

How to make junior programmers write tests? [closed]

Asked 16 years, 4 months ago Modified 1 year, 7 months ago

Viewed 11k times



109



Closed. This question needs to be more [focused](#). It is not currently accepting answers.



Want to improve this question? Update the question so it focuses on one problem only by [editing this post](#).

Closed 10 years ago.

[Improve this question](#)

We have a junior programmer that simply doesn't write enough tests.

I have to nag him every two hours, "have you written tests?"

We've tried:

- Showing that the design becomes simpler
- Showing it prevents defects
- Making it an ego thing saying only bad programmers don't

- This weekend 2 team members had to come to work because his code had a NULL reference and he didn't test it

My work requires top quality stable code, and usually everyone 'gets it' and there's no need to push tests through. We know we can make him write tests, but we all know the useful tests are those written when you're into it.

Do you know of more motivations?

unit-testing

testing

Share

Improve this question

Follow

edited Aug 23, 2008 at 17:53



Chris

6,842 ● 6 ● 53 ● 67

asked Aug 10, 2008 at 17:34



abyx

72.6k ● 19 ● 97 ● 121

16 Hire me at your company! I'd gladly leave mine for one that cared enough about the software to teach me how to better write tests. – [Steven Evers](#) Aug 7, 2009 at 17:57

@SnOrfus - I've changed jobs, sorry ;) – [abyx](#) Aug 7, 2009 at 19:00

1 @Andrew I'd say the rest have more to do with experience, while testing is more of a mind-set? – [abyx](#) May 27, 2010 at 14:55

1 This question appears to be off-topic because it is not a specific programming problem, and would be more suitable on [Software Engineering](#). – [hichris123](#) Dec 16, 2014 at 22:49

1 @hichris123 This question was posted a month before SO went out of beta, 6.5 years ago. There was no Programmers stack exchange. These rules didn't exist yet. Thanks for the down vote though. – [abyx](#) Dec 18, 2014 at 9:29

24 Answers

Sorted by:

Highest score (default)



This is one of *the hardest things* to do. Getting your people to *get it*.

128



Sometimes one of the best ways to help junior level programmers 'get it' and learn the right techniques from the seniors is to do a bit of pair programming.



Try this: on an upcoming project, pair the junior guy up with yourself or another senior programmer. They should work together, taking turns "driving" (being the one typing at the keyboard) and "coaching" (looking over the shoulder of the driver and pointing out suggestions, mistakes, etc as they go). It may seem like a waste of resources, but you will find:

1. That these guys together can produce code plenty fast and of higher quality.
2. If your junior guy learns enough to "get it" with a senior guy directing him along the right path (eg. "Ok, now before we continue, let's write a test for this

function.") It will be well worth the resources you commit to it.

Maybe also have someone in your group give the [Unit Testing 101](#) presentation by Kate Rhodes, I think its a great way to get people excited about testing, if delivered well.

Another thing you can do is have your Jr. Devs practice the [Bowling Game Kata](#) which will help them learn Test Driven Development. It is in java, but could easily be adapted to any language.

Share Improve this answer

Follow

edited Jul 1, 2015 at 14:40



Stephan

42.9k ● 66 ● 244 ● 339

answered Aug 10, 2008 at 18:17



Justin Standard

21.5k ● 22 ● 82 ● 90

The Bowling Game Kata is good! – [Hertanto Lie](#) Dec 12, 2008 at 1:38

The Unit Testing 101 link is broken. I tried searching for webpage archive but found nothing useful. Anyone got this presentation? – [displayName](#) Sep 3, 2015 at 15:33



21

Have a code review before every commit (even if it's a 1 minute "I've changed this variable name"), and as part of the code review, review any unit tests.



Don't sign off on the commit until the tests are in place.



(Also - If his work wasn't tested - why was it in a production build in the first place? If it's not tested, don't let it in, then you won't have to work weekends)

Share Improve this answer

edited Aug 11, 2008 at 2:45

Follow

answered Aug 10, 2008 at 22:59



RodeoClown

13.8k ● 13 ● 55 ● 56



For myself, I have started insisting that every bug I find and fix be expressed as a test:

20



1. "Hmmm, that's not right..."
2. Find possible problem
3. Write a test, show that the code fails
4. Fix the problem
5. Show that the new code passes
6. Loop if the original problem persists

I try to do this even while banging stuff out, and I get done in about the same time, only with a partial test suite already in place.

