The Classroom Mob Programming Game

Paul Gestwicki
pvgestwicki@bsu.edu
Ball State University
Muncie, Indiana, USA

Course Any course involving Test-Driven Development
Programming Language Any
Knowledge Unit Software Development Methods
CS Topics Development Approaches, Testing and Test-Driven Development, Code Reviews

Synopsis

This lab helps students gain experience and proficiency with the agile practice of Mob Programming using Test-Driven Development. Mob Programming involves having an entire team working together to solve one problem at a time. By working closely together, the team shares understanding and enforces quality standards. *The Classroom Mob Programming Game* involves students in all the formal practices of Mob Programming within a single class or lab session.

The game uses tabletop role-playing game design patterns to produce an authentic learning experience for students. Students earn experience points by practicing the different Mob Programming roles, and new abilities are gained as they level up. This provides scaffolding for their learning: students move from simple to more nuanced practices as they gain levels. These abilities represent legitimate Mob Programming techniques, and so the learning objectives and gameplay are intrinsically linked.

Keywords

Mob Programming, Agile, Test-Driven Development, Teamwork, Software Engineering

ACM Reference Format:
Paul Gestwicki. December 2025. The Classroom Mob Programming Game. In *ACM EngageCSEdu.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3786354

ACM EngageCSEdu, December 2025.

ACM ISBN 979-8-4007-2420-6/2025/12
https://doi.org/10.1145/3786354

1 Engagement Highlights

The Classroom Mob Programming Game gets students working together in small teams, learning Mob Programming [7] and practicing Test-Driven Development [3] by authentically engaging in both. The game's short rounds are punctuated by retrospective meetings, and so students get to experience the rhythm of incremental and iterative development in a small scale. That is, students are not just emulating an agile practice but are authentically engaged in one.

A moderator can choose any programming challenge for the game, and it is recommended to choose a simple kata such as FizzBuzz. Likewise, the moderator can choose any programming language and development environment that supports Test-Driven Development. The game involves no submissions nor graded elements: playing the game well is its own motivation and reward.

2 Recommendations

The Classroom Mob Programming Game is designed to be played in a single class session of 50- 75-minutes, the latter being preferred. Completing more than one round of the game helps students see the value of the retrospectives.

Teams should be prohibited from searching the web or generating code, especially if using a classic problem like FizzBuzz. There are innumerable solutions to these problems available, but "solving the problem" is not as important a goal as practicing the process. This can be an important place to make the distinction between rigorous and *ad hoc* software development approaches.

The game assumes students already have some experience programming in a modern IDE using Test-Driven Development, and it has been used extensively in a required sophomore-level class on agile software development, using both Java and Dart. The game works best if a team of four or five students can share one display and easily pass a wireless keyboard and mouse among them, following industrial best practices for Mob Programming [7]. If the environment requires using a single laptop instead, seek to arrange the room so that the people move, not the laptop: moving the computer has a higher chance of breaking the team members' line of sight, inhibiting participation.

3 Background Material

Mob Programming emerged in the mid-2010s as a logical extension of Pair Programming: whereas Pair Programming has developers working in pairs [3], Mob Programming brings the whole team together to solve one problem at a time [7]. Some communities of practice have preferred the terms "Ensemble Programming" or "Software Teaming," but "Mob Programming" remains the most popular term. The limited scholarship on Mob Programming points to its efficacy [6].

This work is inspired by Willem Larsen's *Mob Programming RPG*, which is available under a CC BY-SA-NC license [4]. Larsen adapted portions of Apocalypse World [1] to create a one-session educational game designed to teach Mob Programming. I used Larsen's game for several semesters in a required sophomore-level class and designed the Classroom Mob Programming Game to better align the gameplay with my learning objectives and teaching environment. The most significant difference is in scaffolding of player actions via roles. Larsen's game has a hierarchy of roles where completing one allows a player to add another while retaining all of the XP-earning actions of the previous roles. This contributes to students' cognitive overload, especially considering that Larsen's roles have significantly more XP-earning actions than in this work. My solution was have players gain abilities they level up—which is common in tabletop and digital role-playing games—rather than have them gain new playbooks. This way, as players are starting, they have fewer actions to remember, and practicing simple actions unlocks more advanced ones. The role cards also clearly indicate which actions are new to the player's current level. Some of these are subtle changes in previous actions: for example, a Level 0 Navigator earns XP for describing a failing test, which is the first step of TDD, but a Level 1 Navigator earns XP for holding the team accountable to their current step of the "Red, Green, Refactor" sequence.

4 Gameplay Observations

FizzBuzz is an interesting challenge for students. The problem has been around for decades and yet is still effective as a test of programming ability. Also, because it requires printing numbers, approaching it with TDD requires separating the domain model from the view. That is, a well-formed unit test should be independent of console output, and so teams need to consider how to test the logic in isolation. My experience has been that, even when students have previously engaged with multiple examples of model-view separation, teams will struggle with this important design problem. This exercise can connect to lessons about what it means for tests to be Fast, Independent, Repeatable, Self-validating, and Timely [5].

Following programming sessions with short retrospectives forms an agile cycle. The retrospectives are not only directly helpful for the problem at hand; they also provide an opportunity for students to see agility in action. More fundamentally, it can open a discussion of lifetime learning and what it means to be a reflective practitioner. In such discussions, I often call students' attention to the radical simplicity of the Heart of Agile movement (https://heartofagile.com/).

The game has been designed so that all XP-earning activity represents authentic, legitimate Mob Programming practices. Occasionally, a team of achievers [2] will coordinate to maximize their score, for example, by spamming the Level 3 Mobber ability, "Celebrate a moment of excellence." Such activity is still educationally valuable, especially when appropriately framed within the retrospective.

Moderating the game has been beneficial to my teaching since it allows me to observe how student teams actually work. Even though my context is a third-semester course in a programming sequence, and students have completed several in-class and take-home exercises on TDD, there are inevitably teams who struggle to even get a program to compile. This has allowed me to recognize problems in my curriculum and my own pedagogy. *The Classroom Mob Programming Game* provides insight into players' practices and motivations and thereby provides an excellent framing device for follow-up opportunities.

5 Materials

The archive file contains two entries.

- `ModeratorInstructions.md` provides complete details to the moderator about how to prepare for the game and moderate a session.
- `RoleHandouts.pdf` provides the three playbooks that each player will need to play the game.

6 Auxiliary Materials

(1) https://www.youtube.com/watch?v=nqVUe7ezHhQ: A podcast interview about the game

References

[1] D. Vincent Baker and Meguey Baker. 2016. *Apocalypse World* (2nd ed.). lumpley games.
[2] Richard Bartle. 1996. Hearts, clubs, diamonds, spades: Players who suit MUDs. *Journal of MUD Research* 1, 1 (1996).
[3] Kent Beck and Cynthia Andres. 2004. *Extreme Programming Explained.* Addison-Wesley, Boston.
[4] Willem Larsen. 2016. *Mob Programming RPG.* https://github.com/willemlarsen/mobprogrammingrpg
[5] Robert C. Martin. 2009. *Clean Code.* Pearson Education, Boston.
[6] Makoto Shiraishi, Hironori Washizaki, Yoshiaki Fukazawa, and Joseph Yoder. 2019. Mob Programming: A Systematic Literature Review. In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 2. 616–621.
[7] Woody Zuill and Kevin Meadows. 2022. *Software Teaming* (second ed.). Independently published.