Code by Refactoring Foundations

The Insight Loop

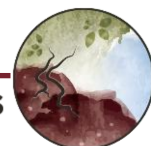
https://insightloop.digdeeproots.com

Facilitator's Guide

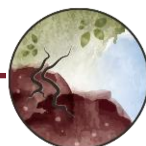
Contents

Advance Planning: Coaching the Habit Shifts 3
Mob Workshop Advance Planning 3
Physical Mobbing Room Set-Up 4
Virtual Mobbing Set-Up 5
Facilitating Mob Cycle #1 – 2 hours 6
- Phase 1: Facilitator Notes 6
  - Instructions 6
  - Reaction 6
  - Technical Expertise: Naming 6
- Phase 2: Facilitator Notes 7
  - Instructions 7
  - Reaction 7
- Phase 3: Facilitator Notes 8
  - Technical Expertise: Safe 8
  - Facilitation Actions 8
  - Technical Expertise: Better 9
  - Facilitation Actions 9
  - Technical Expertise: No Worse 9
  - Facilitation Actions 10
  - Technical Expertise: No Process 11
  - Facilitation Actions 11
- Instructions 11

Facilitating Mob Cycle #2 – 45 min 12
- Facilitator Notes 12
  - Instructions 12



Technical Expertise: Deep Safety 12
Safely Extracting: Facilitator Notes 12
Instructions 12
.... 13
Instructions 13
Facilitating Insight Loop Awareness – 20 minutes 14
Facilitator Notes 14
Technical Expertise: Insight Loop 14
Facilitation Actions 14
Mob Cycle #3 and Beyond! – 30 minutes 15
The Fast Commit Game: Facilitator Notes 15
Instructions 15



Advance Planning: Coaching the Habit Shifts

Request support from the relevant parties to achieve the expected change from this adoption.

- Product Owner: allocates a story for the sprint
- Team Lead: encourages the skill development for the sprint
- Scrum Master: focuses on this skill development for this sprint
- Developers: shows up to learn and help others learn

Schedule a ½ day **mob workshop** (4 hours).

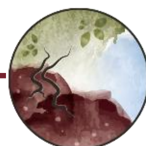
Mob Workshop Advance Planning

Prepare the agenda.

- Mob Cycle #1: 2 hours
- Mob Cycle #2: 45 min
- Insight Loop Cycle: 20 minutes
- Mob Cycle #3: 30 minutes

Send a 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 is 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.**



Physical Mobbing Room Set-Up

- [ ] Prepare mobbing station based on visual below. The intention is to have one developer with hands on keys and anybody else may make suggestions. Rotation is important so everybody is in the code.

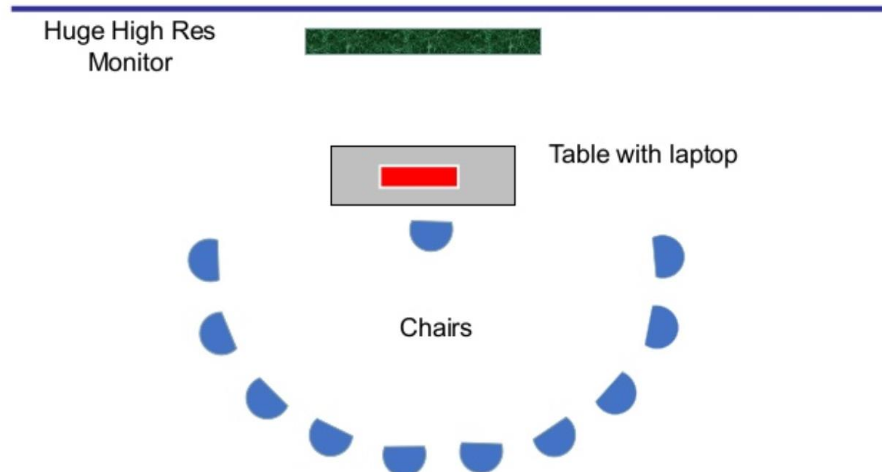
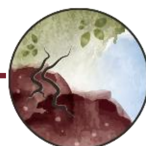


Figure 1: Image Credit /in/camillebell cbell@camillebellconsulting.com

- [ ] Have whiteboard accessibility or flipchart available.
- [ ] Have audio/visual prepared for sharing the expertise videos.
- [ ] Have the expertise videos provided at https://insightloop.digdeeproots.com/ ready to be shown at the right times.

