Agile Project Management
Lessons From the Trenches

10 Expert Tips from Atlassian Experts on How to Avoid an Abysmal Failure
Software development has come a long way in the last few decades. Remember FORTRAN and COBOL? CASE tools? How about the good ‘ol “CMM” (Capability Maturity Model)? Not ringing any bells? Then your age is showing! Whatever generation you started programming in, development has always had a common goal: deliver quality products to market faster and with lower cost. Sounds easy, right?

In 2001, 17 software developers met at the Snowbird, Utah resort to pontificate on lightweight development methods. I’m sure cocktails were flowing and the hot tub waters were whirling too, but the result was their Manifesto for Agile Software Development, which now defines the development approach known as Agile. I won’t spend more time on defining Agile Methodology itself, you can find that anywhere on the web if you’re a newbie... although I will take a moment to pay homage to Martin Fowler, widely recognized as one of the key founders of agile methods...

Yo Marty! NYCE work!

The popularity of agile has skyrocketed in recent years, taking over the mindshare of software development teams and their project management cohorts everywhere. Agile tools to help manage agile process quickly hit the market too, and the tools done well (e.g. the Atlassian toolset) have done a great job riding the proverbial agile coattails.

Why all the buzz?

Agile sells well, frankly. The methodology promises solutions to perennial IT concerns that even some of the best Fortune 500 technology teams can’t seem to crack: projects run over budget and are perpetually late; teams lack effectiveness and cohesion; co-developers don’t collaborate; management can’t see progress; product quality suffers; and ultimately in the end, the business creates more dissatisfied customers than satisfied ones.

Implementing an agile approach can indeed transform quality and time to market, but the agile process is only as good as the people implementing it.

Implementing an agile approach can indeed transform quality and time to market, but the agile process is only as good as the people implementing it. As with any process, it can be mutated and misinterpreted to the point of abysmal failure—and suddenly the methodology is to blame, rather than the integrity of the implementation process itself.

Prefer to avoid an abysmal agile failure?

Great. Then you’re on the right page. We’ve collected 10 tips from our team who collectively have been through hundreds of successful implementations in companies of all sizes, including many Fortune 1000 — all using the agile approach, and many using the Atlassian toolset.
While I wouldn’t dare to claim we have perfected it (the truth is, you are continually learning and the minute you stop learning, you immediately begin your undeniable descent into the ‘failure abyss’), I can say with confidence these are powerful, real-world tips that we have proven to work time and again (and that we often learned the hard way in the early days, in some cases). I’m confident they will have a huge impact on your success rate. And I invite you to contact me with more tips, or with stories on how they have improved your development process.

Here are the 10 best tips from across our team.

1. Define done. (category: sprint planning)

Make sure that everyone on the team and those involved in the business know what it means to actually complete a task. Semantics can get in the way, that’s why you need to be explicit here. In most cases, ‘code complete’ is not done, yet developers often think code is done and immediately ready for a demo. Wrong-O.

Review the definitions of “done” in your initial sprint planning meeting (rinse and repeat as often as needed) and never demo anything in your retrospective code set that isn’t done according to the agreed upon definition. Code complete isn’t necessarily done. Set up automated tests and your own set of definitions, and avoid demos to stakeholders that are too early.

If done right, every piece of code will have a test behind it. Be mindful of how much of your code is covered by unit tests. Are all lines tested, or just some? When you reach the real point of ‘done,’ start the QA and UAT processes. Then make sure you’re clear and consistent going forward against your ‘done’ checklist.

Maybe sounds like a given, but we’ve seen this go haywire on many occasions.

2. Sustainable pace is mission-critical. (category: sprints)

It is unrealistic to think that any resource can spend 100% of their time on new development tasks. Planning a sprint on this assumption is a sure recipe for disaster. It’s far better to start with an estimate of 75% availability as a general rule of thumb, and then reexamine that estimate during every retrospective.

Unless you’re running a team of auto-bots, people are people. They get interrupted regularly. They catch a virus. They maybe even leave for the occasional vacation. Ground your pace in reality and be sure to set the same expectations with stakeholders.

Overbooking resources fuels burnout, literally. Resources become drained mentally and physically, which can lead to mistakes and missteps. It’s human nature.

