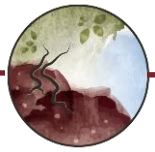
Read by Refactoring

Mindshift 1: The Insight Loop

Facilitator's Guide

Contents



Advance Planning

Schedule a ½ day mob.

Prepare the agenda.

- Mob Cycle #1: 2 hours
- Mob Cycle #2: 1 hour
- Insight Loop Cycle: 30 minutes
- Mob Cycle #3: 15 minutes

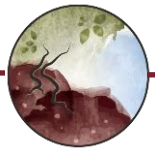
Prepare communication to developers that outlines:

- Empathy of time it takes to de-mystify code
- Intention of facilitating method that will reduce time for stories, reduce bugs, and make it less frustrating for them.
- Explain that this will be a legitimate project and is supported by authority.
- Request that they select the ugliest long method they can find, noting the specific code doesn't matter.
- Attach the agenda.

Add story to the board that represents the team mob.

Mob Preparation

- [ ] Have whiteboard accessibility or flipchart available.
- [ ] Have audio/visual prepared for sharing the expertise videos.
- [ ] Have the expertise videos prepared for easy access at the right times.
  - [ ] Video #1: Honest Names
  - [ ] Video #2: Committing – Safe
  - [ ] Video #3: Committing – Better
  - [ ] Video #4: Committing – No Worse
  - [ ] Video #5: Committing – Just Enough Process
  - [ ] Video #6: Committing – Deep Safety
  - [ ] Video #7: The Insight Loop



Facilitation

Mob Cycle #1 (2 hours)

The first phase is identifying code and pulling it out. The developers will naturally want to find exactly the right code and pull it out in exactly the right way. **Don't let them do that.**

Assure the developers that it doesn't matter if they get the wrong thing or extract it incorrectly. As we practice, extracting will become very fast, and when it's found that the wrong thing has been extracted, you can always undo it with inline method and extract the right thing.

Instructions

Habit Shift
One of These Things is not Like the Other Ones

- ☐ Ask them to identify one thing in the method.
  Use the zoom-out approach to make it easy to see
- ☐ Have them pull it out (extract to a method).

- ☐ Have them scan through and generally identify at least one thing it does.
- ☐ Have them name it:
  probably_[whatever that thing is]_AndStuff

Habit Shift
Names that Don't Suck

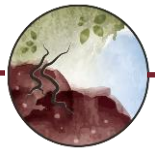
Reaction

They will be very concerned about that name!

Technical Expertise: Naming

To alleviate their concerns, show them Video #1: Honest Names. Once they have watched it, have them name the method as you instructed earlier.

Success Check
We have a second method and it's named according to the pattern provided.



The second phase is very simple: getting to check-in.

At this point, all the objections you'll experience from the developers are addressed as they will occur through the content below.

Instructions

- ☐ Ask them to check it in on to the main that will deploy.

Reaction

They will undoubtedly refuse to check-in.

Facilitate

- ☐ Ask them to list all the reasons why we shouldn't check it in and write them down on the white board. They WILL fall within at least a couple of these categories.

    - ✓ TOO DANGEROUS No testing; may introduce bug
    - ✓ TIME WASTE Too small; not worth it
    - ✓ TOO CRAPPY Name is bad; code isn't ready; embarrassing
    - ✓ NO PROCESS No code review or process

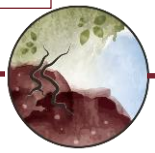
- ☐ If any of these categories do *not* come up, be sure to add them.
- ☐ Tell them that we will address the categories one at a time.

As facilitator, know these categories match these solutions.

- ✓ Too Dangerous – Safe
- ✓ Time Waste - Better
- ✓ Too Crappy – No Worse
- ✓ No Process – mobbing

Addressing these categories will **later** allow you to introduce the mantra "safe, better, and no worse" that they can practice as a new standard to what is committable.

Success Check
Everybody agrees that if those four criteria were met, then it would be safe to commit.



Technical Expertise: Safe

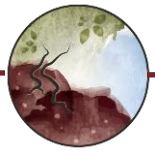
- Tell them that the **too dangerous** issue will be addressed first, by understanding safe.
- Show them Video #2: Committing – Safe.

Facilitate

- During the video, draw levels of proof on the white board.
- After the video, note that yes, this first time we won't achieve top proof. We WILL next time. Meanwhile, let's make this as safe as possible to commit.
- Ask them to use higher levels to prove safe enough.

For this context, safe enough means that it's no more likely to cause a bug than anything else already checked in. As such, allow the developers to do anything they need to meet the definition of "safe enough". Suggest that they use techniques from the levels of proof that are higher than running tests.

Success Check
The developers feel that the objection of "too dangerous" has been resolved.



Technical Expertise: Better

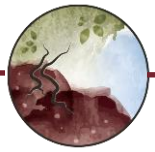
- ☐ Tell them that we will now address the **time waste** issue, with a focus for making it better.
- ☐ Show them Video #3: Committing – Better.

Facilitate

- ☐ After the video, ask what the aspects were that made it better.
  Examples include security, performance, readability, and testability.
  *Your goal is to get developers to list at least **one**.*

- ☐ If they do not feel anything was made better, offer the following items.
  - ✓ They made long method easier to read.
  - ✓ The new method is easier to read and test.
  - ✓ We can track how data flows in and out because it's now visible.

Success Check
The developers agree that the objections that fell within the "time waste" cluster have been resolved.



