.com/questions/41248/how-to-be-a-zero-bug-programmer)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| up vote 102 down vote [favorite](http://programmers.stackexchange.com/questions/41248/how-to-be-a-zero-bug-programmer)  **38** | My boss has always told me that a good programmer should be able to ensure that the code he or she changes is reliable, correct, and thoroughly self-verified; that you should completely understand all the results and impacts your changes will cause. I have tried my best to be this kind of programmer—by testing again and again—but bugs are still there.  How can I be a zero-bug programmer and know what every character of my code will cause and affect?  [code-quality](http://programmers.stackexchange.com/questions/tagged/code-quality)   |  |  |  | | --- | --- | --- | | [share](http://programmers.stackexchange.com/q/41248)[improve this question](http://programmers.stackexchange.com/posts/41248/edit) | [edited Jan 30 '11 at 3:00](http://programmers.stackexchange.com/posts/41248/revisions) | community wiki  [5 revs, 3 users 43% user8](http://programmers.stackexchange.com/posts/41248/revisions) | |
|  | |  |  |  |  | | --- | --- | --- | --- | | |  |  | | --- | --- | | 13 |  | | Become me. That's the only way. –  [rightfold](http://programmers.stackexchange.com/users/4595/rightfold) [Aug 10 '11 at 23:07](http://programmers.stackexchange.com/questions/41248/how-to-be-a-zero-bug-programmer#comment181536_41248) |   show **1** more comment |

**closed as not constructive by** [**gnat**](http://programmers.stackexchange.com/users/31260/gnat)**,** [**Walter**](http://programmers.stackexchange.com/users/101/walter)**,** [**Matthieu**](http://programmers.stackexchange.com/users/7454/matthieu)**,** [**Dynamic**](http://programmers.stackexchange.com/users/34364/dynamic)**,** [**Yusubov**](http://programmers.stackexchange.com/users/57164/yusubov) **Sep 20 '12 at 11:16**

As it currently stands, this question is not a good fit for our Q&A format. We expect answers to be supported by facts, references, or expertise, but this question will likely solicit debate, arguments, polling, or extended discussion. If you feel that this question can be improved and possibly reopened, [visit the help center](http://programmers.stackexchange.com/help/reopen-questions) for guidance.If this question can be reworded to fit the rules in the [help center](http://programmers.stackexchange.com/help/closed-questions), please [edit the question](http://programmers.stackexchange.com/posts/41248/edit).

**29 Answers**

[active](http://programmers.stackexchange.com/questions/41248/how-to-be-a-zero-bug-programmer?answertab=active#tab-top) [oldest](http://programmers.stackexchange.com/questions/41248/how-to-be-a-zero-bug-programmer?answertab=oldest#tab-top) [votes](http://programmers.stackexchange.com/questions/41248/how-to-be-a-zero-bug-programmer?answertab=votes#tab-top)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| up vote 277 down vote | | **Don't code at all.**  That's the only way you can be a zero-bug programmer.  Bugs are unavoidable because programmers are human, all we can do is try our best to prevent them, react quickly when a bug occurs, learn from our mistakes and stay up to date.   |  |  |  | | --- | --- | --- | | [share](http://programmers.stackexchange.com/a/41251)[improve this answer](http://programmers.stackexchange.com/posts/41251/edit) | answered [Jan 29 '11 at 19:47](http://programmers.stackexchange.com/posts/41251/revisions) | community wiki  [wildpeaks](http://programmers.stackexchange.com/posts/41251/revisions) | |
|  | | |  |  |  |  | | --- | --- | --- | --- | | |  |  | | --- | --- | | 56 |  | | +1 Follow up: Or you could become a non-coding (ivory tower) architect and still get paid a lot! That's the best. –  [Martin Wickman](http://programmers.stackexchange.com/users/5692/martin-wickman) [Jan 29 '11 at 19:54](http://programmers.stackexchange.com/questions/41248/how-to-be-a-zero-bug-programmer#comment72980_41251) | | |  |  | | --- | --- | | 15 |  | | To err is human, to fix your bugs divine. –  [Ward Muylaert](http://programmers.stackexchange.com/users/14524/ward-muylaert) [Jan 29 '11 at 23:11](http://programmers.stackexchange.com/questions/41248/how-to-be-a-zero-bug-programmer#comment73016_41251) | | |  |  | | --- | --- | | 5 |  | | I used to have a co-worker who was favored by the boss because she consistently had a small bug count. How did she do it? Simple, she assigned the bugs she didn't want to someone else, and always took on the easiest/smallest features. –  [toby](http://programmers.stackexchange.com/users/14956/toby) [Jan 30 '11 at 6:31](http://programmers.stackexchange.com/questions/41248/how-to-be-a-zero-bug-programmer#comment73179_41251) | | |  |  | | --- | --- | | 28 |  | | I don't know who first said it, but, "If debugging is the process of removing bugs, then programming must be the process of putting them in." –  [Bruce Alderman](http://programmers.stackexchange.com/users/3869/bruce-alderman) [Jan 30 '11 at 7:03](http://programmers.stackexchange.com/questions/41248/how-to-be-a-zero-bug-programmer#comment73182_41251) | | |  |  | | --- | --- | | 4 |  | | Even better: become a boss and make your underlings feel miserable without having to take responsibility for anything. –  [biziclop](http://programmers.stackexchange.com/users/13625/biziclop) [Jan 30 '11 at 19:18](http://programmers.stackexchange.com/questions/41248/how-to-be-a-zero-bug-programmer#comment73348_41251) |   show **4** more comments |
