Has Agile really worked for you as a Developer? [closed]

Asked 15 years, 3 months ago Modified 10 years, 3 months ago

Viewed 10k times

20

Closed. This question is opinion-based. It is not currently accepting answers.

Want to improve this question? Update the question so it can be answered with facts and citations by editing this post.

Closed 9 years ago.

Improve this question

I have met a lot of people for whom Agile has worked really well, and most of them tend to be managers and architects who plan and delegate the work. However I really haven't found much good developers convinced that Agile is working for them.

Of course you can say if Agile isn't working for you, you aren't doing it right. But whatever remixes of Agile are out there, is it working for you as a Developer? And why? Does anyone else think, within a traditional (or close to) team structure, Agile feels more like a form of micromanagement than self-management?

agile methodology

Share

Improve this question

Follow

edited Sep 1, 2009 at 20:49

community wiki
2 revs, 2 users 100%
Langali

6 Bear in mind that "Agile" is the new "object-oriented" - assumed to be good, but ask three people in the field and you'll get four definitions. I'd suggest either narrowing what you are asking for or asking people to define what sort of "Agile" they're talking about. – David Thornley Sep 1, 2009 at 20:54

1 Just like OO is composed of Objects inside objects inside objects, the Agile that I know of means: 1. Short iteration. 2. Stories 3. Daily status meeting. 4. Can't miss deadline like Waterfall. – Langali Sep 1, 2009 at 21:02

@David: In general I agree, however when someone talks about big-a-Agile like this, I just assume they mean some combination of XP and Scrum, and usually that is not too far off. – Matt Briggs Sep 1, 2009 at 21:03

3 @Erlanged: This is agile agilemanifesto.org nothing more, nothing less. The application of those principals are left to real methodologies, and anyone who tells you different doesn't know what they are talking about. – Matt Briggs Sep 1, 2009 at 21:05

13 Answers

Sorted by: Highest score (default)

45

At my first job, we had daily scrums, wrote automated tests, had automated builds, pair programmed, etc. We had been in the agile groove for several years. And for our efforts, we were rewarded with software that I wouldn't touch with 20ft pole. The quality of our product was atrocious: I'd describe as the piecemeal hacking of 100 amateur developers.

What went wrong:

- The company I worked at had a notorious reputation for hiring entry-level developers for the lowest pay ($25-27K/yr was the norm), and frequently we'd outsource work to the lowest offshore bidder. I've heard that agile just doesn't work on inexperienced developers, and I think it showed through the code and our turnover rate.
- No documentation of any sort. No functional documentation, no technical documentation, no requirements, no bug tracking. No one ever wrote things down on persistent media: all requirements and bugs were communicated by email, word of mouth, and psychic mindreading.
- Lousy testing: our automated tests were invaluable, but QA and UAT testing was a disaster. Since we didn't have any formal requirements documentation, QA users didn't know what new functionality they were testing, so all QA consisted more or less of

haphazard end-to-end testing. User acceptance testing was performed in production: we installed the product on our customers servers and reported bugs as they occurred in production.

- Crisis-driven development: bugs were handled by using the "OMG WE HAVE TO FIX THIS AND REDEPLOY PRONTO! NOW NOW NOW! NO TIME FOR TESTING JUST FIX IT!" management methodology.

Although we did everything right and really adhered to agile principles by the book, the methodology failed harder than anything else I've ever seen.

In contrast, the company that I work for now uses a waterfall-like methodology, produces a few hundred pages of documentation for each project, has few automated tests but a sizable QA team. Interestingly, the quality of our product is through the roof, and the work environment is orders of magnitude above and beyond the other company.

I know many people have had the opposite experience. As is usually the case, methodologies are not a golden hammer --- you can probably start a successful project no matter what methodology you choose. In my experience on successful and unsuccessful projects, I get the feeling that methodology doesn't matter as much as environment: comfortable, happy developers and sane project managers are all it takes make a project work.

