Debugging is a bad smell - how to persuade them?

Asked 16 years, 1 month ago Modified 8 years, 8 months ago

Viewed 3k times

15

I've been working on a project that can't be described as 'small' anymore (40+ months), with a team that can't be defined as 'small' anymore (~30 people). We've been using Agile/Scrum (1) practices all along, and a healthy dose of TDD.

I'm not sure if I picked this up from Agile or TDD, more likely a combination of the two, but I'm now clearly in the camp of people that looks at debugging as a bad smell. By 'debugging' I'm not referring to the more abstract concept of figuring out what might be wrong with the system, but the specific activity of running the system in Debug mode, stepping through the code to figure out details that are otherwise inscrutable.

Since I'm fairly convinced, this question is not about whether debugging is a bad smell or not. Rather, I'd like to know how I can persuade my team-mates about this.

People that believe debugging mode is the 'standard' mode tend to write code that can be understood only by debugging through it, which leads to a lot of time wasted

since every time you work an item on top of code developed by someone else, you get to first spend a considerable amount of time debugging it (and, since there's no bug involved.. the term is becoming increasingly ridiculous) - and then silos happen. So I'd love to convince a few of my team-mates that avoiding debug mode is a Good Thing (2). Since they are used to live in Debug mode, however, they don't seem to see the problem; to them, spending hours debugging someone else code before they even start doing anything related to their new item is the norm; they don't see anything wrong with it. Plus, as they spend time 'figuring it out' they know eventually the developer that worked that area will become available and the item will be passed on to them (leading to yet another silo).

Help me come up with a plan to turn them from the Dark Side !

Thanks in advance.

(1) Also referred to as SCRUM (all caps). Capitalization arguments aside, I think an asterisk after the term must be used since - unsurprisingly - our organization 'tweaked' the Agile and Scrum process to fit the perceived needs of all stakeholders involved. So, in all honesty, I won't pretend this has been 100% according to theory, but that's beside the point of my question.

(2) Yes, there will always be times when we'll *have* to get in debug mode, I'm not trying to absolutely avoid it, just..

trying to minimize the number of times we have to dive into it.

debugging tdd

Share

Improve this question

Follow

asked Nov 1, 2008 at 20:13

FOR

4,358 ● 2 ● 27 ● 36

13 Answers

Sorted by: Highest score (default)

17

If you want to persuade your coworkers that your programming practices are better, first demonstrate by your productiveness that you are more effective than they are, at least for some tasks. Then they'll believe you when you explain how you get so much done.

It's also sometimes easier to focus on something concrete. Do your coworkers even talk in terms of "code smell"? Perhaps you could focus on specifics like "When the ABC module fails, it takes forever to debug it; it's much faster to use technique XYZ. Here, let me demonstrate." Then afterwards you can mention your basic principle, which is yeah the debugger is a useful tool, but there's usually other more useful ones.

Share Improve this answer

Follow

answered Nov 1, 2008 at 20:20

lacker

5,550 ● 6 ● 37 ● 39

+1: thanks for the suggestion; sadly it doesn't seem the devs are interested in being more productive (they get paid the same anyway). – FOR Nov 1, 2008 at 21:28

8

This is a cross-post, because the first time around it was more of an aside on someone else's answer to a different question. To this question it's a direct answer.

> Debugging degrades the quality code of the code we produce because it allows us to get away with a lower level of preparation and less mental discipline. I learnt this from an accidental controlled experiment in early 2000, which I now relate:
>
> I took on a contract as a Delphi coder, and the first task assigned was to write a template engine conceptually similar to a reporting engine - using Java, a language with which I was unfamiliar.
>
> Bizarrely, the employer was quite happy to pay me contract rates to spend months becoming proficient with a new language, but wouldn't pay for books or debuggers. I was told to download the compiler and learn using online resources (Java Trails were pretty good).
>
> The golden rule of arts and sciences is that whoever has the gold makes the rules, so I

> proceeded as instructed. I got my editor macros rigged up so I could launch the Java compiler on the current edit buffer with a single keystroke, I found syntax-colouring definitions for my editor and I used regexes to parse the compiler output and put my cursor on the reported location of compile errors. When the dust settled, I had a little IDE with everything but a debugger.
>
> To trace my code I used the good old fashioned technique of inserting writes to the console that logged position in the code and the state of any variables I cared to inspect. It was crude, it was time-consuming, it had to be pulled out once the code worked and it sometimes had confusing side-effects (eg forcing initialisation earlier than it might otherwise have occurred resulting in code that only works while the trace is present).
>
> Under these conditions my class methods got shorter and more and more sharply defined, until typically they did exactly one very well defined operation. They also tended to be specifically designed for easy testing, with simple and completely deterministic output so I could test them independently.
>
> The long and the short of it is that when debugging is more painful than designing, the path of least resistance is better design.