| up vote 89 down vote | Zero bugs is impossible for a non-trivial program.  It's possible to get very close, but productivity suffers. And it's only worthwhile for certain high-risk software. The [Space Shuttle software](http://www.fastcompany.com/magazine/06/writestuff.html) comes to my mind. But their productivity is of the order of a few lines a day. I doubt your boss wants that.  This software is bug-free. It is perfect, as perfect as human beings have achieved. Consider these stats: the last three versions of the program -- each 420,000 lines long - had just one error each. The last 11 versions of this software had a total of 17 errors.  Take the upgrade of the software to permit the shuttle to navigate with Global Positioning Satellites, a change that involves just 1.5% of the program, or 6,366 lines of code. The specifications for that one change run 2,500 pages, a volume thicker than a phone book. The specifications for the current program fill 30 volumes and run 40,000 pages.   |  |  |  | | --- | --- | --- | | [share](http://programmers.stackexchange.com/a/41249)[improve this answer](http://programmers.stackexchange.com/posts/41249/edit) | [edited Jan 30 '11 at 23:15](http://programmers.stackexchange.com/posts/41249/revisions) | community wiki  [3 revs, 2 users 53%](http://programmers.stackexchange.com/posts/41249/revisions) [CodesInChaos](http://programmers.stackexchange.com/users/8669) | | |
|  | |  |  |  |  | | --- | --- | --- | --- | | |  |  | | --- | --- | | 2 |  | | Not impossible. But highly unlikely. –  [Billy ONeal](http://programmers.stackexchange.com/users/886/billy-oneal) [Jan 30 '11 at 1:20](http://programmers.stackexchange.com/questions/41248/how-to-be-a-zero-bug-programmer#comment73061_41249) | | |  |  | | --- | --- | | 25 |  | | What makes you think that FB is bug-free? Facebook's security and privacy model is one huge bug. For example, the Facebook bosses Facebook account got hacked last week due to security weaknesses. –  [Stephen C](http://programmers.stackexchange.com/users/172/stephen-c) [Jan 30 '11 at 2:06](http://programmers.stackexchange.com/questions/41248/how-to-be-a-zero-bug-programmer#comment73075_41249) | | |  |  | | --- | --- | | 6 |  | | @Elaine Facebook's philosophy is "go fast and break things" ([geek.com/articles/news/…](http://www.geek.com/articles/news/mark-zuckerberg-says-you-need-to-move-fast-and-break-things-2009102/)) and they've had countless bugs ([facebook.com/board.php?uid=74769995908](http://www.facebook.com/board.php?uid=74769995908)), including many I've run into myself. Twitter is no different, known for frequently being down ([engineering.twitter.com/2010/02/anatomy-of-whale.html](http://engineering.twitter.com/2010/02/anatomy-of-whale.html)) and bugs such as the "follow bug" ([status.twitter.com/post/587210796/…](http://status.twitter.com/post/587210796/follow-bug-discovered-remedied)). –  [Yevgeniy Brikman](http://programmers.stackexchange.com/users/5939/yevgeniy-brikman) [Jan 31 '11 at 5:39](http://programmers.stackexchange.com/questions/41248/how-to-be-a-zero-bug-programmer#comment73708_41249) | | |  |  | | --- | --- | | 2 |  | | If the last three versions all had exactly one error each, that means the patch for each must have added the subsequent bug. More likely, the third-to-last one had three bugs, the second two, the last one. Except of course there will still be bugs there. –  [l0b0](http://programmers.stackexchange.com/users/13162/l0b0) [Jan 31 '11 at 8:36](http://programmers.stackexchange.com/questions/41248/how-to-be-a-zero-bug-programmer#comment73756_41249) | | |  |  | | --- | --- | | 3 |  | | Don't forget the moving goalposts. A feature in your PerfectProgram 1.0 can be a bug in 2.0 –  [Carlos](http://programmers.stackexchange.com/users/13021/carlos) [Jan 31 '11 at 12:41](http://programmers.stackexchange.com/questions/41248/how-to-be-a-zero-bug-programmer#comment73876_41249) |   show **10** more comments | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| up vote 73 down vote | "Zero-bug programmer" is an oxymoron, like a silent singer, but past 60 or so years of programming has produced some distilled bits of wisdoms, which will make you a better programmer, such as:   * Be humble -- you are and will be making mistakes. Repeatedly. * Be fully aware of the limited size of your skull. Approach task with full humility, and avoid clever tricks like the plague. [Edsger Dijkstra] * Fight [combinatorial explosion](http://en.wikipedia.org/wiki/Combinatorial_explosion) * Get rid of mutable state (where ever possible). Yes, learn functional programming. * Reduce number of possible code paths * Understand (the magnitude of) the size of the input & output spaces (of your functions), and try to reduce them in order to get ever closer to 100% test coverage * Always assume your code is not working -- prove it otherwise!  |  |  |  | | --- | --- | --- | | [share](http://programmers.stackexchange.com/a/41258)[improve this answer](http://programmers.stackexchange.com/posts/41258/edit) | [edited Jan 31 '11 at 1:02](http://programmers.stackexchange.com/posts/41258/revisions) | community wiki  [2 revs, 2 users 95%](http://programmers.stackexchange.com/posts/41258/revisions) [Maglob](http://programmers.stackexchange.com/users/14873) | |
|  | |  |  |  |  | | --- | --- | --- | --- | | |  |  | | --- | --- | | 2 |  | | I'd be glad to prove that my code is working... but testing can only prove it's not. Are you talking formal proof or envisionning the debugging task here ? –  [Matthieu M.](http://programmers.stackexchange.com/users/4853/matthieu-m) [Jan 30 '11 at 16:40](http://programmers.stackexchange.com/questions/41248/how-to-be-a-zero-bug-programmer#comment73302_41258) | | |  |  | | --- | --- | | 1 |  | | @deworde: I understand the idea of testing, however apart from niche situations, the majority of the real work I've done could not be tested exhaustively, simply because the possible number of combinations was too big (and I'm not even adding timing issues). So, apart from niche situations, testing only go so far (it's still useful to check that at least the nominal case pass!) –  [Matthieu M.](http://programmers.stackexchange.com/users/4853/matthieu-m) [Oct 21 '11 at 17:06](http://programmers.stackexchange.com/questions/41248/how-to-be-a-zero-bug-programmer#comment211267_41258) |   show **2** more comments |