(I don't live in a commercial programming environment, and am often the only coder working a particular project.)

answered Aug 22, 2008 at 15:46

**dmckee --- ex-moderator**
kitten

101k ● 25 ● 146 ● 235

1 +1 I think this is an excellent low impact way to start testing. This way, known bugs never get re-introduced accidentally.
– Jonathan Apr 8, 2009 at 9:00

4 It's called regression testing! :) – pfctdayelise Oct 13, 2009 at 12:33

**16**

Imagine I am a mock programmer, named... Marco. Imagine I have graduated school not that long ago, and never really had to write tests. Imagine I work in a company that doesn't really enforce or asks for this. OK? good! Now imagine, that the company is switching to using tests, and they are trying to get me inline with this. I will give somewhat snarky reaction to items mentioned so far, as if I didn't do any research on this.

Let's get this started with the creator:

Showing that the design becomes simpler.

How can writing more, make things simpler. I would now have to keep tabs on getting more cases, and etc. This

makes it more complicated if you ask me. Give me solid details.

Showing it prevents defects.

I know that. This is why they are called tests. My code is good, and I checked it for issues, so I don't see where those tests would help.

Making it an ego thing saying only bad programmers don't.

Ohh, so you think I am a bad programmer just because I don't do as much used testing. I'm insulted and positively annoyed at you. I would rather have assistance and support than sayings.

@[Justin Standard](#): On start of new propect pair the junior guy up with yourself or another senior programmer.

Ohh, this is so important that resources will be spent making sure I see how things are done, and have some assist me on how things are done. This is helpful, and I might just start doing it more.

@[Justin Standard](#): Read [Unit Testing 101](#) presentation by Kate Rhodes.

Ahh, that was an interesting presentation, and it made me think about testing. It hammered some points in that I should consider, and it might have swayed my views a bit.

I would love to see more compelling articles, and other tools to assist me in getting in line with thinking this is the right way to do things.

| @[Dominic Cooney](#): Spend some time and share testing techniques.

Ahh, this helps me understand what is expected of me as far as techniques, and it puts more items in my bag of knowledge, that I might use again.

| @[Dominic Cooney](#): Answer questions, examples and books.

Having a point person (people) to answer question is helpful, it might make me more likely to try. Good examples are great, and it gives me something to aim for, and something to look for reference. Books that are relevant to this directly are great reference.

| @[Adam Hayle](#): Surprise Review.

Say what, you sprung something that I am completely unprepared for. I feel uncomfortable with this, but will do

my best. I will now be scared and mildly apprehensive of this coming up again, thank you. However, the scare tactic might have worked, but it does have a cost. However, if nothing else works, this might just be the push that is needed.

| @[Rytmis](#): Items are only considered done when they have test cases.

Ohh, interesting. I see I really do have to do this now, otherwise I'm not completing anything. This makes sense.

| @[jmorris](#): Get Rid / Sacrifice.

glares, glares, glares - There is a chance I might learn, and with support, and assistance, I can become a very important and functional part of the teams. This is one of my handicaps now, but it won't be for long. However, if I just don't get it, I understand that I will go. I think I will get it.

In the end, the support of my team will play a large part in all this. Having a person take their time to assist, and get me started into good habits is always welcome. Then, afterward having a good support net would be great. It would always be appreciated to have someone come a few times afterward, and go over some code, to see how everything is flowing, not in a review per se, but more as a friendly visit.

Reasoning, Preparing, Teaching, Follow up, Support.

Share Improve this answer

edited May 23, 2017 at 11:54

Follow



Community Bot

1 ● 1

answered Aug 11, 2008 at 2:07



Marcio DaSilva

161 ● 1 ● 6



12



I've noticed that a lot of programmers see the value of testing on a rational level. If you've heard things like "yeah, I know I should test this but I really need to get this done quickly" then you know what I mean. However, on an emotional level they feel that they get something done only when they're writing the real code.

