How To Fail With Agile:

Twenty Tips to Help You Avoid Success

- July 1, 2008
- by Mike Cohn & Clinton Keith
- 38 Comments
- originally published in Better Software on 2008-07-01

The agile process is now accepted as valid alternatives to traditional software development processes. Most people who adopt agile do so to realize the benefits of faster delivery, higher quality, products that more closely match user needs, and so on.

Not everyone is so enamored of agile. Some teams and individuals balk when a mandate to "become agile" is passed down from some "higher-up" in the organization or when some young go-getter decides to start an idealistic grassroots movement to effect change. A switch to agile often conflicts with personal goals such as maintaining the status quo, avoiding career risk, working no harder than necessary, or maintaining a large fiefdom of direct reports.

It is to these individuals—those who have to become agile but don't want to—that we would like to direct our advice. Don't worry. We're not going to try to seduce you into trying agile, convince you of its merits, or tell you how to succeed. No, we're going to help you ensure agile failure. Then you'll be done with it and can go back to your comfort zone.

Although there are many ways to sabotage your agile project, for convenience we have grouped them into four categories: management issues, team issues, product owner issues, and process issues. In each instance, we will cite an example of someone who successfully caused agile failure, list the general guidelines for failure that the example is meant to demonstrate, and then list alternative techniques you can try to help you replicate the process. We hope this approach will allow you to fail quickly and avoid potential success.

Management Issues

Drew had seen management fads come and go. In his mind, agile was no different. A quick learner, he read a number of books and even took a class on agile. He didn't trust it, but, as a team player, it was his obligation to give it a try.

Drew picked team members and told them to "be agile." He told them that they would need to meet daily, estimate their work, and produce versions of their product (a database tool for storing artwork) every month.

Since Drew didn't trust agile or his team's ability, he attended every daily scrum, paying close attention and pointing out what the team was doing right and what it was doing wrong. Soon the daily meetings became a model of brevity and procedural correctness. As a bonus, no one spoke up about problems—especially in front of Drew. Drew had successfully followed our first guideline to agile failure.

GUIDELINE 1: ***Don't trust the team or agile. Micromanage both your team members and the process.***
To no one's surprise, the team did not produce impressive results. It didn't meet all of its iteration goals and was no more productive than it had been before. Drew conducted retrospectives that did not reveal any problems that he could fix. As a result, Drew threw away all the books he had read and directed the team to return to the old way of developing the project. Drew was following guideline 2.

GUIDELINE 2: ***If agile isn't a silver bullet, blame agile.***
While Drew went to one extreme by micromanaging his team, it is equally effective to go to the opposite extreme and not provide any guidance at all. Remember: While self-organizing agile teams are also self-managing, they are not self-leading. An agile team needs the type of leadership that provides a vision to work toward and motivation for achieving that vision. A strong agile leader, often in the form of a product owner, knows how to motivate a team with a description of an extremely desirable product that is just beyond what the team may think it can do. Freed to pursue that goal and provided with ongoing guidance from a product owner, an agile team can become truly high performing. Don't give your team that opportunity! If micromanagement isn't your style, follow guideline 3.

GUIDELINE 3: ***Equate self-managing with self-leading and provide no direction to the team whatsoever.***
While support for using agile may come from the highest levels of a company, often the adoption of agile will be driven by the agile team itself. Don't worry. You still have plenty of opportunities to create failure in those cases, especially if you are the manager. You

may want to start by undermining the evangelist on the team—the one who has read all the agile books and is taking the chance to promote agile. Brush off the rules he is asking you to follow. Interrupt the daily scrum with new directions. Change the priority of the iteration goals. It works well and is encapsulated in guideline 4.

GUIDELINE 4: ***Ignore the agile practices*****.**
They don't apply to management. If you want to be sure that agile doesn't take root, go straight to the agile team members themselves and let them know you think agile is a fad. Some of them will be skeptical to begin with, so it won't take much to convince them to ignore the practices. Remember, like Barney Fife, you have the power to nip this thing in the bud. Just follow guideline 5.