| up vote 20 down vote | **TDD**  The point of TDD is that you don't write a single line of code if there is not a test requiring that line of code. And to take it to the extreme you always start developing a new feature by writing an acceptance test. Here I have found that writing [cucumber](http://cukes.info/) style tests is ideal.  The TDD approach gives at least two benefits.   * All code is written to solve a specific feature, thus no unnecessary overproduction. * Whenever you change an existing line of code, if you break a feature, you will be notified   It doesn't prove zero bugs, as that would be impossible (have already been pointed out by countless other answers). But after having learned TDD and become good at it (yes, it is also a skill that needs practice), I have a much higher confidence in my own code because it is thoroughly tested. And more importantly, I can change existing code I don't completely comprehend without worrying about breaking functionality.  But TDD doesn't help you all the way. You cannot write bug-free code if you do not thoroughly understand the architecture of the system, and the pitfalls of that architecture. E.g. if you are writing a web application that handles multiple requests simultaneously, you must know that you cannot share mutable data between multiple requests (don't fall into the newbie trap to cache mutable data to improve performance).  I believe that the development teams that are good at TDD deliver the code with the fewest defects.   |  |  |  | | --- | --- | --- | | [share](http://programmers.stackexchange.com/a/41378)[improve this answer](http://programmers.stackexchange.com/posts/41378/edit) | [edited Jan 31 '11 at 15:23](http://programmers.stackexchange.com/posts/41378/revisions) | community wiki  [3 revs](http://programmers.stackexchange.com/posts/41378/revisions) [Pete](http://programmers.stackexchange.com/users/12902) | |
|  | show **1** more comment |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| up vote 15 down vote | | The thing is, bugs are the things that you *don't* recognize. Unless you are have some kind of encyclopedic knowledge of the programming language/compiler as well as all of the environments your application will run in, you really cannot expect to produce 100% bug-free code.  You can reduce your bug count through lots of testing, but in the end there will likely be some fringe case that will not be accounted for. Joel Spolsky wrote a particularly nice article on [bug-fixing](http://www.joelonsoftware.com/articles/fog0000000014.html).   |  |  |  | | --- | --- | --- | | [share](http://programmers.stackexchange.com/a/41250)[improve this answer](http://programmers.stackexchange.com/posts/41250/edit) | answered [Jan 29 '11 at 19:47](http://programmers.stackexchange.com/posts/41250/revisions) | community wiki  [Glenn Nelson](http://programmers.stackexchange.com/posts/41250/revisions) | |
|  | | add comment |
| up vote 8 down vote | Personally I think that striving for bug free programming seems to be more expensive (in both time and money). In order to reach zero bug, or even near-zero bug, you need to have the developers test thoroughly. This means regression test everything before submitting any code for patch review. This model doesn't strike me as being cost effective. Your better off having the developers conduct due diligent testing and leave the in-depth testing up to the QA team. Here is why:   * Developers suck at testing. It's true and you know it. (I'm a developer!) A good QA team will always find the edge cases that developers never think about. * Developers are good at coding. Let them get back to what they excel at (and usually what they prefer to do anyway.) * The QA team can find bugs related to multiple developer tasks in one pass.   Accept that when you write code, there will be bugs logged against it. That's why you have a QA process, and it's all part of being a developer. Of course this doesn't mean you submit something as soon as you write your last semi-colon. You still need to ensure quality in your work, but you **can** overdo it.  How many professions can you name that always get their task done right the first time without peer review and/or testing?   |  |  |  | | --- | --- | --- | | [share](http://programmers.stackexchange.com/a/41276)[improve this answer](http://programmers.stackexchange.com/posts/41276/edit) | answered [Jan 29 '11 at 23:30](http://programmers.stackexchange.com/posts/41276/revisions) | community wiki  [Sebastien Martin](http://programmers.stackexchange.com/posts/41276/revisions) | | |
|  | add comment | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| up vote 8 down vote | Yes, it is impossible to never have a bug in your code but it's not impossible to have less bugs. The attitude that "That's dumb, You're always gonna have bugs" is just a cop out to avoid reducing the number of bugs in your code. No one is perfect but we can and should strive to be better. In my own efforts to improve I've found the following points helpful.   * This is not easy. You will not improve over night. So don't get discouraged and don't give up. * Write less and write smarter. Less code is typically better code. It's natural to want to plan ahead and try to make awesome design patterns but in the long run just writing what you need saves time and prevents bugs. * Complexity is the enemy. Less does not count if it's an obscure complicated mess. Code golf is fun but it is hell to understand and a worse hell to debug. Whenever you write complicated code you open your self up to a world of problems. Keep things simple and short. * Complexity is subjective. Code that was once complicated becomes simple once you become a better programmer. * Experience matters. One of the two ways to become a better programmer is to practice. Practice is NOT writing programs you know how to write with ease, it's writing programs that hurt a little and make you think. * The other way to get better is to read. There are a lot of hard topics in programming to learn but you'll never be able to learn them by just programming, you need to study them. You need to read the hard stuff. Things like security and concurrency are impossible it learn correctly from just writing code unless you want to learn by cleaning up disasters. If you don't believe me look at the epic security issues sites like gawker had. If they took the time to learn how to do security correctly and not just make something that worked that mess would never have happened. * Read blogs. There are a ton of interesting blogs out there that will give you new and interesting ways to look at and think about programming this will help you to... * Learn the dirty details. The minor details of how obscure parts of your language and application work are very important. They could hold secrets that help you avoid writing complicated code or could be parts that have there own bugs you need to avoid. * Learn how the users think. Sometimes your users are flat out insane and will work with your app in ways you don't understand and can't predict. You need to get in to their heads enough to know the stranger things they might try and make sure your app can handle it.  |  |  |  | | --- | --- | --- | | [share](http://programmers.stackexchange.com/a/41467)[improve this answer](http://programmers.stackexchange.com/posts/41467/edit) | [edited Jan 31 '11 at 4:25](http://programmers.stackexchange.com/posts/41467/revisions) | community wiki  [2 revs](http://programmers.stackexchange.com/posts/41467/revisions) [AmaDaden](http://programmers.stackexchange.com/users/8121) | |