Share Improve this answer Follow

answered Sep 1, 2009 at 21:20

community wiki
Juliet

10 scrapping the designing,speccing and testing is hardly adhering to agile development. Implementing the buzz-words and forgetting about common knowledge things you were, frankly, doomed from the start. Agile, does not mean "stop thinking and get coding" – NomeN Sep 1, 2009 at 21:25

2 +1 - Quality answer from experience. – DarkSquid Sep 1, 2009 at 21:40

2 @Juliet I have to agree with your conclusion. I think we put an unfair amount of importance to methodology compared to environment, personalities, and the company vibe. – Langali Sep 1, 2009 at 21:43

3 Also, not all projects are alike. I'd rather use Lisp or a scripting language if I didn't understand what I was doing, and C++ if I was. Waterfall is efficient if the problem is well understood, but Agile does better with constantly changing requirements. – David Thornley Sep 1, 2009 at 22:03

2 Having spent a couple hours reading the agile tagged posted on SO, it's seems anti-agile/agile-failure posts gets the highest votes. Why do I get the feeling that most developers have had bad experiences with "agile"? :) – Jack Ukleja Sep 20, 2009 at 10:33

11

At my company, we made a wholesale switch to agile practices about 4 years ago when a new VP came in. This VP had experienced success with Agile in the past, and decided it was what we needed.

As it turns out, he was right. I was a developer at the time (albeit a somewhat junior one), and I loved the practices. Pair programming really aided knowledge transfer and prevented the formation of knowledge silos. Unit testing, test driven development, and test emphasis in general made for more robust code that wasn't over-engineered. No Big Design Up Front meant that instead of spending 6 months writing requirements documents (by which time the market had passed us by), we were prototyping and delivering real value to customers in a timely matter. Working closely with a customer surrogate (in our case, a technical product manager) greatly shortened cycle feedback time, which helped us deliver what the customer actually wanted.

Our organization had quite a few talented developers, but we were very prone to cowboy coding. A few developers didn't like the new practices ("What do you mean, write tests? I'm a developer!"), but generally everyone loved the changes. Defect rates went down, customer satisfaction rates went up, and our office became very well regarded in our company.

About a year ago I became a manager, and I heavily use Agile practices, incorporating some Lean principles as well (value stream analysis, waste elimination, kanban).

Tightening up release cycles has been an ongoing activity, and my team now releases as often as possible (with quality!) - often every two weeks. We have no field reported defects from my team in the past year, and the sales people and product management love the shorter release cycles.

As a developer, Agile really increased my confidence in working with various areas of code (I now feel nervous whenever I'm changing anything in a package that DOESN'T have 100% unit test coverage!), taught me to be a more well-rounded programmer (thinking of test implications, business impacts, etc.), and helped me write simple, self-documenting code. As a manager, Agile and kanban gives me predictability, lower lead times, lower defect rates, and an empowered team. This is not theory, or speculation, or hand waving - our team morale, defect rate, customer satisfaction, and balance sheet have proven that Agile can do wonderful things for an organization.

Share Improve this answer Follow

answered Sep 2, 2009 at 16:59

community wiki
Chris Simmons

6

To comment on the Principles of the Agile Manifesto from my experience at a site that *tried* it.

> Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

This was a double-edged sword for my last site -- valuable was taken to mean 100% perfect and bug-free.

> Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

I still communicate with that site and just today, their rock-hard deadline date, they were given a requirement change. That was just the way things were there; it's almost as if they wanted failure.

> Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

The norm for many years was basically build and deploy daily, hourly, near real-time...

> Business people and developers must work together daily throughout the project.

Some of the meetings/reviews with respect to this were hilarious. We were reprimanded for not working with the people (because they asked us not to because they were already working 9-10 hour days) and then for bothering them because they were busy.

> Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

Ahh, here's our problem... We had top-of-the-line PCs but the business side wasn't supportive. The positive morale essentially got beaten out of you after about a year or so... This also negates your micromanagement concern (if implemented correctly).

> The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

This worked out well. Personally I prefer email because I hate taking notes.

> Working software is the primary measure of progress.

No doubt here.

> Agile processes promote sustainable development. The sponsors, developers, and

> users should be able to maintain a constant pace indefinitely.