The goal, then, should be to somehow get them to understand that testing is in fact the **only** way to measure when something is "done", and thus give them the intrinsic motivation to write tests.

I'm afraid that's a lot harder than it should be, though. You'll hear a lot of excuses along the lines of "I'm in a real hurry, I'll rewrite/refactor this later and then add the tests" -- and of course, the followup never happens because, surprisingly, they're *just as busy the next week*.

[Share](#) [Improve this answer](#)

[Follow](#)

answered Aug 10, 2008 at 18:00



[Rytmis](#)

32k ● 8 ● 61 ● 69



9



Here's what I would do:

- First time out... "we're going to do this project jointly. I'm going to write the tests and you're going to write the code. Pay attention to how I write the tests, coz that's how we do things around here and that's what I'll expect of you."
- Following that... "You're done? Great! First let's look at the tests that are driving your development. Oh, no tests? Let me know when that is done and we'll reschedule looking at your code. If you're needing help to formulate the tests let me know and I'll help you."

Share Improve this answer

answered Aug 10, 2008 at 19:53

Follow



[Mark Harrison](#)

304k ● 131 ● 350 ● 489



6



He's already doing this. Really. He just doesn't write it down. Not convinced? Watch him go through the standard development cycle:

- Write a piece of code
- Compile it
- Run to see what it does
- Write the next piece of code

Step #3 is the test. He already does testing, he just does it manually. Ask him this question: "How do you know

tomorrow that the code from today still works?" He will answer: "It's such a little amount of code!"

Ask: "How about next week?"

When he hasn't got an answer, ask: "How would you like your program to tell you when a change breaks something that worked yesterday?"

That's what automatic unit testing is all about.

Share Improve this answer

answered Feb 24, 2009 at 14:25

Follow



[Aaron Digulla](#)

328k ● 110 ● 622 ● 834



5



As a junior programmer, I'm still trying to get into the habit of writing tests. Obviously it's not easy to pick up new habits, but thinking about what would make this work for me, I have to +1 the comments about code reviews and coaching/pair programming.



It may also be worth emphasising the long-term purpose of testing: ensuring that what worked yesterday is still working today, and next week, and next month. I only say that because in skimming the answers I didn't see that mentioned.

In doing code reviews (if you decide to do that), make sure your early-career dev knows it's not about putting them down, it's about *making the code better*. Because that way their confidence is less likely to get damaged.

And that's important. On the other hand, so is knowing how little you know.

Of course, I don't really know anything. But I hope the words have been useful.

Edit: [[Justin Standard](#)]

Don't put yourself down, what you have to say is pretty much right on.

On your point about code reviews: what you will find is that not only will the junior dev learn in the process, but so will the reviewers. Everyone in a code review learns if you make it a collaborative process.

Share Improve this answer

edited May 1, 2023 at 1:05

Follow

answered Aug 14, 2008 at 0:28




[Lucas Wilson-Richter](#)

2,314 ● 1 ● 19 ● 24

-
- 2 I agree with the comments above, but would urge people to be a little less formal with code reviews, I had a code review sprung on me when I was a junior without knowing about it previously. The reviewer sat another dev and me down and took out pages of code with red pen scrawlel over it. It made both of the devs feel slightly betrayed and we both felt it was an excercise to demean us. I would recommend more informal chats walking through the code so that something

can be learned instead of finger pointing. – [Burt](#) Jan 24, 2010 at 16:04

I totally agree, Burt. Code reviews should be anything but intimidating or demeaning. That said, they should still leave the code improved. But having some code that runs a little slower is nowhere near as harmful as spending good money on a junior dev who won't do anything because they're afraid they'll cop hell in a review. – [Lucas Wilson-Richter](#) Feb 3, 2010 at 4:56 



4



Frankly, if you are having to put *that* much effort into getting him to do something then you may have to come to terms with the thought that he may just not be a good fit for the team, and may need to go. Now, this doesn't necessarily mean firing him... it may mean finding someplace else in the company his skills are more suited. But if there is no place else...you know what to do.