> What turned this from an observation to a certainty was the success of the project. Suddenly there was budget and I had a "proper" IDE with an integrated debugger. Over the course of the next two weeks I noticed a reversion to prior habits, with "sketch" code made to work by iterative refinement in the debugger.
>
> Having noticed this I recreated some earlier work using a debugger in place of thoughtful design. Interestingly, taking away the debugger slowed development only slightly, and the finished code was vastly better quality particularly from a maintenance perspective.
>
> Don't get me wrong: there is a place for debuggers. Personally, I think that place is in the hands of the team leader, to be brought out in times of dire need to figure out a mystery, and then taken away again before people lose their discipline.
>
> People won't want to ask for it because that would be an admission of weakness in front of their peers, and the act of explaining the need and the surrounding context may well induce peer insights that solve the problem - or even better designs free from the problem.

So, FOR, I not only agree with your position, I have real data from a controlled experiment to support it. It is, however, a rather small sample. More elaborate tests are required before my conclusions are supportable.

Why don't you take what I've said to your team and suggest trials. You have more data than they do (I just gave it to you) and in order to have a credible basis for disagreeing with you they basically have to test the idea, and the only way to do that is to give your idea a go.

You should be ready for it to all fall apart, though, because the whole thing is predicated on the assumption that the developers have the talent and experience to rise to the challenge of stronger design in the absence of step-through debugging.

Step-through debugging was created to make debugging easier. The direct effect of lowering the bar is that people with less talent can participate - if you build a tool that even jackasses can use, you will get jackasses using it -- a lot of them, if the newly accessible activity is well-remunerated.

This causes an exodus of people with talent because they generally use that talent to do rare and precious things in order to be well paid without working too hard, and the market doesn't want to pay for excellence because it cannot distinguish talent well enough to know when paying for it is justified.

Another thought: more recent work with problems on production servers, where it was impossible to install a debugger, has shown the importance of having a codebase for which maintenance doesn't depend on the availability of a debugger. Code that's grown in the absence of debuggers is much less hassle. Choose not to use them when you can change your mind, and then when you can't change your mind it won't be so awful.

Share Improve this answer

Follow

edited May 23, 2017 at 12:13

Community Bot

1 • 1

answered Nov 2, 2008 at 12:10

Peter Wone

18.7k • 14 • 94 • 147

Ah.. direct, practical, concrete experience of the issue at hand - thank you for sharing; this sort of things might help me and my team out! – FOR Nov 2, 2008 at 13:18

I'd give you an extra +1 if I could, for remembering this topic 4 years later :) Thanks for the added info. The "can't use a debugger" scenario is very intriguing. – FOR Jun 28, 2012 at 0:42

6

> Since I'm fairly convinced, this question is not about whether debugging is a bad smell or not.

Well, your local Church might be more appropriate place for your question then.

That aside, convince them by arguments. You might want to reconsider your fundamentalist stance, however, because this is the very opposite of persuasive. One thing you might want to do is drop the term “debugging” in your whole discussion and replace it by “stepping through the code” or the likes, emphasizing that you oppose the uninformend guesswork/patchwork practice of probing that you condemn rather than an informed reflection about the code.

(I would still disagree with you, but that's besides the point since you didn't want a discussion.)

Share Improve this answer

Follow

answered Nov 1, 2008 at 20:19

Konrad Rudolph

545k ● 139 ● 956 ● 1.2k

Are you implying that people of faith are unwilling to engage in real discussions? Perhaps I'm misinterpreting your comment about the Church, but it seems uncalled for if you mean what I think you mean. (There are plenty of atheists who are firmly convinced and won't listen to discussion either...) – Jon Skeet Nov 1, 2008 at 20:29

@Jon, this wasn't a snide remark on Churches. I think most people agree that strongly held convictions are usual/necessary in Church but rather less useful in a technical context where absolutes are rare. So no, I'm actually not implying what you think. – Konrad Rudolph Nov 1, 2008 at 20:47

Goodo - feel free to delete this comment, my original one and yours then :) – Jon Skeet Nov 1, 2008 at 20:58

I respect different opinions, but wanted to get some answer to my actual question. Sorry if it sounded close-minded. – FOR Nov 1, 2008 at 21:30

1 1. He said he was "convinced", and I like to think the best of people (that he is convinced by reason, not dogma); 2. he didn't say he'd never discuss it; just that he intends this to be a different discussion. It's hard to base a question on a controversial premise. People can't resist. – Alan Hensel Nov 1, 2008 at 23:55

5

I think the real problem here is

> People that believe debugging mode is the 'standard' mode tend to write code that can be understood only by stepping through it

This, if true, should be self evidently wrong and there should be no need to discuss it. If it's not evident it's because they don't see how the badly written code could be improved. Show them, do code reviews where you show how that code could be refactored in a way that is clear without stepping through it.