I agree with this 100%; the problem with the last business team I worked with was the expectation of 30-hour days, 10-day weeks, and 400-day years was not a pace I agreed with.

> Continuous attention to technical excellence and good design enhances agility.

This is where some developer morale & education was needed.

> Simplicity--the art of maximizing the amount of work not done--is essential.

I love this one and it's long been one of my goals. However, there was a "if you're not typing, you're not working" attitude that was tough to overcome.

> The best architectures, requirements, and designs emerge from self-organizing teams.

I agree with this about 90% -- my only caveat is that they must be well-educated and well-informed teams.

> At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

We just failed here and it likely caused a lot of other problems we had. The business side was really good at saying "you need to do what we say needs to be done."

To wrap it up, if you're working somewhere where everyone is informed and on board with an Agile methodology, it should be a great place to work. When the goal is great software, momentum alone will carry any project.

Share Improve this answer Follow

answered Sep 1, 2009 at 22:28

community wiki
Austin Salonen

1 +1. Well written answer, and I totally agree with your conclusion. When agile is appropriate to your business and implemented well, it rocks. – NomeN Sep 2, 2009 at 9:35

nicely sums up why agile fails in most places - its needs more than buy-in from developers, everyone has to be on board – Jack Ukleja Sep 20, 2009 at 10:19

@Austin Salonen, and what's wrong with shipping only bug-free software? In fact, when is it ever right to ship buggy software? – Asclepius Jun 14, 2014 at 7:35

@A-B-B: It's rarely correct to ship bug-ridden software though it's impossible to know if you've caught every bug (it's an extension of the Halting Problem). – Austin Salonen Jun 15, 2014 at 2:31

6

Agile hasn't worked for me, the main reason being that the systems I usually develop need a well-defined and well-thought architecture, which is hardly realisable by an agile approach. Agile approaches tend to write as little code as necessary to pass the applicable tests, and thus to grow the system organically. This can be nice from many perspectives, but it wreaks havoc from the architectural viewpoint.

Share Improve this answer Follow

answered Nov 17, 2009 at 4:05

community wiki
CesarGon

6

From my personal experience, Agile methodology tends to create a huge technical debt in the long term, and while it might save you (as a business owner/management) a couple of bucks short term, in the long term it will come back and bite you. Whatever you do not fix now will eventually cost you many hours of work to fix at a much higher cost than it would have cost you to invest some more hours into the original problem.

Agile is always great from the point of view of beginners and management, but I do not know one experienced programmer who actually loves it. The whole point of Agile is to save development money for a company, it has nothing to do with actual product quality. In fact most of the methodology promotes bad code done quick over well-engineered code. The difference is that a few years down the road, you have to do the whole work all over again whereas the well-engineered code can last decades without corrections. Good programmers do not need Agile methodology most of the time.

We have a business logic library written 22 years ago here by a single team of 3 programmers using waterfall methodology, and that code hasn't needed a single correction since. Because it was tought properly, was well-engineered, and the three programmers took their time and were careful with their implementation. Agile methodology would instead ask of those three to do the strict minimum to make sure some ill-defined tests passed, and then wait until the next time you hit a wall to add some more duct tape code. It's a ridiculous way to work and goes against every engineer fiber in my body.

To this day I refuse to work in an Agile environment, because frankly I do not need it, and I do not need an employer who thinks I do need it.

Share Improve this answer Follow

edited Sep 24, 2014 at 18:41

community wiki
2 revs
Drunken Code Monkey

5

Agile has worked awesomely for me as a Developer in my current environment. Here are some practices and why it seems to work so well:

- Pair programming - This prevents anyone from feeling an individual ownership of the code. Pairs of developers tend to make better code than one person's "mad science" code that seems to happen if one person writes a bunch of code in isolation. This also allows for someone else to be brought in if someone goes away and that feature or enhancement has to get done while the person is away. Sometimes, one developer may think something will be great but if no one else can understand the code, it is useless to have unless it is perfect and futureproof which isn't likely.
- Scrum - This creates both an accountability and communication so that each person knows what the other is doing. If someone wants to know how the sprint is going, just show up at the stand up. The standup is really simple in that it is just 3 questions: What did I do yesterday, what I am doing today and what would prevent me from getting that done?
- Test-driven development - A variation on this is practiced where I work in that we generally try to have tests for most of the plumbing code we are

writing to customize a CMS in the big project we are doing. This mindset can be tricky to get into though it does get easier as one practices it more.