I'm assuming he is also a fairly new hire (< 1 year) and probably recently out of school...in which case he may not be accustomed to how things work in a corporate setting. Things like that most of us could get away with in college.

If this is the case, one thing I've found works is to have a sort of "surprise new hire review." It doesn't matter if you've never done it before...he won't know that. Just sit him down and tell him your are going to go over his performance and show him some real numbers...take your normal review sheet (you do have a formal review process right?) and change the heading if you want so it

looks official and show him where he stands. If you show him in a very formal setting that not doing tests is adversely affecting his performance rating as opposed to just "nagging" him about it, he will hopefully get the point. You've got to show him that his actions will actually affect him be it pay wise or otherwise.

I know, you may want to stay away from doing this because it's not official... but I think you are within reason to do it and it's probably going to be a whole lot cheaper than having to fire him and recruit someone new.

Share Improve this answer

answered Aug 10, 2008 at 17:58

Follow



Adam Haile

31.3k ● 60 ● 195 ● 290



4

As a junior programmer myself, I thought that I'd reveal what it was like when I found myself in a similar situation to your junior developer.



When I first came out of uni, I found that it had severely un-equipped me to deal with the real world. Yes I knew some JAVA basics and some philosophy (don't ask) but that was about it. When I first got my job it was a little daunting to say the least. Let me tell you I was probably one of the biggest cowboys around, I would hack together a little bug fix / algorithm with no comments / testing / documentation and ship it out the door.

I was lucky enough to be under the supervision of a kind and *very* patient senior programmer. Luckily for me, he

decided to sit down with me and spend 1-2 weeks going through my very hacked together code. He would explain where I'd gone wrong, the finer points of C and pointers (boy did that confuse me!). We managed to write a pretty decent class/module in about a week. All I can say is that if the senior dev hadn't invested the time to help me along the right path, I probably wouldn't have lasted very long.

Happily, 2 years down the line, I would hope that some of my colleagues might even consider me an average programmer.

Take home points

1. Most Universities are very bad at preparing students for the real world
2. Paired programming really helped me. That's not to say that it will help everyone but it worked for me.

Share Improve this answer

answered Aug 18, 2008 at 19:42

Follow



TK.

47.8k ● 47 ● 121 ● 148



3



Assign them to projects that don't require "top quality stable code" if that's your concern and let the jr. developer fail. Have them be the one to 'come in on the weekend' to fix their bugs. Have lunch a lot and talk about software development practices (not lectures, but discussions). In time they will acquire and develop the best practices to do the tasks they are assigned.



Who knows, they might even come up with something better than the techniques your team currently uses.

Share Improve this answer

answered Sep 18, 2008 at 1:05

Follow



Jeff

1,053 ● 1 ● 8 ● 14



If the junior programmer, or anyone, doesn't see the value in testing, then it will be hard to get them to do it...period.

2



I would have made the junior programmer sacrifice their weekend to fix the bug. His actions (or lack there of) are not affecting him directly. Also, make it apparent, that he will not see advancement and/or pay increases if he doesn't improve his skills in testing.



In the end, even with all your help, encouragement, mentoring, he might not be a fit for your team, so let him go and look for someone who does get it.

Share Improve this answer

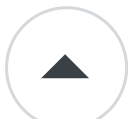
answered Aug 10, 2008 at 17:49

Follow



jsmorris

319 ● 2 ● 11



I second RodeoClown's comment about code reviewing every commit. Once he's done it a fair few times he'll get in the habit of testing stuff.

2



I don't know if you need to block commits like that though. At my workplace everyone has free commit to everything,



and all SVN commit messages (with diffs) are emailed to the team.



Note: you *really* want the [thunderbird colored-diffs addon](#) if you plan on doing this.

My boss or myself (the 2 'senior' coders) will end up reading over the commits, and if there's any stuff like "you forgot to add unit tests" we just flick an email or go and chat to the person, explaining why they needed unit tests or whatever. Everyone else is encouraged to read the commits too, as it's a great way of seeing what's going on, but the junior devs don't comment so much.

You can help encourage people to get into the habit of this by periodically saying things like "Hey, bob, did you see that commit I did this morning, I found this neat trick where you can do blah blah whatever, read the commit and see how it works!"

