

Give me ideas and steps on how to create a "scheme of learning/work" for someone who is learning to be a technical coach (e.g. samman practitioner) for a team of C#.NET software developers who have different levels of experience. Base the sessions around bi-weekly "learning hours" interspersed with ensemble coding sessions that reinforce the learning hours. The scheme of work should be constructed as a table with the following columns

- Cycle – a number that represents two activities (a learning hour and ensemble session)
- Activity – either a "Samman Learning Hour" or "Ensemble Session"
- Focus/Topic – TDD aspect being studied
- Connect Activity – A short activity to attract attention and pique interest (5 - 10 mins)
- Concepts – coach led session usually a demonstration (10 – 20 mins)
- Concrete Practice – An on-hands activity to apply the concepts (often a code Kata) (~ 20 mins)
- Conclusion – (~5 mins)

The Connect, Concepts, Concrete and Conclusion topics are based on Sharon Bowman's 4 C's training model.

The "Ensemble Sessions" don't need the Connect, Concepts, Concrete and Conclusion columns, just a moderately detailed account of what the session will involve.

Here's a "**Scheme of Learning/Work**" designed for someone coaching a C# .NET development team in **Test-Driven Development (TDD)** using the **Samman approach**, structured with **bi-weekly learning hours** and **ensemble sessions**. This plan assumes varied levels of experience and aims to build up from fundamentals to advanced practices while reinforcing concepts through collaborative coding.

Scheme of Learning/Work – Technical Coach Plan (12-Week Cycle)

Cycle	Activity	Focus/Topic	Connect Activity	Concepts	Concrete Practice	Conclusion
1	Samman Learning Hour	What is TDD & The Red-Green-Refactor Cycle	Discuss "What's one habit you've recently adopted?"	Live demo: Solve a small problem using TDD	String Calculator Kata (first few requirements)	Reflect on "What surprised you?"
2	Ensemble Session	Practice Red-Green-Refactor basics	–	–	Team applies TDD to a toy problem (e.g. FizzBuzz)	–
3	Samman Learning Hour	Writing the First Test & Baby Steps	"Guess the Output" game from a badly written test	Demo: Driving design through the first test	Bowling Game Kata – frame-by-frame TDD	Share takeaways on clarity of intent
4	Ensemble Session	Incremental TDD applied to a feature	–	–	Apply TDD to a real story/feature in the backlog	–

5	Samman Learning Hour	Refactoring Techniques & Naming	Show before/after code and ask "Which would you prefer to maintain?"	Demo: Rename, extract method, introduce parameter	Refactor messy code (Gilded Rose / Legacy code kata)	What was your favorite refactor?
6	Ensemble Session	TDD with Refactoring Focus	–	–	Work on code with deliberate naming and refactoring	–
7	Samman Learning Hour	Outside-In TDD / London School	Puzzle: "Can you guess what I'm mocking?"	Demo: Testing behavior with mocks	TDD with mocking (e.g. use FakeClock in an alarm clock)	Review benefits of mock-based design
8	Ensemble Session	Practice Outside-In with Service Layer	–	–	Apply to a mockable business logic scenario	–
9	Samman Learning Hour	Working with Legacy Code	Share horror stories of legacy code	Show: Characterization tests and seams	Legacy Code Kata – wrap code in tests	What was the hardest part to wrap?
10	Ensemble Session	Characterization Tests in Practice	–	–	Ensemble work: Add tests to poorly covered production code	–

11	Samman Learning Hour	Design Pressure & Test Smells	"Name that smell" – Identify smells in tests	Highlight code/test smells via real examples	Test Smell cleanup exercise	"What smell will you watch for?"
12	Ensemble Session	Choose & Apply Any Technique	–	–	Group picks a past challenge and applies techniques	–

Notes for Implementation:

- **Tools:** Use shared IDEs, screen sharing, or mob tools like [Mob](#) for ensemble sessions.
- **Leveling:** Encourage pairing/mobbing with mixed-experience levels to foster mentoring.
- **Rotation:** Rotate the "driver" and "navigator" roles during ensemble sessions.
- **Retrospectives:** Consider mini retros every 3 cycles for feedback and calibration.

This is good but it's safe to assume everyone knows how to create a basic unit test. Could you regenerate the workflow basing it more on the following:

- TDD workflow: Red → Green → Refactor cycle
- Turning Requirements into tests
- Writing unit and integration tests with xUnit
- Handling dependencies
- Test doubles: stub and mocks
- Test doubles and verification testing
- Untestable code
- Mutation testing
- Safe refactoring practices guided by tests

It doesn't matter how many weeks it covers but could you mark each of the learning hour/ensemble pairs as a single cycle.

Absolutely! Below is a **revised scheme of learning/work** tailored for a coach guiding C# .NET developers in applying **TDD** more deeply. This version assumes the developers already understand how to write basic unit tests and focuses instead on practical and nuanced aspects of TDD, xUnit, test doubles, handling dependencies, and writing testable code.