- YAGNI - The "You Aren't Gonna Need It" principle that can really be hard if you've been a perceptive programmer that generally puts in 101 things as a "Well, I might need this someday..." mindset. Another way to put this is a "Keep It Simple, Stupid" idea.
- Sprints - The idea here just seems to prevent a sense of being overwhelmed as we are just working for 2 weeks on this or that, and don't look too far forward as it may well change.
- Demos - Showing off what we have done, getting feedback on what is good and what isn't is crucial for getting things better and having a mindset that we want "continuous improvement" in the project and what is good enough today, won't be good enough tomorrow and get better at what we do.
- IPM, Retrospectives - The ability to look back at what was done in retrospectives is useful for venting frustrations, celebrating things working well and finding ways to address pain points. IPM is where we determine our future for the next sprint in terms of what will be the goals and how long do we think various things will take by using a couple of different estimation tools, one for points on "epics" as we call them and the other for hours on an individual task or card which is part of a story that is something between the epic and a small piece of work.

- Storywall and user stories - Now this low tech tool since it is just a few whiteboards, with dividers and post its provides some structure to things. We have lanes for each of the epics and various columns for states of work: Backlog, in development, on dev, or on test. There are also places for the task backlog, blocked cards, questions, standards and practices and a few other things that may be useful for managers to see to get an overview on the current status if they want more of a bigger picture than what they would get at standup.
- Broken windows/technical debt/tasks - These are similar in some respects and are useful as a way to illustrate that quality matters,i.e. we don't want broken windows that can be easily explained in non-technical terms by either using a house in a neighbourhood or the New York Subway sytem as starting points. Technical debt being something that doesn't immediately add business value that is sometimes an important thing to use to prevent some problems as there may be problems with a particular architecture and so part of a sprint may be spent doing a re-arch that has to be communicated so that if there is a sprint with little to demo this is why.

I don't know if the idea of a "self-organizing" or "self-managing" team is part of Agile, but it has been a bit of a challenge for me to have enough faith and trust in my co-workers that things will work out fine. The professionals that are the rest of my team know what has to be done, are reasonable, honest people with integrity to just get the work done and not be jerks about getting things done. There aren't egos or bad attitudes which really do help foster building a team.

Share Improve this answer Follow

answered Sep 2, 2009 at 16:17

community wiki
JB King

When two professionals are working, using pair programming by default is a nasty idea that takes away from developer freedom and independence. It makes more sense when one is a junior newbie and is learning from an expert. Regarding Scrum, as idealistic as your definition of the standup maybe, it's going to degenerate into "why did you not finish xyz yet??" – Asclepius Jun 14, 2014 at 7:55

How much experience do you have with pair programming? Where I've worked it has been quite good at keeping things focused and preventing someone from writing a pile of code that many hours have to be spent so that others understand it. What kind of experience do you have with Scrum as those questions are the heart of the standup you do realize, right? – JB King Jun 14, 2014 at 19:24

If code is modular and well documented, it can generally be well understood. Pair programming is no guarantee that the rest of the team will understand it anyway. Regarding the standup, well that's just a daily humiliation. If you don't trust your employees to get work done, you shouldn't hire them. – Asclepius Jun 14, 2014 at 20:11

That's a mighty big if there. – JB King Jun 14, 2014 at 20:47

Then isn't that where the focus should be? Also, a review process is more useful than pair programming. Programming

is a creative process, and creative work is not done in pairs.
– Asclepius Jun 15, 2014 at 1:48

3

Agile is not a methodology, it is a subset of methodologies that have a common set of goals, and more often then not those methodologies have wildly varying results based on team makeup, corporate culture, and implementation.

