Agile Anti-patterns

Andrew Cox
@coxandrew

What is an anti-pattern?

“A pattern that may be commonly used but is ineffective and/or counterproductive in practice.”

— Wikipedia

The term was coined in 1995 by Andrew Koenig,[3] inspired by Gang of Four's book Design Patterns, which developed the concept of design patterns in the software field.

In the old days, they just called these 'bad ideas'. The new name is much more diplomatic.

See Also

- Scrumbut
- Process Smells
- Cargo Cult Agile

Why learn anti-patterns?

Identifying bad practices can be as valuable as identifying good practices.

2 key elements:

- A repeated pattern of action that appears beneficial, but ultimately produces more bad consequences than beneficial results
- A refactored solution exists that has been proven in actual practice and is repeatable

There must be at least two key elements present to formally distinguish an actual anti-pattern from a simple bad habit, bad practice, or bad idea:

“Scrum is like your mother-in-law. It's constantly pointing out your shortcomings.”

— Ken Schwaber

Ken Schwaber at an Agile Vancouver conference in 2008.

Ken Schwaber is the cofounder of Scrum (w/ Jeff Sutherland) and a founder of the Agile Alliance.

The trick is that you're supposed to learn from that feedback and fix your problems.

The Anti-patterns

We Tried Baseball and It Didn't Work

Or, "Premature rejection of process".

Smell: The people rejecting the practice have never read anything about it.

This is often the case of not understanding the underlying philosophy behind a process or not implementing it correctly. If you find yourself in this situation for a practice that other development teams seem to get a lot of value out of, you should examine how you're doing the practice more closely.

Of course, it might *not* work for you and you should never just do a practice because it's prescribed by a book, blog article, or your local agile user group :)

Antidote:

* Learn as much as you can about the practice
* Give it a try for at least 2 sprints
* Retrospect on whether the practice is worth continuing
* Try it again

Agile Coding Sandwich

Smells:

* Design > Coding > Testing > Documentation > Release
* Requirements don’t change
* A “bug fixing” sprint

Antidote:

* JIT design
* Integrate QA and Docs into your Definition of “Done”
* Cross-functional teams

Kitchen Sink Product

Smells: Everything is a must-have.

This usually arrises out of a fear that if features don’t make it in the initial release, they’ll never get in. This fear is natural since that’s the way waterfall development taught us that software development works.

Why bad:

You don’t learn about your product from real world use. See: Lean Startup

Antidote:

The MVP - release before you’re ready to small group and listen to their feedback to inform your next moves.

Deadlines

Specifically, deadlines with fixed scope.

Smells:

* “Bug fixing” sprints
* Lots of bugs

Why bad:

* Force you to compromise on quality

Antidote:

* If you must have deadlines, flex on scope rather than quality
* Don’t use hard deadlines

The Absentee Product Owner

Smell:

* Developers establish priority of stories

Antidote:

* Get a product owner that can be accessible - even if it means making one of your developers be the official Product Owner.

Someone needs to be responsible for the priority of the stories and talking with customers. Your project needs that “single wringable neck” that has the appropriate information and authority to make decisions on priority.

Customer Driven Development

Smell:

* Prioritizing customer requests above all other work.
* Checklist product

Why Bad:

You're always reactive. The customer doesn't always know what they want. Feature checklists lead to complex, cluttered design that doesn't please anyone.

Antidote:

* Say "thank you for your feedback"!
* Don't be afraid of your customer becoming too advanced for your product (37Signals philosophy)

Double Duty

Smells:

* Developer as Product Owner
* Manager as Scrum Master
* Scrum Master + Product Owner

Why bad:

Conflict of interest

Antidote:

Don’t :)

Watching the Runners

(aka “Maximizing Capacity”)

Smell: Management concern over idle developers

Why bad:

In an agile project, you want to watch the baton, not the runners.

Maximizing capacity will skew the stories toward your available skills rather than toward the most important stories for your product.

Antidote:

* Allow slack time
* Encourage pair programming
* WIP limits
* Establish “shave the yak” time where people can add items to their list of things to do when they’re idle

See also:

* Silo People
* Layer Cake Stories

Retrospective Actionless Items

Smells:

* Not following through with action items
* More than 2 action items from a retrospective

Antidote:

Decide on only 1 or 2 things you want to improve with your process and make sure to evaluate the effectiveness of the change(s) at the next retrospective.