Technical Expertise: No Worse

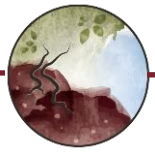
- ☐ Tell them that we will now address the crappy code issue, with a focus for making it no worse than it was.
- ☐ Show them Video #4: Committing – No Worse.

Facilitate

- ☐ After the video, ask what are all the aspects of code that matter.
- ☐ Write on whiteboard or flipchart each aspect they list.
  Examples will include security, performance, readability, and testability, and you can use these to prompt the start of that list.

- ☐ If they feel that things were made worse, ask what the minimum change would be to make to the commit in order to not make it worse.

Success Check
The developers agree that the objections that fell within the "crappy code" cluster have been resolved.



Technical Expertise: No Process

- ☐ Tell them that we will now address the no process issue, with a focus on replacing that with mobbing.
- ☐ Show them Video #5: Committing – Just Enough Process.

Facilitate

- ☐ Lead a discussion with the team on the following questions.
  - ✓ What piece of the process are no longer necessary because of the way we're working (mobbing and safely committing)?
  - ✓ Is there anything else we need to do?
  - ✓ Is there anything else we need to do?

- ☐ Have them perform the minimum necessary to address any remaining concerns.

This is the opportunity for the developers to realize how much they can bypass in their regular process when they are 1) working in mob and 2) checking in a single safe refactoring.

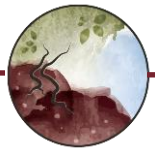
There are parts of the process they will likely feel are necessary, but you should challenge them to reconsider. These are code reviews, checking with designer or Product Owner, or running the full test suite again.

Success Check
The developers agree that the objections that fell within the "no process" cluster have been resolved.

Instructions

- ☐ Ask them to check it in.

Success Check	Habit Shift
A commit was executed.	Tiny Commits



Mob Cycle #2 (1 hours)

This entire cycle is about safety. The first mob was about covering better and no worse, and we did safety as much as we could after the fact. Now we are going to focus on safety from the beginning. Developers will be very happy to hear this! They will also become more pedantic, but so you will through these instructions.

There are some actions for you to notice and adjust their behavior. These include:

- Changing code by editing text.
  Instead, have them copy and paste or use tool capabilities.

- Combining or skipping steps.
  Instead, ask them how they would prove that their transformation is bug-for-bug compatible. Since it's not possible, they will undo the combination step.

- Reading, analyzing, or understanding the code in order to create safety.
  Instead, remind them that we are creating a **process** for safety that will works without understanding the code.

Instructions

- ☐ Ask them to identify one thing to extract.
- ☐ Explain that now we want to do this with deep safety at the top level.

Technical Expertise: Deep Safety

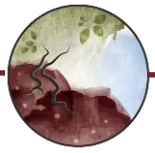
In preparation to extract with safety, show them Video #6: Committing – Deep Safety.

Instructions

- ☐ Ask them if they will be using the tool approach or the recipe approach.

RECIPE APPROACH
If it's the recipe approach, open the recipe on the screen and do the following.

Habit Shift
Bug-for-Bug Compatibility



- Have them fork it on Github and clone. If they don't have experience on Git, then fork on Github and edit on Github.
- Going one step at a time through the recipe, have them do each action below for **every step**.
  - Remove, adjust, or change to fit their language
  - Commit any changes to the recipe
  - Ensure everybody in the MOB is comfortable with the step)

TOOL APPROACH

If it's the tool approach, have them talk about how they will discover and handle tool limitations.

- Have them write the limitations down as a set of check steps and what kinds of code to check for.

☐ Extract to a method using the tool or recipe and name just like last time.

Success Check
Everybody agrees that this extraction is safe and doesn't require running tests.

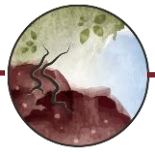
Instructions

☐ Check if ready to commit. Is it:

a. Safe?
b. Better?
c. No Worse?

☐ Ensure the commit is performed.

Success Check
A commit was executed.



Insight Loop Discussion (30 minutes)

Now that you have practiced a set of techniques twice, it's time to articulate how they all come together. While each action is a good independent habit shift, all four of them combined culminate into a bigger mind shift, called the Insight Loop.

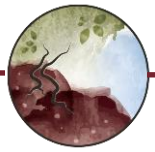
Technical Expertise: Insight Loop

In preparation for the de-brief, show them Video #7: The Insight Loop.

Facilitate

- ☐ While the video is playing, draw the insight loop cycle as described in the video on the whiteboard or flipchart.
- ☐ Lead a discussion that ensures they realise these two things:
  - ✓ Each of the first 2 mobbing iterations was one cycle of the insight loop.
  - ✓ It is cognitively easier to work this way.
- ☐ Keep the insight loop drawn on the whiteboard through remaining rounds of mob cycles for this session.

Success Check
Developers exhibit an aha moment.



Mob Cycle #3 and beyond! (30 minutes)

This cycle is about fluency. The Insight Loop is the most effective when developers do many cycles that are small instead than trying to do a lot in a single cycle. During this session, developers will likely get down to 6-8 min per cycle. With practice over the next two weeks, they will get to 1-3 minutes each cycle.

Instructions

- ☐ Set it up as a game:
  How many iterations can we complete in the next 30 min?
- ☐ Extract another method, but guide them only by helping them keep track of where they are in the insight loop (it's on the whiteboard).
- ☐ Have them repeat the cycle as many times as possible.

Success Check
The last iteration is less than 8 minutes long.