Off the top of my head, purely developer agile practices would include pair programming, TDD, user stories over specs, the assumption that all code is going to be refactored several times (although this is part of TDD) and code reviews more then anything. Things that impact us are daily standups, being engaged with users regularly and directly, postmortem introspections, and very tight development cycles.

Share Improve this answer Follow

edited Sep 1, 2009 at 21:07

community wiki
2 revs
Matt Briggs

3

I'm a developer and a manager at the same time, so I either have special insight or my opinion is totally invalid. ;)

I will say that Agile means a lot of things. It's actually a whole family of methodologies and guidelines at this point.

Exposing yourself to all these interesting ideas is really the thing. As a manager, it's very hard for me to decree that a whole team suddenly adopt a whole methodology, but if they see me constantly trying to improve every aspect of my game, I think they appreciate that. And hopefully, if they like what they see, they follow my example.

I've managed to slowly implement a bunch of things from Scrum without (hopefully) coming off as a tool. Burn down reports, stand-up meetings, and story cards on the whiteboard have really made us better at shipping software. For instance, on any project tasks are constantly being done ahead of schedule or late. In a really big project, it can become very difficult to tell what that's doing to your ship date. With burn down reports, I can tell what a slip does to our ship date, and if we need to just start cutting features in order to meet a deadline.

That's more of a management thing, but the other devs here care about it because it might mean they get to keep their jobs or avoid a death march. :)

But it's not just management stuff. There's tons in Agile about best practices for source control, unit testing, etc. Just good solid best practices. As an industry, we are pretty terrible about mentoring, so it's good that this information is out there.

Share Improve this answer Follow

edited Sep 1, 2009 at 21:19

community wiki
2 revs
Brian MacKay

2

From the developers perspective I think it works well. In my point of view agile techniques have in common that the loop between defining the task, working on the task and getting feedback from that task is a very small one as compared to a non-agile approaches.

Take TDD as an example: Code the test, red bar, code the functionality, green bar, refactor, red bar, fix, green bar.

From the managers perspective this faster feedback loop is also true: Daily meeting one, status green, daily meeting two, status yellow, countermeasures / re-assign ressources, daily meeting three, status green.

Immediate feedback and knowing where you are heading gives a feeling of safety.

Share Improve this answer Follow

edited Sep 2, 2009 at 3:50

community wiki
2 revs
Theo Lenndorff

Due to need for specialized knowledge and/or lack of resources, "countermeasures / re-assign ressources" almost always tend to be working extra hours and weekends to keep the team goal alive. Most agile developers I see are overworked. – Langali Sep 1, 2009 at 21:37

@Erlanged - is this true only of "agile" developers? Most of the good coders I know are overworked by any reasonable standard – DarkSquid Sep 1, 2009 at 21:42

@DarkSquid There is a big difference between voluntarily overworking on projects that you are passionate about, and overworking to meet someone elses unrealistic deadlines. – Langali Sep 1, 2009 at 21:52

1 @Erlanged - Again, is there any difference between Agile and non-Agile developers here? You can be overworked in a hopeless attempt to reach somebody else's impossible deadline in any methodology; the phrase "death march" is older than any coherent description of Agile development. – David Thornley Sep 1, 2009 at 21:58

If you're overworked, you're doing it wrong. Sure, there's always hot phases where you stay longer to finish things in time, but chronic overtime is a symptom of either un-agile bad management or exploitation. One of the core tenets of XP is: 40 hour week is the norm, to be exceeded only in exceptional circumstances. Exhausted people aren't productive. – Michael Borgwardt Sep 1, 2009 at 22:38

0

In the so called 'traditional team', Agile development would increase the visibility of individual developers to management. That would probably allow managers and architects to plan their work better. Ofcourse that could be interpreted as micromanagement.

But from an organizational perspective, if it produces results, who cares.

Share Improve this answer Follow

answered Sep 1, 2009 at 21:05

community wiki
Alterlife

Even if the productivity isn't sustainable? – Langali Sep 1, 2009 at 21:16

@Erlanged Non-sustainable productivity is getting things out quickly but full of bugs and pulling overtime because of over-committed deadlines. These things shouldn't happen in a well-functioning agile team. – Chris Simmons Sep 2, 2009 at 17:29

0

I guess what makes an "agile" project agile, is the methodology: "Design for today not for tomorrow".