Code stepping will automatically diminish once better code is written, *it just doesn't work the other way around*. People will still write bad code and if they avoid stepping through it that will only lead to more wasted time (damn I wish I could step through this spaghetti mess), not to better code.

Share Improve this answer

Follow

answered Nov 1, 2008 at 20:38

Vinko Vrsalovic

340k ● 55 ● 340 ● 373

> Vinko I've been thinking about this for decades. Everything you say is true but I think this is a good way to separate those who *can* step up to the mark from those who can't. You could then either cut your staff or change the hierarchy.
> – Peter Wone May 11, 2012 at 4:33

5

There is something wrong here, but it's hard to put my finger on it. Perhaps the real issue is that the code has other smells that make it difficult to readily understand. I agree that with TDD one ought to use the debugger less rather than more, since you'll be developing the code in small increments. But, if you can't look at the code and understand it, perhaps it's because the design is too coupled -- there are too many interrelated classes required to make things work.

If the code really needs to be so complex that observation won't suffice, then maybe you need to invest in some good commenting, explaining what is happening -- though I would prefer to see things refactored to the point where comments are not needed. My suspicion is that the debugger may be a symptom rather than the problem.

I know that for me, switching from traditional, code-first development to test-first development has resulted in less time spent debugging...and it's not something I miss.

Typically I'll only involve the debugger when its not obvious why the code I just wrote to pass a test, didn't.

Share Improve this answer Follow

edited Nov 1, 2008 at 21:12

answered Nov 1, 2008 at 20:29

tvanfosson

532k ● 102 ● 699 ● 798

3

This is going to sound like the argument you said you don't want to have, but I think if you want to convince your teammates, you're going to have to make a stronger case. I don't understand your objection. I frequently step through code I'm trying to understand with the debugger. It's a great way to see what's going on. You have not established your claim that people who use the debugger in this way tend to write code which is otherwise difficult to understand. The only convincing way to do so would be through some kind of case/control study which tried to measure and compare the readability of code written by people with varying approaches to the debugger. And you have not even told a plausible story explaining why you think using a tool to understand code execution tends to lead to sloppier code construction. For me it's a complete *non sequitur*.

Share Improve this answer Follow

answered Nov 1, 2008 at 22:37

Alex Coventry

70.7k ● 5 ● 39 ● 40

Sorry - I guess I didn't make a string case because I was interested in the persuading aspect. Please read my question as an hypothetical: 'Assuming Debugging is a bad smell, how would you persuade your team-mates?' Still, your point is well taken. – FOR Nov 1, 2008 at 22:57

1

A "plan" to convince them of the advantage of another approach is by establishing ***metrics*** linked to the number of time you debug the *same* function for *different* bugs.

By analysis the *trend* of that metric, you may convince them that ***non-regression*** tests are more useful to spend time writing, and will help them to debug more efficiently.

That way, you do not write completely off the "debug" habit, but you convince them of establishing a solid set of test, allowing them to focus on really useful debug session, if needed.

Should you consider this course of action (metrics), you should know its implementation involves the all hierarchy (stakeholder, project manager, architect, developers). They all need to be implicated in those metrics in order to act on them.

Regarding developers, you could try to suggest:

- some new ways of closing a bug case (close it only with the test scenario played to reproduce that bug, meaning they need an independent test in order to, if needed, launch their debug session)

- a clear relationship between those metrics and their evaluation by the management (it would be a bad practice to debug over and over the same function)
- a larger involvement in *architectural* decisions: sometimes, knowing some functional or applicative features rather than just classes and code can incite a developer to think more in term of black-box test rather than white-box (which can more easily lead to debug session)
- a participation into "operational architecture" process (where you need to deploy your app, and make full front-to-back integration test). Again, a larger picture of the all system can help a developer to get more interested in features rather than 'lines of code'

edited May 23, 2017 at 11:45

Community Bot

1 ● 1

answered Nov 1, 2008 at 20:28

VonC

1.3m ● 558 ● 4.7k ● 5.6k

+1: thanks for the suggestion; now.. how to persuade them that establishing this sort of metric would be useful? I'm dealing with an interesting group of people here, can you tell ? – FOR Nov 1, 2008 at 21:33

Thanks for the added ideas. Increased involvement in the architectural decisions, in particular, might apply to our team and lead to some good introspection. Thanks! – FOR Nov 1, 2008 at 22:59

1

I think a better phrasing of this question would be "Is non-TDD a code smell?" TDD seems to lead to less time spent in the debugger due to more time spent writing/failing/passing tests. Without TDD, you are more likely to spend time in the debugger to diagnose errors.

