Iterating As If You Meant It

Following are the notes from the Agile Islands 2019 version of this talk.

Outline

- Do you know "TDD As If You Meant It"?
    - Focus on *deliberate practice* as a way to increase and refine skill.
    - Practise test-first programming with special emphasis on what makes it effective: tiny steps, relentless refactoring / defer generalization to the last moment, microcommitting
    - Some people treat this as an *étude* to practise entirely outside "real work"; some people treat this as a reminder of how to do "real work" more effectively. You choose.
- "Heart of Agile" and "Modern Agile" both express clearly a reaction to the shift in meaning of "Agile", trying to capture the essence of what makes the Agile philosophy work well.
    - Alistair Cockburn has framed this as a return to the essence of Agile.
    - Joshua Kerievsky has framed this as an evolution of the essence of Agile.
    - I have a terrible marketing department and suffer from selective idealism, so I merely try to do "good things" and persist in calling it "Agile" and even "XP". I even like to use the term "Lightweight" to distinguish "Agile as we meant it" from "Agile as she is spoke".
- I propose that we Iterate As If We Meant It.
- Among the problems with "Agile as practiced", I notice that organizations routinely *play* at working iteratively and *almost ignores* delivering incrementally.
    - Milestones become "iterations".
    - "Sprints" rarely end in a shippable increment of anything.
    - Introspection rarely happens out of fear or cynicism; adapting typically only happens when they have exhausted all other options.
    - All this leads to changing the words but keeping the results, which leads to various forms of chaos: failed projects, disheartened workers, cynical managers, and a soiled brand.
- What would happen if we simply iterated as if we meant it?
    - We would *ship* an increment of the product every iteration, even when we thought we couldn't possibly do it.
    - We would actually *need* to change the way we worked, so we would have an *incentive* to achieve something of value from retrospectives.
    - We would have practical reasons for many of the practices that we generally agree would help, but "would never work here" or "we never have time to do". We would do them because

    we need them, rather than out of some sense that they might help, or worse, out of some sense of duty to follow “the correct rules”.
  - We would increase cash flow, initiate cash flow sooner, show greater results to the rest of the company, gain trust and credibility, and then maybe improve our lives at work.
  - (Of course, we might make other people look bad. This is a problem that we can’t handle in this talk.)
- Why *iterate* as if we meant it? Why focus here? Frankly, because nothing else seems to be working, so we need to try something else. Maybe enough people can agree that iterating would help them, so we could more-easily argue that they should just try focusing on this one thing for a few months to see what happens. It seems easier than reinventing an entire brand to rearticulate our long-held values and beliefs about how to build better companies.

What If We Iterated?

- We could turn the single-pass Waterfall into Lean Startup in a few easy steps. This, alone, provides some clearer motivation to adopt practices like test-first programming, TDD, BDD, Continuous Delivery, and even Lean Startup. See “How Test-Driven Development Works (And More!)” for a recap of this section of the talk, albeit an earlier version from 2009.

The talk, as I performed it at Agile Islands 2019, ended here. Following are the topics that I had available to discuss in detail. I mentioned some of them briefly, but not them all and not in much detail. See the References to learn more about them.

How Do We Start?

- I propose Theory of Constraints as a guiding principle for improving how we iterate. We start with the proposition that bottlenecks restrict us from iterating effectively. We can also frame this as “time-boxed iterating will make the bottlenecks easier to spot and harder to ignore”.
  - Describe Throughput Accounting and Bottleneck Theory, the very short version.
  - We can use ToC to guide us to iterate on adopting practices from our “Agile bag of tricks”. Example: From Waterfall to Lightweight Operations in 6 easy steps.
- How do we adopt various practices once we’ve chosen them? Combine two strategies: Immediate Rapid Iteration in the small and Cut Feedback Loops In Half everywhere else.
- Where you have total control over the way you work, jump immediately to rapid iteration: TDD/TCR, Talking in Examples, Continuous Build, Continuous Integration (Main-Line

Development by default), Big Visible Charts, Daily Risk Management, Relentless Focus on the Task (GTD, Pomodoro, Monotasking, …).

  - Where you have to negotiate with others, struggle to cut feedback loops in half. Map the value stream, look for waiting or rework. Cut waiting in half with increased communication and collaboration. Cut rework in half by delivering in smaller increments/batches with more-frequent review.
  - Cutting a feedback loop in half, rather than jumping directly to microiterating, tends to balance two competing forces: pushing the group out of its comfort zone to change how they work, providing “just enough” challenge to avoid a feeling of learned helplessness. This makes it a suitable strategy for areas where you don’t have significant authority, but rather need to engage in ongoing influence.