Each **cycle** contains:

- A **Samman Learning Hour** (structured using Sharon Bowman's 4Cs model).
- A follow-up **Ensemble Session** where the concepts are applied collaboratively.

Technical Coaching Scheme of Work: Intermediate–Advanced TDD in C#/.NET

Cycle	Activity	Focus/Topic	Connect Activity	Concepts	Concrete Practice	Conclusion
-------	----------	-------------	------------------	----------	-------------------	------------

1	Samman Learning Hour	TDD Workflow: Red → Green → Refactor	"What do you do when you don't know how to start?"	Live code: small problem using disciplined R-G-R	Roman Numerals Kata (TDD)	Share: "Where did you hesitate most?"
	Ensemble Session	Practice the R-G-R Loop in real time	–	–	Apply R-G-R to a toy problem like Leap Year	–
2	Samman Learning Hour	Turning Requirements into Tests	Group brainstorm: Break a vague requirement into testable pieces	Demo: Deconstructing a user story into tests	Write tests for a simulated feature spec (e.g. account rules)	What made it easy/difficult?
	Ensemble Session	Apply "requirements → tests" to a real story	–	–	Choose a story, identify test cases before coding	–
3	Samman Learning Hour	Unit & Integration Testing with xUnit	"What's the riskiest kind of bug you've seen recently?"	Explain & show: Difference between unit and integration tests using xUnit	Drive a feature from unit test → integration test using xUnit	Share: "Did anything surprise you about integration coverage?"
	Ensemble Session	Apply layered testing with xUnit	–	–	Ensemble adds both unit and	–

					integration tests for a small feature	
4	Samman Learning Hour	Handling Dependencies	Code puzzle: What's wrong with this constructor?	Show: Constructor injection, parameter object, interface isolation	Refactor to inject dependencies into testable code	"What makes code easier to wire up?"
	Ensemble Session	Refactor dependency-laden code	–	–	Refactor production code to be more test-friendly	–
5	Samman Learning Hour	Test Doubles: Stubs and Mocks	"What's the difference between pretending and spying?"	Show: Stub vs Mock in xUnit using Moq/FakeItEasy	Drive behavior using test doubles for a service interaction	Which double felt hardest to work with?
	Ensemble Session	Apply test doubles to real service code	–	–	Add tests to code that integrates with external service	–
6	Samman Learning Hour	Test Doubles & Verification Testing	Game: "Did the spy see this call?" (mock verification guessing game)	Show: When/why to use verification with mocks	Add verification tests to validate correct behavior under rules	"Is this test really testing <i>behavior</i> ?"

	Ensemble Session	Apply verification to business logic	–	–	Ensemble uses mocks to verify critical side effects	–
7	Samman Learning Hour	Dealing with Untestable Code	“What code would you <i>never</i> want to test?”	Show: Extract seams, wrap static calls, subclass for testability	Refactor legacy-style class to become testable	“What made the biggest difference?”
	Ensemble Session	Tame real or legacy code	–	–	Ensemble refactors toward testability before adding tests	–
8	Samman Learning Hour	Mutation Testing	“What’s a test that passed but shouldn’t have?”	Explain & demo: Mutation testing with Stryker.NET	Run Stryker on small test suite and improve tests	“What mutation caught you off guard?”
	Ensemble Session	Apply mutation testing to a real feature	–	–	Run mutation tests on production logic, fix weak tests	–
9	Samman Learning Hour	Safe Refactoring with Tests	“When do you feel safe to refactor?”	Demo: Refactor with full test suite & mutation testing safety net	Practice refactoring steps: Rename, Extract, Introduce Parameter	“What did tests let you do with confidence?”
	Ensemble Session	Refactor real feature with tests as safety	–	–	Carefully refactor production code,	–

					relying on safety nets	
--	--	--	--	--	------------------------	--

In your previous suggestion you referred to a number of coding Katas but you've barely mentioned any this time. Please correct this.

Here's a list of other things I'd like you to look at:

Cycle 1 ensemble: the Leap Year challenge will be too trivial can you suggest something with a bit more of a challenge

Cycle 2 ensemble: Can you suggest a moderately challenging story to identify test cases from.

Cycle 3 ensemble: can you suggest a moderately challenging feature to use for the exercise

Cycle 4 ensemble: Can you suggest something other than production code

Cycle 5 ensemble: Can you suggest a moderately challenging scenario

Cycle 6 ensemble: Can you suggest a moderately challenging scenario

Cycle 7 ensemble: Can you suggest something other than real or legacy code. A kata perhaps?

Cycle 8 ensemble: Can you suggest something other than real code. A kata perhaps?