|  | add comment |
| up vote 7 down vote | Zero bugs? It sounds like [you need Lisp](http://landoflisp.com/) (follow the skeptic path and avoid the music video).  It is extraordinarily hard to achieve bug-free code in the mainstream coding environments (Java, C#, PHP, etc.). I would focus on **producing well tested and peer-reviewed code** in short controlled iterations.  **Keeping the code as simple as you can** will help you to avoid bugs.  Make sure that you are using **code analysis tools** (such as [FindBugs](http://en.wikipedia.org/wiki/FindBugs), [PMD](http://en.wikipedia.org/wiki/PMD_%28software%29) and so on) which - combined with strict compiler warnings - will reveal all sorts of issues with your code. Take note of what they are telling you, really strive to understand what the nature of the bug is, and then take steps to change your programming idiom so that it feels unnatural to code in a manner that introduces that bug again.   |  |  |  | | --- | --- | --- | | [share](http://programmers.stackexchange.com/a/41255)[improve this answer](http://programmers.stackexchange.com/posts/41255/edit) | [edited Jan 30 '11 at 23:07](http://programmers.stackexchange.com/posts/41255/revisions) | community wiki  [3 revs, 2 users 90%](http://programmers.stackexchange.com/posts/41255/revisions) [Gary Rowe](http://programmers.stackexchange.com/users/7167) | |
|  | |  |  |  |  | | --- | --- | --- | --- | | |  |  | | --- | --- | | 3 |  | | Sometimes bugs are like hidden monsters living there, they hide during my repeatedly testing and self code review... but once I do peer-review, I found the monsters were unbelievable visible and jumped out to me suddenly. It's really weird. Depending only on human 'eye' and 'brain' seems impossible to touch the bug-free line. unit-test is not workable for all cases. Code analysis tools? sounds exciting, I've never used that, I will do research on that field. –  [Elaine](http://programmers.stackexchange.com/users/4841/elaine) [Jan 30 '11 at 1:54](http://programmers.stackexchange.com/questions/41248/how-to-be-a-zero-bug-programmer#comment73073_41255) | | |  |  | | --- | --- | | 1 |  | | @Elaine Where I work is a Java environment and code can only be shared with the team if it has passed through Findbugs and PMD. New team members pass through five stages: denial, anger, bargaining, depression and then acceptance. Afterwards they never look back. –  [Gary Rowe](http://programmers.stackexchange.com/users/7167/gary-rowe) [Jan 30 '11 at 9:01](http://programmers.stackexchange.com/questions/41248/how-to-be-a-zero-bug-programmer#comment73204_41255) |   show **6** more comments |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| up vote 6 down vote | | Either don't write anything more complicated than "Hello World!" or if you do tell everyone never to use it.  Ask your boss for some examples of this so called bug free code.   |  |  |  | | --- | --- | --- | | [share](http://programmers.stackexchange.com/a/41256)[improve this answer](http://programmers.stackexchange.com/posts/41256/edit) | answered [Jan 29 '11 at 20:08](http://programmers.stackexchange.com/posts/41256/revisions) | community wiki  [JeffO](http://programmers.stackexchange.com/posts/41256/revisions) | |
|  | | add comment |
| up vote 6 down vote | As the other comments already correctly pointed out, there is no non-trivial software without bugs.  If you want to test the software always keep in mind, that tests can only prove the presence of bugs not their absence.  Depending on your domain of work you might try formal verification of your software. Using formal methods you can become pretty sure that your software exactly meets the specification.  That ofcourse does not mean that the software exactly does what you want. Writing a complete specification is in almost all cases also not possible. It basically shifts the place where errors can occur from implementation to specification.  So depending on your definition of "bugs" you can try formal verification or just try to find as many bugs as you can in your software.   |  |  |  | | --- | --- | --- | | [share](http://programmers.stackexchange.com/a/41253)[improve this answer](http://programmers.stackexchange.com/posts/41253/edit) | [edited Jan 29 '11 at 20:17](http://programmers.stackexchange.com/posts/41253/revisions) | community wiki  [2 revs](http://programmers.stackexchange.com/posts/41253/revisions) [FabianB](http://programmers.stackexchange.com/users/14928) | | |
|  | add comment | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| up vote 5 down vote | | I agree with the other ones. Here's how I would approach the problem   * Get a tester. See the Joel Test for why. * Use libraries extensively; the probably have been debugged better. I'm a big fan of CPAN for Perl.  |  |  |  | | --- | --- | --- | | [share](http://programmers.stackexchange.com/a/41279)[improve this answer](http://programmers.stackexchange.com/posts/41279/edit) | answered [Jan 29 '11 at 23:56](http://programmers.stackexchange.com/posts/41279/revisions) | community wiki  [Brian Carlton](http://programmers.stackexchange.com/posts/41279/revisions) | |
|  | | |  |  |  |  | | --- | --- | --- | --- | | |  |  | | --- | --- | | 1 |  | | … but if you use libraries, make sure their bugs can't drag you down (for example, by having the source so you can audit it or fix the bugs yourself if need be). –  [millenomi](http://programmers.stackexchange.com/users/14979/millenomi) [Jan 30 '11 at 8:44](http://programmers.stackexchange.com/questions/41248/how-to-be-a-zero-bug-programmer#comment73202_41279) |   add comment |