NB: We have 2 'senior' devs and 3 junior ones. This may not scale, or you might need to adjust the process a bit with more developers.

Share Improve this answer

Follow

answered Aug 11, 2008 at 0:35



[Orion Edwards](#)

123k ● 66 ● 245 ● 339



It's his Mentor's responsibility to Teach him/her. How well are you teaching him/her HOW to test. Are you pair



programming with him? The Junior more than likely doesn't know HOW to set up a good test for xyz.



As a Junior freshout of school he knows many Concepts. Some technique. Some experience. But in the end, all a Junior is POTENTIAL. Almost every feature they work on, there will be something new that they have never done before. Sure the Junior may have done a simple State pattern for a project in class, opening and shutting "doors", but never a real world application of the patterns.

He/she will only be as good as how well you teach. If they were able to "Just get it" do you think they would have taken a Junior position in the first place?

In my experience Juniors are hired and given almost same responsibility as Seniors, but are just paid less and then ignored when they start to falter. Forgive me if i seem bitter, it's 'cause i am.

Share Improve this answer

answered Aug 13, 2008 at 19:20

Follow



[Brian Leahy](#)

35.4k ● 12 ● 46 ● 60



2



1. Make code coverage part of the reviews.
2. Make "write a test that exposes the bug" a prerequisite to fixing a bug.
3. Require a certain level of coverage before code can be checked in.



4. Find a good book on test-driven development and use it to show how test-first can speed development.

Share Improve this answer

answered Aug 29, 2008 at 18:55

Follow



[joel.neely](#)

30.9k ● 9 ● 57 ● 64



2



Change his job description for a while to solely be writing tests and maintaining tests. I've heard that many companies do this for fresh new inexperienced people for a while when they start.

Additionally, issue a challenge while he's in that role:

Write tests that will a) fail on current code a) fulfill the requirements of the software. Hopefully it'll cause him to create some solid and thorough tests (improving the project) and make him better at writing tests for when he re-integrates into core development.

edit> fulfill the requirements of the software meaning that he's not just writing tests to purposely break the code when the code never intended or needed to take that test case into account.

Share Improve this answer

answered Feb 24, 2009 at 14:35

Follow



[Steven Evers](#)

17.2k ● 22 ● 81 ● 132



If your colleague lacks experience writing tests maybe he or she is having difficulty testing beyond simple situations,

1

and that is manifesting itself as inadequate testing. Here's what I would try:



- Spend some time and share testing techniques, like dependency injection, looking for edge cases, and so on with your colleague.
- Offer to answer questions about testing.
- Do code reviews of tests for a while. Ask your colleague to review changes of yours that are exemplary of good testing. Look at their comments to see if they're really reading and understanding your test code.
- If there are books that fit particularly well with your team's testing philosophy give him or her a copy. It might help if your code review feedback or discussions reference the book so he or she has a thread to pick up and follow.

I wouldn't especially emphasize the shame/guilt factor. It is worth pointing out that testing is a widely adopted, good practice and that writing and maintaining good tests is a professional courtesy so your team mates don't need to spend their weekends at work, but I wouldn't belabor those points.

If you really need to "get tough" then institute an impartial system; nobody likes to feel like they're being singled out. For example your team might require code to maintain a certain level of test coverage (able to be gamed, but at least able to be automated); require new code to have

tests; require reviewers to consider the quality of tests when doing code reviews; and so on. Instituting that system should come from team consensus. If you moderate the discussion carefully you might uncover other underlying reasons your colleague's testing isn't what you expect.

Share Improve this answer

answered Aug 10, 2008 at 18:19

Follow



Dominic Cooney

6,505 ● 1 ● 27 ● 39



1



@ jsmorris

I once had the senior developer and "architect" berate me and a tester(it was my first job out of college) in email for not staying late and finishing such an "easy" task the night before. We had been at it all day and called it quits at 7pm, I had been thrashing since 11am before lunch that day and had pestered every member our team for help at least twice.

I responded and cc'd the team with: "I've been disappointed in you for a month now. I never get help from the team. I'll be at the coffee shop across the street if you need me. I'm sorry i couldn't debug the 12 parameter, 800 line method that just about everything is dependent on."