GUIDELINE 5: ***Undermine the team's belief in agile.***

Team Issues

Not all of us are managers. Don't worry, non-managers can wreak havoc at the team level, as well. Just take the case of the NotQuite agile team, tasked with developing inventory-management software. This team shows the power of consistency in bringing down an agile project. For its first iteration, NotQuite committed to completing six items from the product backlog; it finished four. Because it was the first iteration and most teams overcommit in their first iteration, the product owner cut the team a little slack. This didn't faze the NotQuite team.

For the second iteration the team again planned to finish six items; it finished five. The slight improvement only lulled the product owner into a false sense of security. NotQuite continued to chronically overcommit, falling short in the third, fourth, and fifth iterations. Soon, the product owner learned not to trust the team, and this undermined any success it may have had with agile—a fantastic implementation of guideline 6 for agile failure.

GUIDELINE 6: ***Continually fail to deliver what you committed to deliver during iteration planning.***
When falling short, don't make the mistake of going all-out on every iteration, reaching the last day panting with exhaustion time after time. A team like that could almost be forgiven for never quite delivering what it planned. A key to NotQuite's failure was its cavalier attitude toward missed commitments. Team members made it clear that it really didn't matter if something was finished on the last day of the iteration (as had been committed) or a few days into the next iteration. What's a few days between friends? Remember, a few days here and there can add up to quite a lot. If an agile team continually misses its commitments, it makes it impossible for the product owner to make plans and external commitments. This leads to guideline 7 for how to fail at agile.

GUIDELINE 7: ***Cavalierly move work forward from one iteration to the next. It's good to keep the product owner guessing about what will be delivered.***
Perhaps the best way to cause an agile project to fail is to follow guideline 8.

GUIDELINE 8: ***Do not create cross-functional teams. Put all the testers on one team, all the programmers on another, and so on.***
Merrilynn was able to use this guideline to kill her company's pilot agile project. Her organization was developing an application that would have separate Windows and Web-based clients. As a development director, Merrilynn had control over team composition and was able to create three separate teams: a Windows team, a Web team, and a test team. This team structure worked against the goals of agile. If Merrilynn had wanted to succeed, she would have instead created three teams that each included Windows, Web, and test skills. Because Merrilynn kept the teams separate, she made it impossible for any team to deliver the working software that an agile team is expected to deliver at the end of each iteration. Nicely done, Merrilynn.

Another option open to Merrilynn was putting all twenty of her people on one team. This would have violated the standard agile advice of creating teams of five to nine people. She could have justified it to anyone who questioned the decision by stressing the unique twoclient nature of her team's product. If she had chosen to create one large team instead of three reasonably sized teams, Merrilynn would have substantially increased the communication overhead of her team. This would have slowed progress and created complaints about how all the conversations in agile were a tremendous burden. If separating teams is too hard to justify, you can bog down a project very easily by following guideline 9.

GUIDELINE 9: ***Large projects need large teams. Ignore studies that show productivity decreases with large teams due to increased communication overhead. Since everyone needs to know everything, invite all fifty people to the daily standup.***

Product Owner Issues

As you can see, agile failure at the product owner level is easily achieved through miscommunication, general ignorance of the team's progress, and lack of education.

If the management and team guidelines aren't available to you, there is another route to take: Consider a takedown from the product owner angle. A product owner has many options at her disposal to bring an agile project to its knees.

Take Kathy, for example, who was the product owner for a team working on a video game. The team was making great progress on features with every iteration and showing more player "fun" every time. Kathy let team members keep thinking that this was all they needed to do. She never attended reviews, rarely tried the game, and requested stories that were meant to steer the game toward the product she imagined. If that weren't enough, Kathy didn't share her own vision with the team or the other customers to whom she reported (such as marketing).