| up vote 5 down vote | You can strive to be a zero bug programmer. I strive to be a zero bug programmer whenever I'm writing code. However, I don't   * engage multiple testing techniques (other that ATDD) * create formal verifications of our software * have a separate QA team * do hard analysis on each change made to the code base * use a language that leans more towards safety and caution   I don't do these things because they are cost prohibitive for the software I write. If I did these things I would probably be further along towards zero bugs, but it wouldn't make business sense.  I create internal tools that a large portion of our infrastructure use. My standards for testing and coding are high. However, there is a balance. I do not expect zero bugs because I can't have people put that kind of time into one piece of work. Things might be different if I were creating the software to control an X-Ray machine, jet engines, etc. No lives are on the line if my software breaks, so we don't engage in that level of assurance.  I would match the level of assurance to the intended use of the software. If you are writing code that the NASA shuttle will use maybe zero bug tolerance is reasonable. You just need to add a bunch of additional and very expensive practices.   |  |  |  | | --- | --- | --- | | [share](http://programmers.stackexchange.com/a/41289)[improve this answer](http://programmers.stackexchange.com/posts/41289/edit) | answered [Jan 30 '11 at 0:41](http://programmers.stackexchange.com/posts/41289/revisions) | community wiki  [dietbuddha](http://programmers.stackexchange.com/posts/41289/revisions) | | |
|  | add comment | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| up vote 4 down vote | All the "Don't code at all." answers are completely missing the point. Also, your boss does definitely not seem to be a moron!  I cannot recall how often I have seen programmers who simply did not know what their code does. Their sole development philosohpy seemed to be trial and error (and quite often also copy/paste/modify). While trial and error is a valid way to go about some problems, often you can analyze the problem domain and then apply a very specific solution based on your understanding of the tools you use and with a little bit of discipline and diligence you will have not only solved the problem but also most corner-cases (potential bugs) before you deploy it for the first time. Can you guarantee that the code is bug free? Of course not. But with every bug you encounter or read about you can add it to the things you might want to think about next time you write/change something. If you do this you will consequently gain a lot of experience on how to write code that is almost bug free. - There are tons of resources available on how to become a better programmer that can help you on the journey...  Personally, I would never commit code where I cannot explain every single line. Every line has a reason to be there, otherwise it should be removed. Of course sometimes you will make assumptions of the inner workings of the methods you call, otherwise you would need to know the inner logic of the whole framework.  Your boss is completely right to say that you should understand the results and impact of the code you write on existing system. Will bugs occcur? Yes, of course. But these bugs will be due to your incomplete understanding of the system/tools you work with and with every bug fix you will have better coverage.   |  |  |  | | --- | --- | --- | | [share](http://programmers.stackexchange.com/a/41698)[improve this answer](http://programmers.stackexchange.com/posts/41698/edit) | answered [Jan 31 '11 at 6:18](http://programmers.stackexchange.com/posts/41698/revisions) | community wiki  [Patrick Klug](http://programmers.stackexchange.com/posts/41698/revisions) | |
|  | show **1** more comment |
| up vote 3 down vote | I think a good first step towards becoming a "zero-bug" programmer is to change your attitude towards bugs. Instead of saying things "they happen," "get better QA and testers," or "developers suck at testing," say:  **Bugs are not acceptable, and I will do everything within my power to eliminate them.**  Once this becomes your attitude, the bugs will drop quickly. In your search to find ways to eliminate bugs, you'll come across test-driven-development. You'll find plenty of books, blog posts, and people offering free advice on better techniques. You'll see the importance of improving your skills through **practice** (like coding katas, or trying new things at home). You'll start to perform better at work because you'll start working on your craft **at home.** And, hopefully, once you see that it **is** possible to write good software, your passion for your craft will grow.   |  |  |  | | --- | --- | --- | | [share](http://programmers.stackexchange.com/a/41423)[improve this answer](http://programmers.stackexchange.com/posts/41423/edit) | answered [Jan 30 '11 at 14:13](http://programmers.stackexchange.com/posts/41423/revisions) | community wiki  [Darren](http://programmers.stackexchange.com/posts/41423/revisions) | |
|  | add comment |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| up vote 2 down vote | | In a sense, your boss is right. It is possible to write software that approaches zero bugs.  But the problem is that the *cost* of doing writing (almost) zero-bug programs is *prohibitively high*. You need to do things like:   * Use formal specifications of your requirements. Formal, as in use of Z or VDM or some other mathematically sound notation. * Use theorem proving techniques to formally prove that your program implements the specification. * Create extensive unit, regression and system test suites and harnesses to test every which way for bugs. (And this is not sufficient in itself.) * Have *many* people review the requirements (formal and informal), software (and proofs). tests, and deployments.   It is extremely unlikely that your boss is prepared to pay for all of this ... or put up with the time it takes to do it all.   |  |  |  | | --- | --- | --- | | [share](http://programmers.stackexchange.com/a/41297)[improve this answer](http://programmers.stackexchange.com/posts/41297/edit) | [edited Jan 30 '11 at 22:54](http://programmers.stackexchange.com/posts/41297/revisions) | community wiki  [2 revs](http://programmers.stackexchange.com/posts/41297/revisions) [Stephen C](http://programmers.stackexchange.com/users/172) | |