After cooling off at the coffee shop for an hour, i went back in the office, grabbed my crap and left. After a few

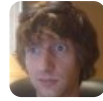
days they called me asking if I was coming in, I said I would but I had an interview, maybe tomorrow.

"So your quitting then?"

Share Improve this answer

answered Aug 13, 2008 at 19:34

Follow



Brian Leahy

35.4k ● 12 ● 46 ● 60



1



On your source repository : use hooks before each commits (pre-commit hook for SVN for example)

In that hook, check for the existence of at least one use case for each method. Use a convention for unit test organisation that you could easily enforce via a pre-commit hook.

On an integration server compile everything and check regularly the test coverage using a test coverage tool. If test coverage is not 100% for a code, block any commit of the developer. He should send you the test case that covers 100% of the code.

Only automatic checks can scale well on a project. You cannot check everything by hand.

The developer should have a mean to check if his test cases covers 100% of the code. That way, if he doesn't commit a 100% tested code, it is his own fault, not a "oops, sorry I forgot" fault.

Remember : People never do what you expect, they always do what you inspect.

Share Improve this answer

answered Aug 29, 2008 at 19:19

Follow



Jérôme Radix

10.5k ● 4 ● 36 ● 42



1



Lots of psychology and helpful "mentoring" techniques but, in all honesty, this just boils down to "write tests if you want to still have a job, tomorrow."

You can couch it in whatever terms you think are appropriate, harsh or soft, it doesn't matter. But the fact is, programmers are not paid to just throw together code & check it in -- they're paid to carefully put together code, then put together tests, then test their code, THEN check the whole thing in. (At least that's what it sounds like, from your description.)

Hence, if someone is going to refuse to do their job, explain to them that they can stay home, tomorrow, and you'll hire someone who WILL do the job.

Again, you can do all this gently, if you think that's necessary, but a lot of people just need a big hard slap of *Life In The Real World*, and you'd be doing them a favor by giving it to them.

Good luck.

Share Improve this answer

answered Nov 7, 2008 at 7:13

Follow



Olie

24.7k ● 18 ● 101 ● 132



1



First off, like most respondents here point out, if the guy doesn't see the value in testing, there's not much you can do about it, and you've already pointed out that you can't fire the guy. However, failure is not an option here, so what about the few things you **can** do?



If your organization is large enough to have over 6 developers, I strongly recommend having a Quality Assurance department (even if its just one person to start). Ideally, you should have a ratio of 1 tester to 3-5 developers. The thing about programmers is ... they are programmers, not testers. I have yet to interview a programmer that has been formally taught proper QA techniques.

Most organizations make the fatal flaw of assigning the testing roles to the new-hire, the person with the LEAST amount of exposure to your code -- ideally, the senior developers should be moved to the QA role as they have the experience in the code, and (hopefully) have developed a sixth sense for code smells and failure points that can crop up.

Furthermore, the programmer that made the mistake is probably not going to find the defect because its usually not a syntax error (those get picked up in the compile), but a logic error -- and the same logic is at work when they write the test as when they write the code. Don't

have the person who developed the code test that code -- they'll find less bugs than anyone else would.

In your case, if you can afford the redirected work effort, make this new guy the first member of your QA team. Get him to read "Software Testing In The Real World: Improving The Process", because he obviously will need some training in his new role. If he doesn't like it, he'll quit and your problem is still solved.

A slightly less vengeful approach would be let this person do what they are good at (I'm assuming this person got hired because they are actually competent at the programming part of the job) , and hire a tester or two to do the testing (University students often have practicum or "co-op" terms, would love the exposure, and are cheap)

Side Note: Eventually, you'll want the QA team reporting to a QA director, or at least not to a software developer manager, because having the QA team report to the manager who's primary goal is to get the product done is a conflict of interest.

If your organization is smaller than 6, or you can't get away with creating a new team, I recommend paired programming (PP). I'm not a total convert of all the extreme programming techniques, but I'm definitely a believer in paired programming. However, both members of the paired programming team have to be dedicated, or it simply doesn't work. They have to follow two rules: the inspector has to fully understand what is being coded on

the screen or he has to ask the coder to explain it; the coder can only code what he can explain -- no "you'll see" or "trust me" or hand-waving will be tolerated.