A year into development, the game was demonstrated to a group of executives who were shocked at the direction the game had taken. It was not what they wanted to market. The disconnect between Kathy, the team, and senior management caused the project to lose six months of progress. Well done, Kathy!

Kathy demonstrated several guidelines for how an agile project can fail at the hands of a product owner.

GUIDELINE 10: ***Don't communicate a vision for the product to the agile team or to the other stakeholders.***

GUIDELINE 11: ***Don't pay attention to the progress of each iteration and objectively evaluate the value of that progress.***

GUIDELINE 12: ***Replace a plan document with a plan "in your head" that only you know.***
One of the tenets of iterative development is the discovery of the value of features being added as part of the whole. This is the reason that every iteration produces a potentially shippable release of the product. This is in stark contrast to plan-driven projects, which attempt to predict the utilization of resources so the product emerges complete from all the separate parts only at the end. When a product owner does not make that change, the agile team can quickly fail. It is just as critical to educate the product owner as it is to educate the team. Generally speaking, if you want to enusre agile failure quickly, avoid training at all.

The crucial role of product owner often is balanced by someone else who acts as the team's ScrumMaster or coach. On many successful projects, a certain amount of naturally occurring tension exists between product owner and ScrumMaster. A product owner always desires more, more, more features. The coach, by contrast, is responsible for monitoring the health of the team. If the team is being pushed too hard and is beginning to get sloppy due to fatigue, the coach pushes back against the product owner's desires for more. A good way to fail at agile is to eliminate this push-pull tension between the coach and product owner by following guideline 13.

GUIDELINE 13: ***Have one person share the roles of ScrumMaster (agile coach) and product owner. In fact, have this person also be an individual contributor on the team.***
As you can see, agile failure at the product owner level is easily achieved through miscommunication, general ignorance of the team's progress, and lack of education. You can compound that, if necessary, by having one person act in roles that are designed to balance each other. Following these guidelines to the letter is a great way to fail.

Process Issues

> Rather than align pay, incentives, job titles, promotions, and recognition with agile, create incentives for individuals to undermine teamwork and shared responsibility.

If all else succeeds, careful misapplication of process issues can bring down almost any agile project. Jon is a terrific example of a process nightmare, and he did most of his best sabotage without even knowing he was doing it. Jon was the lead developer for a Chicago-based team developing software designed to approve or reject loan applications. In addition to being the lead developer, he was also the ScrumMaster (note how he began by embracing guideline 13, which by itself can wreak havoc).

Jon and his team were new to agile and were anxious to get rid of its unneeded parts. They immediately dispensed with daily standup meetings, reasoning that since the team sat in the same general area, most conversations could be heard over the six-foot-high cubicle walls.

They also decided that having automated unit tests was unnecessary. Since theirs was a new application, there was no chance of breaking old code, and since all new code would be fresh in everyone's minds there would be little chance of accidentally breaking it. However, Jon and his coworkers did embrace refactoring and collective code ownership.

Their new rule was that any programmer could change the code of any other programmer at any time. They soon learned that refactoring and collective code ownership can be very dangerous without the safety net of automated unit tests to make sure you aren't breaking things while improving them. Jon and his team had unwittingly stumbled on these two guidelines for causing agile

failure.

GUIDELINE 14: ***Start customizing an agile process before you've done it by the book.***

GUIDELINE 15: ***Drop and customize important agile practices before fully understanding them.***
An alternative to these guidelines is to dive into the practices without understanding why you're doing them. As coaches, we encounter many teams who have learned a technique or been told to do something by someone and who then continued to do it even when they'd outgrown the technique (a subtle, yet effective, subterfuge). This brings to mind the story of the newlywed wife who cuts a quarter inch off both ends of every roast she cooks.