Virtual Mobbing Set-Up

- [ ] Access and follow the guidelines for setting up the remote mobbing environment. https://www.remotemobprogramming.org/
- [ ] Have Miro (https://miro.com/) or preferred shared whiteboarding available.
- [ ] Verify that the video conferencing software has the option to share computer sound, such as Zoom (https://zoom.us/).
- [ ] Have the expertise videos provided at https://insightloop.digdeeproots.com/ ready to be shown at the right times.



Facilitating Mob Cycle #1 – 2 hours

Phase 1: Facilitator Notes

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.**

The mindset for this mob cycle is that it doesn't matter if they get the wrong thing or extract it incorrectly. As they practice, extracting will become extremely fast, and when it's found that the wrong thing has been extracted, they can always undo it with inline method and extract the right thing.

Instructions

Habit Shift

One of These Things is not Like the Other

- ☐ Verify that they brought painful code and ask them to bring it up.
- ☐ Ask them to zoom out to the point they can't read it.
- ☐ Now that they can't read it, ask them what parts are interesting.
- ☐ When they identify a part, have them zoom back in and extract it.
- ☐ Make a point that this is not safe; this is just getting them used to **finding things that are not like the others**, and we'll practice safety in the next cycle.
- ☐ Recommend that they make it a nonsense name (such as "foo"). This is so that we can do the extraction quickly, and then name it well in the next step.

Habit Shift

Names that Don't Suck

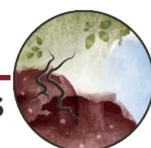
- ☐ Note that "Foo" is not very descriptive, so have them scan through and generally identify at least one thing it does.
- ☐ Have them name it: **probably_[whatever that thing is]_AndStuff**

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.

Phase 2: Facilitator Notes

The second phase is quite simple: getting them 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. Your job will be to facilitate through their concerns with curiousity, knowing that they will be categorized and addressed one by one.

Instructions

Ask them to check it in on to the main branch, such that will deploy to production. If they are willing, they probably feel safe because it will be just checked in on a branch. Clarify that the check in is to the **main**.

Reaction

They will undoubtedly refuse to check-in. Your next hour will be spent facilitating through all the very valid objections!

Facilitate

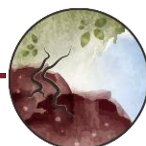
- ◻ Ask them to list all the reasons why we shouldn't check it in and write them down on the white board or virtual board. Write down everything, but they WILL fall within at least a couple of these categories.

  - ✓ TOO DANGEROUS No testing; may introduce bug
  - ✓ TOO CRAPPY Not worth it; name is bad; code isn't ready; embarrassing
  - ✓ SYSTEMATICALLY BAD Hurt performance; hurt debuggability
  - ✓ 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.

Success Check

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



Phase 3: Facilitator Notes

The third phase addressing each of these four categories so that the developers can feel safe about checking in the method they extracted and named with the new naming convention.

As facilitator, know these categories match these solutions.

- ✓ Too Dangerous – Safe
- ✓ Too Crappy - Better
- ✓ Systematically Bad – 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.

Technical Expertise: Safe

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

Facilitation Actions

While you want to watch all the videos in advance, it's especially critical to watch this one in advance. **During the video,** draw these levels of proof on the white board or shared screen.

Level	How to Prove
Bug for bug compatibility	Static analysis
Quantified	Telemetry
TDD a story change	Unit tests

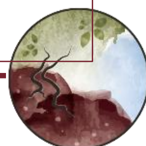
After the video, note that our old safety standard is now the lowest standard. While we can't get the top safety of bug-for-bug compatibility, We WILL next time. We can't do telemetry during the mob, **so have them perform static analysis on the code to achieve safe enough.**

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".

What if they get stuck? Ask these questions:

1. What changes did you make? They'll say extract method and rename.
2. How could those have caused a bug? They'll give answers you don't care about.
3. What static analysis techniques can you use to show that those problems didn't happen?