|  | | add comment |
| up vote 2 down vote | I have attained the "zero bug" status. I tell my users it's an un documented feature, or they are asking for a new feature and therefore it's an enhancement. If neither of these are accepted responses I merely tell them they have not understood their own requirements. Thus, there are no bugs. Programmers are perfect.   |  |  |  | | --- | --- | --- | | [share](http://programmers.stackexchange.com/a/41370)[improve this answer](http://programmers.stackexchange.com/posts/41370/edit) | [edited Jan 30 '11 at 23:08](http://programmers.stackexchange.com/posts/41370/revisions) | community wiki  [2 revs, 2 users 50%](http://programmers.stackexchange.com/posts/41370/revisions) [giulio](http://programmers.stackexchange.com/users/14963) | | |
|  | add comment | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| up vote 2 down vote | Here is the steps to make a bug free-program:   1. Never start coding unless you have UNAMBIGUOUS SPECIFICATIONS for your functionality. 2. DO NOT TEST or or if you do DO NOT RELY ON TESTING to catch defects in software. 3. Apply all FEEDBACK from defects discovered during testing, reviews and production to a process and developers who inserted defect in first place. Discard all defective components completely as soon as defects are found. Update your checklists and retrain your developers so they don't make mistakes like that again.   Testing can only prove that you have bugs, but it is usually useless to prove otherwise. Regarding the feedback - if you have coin-making machine that makes coins and every 10s coin on average has a defect. You can take that coin, flatten it up and insert into machine again. coin that made out that recycled blank will be not as good, but maybe acceptable. every 100s coin will have to be re-stamped 2 times and so on. Would it be easier to fix the machine?  Unfortunately people are not machines. To make good, defect-free programmer you have to invest a lot of time, training and iterating over every defect made. Developers need to be trained in formal verification methods which are often hard to learn and apply in practice. Economics of software development is also working against it - would you invest 2 years into training a programmer who can make fewer defects just to see him jumping to another employer? You can buy machine that make perfect coins, or hire 10 more code monkeys to create bunch of tests at same cost. You can perceive this exhaustive process as your "machine", your asset - investing in extensive training of excellent developers does not pay off.  Pretty soon you will learn how to develop software of acceptable quality, but probably you will never be one who is defect free for simple reason that there is no market for developer who makes perfect code because it is slow.   |  |  |  | | --- | --- | --- | | [share](http://programmers.stackexchange.com/a/41670)[improve this answer](http://programmers.stackexchange.com/posts/41670/edit) | answered [Jan 31 '11 at 5:07](http://programmers.stackexchange.com/posts/41670/revisions) | community wiki  [Alexei Polkhanov](http://programmers.stackexchange.com/posts/41670/revisions) | | |
|  | show **1** more comment | |
| up vote 1 down vote | | Could this be a result of misunderstanding a *good* methodology, and not just generic boneheadedness?  What I mean is that it is possible that your boss heard of ["zero defect methodology"](http://www.joelonsoftware.com/articles/fog0000000043.html) (see section no.5), and didn't bother understanding what that meant? Of course, it's inconvenient for management to postpone development of new features, in favor of bugs that *you* shouldnt have put in... And of course, that threatens his bonus, so of course you wont get one because "good programmers don't have bugs"...  It's fine to *make* bugs, as long as you can *find* them and *fix* them (within reason, of course).   |  |  |  | | --- | --- | --- | | [share](http://programmers.stackexchange.com/a/41538)[improve this answer](http://programmers.stackexchange.com/posts/41538/edit) | answered [Jan 30 '11 at 20:00](http://programmers.stackexchange.com/posts/41538/revisions) | community wiki  [AviD](http://programmers.stackexchange.com/posts/41538/revisions) | |
|  | | add comment |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| up vote 0 down vote | | I only see answers about us being human and prone to err, which is very true... but I see your question from another viewpoint.  I think you *can* write bug-free programs, but those typically are programs that you have written already 10 or 12 times. The 13th time you write the same program from scratch, you already know how to do it: you know the problem, you know the techniques, you know the libraries, the language... you *see* it in your mind. All the patterns are there, at all levels.  This happens to me with very simple programs because I teach programming. They are simple for me, but hard for the students. And I am not talking about solutions to problems I have done many, many times in the blackboard. Of course I know those. I mean ~300-line programs that solve something using concepts I know really well (the concepts I teach). I write these programs with no planning and they just work, and I feel I know all the details, I don't need TDD at all. I get a couple or three compilation errors (mostly typos and other things like that) and that's it. I can do this for small programs, and I also believe that some people can do that for more complicated programs. I think people like Linus Torvalds or Daniel J. Bernstein have such clarity of mind, they are the closest you can get to a bug-free coder. If you *understand* things deeply I think you can do it. I can only do this for simple programs, like I said.  