At least within Visual Studio, using the debugger is not that painful, so the challenge for you would be to explain to your teammates how TDD would make their development more enjoyable, productive and successful. Just avoiding the debugger is probably not reason enough for a team to switch their development methodology.

Share Improve this answer

Follow

answered Nov 1, 2008 at 22:03

Peter M

472 ● 5 ● 16

Right on roadwarrior. debugging isn't the problem, it's poorly commented and or documented code and bad

1

archetecture. I work on a smaller team but when a bug does surface, I do step through the code. frequently it's a very small job because the app is well planned out and the doc's on the code are clear.

That said lets get to my point. Want the team to not debug... comment, comment comment. Nothing beats down the urge to debug faster. Sure they'll still do it, but they'll be more likely to step over well documented code.

Oh and though it should go without saying, I'll do it anyway. don't have bugs in your code. :)

Share Improve this answer

Follow

answered Nov 2, 2008 at 2:42

baash05

4,506 ● 11 ● 62 ● 99

1

I agree with those above who expressed the relative irrelevance of this "debugger issue."

IMO, the 2 most important goals of a developer are:

1) Make the software do what it's supposed to do.

2) Write the code so that a maintenance developer 2 years down the road enjoys the experience of changing existing or adding new features.

Share Improve this answer

Follow

answered Nov 2, 2008 at 3:32

Tim Tonnesen

31 ● 5

Before you make a plan, you should decide how important this change is to you. Although I agree that debugging is a smell, it is also a very well accepted and ingrained practice for developers, so convincing them that they should stop doing it won't be easy or quick - and for good reasons. How much energy do you want to put into this topic?

Second, why do you want to persuade them in the first place? If your motivation is to help them, is it really their top priority problem? When you help people in ways they *want to be helped*, change becomes easy.

Once you have decided that you want to go on with your change initiative, you need to take into account that different people are convinced by different things. Some people will already be convinced by trying something new and exciting. Some will be convinced by numbers (metrics). Some by getting told about it while eating their favorite type of cookie (seriously!), some by hearing about it from their favorite guru. Some by reading about it in a magazine. Some by seeing that "everyone else is doing it, too". Etc. pp.

There is an insightful interview with Linda Rising on this topic at InfoQ: http://www.infoq.com/interviews/Linda-Rising-Fearless-Change. She can say it much better than me. The book is quite good, too.

Whatever you do, don't press too much, but also don't give up. Change can happen - especially if you take

resistance as a resource -, and sometimes it happens at unexpected times, so always keep a sense of wonder.

Share Improve this answer Follow

edited Nov 2, 2008 at 11:30

answered Nov 2, 2008 at 8:39

Ilja Preuß

2,421 ● 17 ● 15

The problem is that after an extended period of working on rubbish code, people switch from craft mode to drudge mode. Because they cannot care about quality they refocus on getting paid. – Peter Wone Nov 2, 2008 at 12:09

That might be *a* problem, but I don't think it's *the* problem. There are a lot of other reasons to think that debugging is ok. – Ilja Preuß Nov 2, 2008 at 13:04

0

@FOR : You have a second problem too, here it is :

> sadly it doesn't seem the devs are interested in being more productive (they get paid the same anyway)

How do you intend to **make them want** to be more productive when there is nothing (visible) for them to gain?

Share Improve this answer

answered Nov 4, 2008 at 1:14

Follow

Andrei Rînea
20.7k ● 18 ● 121 ● 169

> Good point - my intent was to sidestep that issue since I believe *I* can't make them want to be more productive. But maybe, I can convince them to do things that make their life easier (like developing a system that doesn't require code-stepping). – FOR Nov 4, 2008 at 16:20

0

Designing software by debugging is a good practice.

The number of environments supporting this way of developing is very small: the best known is Smalltalk. In Smalltalk, you can write a test describing your objects protocol without the methods being implemented. Running this test will then trigger the debugger, and you can add the method to the right class in the debugger, and can continue stepping through the code until all functionality is implemented and the test is green.

This needs a compiler to be available at run-time, and first-class invocations. It offers a very short feedback cycle, and is one of the primary reasons for Smalltalks' productivity

Share Improve this answer

Follow

answered Apr 8, 2016 at 16:42

Stephan Eggermont
15.9k ● 1 ● 40 ● 65