For any not life-critical software systems this is a way to keep programmers coding in stead of discussing ages about design. Please note that design is not scrapped, it is just done in smaller and therefore more overseeable chunks.

All other techniques that are associated with agile, like pair programming, are more borrowed ideas that could also be used effectively in any other methodology.

Now, does this technique 'work'? Yes! If applied correctly, the technique promotes that the software product will be ready for shipping at any time to react to competition.

On the other hand, because programmers are feeling they are coding more, they are generally happier. And they are less irritated by writing specs because this phase is inherently always small.

But again, if you know exactly what your product is going to be and especially if it is life-critical like the space shuttle, agile development is not what you want.

Share Improve this answer Follow

answered Sep 1, 2009 at 21:21

community wiki
NomeN

0

Nearly every management is aware of "Agile" by now: *It's this thing, you know?* Alone by your initial question I would assume that something is really going wrong. I really recommend you reading a book like Practices of an

Agile Developer (as the title suggests - it's about what's in for you).

Some managers read a book and then will *know what agile is all about*. They are telling you what to do and everything is fine, isn't it?

If you look around, there are a lot of developers (in Agile companies) who can't tell you within a second what the *purpose of a stand-up* is - and that's an issue. If you (and maybe even nobody else) don't know the *why* the StandUp won't make things better.

Take a look at *time tracking* (and time estimation) - there are some managers who think it's about *measuring how much work you do*: *Hey, you have a 40h contract but the time tracking tool says that you have only be working for 38h this week!* That's not how it was meant to be used.

The only thing you can do about that: you need to learn what agile methods are out there. Mediocre managers will pick the ones they find interesting. Good managers will grasp the *why* and not only choose the methods for their direct benefit - but also those which will make the team more happy / efficient / teamish (Team vs Workgroup).

P.S. Something you really need to take care of: In agile there is no place for slackers. Everybody has to do stuff on their own. You have to put personal interest into the success of the product. If you don't do things on your own, somebody will tell you what to do (and then there's micromanagement).

Share Improve this answer Follow

answered Sep 1, 2009 at 21:47

community wiki
Marcel Jackwerth

1 For your info, I have read Venkat's book and several other articles. The problem I see is not with developers but with traditional managers who still don't demonstrate servant-leadership. What happens in turn is that status meetings meant to update each other turn into report status to your leadership. Estimation becomes commitment. Work logs becomes the tool for performance monitoring. And if there are unexpected technical problems, you end up working nights and weekends to make up for it. – Langali Sep 1, 2009 at 22:07

Agile from top can be desasterous. Actually that's when I would look for a new company to work for. Or (which might be quite difficult) convince the management that the status-quo reduces the team's effectiveness - and will result in failure / burnout any time soon. – Marcel Jackwerth Sep 1, 2009 at 22:20

-2

Has Agile really worked? "***Yes***."

Before there was "Agile Programming" there were equivalent largely unrecognized methodologies. I thought these were called *incremental prototyping* but apparently this has been split into that and *evolutionary prototyping*.

I suspect that many or most of the successful systems were so constructed. Just because the methodology grew a new name doesn't mean that it suddenly appeared.

It's just that Waterfall and other broken management techniques that got all the press.

I say Agile works. I say it's the only thing that ever worked.

edited Jun 14, 2014 at 8:04

community wiki
2 revs, 2 users 83%
DigitalRoss

Well said - the more I read about Agile (and the concept of agility in general), the more I see generic applications in it. – Chris Simmons Sep 2, 2009 at 17:28

"I say its the only thing that ever worked". As agile only appeared in the mainstream in the last ten years, that would imply that no successful software was written before then. Which is clearly not true. In fact most of the biggest successful projects probably used waterfall. news.slashdot.org/story/13/05/25/139218/… – wobbily_col Apr 16, 2015 at 14:20