When her husband asks why she's trimming the roast that way, she has no ready answer; she does it that way because it's the way her mother always did it. Curious as to her mother's rationale, the wife calls her mother and asks why she taught her to cut the ends of the roast. Her mother says she only does it that way because her own mother taught her to do so. The young wife next calls her grandmother and asks why she cut a quarter inch off the end of every roast. Her grandmother tells her, "Because my roasting pan was too small. The roast wouldn't fit any other way." We capture this as our next guideline for agile failure.

GUIDELINE 16*: Slavishly follow agile practices without understanding their underlying principles.*
If you haven't been able to implement guidelines 14 through 16 and your agile project is succeeding despite your best efforts, you can bring even a successful project to a halt simply by changing nothing. What, you say? Change nothing? Follow the example of the StatusQ team. StatusQ, assigned to build a new Web-based reservation system, got off to a good start. Team members were new to agile but did a good level of research and sent a few of their people off to become certified ScrumMasters.

The project quickly benefited from the new practices. In a month, StatusQ had a simple Web site up and running and was able to demonstrate a few key interfaces that gave its customers a lot of confidence that their vision of an accessible and powerful reservation system would work.

StatusQ never held a retrospective at the end of each sprint. The ScrumMaster didn't push for it because he didn't see the need. Everything was already working wonderfully well. You can encourage this behavior on your own team by complaining loudly to your ScrumMaster and team members if they try to hold a retrospective. Tell them that it's a waste of time to sit and talk about a project that's going well. Tell them that retrospectives only make sense when things are going wrong.

Over time, things at StatusQ started to slow down. The rate of change and the growing code base were creating maintenance problems. Changes to the system from the customers were supposed to be a benefit to them, but the code couldn't keep up. Code stability became so bad that the project seemed to be moving backward. Finally company management stepped in, put a halt to the changes, and finished the contract at half price.

This team had unknowingly demonstrated our next two guidelines for agile failure.

GUIDELINE 17: ***Don't continually improve.***

GUIDELINE 18: ***Don't change the technical practices.***
Company process issues that at first seem unrelated to the project can have a negative impact on morale and motivation. Consider the case of Dave, an up-and-coming artist who worked for a mobile phone game developer. He welcomed his company's adoption of agile, as it made a lot of sense to him. The new agile teams consisted of programmers, artists, designers, and a number of other people from numerous disciplines. Everyone relied on each other to create iterations of their game. If the artists did a good job, then the entire team looked good. Dave often dropped what he was doing to help his teammates iterate on the art to improve the game. This often came at the expense of his own work, but, for Dave, team goals came first.

Dave's team did a great job and produced a hit title that sold many thousands of copies and earned the company substantial profits.

At the end of the year, Dave had his first performance review with the company's lead artist. Dave was shocked to learn that his yearly bonus was small. Dave was told that he was judged by the senior artist in the company to have missed his art production goals. As it turns out, the senior artist was counting the number of iteration task cards that Dave completed and based his judgment on that rather than on the real amount of work Dave completed.

Chagrined, Dave returned to his team vowing to make sure his task cards took priority over the needs of the team. Guideline 19 can be your backup plan for any enthusiastic agilists in your company.

GUIDELINE 19: ***Rather than align pay, incentives, job titles, promotions, and recognition with agile, create incentives for individuals to undermine teamwork and shared responsibility.***
A tenet shared by all agile processes is that work is prioritized based on the value provided by each feature. Other factors, such as risk and knowledge creation, are considered, but the amount of value delivered remains the dominant factor. A sure fire way to

ensure agile failure is to ignore this tenet and instead follow guideline 20.

GUIDELINE 20: ***Convince yourself that you'll be able to do all requested work, so the order of your work doesn't matter.***
There are, of course, other ways to fail in addition to those collected here. In your effort to undermine a successful agile project, you may have already discovered some on your own. Of course, you are probably keeping quiet about them because it's critical that the sabotage not be detected until the bridge is blown. We are confident that, through diligent application of the guidelines here or those that only you know—or both— you will be able to plot the downfall of your next agile project and ensure agile failure.