I only recommend PP if your team is capable of it, because, like testing, no amount of cheering or threatening will persuade a couple of ego-filled introverts to work together if they don't feel comfortable doing so. However, I find that between the choice of writing detailed functional specs and doing code reviews vs. paired programming, the PP usually wins out.

If PP is not for you, then TDD is your best bet, but only if its taken literally. Test Driven Development mean you write the tests FIRST, run the tests to prove they actually do fail, then write the simplest code to make it work. The trade off is now you (should) have a collection of thousands of tests, which is also code, and is just as likely as production code to contain bugs. I'll be honest, I'm not a big fan of TDD, mainly because of this reason, but it works for many developers who would rather write test scripts than test case documents -- some testing is better than none. Couple TDD with PP for a better likelihood of test coverage and less bugs in the script.

If all else fails, have the programmers equivalence of a swear jar -- each time the programmer breaks the build, they have to put \$20, \$50, \$100 (whatever is moderately painful for your staff) into a jar that goes to your favorite (registered!) charity. Until they pay up, shun them :)

All joking aside, the best way to get your programmer to write tests is don't let him program. If you want a programmer, hire a programmer -- If you want tests, hire a tester. I started as a junior programmer 12 years ago doing testing, and it turned into my career path, and I wouldn't trade it for anything. A solid QA department that is properly nurtured and given the power and mandate to improve the software is just as valuable as the developers writing the software in the first place.

Share Improve this answer

answered Nov 7, 2008 at 8:58

Follow



[dennisjbell](#)

326 ● 2 ● 5



0



This may be a bit heartless, but the way you describe the situation it sounds like you need to fire this guy. Or at least make it clear: refusing to follow house development practices (including writing tests) *and* checking in buggy code that other people have to clean up will eventually get you fired.



Share Improve this answer

answered Aug 18, 2008 at 20:12

Follow



[Chris Conway](#)

56k ● 43 ● 131 ● 155



0

The main reason junior engineers/programmers don't take lots of time to design and perform test scripts, is because most CS certifications do not heavily require this, so other areas of engineering are covered further in college programs, such as design patterns.



In my experience, the best way to get the junior professionals into the habit, is to make it part of the process explicitly. That is, when estimating the time an iteration should take, the time of design, write and/or execute the cases should be incorporated into this time estimate.

Finally, reviewing the test script design should be part of a design review, and the actual code should be reviewed in the code review. This makes the programmer liable for doing proper testing of each line of code he/she writes, and the senior engineer and peers liable to provide feedback and guidance on the code and test written.

Share Improve this answer

answered Aug 27, 2008 at 3:52

Follow



[axs6791](#)

707 ● 1 ● 6 ● 12



0



Based on your comment, "Showing that the design becomes simpler" I'm assuming you guys practice TDD. Doing a code review after the fact is not going to work. The whole thing about TDD is that it's a design and not a testing philosophy. If he didn't write the tests as part of the design, you aren't going to get a lot of benefit from writing tests after the fact - especially from a junior developer. He'll end up missing a whole lot of corner cases and his code will still be crappy.

Your best bet is to have a **very** patient senior developer to sit with him and do some pair programming. And just

keep at it until he learns. Or doesn't learn, in which case you need to reassign him to a task he is better suited to because you will just end up frustrating your real developers.

Not everyone has the same level of talent and/or motivation. Development teams, even agile ones, are made up of people on the "A-Team" and people on "B-Team". A-Team members are the one who architect the solution, write all the non-trivial production code, and communicate with the business owners - all the work that requires thinking outside the box. The B-Team handle things like configuration management, writing scripts, fixing lame bugs, and doing maintenance work - all the work that has strict procedures that have small consequences for failure.

Share Improve this answer

Follow

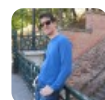
edited Nov 22, 2009 at 21:42



abyx

72.6k ● 19 ● 97 ● 121

answered Aug 13, 2008 at 23:55



Jim

3,170 ● 4 ● 32 ● 38