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



Technical Expertise: Better

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

Facilitation Actions

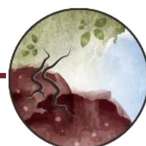
After the video, ask what the aspects were that made it better.
Examples include security, performance, readability, and testability.
*Your goal is to get developers to list at least **one.***

If they do not feel anything was made better, offer the following items.

- ✓ Is the originals method easier to read?
- ✓ Is the new method easier to read and test?
- ✓ What data flows in this new method and comes out of it?
  When they answer, ask how obvious that was before they extracted it.
- ✓ Does the name represent **exactly what is known** about that method?

They will likely agree that it's better, but that performance is worse. Assure them that this will get addressed in the next category of making code no worse.

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



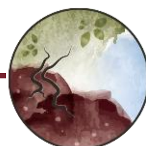
Technical Expertise: No Worse

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

Facilitation Actions

- ◻ 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.
- ◻ Tag each aspect or "-ility" that could have been made worse.
- ◻ 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 "systematically bad" cluster have been resolved.



Technical Expertise: No Process

Tell them that we will now address the **no process** issue, based on the processes they have been using. Show them [Video #5: Committing – Just Enough Process].

Facilitation Actions

- ◻ 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 committing only one provable refactoring)?
  - ✓ 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.

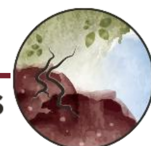
There are parts of the process they will likely feel are necessary, but you should challenge them to reconsider. These are code reviews, checking with designer or Product Owner, or running the full test suite again.

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



Facilitating Mob Cycle #2 – 45 min

Facilitator Notes

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.

Instructions

- ☐ Ask them to identify one thing to extract.
- ☐ Explain that now we want to do this with deep safety: bug-for-bug compatibility.

Technical Expertise: Deep Safety

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

Safely Extracting: Facilitator Notes

Developers will be happy now that deep safety is on the table! They will also become more pedantic, but so will you through these instructions.

> **Habit Shift**
>
> Bug-for-Bug Compatibility

Instructions

Ask them if they will be using the tool approach or the recipe approach. The recipe we provide will work for C++ and Javascript. If the MOB is using C# or Java, they have good tools, so use that approach. These 4 languages cover most teams.

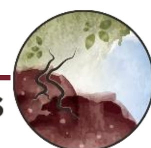
RECIPE APPROACH

If it's the recipe approach, open the linked recipe to be on the right half of the screen, and their code on the left half of their screen.

- Going one step at a time through the recipe, have them do each action below for **each step**.
  - Ensure everybody in the MOB is comfortable with the step.
  - Take it.

FACILITATOR NOTES

While you will not be able to follow the coding practices, follow their actions attentively to catch and rectify any of the following behaviors.



- **Changing code by editing text.**
  Instead, have them copy and paste or use tool capabilities. Editing text creates potential error and the compiler may not catch it.
- **Combining or skipping steps.**
  Instead, ask them how they would prove that their combined transformation is bug-for-bug compatible. Since it's not possible, they will undo the combination step. However, this won't likely happen the first time they use a recipe.
- **Reading, analyzing, or understanding the code to create safety.**
  Instead, remind them that we are creating a **process** for safety that will works without understanding the code. Trust the recipe!

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

- Ask the question, "how will you find bugs in your tool, and remember them?"
- Have them write the list of bugs down as a set of check steps when doing a refactoring.
- Extract to a method using the tool and name just like last time.

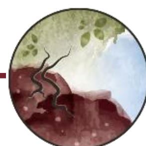
Success Check
Everybody agrees that this extraction is safe and doesn't require running tests.

Instructions

- ☐ Check if ready to commit. Is it:
  a. Safe?
  b. Better?
  c. No Worse?

Success Check
A commit was executed.

- ☐ Ensure the commit is performed.



Facilitating Insight Loop Awareness – 20 minutes

Facilitator Notes

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.

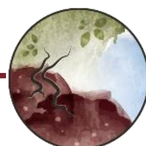
Technical Expertise: Insight Loop

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

Facilitation Actions

- ☐ 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

The Fast Commit Game: Facilitator Notes

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.