A “Bug Fixing” Sprint

Smell: You need time for your product to stabilize before a release

Antidote:

* Continuous Integration
* Demo your product at the end of each sprint

Sword of Damocles Standups

The Damocles of the anecdote was an obsequious courtier in the court of Dionysius II of Syracuse, a fourth century BC tyrant of Syracuse, Italy. Pandering to his king, Damocles exclaimed that, as a great man of power and authority surrounded by magnificence, Dionysius was truly extremely fortunate. Dionysius then offered to switch places with Damocles, so that Damocles could taste that very fortune first hand. Damocles quickly and eagerly accepted the King's proposal. Damocles sat down in the king's throne surrounded by every luxury, but Dionysius arranged that a huge sword should hang above the throne, held at the pommel only by a single hair of a horse's tail. Damocles finally begged the tyrant that he be allowed to depart, because he no longer wanted to be so fortunate.[2][5]

Dionysius had successfully conveyed a sense of the constant fear in which the great man lives. Cicero uses this story as the last in a series of contrasting examples for reaching the conclusion he had been moving towards in this fifth Disputation, in which the theme is that virtue is sufficient for living a happy life.[6] Cicero asks:

> Does not Dionysius seem to have made it sufficiently clear that there can be nothing happy for the person over whom some fear always looms?

Smell:
* Management tracks hours worked

Antidote:
* Watch the baton

Layer Cake Stories

Smells: Integration stories

Why bad:

* Front-end and back-end get out of sync
* You often aren’t in a releasable state

Antidote: Slice stories vertically

Premature Allocation

Smells: Assigning users to individual stories at the start of an iteration.

Why bad:

* Bottlenecks
* Bus factor
* Silo People

See also:

* Silo People

Silo People

Smells:

* Premature Allocation
* Prioritizing stories based on team skills

Why bad:

Also known as “stove pipe”. What are the dangers of having developers that are overly specialized?

* Stories will be prioritized based on the skills of the developers - not customer need
* Low “bus factor”, which means bottlenecks when specialized developers are unavailable

Antidote:

* Prioritize stories based on product / market needs
* Pair program to share knowledge

Multitasking

Smells:

One developer is working on many stories at once

Why bad:

* Bottlenecks
* Not finishing stories in a sprint

Antidote:

* WIP limits

Agile as Project Management

(aka "Not using XP practices")

Agile is a holistic process. It is tempting to take the rituals of Agile like sprints, standups, story points and retrospectives, but not implement the engineering practices described in eXtreme Programming.

> the solid agile engineering practices included in Extreme Programming pay for themselves within the first few months. Without XP's agile engineering practices, code quality and productivity asymptotically decreases over time. With them, productivity starts lower, but then it asymptotically increases. I can't prove it, but my sense is that the two curves cross at about the eight-week mark. Maybe sooner. Agile engineering practices aren't only important--they pay for themselves! Doing anything else is pure negligence... if you understand your options. Scrum is silent on the matter.

The updated Scrum Guide for 2011 includes a recommendation that Agile will not work without complementary XP practices, such as <citation needed>:

* Test Driven Development
* Pair Programming
* Continuous Integration
* Refactoring
* Coding Standards
* Simple Design

XP Practices

- Test Driven Development
- Pair Programming
- Continuous Integration
- Coding Standards
- Simple Design
- Refactoring

The Nots

These don’t fit the formal definition of an anti-pattern, but are worth an honorable mention.

These are things that are often easy to neglect in an agile project, but I think are fundamental to a well-functioning agile team.

- Prioritized backlog
- Definition of “Done”
- Spikes for risky stories
- Retrospectives
- Design
- Prioritizing technical debt

Further reading

- *Out of the Crisis* by W. Edwards Deming
- *Scaling Lean & Agile Development* by Craig Larman and Bas Vodde
- *The Decline and Fall of Agile* by James Shore
- *The Scrum Guide*
- *We Tried Baseball and It Didn't Work*

Upcoming meetups

- November - *How to design stuff that matters fast* (Ariadna Font)
- December - *Experiment with Story Splitting* (Jake Goulding)
- January - *THIS COULD BE YOU!*

Retrospective