Specific Techniques

- Talking in Examples creates a bias in favor of smaller slices of features, so we have more options to deliver a smaller increment sooner, get feedback sooner, then adjust. Sometimes we figure out that delivering 20 kg of a feature is enough to start the inflow of cash, and we can defer the remaining 80 kg by a few months!
  - You can start *right now* by learning how to talk in examples and practising it. No agreements required, no process document, no formal approvals… just start. When someone describes a feature or clarifies the meaning of some acceptance criteria, *check your understanding with an example!* Keep doing it. Maybe you turn some of those examples into running customer tests, if doing that helps you feel more confident that you built what they asked for.
- The Two-Minute Rule and the Inbox technique from *Getting Things Done* combine very well with working in short bursts (Pomodoro, Monotasking) to create and sustain focus on the task at hand. This facilitates working on smaller increments of value!
  - Many people resist delivering incrementally because it takes 2 days to do anything. They don’t practise working in short bursts. They believe that *flow* requires *time*, but it really requires more *focus* than time. I have improved my focus by iterating as if I meant it! By training myself to work in short bursts, I get into a deep focus state relatively quickly, then if I can sustain that over a few hours, I find it relatively easy to achieve *flow*.
- I love TDD, but let’s remember that we do it “only” to sustain evolutionary design (and architecture!). We do this to protect our capacity to deliver features *and* to practise and become more comfortable with delivering work in smaller increments as well as iterating. Writers call it “editing”, while programmers call it “refactoring”. If we iterate over our designs as if we meant it, then gradually we get more (and more stable) capacity to deliver features.

- Combine these three practices to unleash the awesome power of iterative and incremental work! There is more, but this is already enough to start.

So Do This!

Are you still worried about convincing the people around you? I understand. You can just start here:

1. Practise working iteratively (short bursts, time them, get things out of your head).
2. Practise working incrementally (focus on completing things, and if you can't, then slice them more thinly in order to increase your chances).
3. Profit (notice the next time you find it easy to do something in just 20 minutes or when you stop early, show it to a customer, and they love it).

If you're not sure how to start in your environment, then talk to me and perhaps we can figure it out together.

References

J. B. Rainsberger, "How Test-Driven Development Works (And More!)". The article that corresponds with the second half of the talk.

Naresh Jain, "Using Theory of Constraints and Just in Time Practice to Coach Agile Teams". Naresh and I have performed workshops together on this topic. In these presentation slides, you'll find an earlier version of what we did together. Does not include an introduction to Theory of Constraints.

Eliyahu Goldratt, *The Goal*. An introduction to Theory of Constraints, explaining Throughput Accounting and Bottleneck Theory. Although some people criticize applying Theory of Constraints (linear modeling) to inherently complex adaptive systems (such as a group of people delivering software), I believe that most organizations can benefit from analyzing their value stream even to "only" a linear approximation.

J. B. Rainsberger, "Three Steps To a Useful Minimal Feature". A worked example of the value of Talking in Examples, which outlines the "Contract, then Expand" approach I teach to exploring features. The larger technique overlaps with User Story Mapping, so I consider them different styles of the same approach. Includes a handful of examples of Examples.

J. B. Rainsberger, "The Two-Minute Rule". When a request for work finds you, if you can complete it in

two minutes, then do it now; otherwise, schedule it for later.

J. B. Rainsberger, “Getting Started with Getting Things Done”. If you don’t have the time nor energy to read a 300-page book, maybe you an read a handful of pages to start!

J. B. Rainsberger, “A Proven Path to Effectiveness”. A single diagram showing how an individual programmer can “find time” to improve.

Contact Me

- J. B. Rainsberger. Call me “J. B.” or “Joe”, whichever you prefer.
- https://ask.jbrains.ca :: https://tell.jbrains.ca
- https://www.twitter.com/jbrains
- Read more: https://www.jbrains.ca, https://blog.thecodewhisperer.com (mostly programming, design, testing) https://blog.jbrains.ca (mostly other topics)
- Learn TDD now: https://tdd.training
- Custom remote workshops are available in 2020 to adventurous organizations looking to improve. Please contact me to find out how to get started!