Part of pace setting also means keeping resources focused on singular tasks in the sprint.

Part of pace setting also means keeping resources focused on singular tasks in the sprint. If they end up below 75% capacity, they are likely getting dragged into too many other things. Outcome equals: not good.

As time progresses, you’ll get better and better at estimating. Don’t beat yourself up, just commit to improving with every iteration.
Create purpose-built daily stand-ups. (category: scrum)

It’s not uncommon for stand-ups to become 30 minutes, or holy forbid, even longer discussion sessions. When this happens they are often transitioning from true Agile, to truly wasteful.

While it’s easy (and human nature) to quickly get into problem-solving mode, it’s the job of the Scrum Master to have enough backbone and leadership abilities to push discussions offline as appropriate and keep the stand up to 5 to 15 minutes max. Get it on the table, then get back to work.

In the stand-up, every team member should answer three simple questions...

1. What did I get done since the last meeting?
2. What will I get done before our next meeting?
3. What’s in the way / what are my blockers?

Scrum is intended to give everyone info on status so there’s transparency. Yes, you can swarm after the scrum to get the right folks to start problem solving on the right issues. Finally, be sure it’s developers only. If management is brought in, the purpose of the meeting changes, and it’s officially a management update, not a stand-up.

Yes, length really does matter. (category: sprints)

If you think two week sprints are great, start getting your mind around, “one week is better, three weeks is utterly MEH, and four weeks is the wasteland.” Longer sprints will undoubtedly force business changes to begin leaking toxic waste into your sprint plan. This will have a negative impact on your burn-down and throw your future sprint planning off.

It’s also important to note that you should not change priorities or requirements mid-sprint. A sure-fire no-no.

Keeping the sprints short will help keep the business at bay. Generally, the business will start asking for changes a week or so after the sprint has started, if the sprint is only two weeks they don’t have to wait long for this new urgent requirement to be addressed.

QA what? (category: sprint planning)

Make sure QA is part of the sprint planning session. The sprint is not complete until the QA is complete. Every organization with a serious development effort has a QA process after development is complete. Development will often finish two weeks of work, then QA steps in to test as soon as there is code.

QA should be engaged at the beginning of the sprint. They need to start requirements with the stories (aka: use cases) before the code is complete. The stories will have a natural business language and framework from which QA starts writing their test plan, and they should stay in lock-step with developers as the code marches forward. In reality, there should always be something to test against.

Try to automate as much as possible, and remember from my earlier point: nothing is done until it has been through QA. Unit tests and functional tests should be a slam-dunk to automate and execute. In the world of Continuous Integration (CI), builds should fail if automated test code coverage does not meet a minimum threshold.
6 Eat your carrots & get 20/20. (category: sprint)

Getting 20/20 vision may be more important than you think. Do a sprint retrospective and remember the Scrum process itself is iterative. Always be looking for ways to improve the process; this will assist with future sprint estimating and help the team avoid distractions and more clearly identify blockers.

A word of caution: try not to be too reactionary in changing the process. If you make adjustments, give them two to three sprints to work through the process before changing them again. You need to get enough data to triangulate a fact-based decision about the effectiveness of the change, otherwise your case will be off and the ripple effect ensues.

As I mentioned before, you should always be reexamining the process, and the minute you think you know everything is the minute you’ve officially checked out of agile. Our brains are physically wired to learn continuously, but it’s still a conscious choice you need to make.

The value of retros can never be overstated, and true agile technicians understand the continuous learning aspect of the methodology and use it to their advantage.

If you make adjustments, give them two to three sprints to work through the process before changing them again.

7 Don’t fool yourself: working software is how all things are judged. (category: continuous integration)

Your software should always be build-able, tested and therefore, ship-able. This is the holy grail of development in the first place, isn’t it? Don’t lose sight of this while you’re in the daily sprint weeds. Ultimately, the business is depending on you for output that not only meets the requirements, but makes everyone look good and creates more happy customers. If you’ve done everything right, the software fulfills the user stories, everything works as it should, and you’ve passed your tests.

Documentation is important, but still secondary to a functional, shippable product that meets the deadlines, meets (or beats!) the budget, and drives revenue.