My belief is that if you always try to do programs that are far above your level (I've spent years doing just that), you will get confused and make mistakes. Big mistakes like those in which you suddenly realise that your solution cannot work, when you finally understand the problem, and have to make changes so complicated that they might stop you from solving your problem or make the code awful. TDD is for this cases, I believe. You know that you don't grok the problem you are tackling and therefore put tests everywhere to make sure that you have a strong base. TDD doesn't solve the 10,000 feet vision, though. You might walk in circles with perfectly clean code all the time.  However, if you try to do something that is new but that is *just* above your level, you might get your program perfect or almost perfect. I think it is really difficult to know what programs are in your "knowledge frontier", but in theory that is the best way to learn. I rewrite programs from scratch a lot, actually. Some people do, but you need a lot of time and patience because the third time you repeat a non-trivial program you don't get excited like the first time.  So my advice is: don't think you understand something until you can write a program bug-free just for that thing. And then try to combine two of those concepts you know deeply into the same program. I'm almost sure you will get it right the first time. One of the best ways is to rewrite non-trivial software, something that took a lot of effort the first time (I am doing this with Android apps right now). Every time I start again I change something or add stuff, just to add a little fun, and I can tell you I get better and better and better... maybe not bug-free but really proud.   |  |  |  | | --- | --- | --- | | [share](http://programmers.stackexchange.com/a/41436)[improve this answer](http://programmers.stackexchange.com/posts/41436/edit) | answered [Jan 30 '11 at 15:42](http://programmers.stackexchange.com/posts/41436/revisions) | community wiki  [Pau Fernández](http://programmers.stackexchange.com/posts/41436/revisions) | |
|  | | add comment |
| up vote 0 down vote | Program defensively: <http://en.wikipedia.org/wiki/Defensive_programming>  If someone follows the conventions of programming defensively, then changes will be easily traceable. Combine this with rigorous bug reports during development, and solid documentation, like with doxygen, and you should be able to know exactly what all of your code is doing and fix any bugs that come up, very efficiently.   |  |  |  | | --- | --- | --- | | [share](http://programmers.stackexchange.com/a/41536)[improve this answer](http://programmers.stackexchange.com/posts/41536/edit) | answered [Jan 30 '11 at 19:52](http://programmers.stackexchange.com/posts/41536/revisions) | community wiki  [Jason McCarrell](http://programmers.stackexchange.com/posts/41536/revisions) | | |
|  | add comment | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| up vote 0 down vote | imho bugs and sudden, misterious algorithm artifacts **must** appear during the coding process: they inspire and force evolution of code.  however it's possible (usually after some testing) to check every variable that might be used before declaration, to handle every error wherever it may appear - to make the program zero-bug... until you recieve a request for a feature that was considered impossible when you were discussing program architecture ;)   |  |  |  | | --- | --- | --- | | [share](http://programmers.stackexchange.com/a/41640)[improve this answer](http://programmers.stackexchange.com/posts/41640/edit) | answered [Jan 31 '11 at 0:42](http://programmers.stackexchange.com/posts/41640/revisions) | community wiki  [www0z0k](http://programmers.stackexchange.com/posts/41640/revisions) | | |
|  | add comment | |
| up vote 0 down vote | | One of the fundamental concepts of software testing is that you can NEVER be absolutely sure that the program is perfect. You can validate it forever, but that never proves that the program is complete because it quickly becomes infeasible to even test against all input / variable combinations.  Your boss seems like one of those who "doesn't understand whats so hard about programming, since its just typing"   |  |  |  | | --- | --- | --- | | [share](http://programmers.stackexchange.com/a/41714)[improve this answer](http://programmers.stackexchange.com/posts/41714/edit) | answered [Jan 31 '11 at 7:20](http://programmers.stackexchange.com/posts/41714/revisions) | community wiki  [Zaphod42](http://programmers.stackexchange.com/posts/41714/revisions) | |
|  | | add comment |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| up vote 0 down vote | | If we assume big software houses *know* how to get top notch developers (as in *zero bugs programmer*) we can deduct that Microsoft's software **must be** without bugs. Yet we know that's far from the truth.  The develop their software and when they reach certain level of low priority bugs they simply release the product and solve those later.  Unless you're developing something more complex than a simple calculator, It's not possible to avoid bugs all together. Hell even NASA has redundancy on their vehicles and bugs as well. Although they have much rigorous testing to avoid catastrophic failures. But nonetheless even they have bugs in their software.  **Bugs are inevitable** just as is human nature to err.  Having no bugs is like having a 100% secure system. If a system is 100% secure it definitely isn't useful any more (it probably lies inside tons and tons of concrete and isn't connected to the outside at all. Not wired nor wireless. So just as there are no perfectly secure system, there's no complex bug-less system.   |  |  |  | | --- | --- | --- | | [share](http://programmers.stackexchange.com/a/41743)[improve this answer](http://programmers.stackexchange.com/posts/41743/edit) | [edited Jan 31 '11 at 9:41](http://programmers.stackexchange.com/posts/41743/revisions) | community wiki  [2 revs](http://programmers.stackexchange.com/posts/41743/revisions) [Robert Koritnik](http://programmers.stackexchange.com/users/1894) | |
|  | | add comment |