8 Reality is the board, reality is the board. (category: scrum)

The Scrum Master should be monitoring the scrum board. There’s a reason that role gets their title capitalized. If a team member says a task is ‘in progress,’ the board should reflect that fact. If a team member says “I’m blocked,” the board should reflect that too. Everything logged in the board should be current and consistent with reality.

Every blocker should be documented with some ‘meat.’ Don’t spend time just explaining in the stand-ups, be sure JIRA (or whatever tool you’re using) is current and accurate. This one can’t be understated either. If the board doesn’t reflect reality, you’re only fooling yourselves and ultimately, misleading your stakeholders. For you visionaries out there, you can predict just how much pain and suffering this can save you. Better to face facts and issues head-on, sooner vs. later, than to deal with the aftermath of misleading everyone down “la-la-land.”
9 Every bug should have an automated test named after it. (category: continuous integration)

When observing a continuous integration (CI) build, you should be able to see one automated test after another scroll by that is named after JIRA ‘bug’ issues. Old bugs should never, ever resurface. When QA finds an issue (defect, bug – whatever you choose to call it) from anywhere in the code, there should be a test developed to identify and resolve that bug going forward.

The best teams write a test every night for the bugs found that day. That means (in theory!) you should never see the same bug twice. The first step is to write the code that duplicates the bug. This becomes the foundation for your automated test.

10 Business/product owners must participate in the agile process, really. (category: management)

In our experience, it’s really common in organizational structures that the product and business owners will participate lightly in sprint planning, the daily stand-up, and the process overall. We’ve seen this time and again, and without fail, they are invariably befuddled when the software they asked for isn’t functioning as they expected it would.

A key component of a truly agile process is staying in constant communication and continually gathering feedback. If you take a step back and look at the big picture, you can see that was the real intent of the agile founders. Why else would you have such a painstakingly detailed process baked into the methodology? If the product owner is half-in and half-out, there’s tremendous risk (which, by the way, only increases over time), and the resulting product produced is not likely to succeed.

Put another way – they bring value to your efforts by helping to regularly re-order the backlog and drive business inputs into the process at each step. Ok, so maybe that sounds just plain annoying to you, but trust me, it’s worth it. If you prefer to work alone in a closet, I might recommend freelance work or hacking.

So get them in the game, on board and fully committed. Ideally it’s a director level business colleague or above. Lower level folks may feign more authority, insight or decision-making power than they really have and set you on a dead-end course to nowhere. Director or above folks can also serve as your dedicated team of ‘buffers’ to the rest of the business, helping you stay focused and still in the know with the end goals.

In the end, your stakeholders will thank you, even if they don’t appreciate the “what’s” and the “why’s” along the way.
SO...
THERE YOU HAVE IT

Lessons from the trenches from real-world development projects compiled for you in one, neat little package. I hope these tips shed new thinking on your agile journey, whatever stage you're in, and inject integrity and fuel into your process that improves your quality, speed and success rate. And, hopefully you're a little wiser too, and can drive these tips home to the naysayers on your team about the value and effectiveness of agile overall.

As always, don't hesitate to contact us with questions for more detail or clarity around any of these tips. We can be found online, at our Geek Nights in Phoenix, at our user groups in Las Vegas, and... all over the place.

ABOUT ISOS TECHNOLOGY

Isos Technology is a leader in solving complex business, people and technology challenges in software development, mobile and information technology. The company’s proprietary people assessment methodology helps find leading engineering talent to drive advanced software and information technology solutions for enterprise clients.

The company shares its expertise on the corporate blog as well as on Twitter, LinkedIn, Google+ and Facebook. For more information visit www.isostech.com.

ABOUT THE AUTHOR
THAD WEST, Founding Partner & CEO, Isos Technology

As a Founding Partner and CEO of Isos Technology, Thad West is responsible for overall vision, strategy and execution of the business.

Thad has nearly two decades of extensive experience in product management, development team leadership, enterprise architecture and business application development. Prior to founding Isos Technology, Thad held the position of VP of Application Development for a Fortune 100 financial company and Director of Product Management for a mid-size software company specializing in mobile applications for the construction industry.

Thad can be found online on the Isos corporate blog, and on Facebook, Twitter and Google+.