| up vote 0 down vote | Maybe think more about the nature of the bugs that you get. If the bugs are generally minor oversights then a focus on better testing and a bit of code proof-reading would help.  If the bugs tend to be due to suboptimal programming decisions, though, then maybe putting more effort into better design is required. In this case, I think it's possible to be too dependent on testing to raise the quality of the software, because applying a patch to deficient code can just make future maintenance more complicated. On the one hand you get less bugs as you find and fix them, but on the other hand you prepare the ground for future bugs.  One way to judge if you have an issue with oversights or an issue with design might be to consider how much effort is required to fix the bugs. If the fixes tend to be large, or you feel that you don't understand them well, that points the figure at code design that can be improved.  That I think comes down to a sort of good taste about code, which you can develop with practice and review, and reading about people with similar problems.  Ultimately, it is futile to expect no bugs at all, but there's no harm in trying to reduce your bug count unless you already have it at some low level, and then it becomes a trade-off between your time and the time of whoever finds bugs that you don't catch.   |  |  |  | | --- | --- | --- | | [share](http://programmers.stackexchange.com/a/41758)[improve this answer](http://programmers.stackexchange.com/posts/41758/edit) | answered [Jan 31 '11 at 10:38](http://programmers.stackexchange.com/posts/41758/revisions) | community wiki  [John Bickers](http://programmers.stackexchange.com/posts/41758/revisions) | | |
|  | add comment | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| up vote 0 down vote | Your boss is right. A good programmer *should* be able to ensure that the code he or she changes is reliable, correct, and thoroughly self-verified; that he should completely understand all the results and impacts the changes will cause.  Now if you wrote *every* character of the code yourself, you should be aware of where a change needs to be made for a specific enhancement and how such a change can/will affect your existing program and whether it will cause any adverse side-effects. Its your responsibility.  Lastly remember that system tend to grow more complex with time, and eventually it becomes harder to keep a track of how the modules behave (and are expected to behave) and how they interact with others. In that case right tools and proper documentation might help you deal with the problem to some extent. Remember there are programmers that pay attention and programmers that do not -- there are no zero-bug programmers.   |  |  |  | | --- | --- | --- | | [share](http://programmers.stackexchange.com/a/41777)[improve this answer](http://programmers.stackexchange.com/posts/41777/edit) | answered [Jan 31 '11 at 11:46](http://programmers.stackexchange.com/posts/41777/revisions) | community wiki  [Salman A](http://programmers.stackexchange.com/posts/41777/revisions) | | |
|  | add comment | |
| up vote -1 down vote | | If I you mean: "zero bugs while writing the code" -> that's a nice goal but pretty impossible.  But if you mean: "zero bugs on delivered code" -> that's possible, and I worked in such environments.  All you need is: insanely high code quality and near 100% test coverage (unit tests + acceptance tests + integration tests).  In my opinion the best book to learn that is: [GOOS](http://www.growing-object-oriented-software.com/). But of course a book is not enough. You need to go to some user group and discuss about this. Courses, conferences, etc. Zero-bugs quality is not easy.  First of all you need a boss who's really interested in high quality and willing to pay for it.   |  |  |  | | --- | --- | --- | | [share](http://programmers.stackexchange.com/a/41383)[improve this answer](http://programmers.stackexchange.com/posts/41383/edit) | [edited Jan 30 '11 at 23:11](http://programmers.stackexchange.com/posts/41383/revisions) | community wiki  [2 revs, 2 users 88%](http://programmers.stackexchange.com/posts/41383/revisions) [Uberto](http://programmers.stackexchange.com/users/9115) | |
|  | | add comment |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| up vote -2 down vote | | Programmer's solution:   * Stop programming. * Build a mechanical computer. * Replace it every 5 years so that mechanical wearing does not come into play.   User's solution:   * Stop using computers. * Do everything manually. * Always have a second person double-check the results.  |  |  |  | | --- | --- | --- | | [share](http://programmers.stackexchange.com/a/41290)[improve this answer](http://programmers.stackexchange.com/posts/41290/edit) | answered [Jan 30 '11 at 1:00](http://programmers.stackexchange.com/posts/41290/revisions) | community wiki  [rwong](http://programmers.stackexchange.com/posts/41290/revisions) | |
|  | | add comment |
| up vote -2 down vote | Pair with another engineer. Write a failing test. Have every character you type be necessary to make the failing test pass. Refactor your code to make it simpler. Write another failing test, and so on an so forth.   |  |  |  | | --- | --- | --- | | [share](http://programmers.stackexchange.com/a/41366)[improve this answer](http://programmers.stackexchange.com/posts/41366/edit) | answered [Jan 30 '11 at 6:30](http://programmers.stackexchange.com/posts/41366/revisions) | community wiki  [Carl Coryell-Martin](http://programmers.stackexchange.com/posts/41366/revisions) | | |
|  | add comment | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| up vote -3 down vote | I agree that to be a zero-bug programmer you simply cannot program/code. It's part of every programmer's life to encounter and develop bug. No seasoned developer can say, hand on heart, they've never encountered a bug in their code.   |  |  |  | | --- | --- | --- | | [share](http://programmers.stackexchange.com/a/41838)[improve this answer](http://programmers.stackexchange.com/posts/41838/edit) | answered [Jan 31 '11 at 15:35](http://programmers.stackexchange.com/posts/41838/revisions) | community wiki  [dbramhall](http://programmers.stackexchange.com/posts/41838/revisions) | |
|  | add comment |

**Not the answer you're looking for? Browse other questions tagge**