Excellent feedback — thank you. You're right that Katas play a crucial role in effective TDD coaching, especially when aiming for **focused, repeatable, and safe practice** that encourages deep thinking and learning without production pressure.

Below is the **revised and improved Scheme of Work**, incorporating:

- A more structured use of appropriate code katas.

- **More meaningful ensemble session scenarios**, particularly where earlier ones were too trivial.
- Cycle-specific recommendations based on your feedback.

Technical Coaching Scheme of Work – Intermediate/Advanced TDD with Katas and Scenarios

Cycle	Activity	Focus/Topic	Connect Activity	Concepts	Concrete Practice	Conclusion
1	Samman Learning Hour	TDD Workflow: Red → Green → Refactor	"What do you do when you don't know how to start?"	Live demo of R-G-R using a small kata	Roman Numerals Kata	"Where did you hesitate most?"
	Ensemble Session	Practice full TDD loop	–	–	Yatzy Kata — moderately complex scoring logic	–
2	Samman Learning Hour	Turning Requirements into Tests	"What makes a requirement testable?"	Demo: Break down a user story into testable cases	Tennis Game Kata — interpret scoring rules as requirements	"What made it easy/difficult?"
	Ensemble Session	Define test cases from requirements	–	–	Parking Lot Pricing scenario: Determine fee based on entry/exit times, days, discounts	–
3	Samman Learning Hour	Unit vs. Integration Testing with xUnit	"What's the riskiest kind of bug you've seen recently?"	Compare test scopes in xUnit	Supermarket Receipt Kata — blend of unit and integration testing (discount engines, item totals)	"What surprised you?"
	Ensemble Session	Layered testing of business rules	–	–	Shopping Cart feature: Integrate pricing rules and test receipt outputs	–

4	Samman Learning Hour	Handling Dependencies	Code quiz: "What's wrong with this constructor?"	Show: Injecting dependencies cleanly in C#	Dependency Breaker Kata (variation of Gilded Rose)	"What made testing easier?"
	Ensemble Session	Testable design without production pressure	–	–	Invoice Generation Kata (designed for teaching DI and interface design)	–
5	Samman Learning Hour	Test Doubles: Stubs and Mocks	"What's the difference between pretending and spying?"	Show: Stub vs Mock in xUnit using Moq	Cash Register Kata — simulate discount providers, receipt printers	"Which double was hardest to write?"
	Ensemble Session	Apply test doubles to scenario	–	–	Subscription Billing System: Add test doubles for clock, payment provider, and notifier	–
6	Samman Learning Hour	Verification Testing with Mocks	"Can you guess what the spy saw?" (fun quiz)	When to verify calls using mock frameworks	Mockist Ping-Pong Kata — design driven by interaction, not state	"What was most confusing?"
	Ensemble Session	Verify behavior of dependent services	–	–	Notification Service: Verify alerts are sent correctly via email/SMS APIs under different triggers	–
7	Samman Learning Hour	Dealing with Untestable Code	"What code would you <i>never</i> want to test?"	Demo: Extract seams, subclass for seams, wrapper classes	Gilded Rose Kata — introduce seams and characterization tests	"What made the biggest difference?"
	Ensemble Session	Refactor toward testability	–	–	Log Analyser Kata (simulate tight coupling; refactor for testability)	–

8	Samman Learning Hour	Mutation Testing	"What's a test that passed but shouldn't have?"	Demo: Stryker.NET applied to a naive test suite	Apply mutation testing to Bank Account Kata	"What mutation caught you off guard?"
	Ensemble Session	Improve weak test suite using mutation testing	–	–	Mars Rover Kata with intentional weak tests — mutate and improve	–
9	Samman Learning Hour	Safe Refactoring with Tests	"When do you feel safe to refactor?"	Demo: Rename, Extract Method, Parameter Object, etc	Use tests + mutation testing to refactor Yahtzee Kata codebase	"What did the tests let you change safely?"
	Ensemble Session	Refactor safely with tests as safety net	–	–	Legacy Feature Extract Kata: Isolate and refactor a complex rule hidden in code	–

✚ Kata & Scenario Descriptions (New or Specific Recommendations)

- **Yatzy Kata:** Scoring rules with multiple edge cases — good for introducing complexity in TDD.
- **Parking Lot Pricing Scenario:** Realistic pricing based on rules with edge cases (overnight, weekend, etc.).
- **Supermarket Receipt Kata:** Combines product data, discounts, totals — mix of unit and integration test examples.
- **Invoice Generation Kata:** Used to practice constructor injection and dependencies without coupling to actual systems.
- **Cash Register Kata:** Use stubs for pricing engine and mocks for printer/email receipt system.
- **Subscription Billing System:** Service interactions with time-sensitive logic (use mocks for time and email).
- **Notification Service:** Triggers behavioral responses based on input events (mock/verifiable interactions).
- **Log Analyser Kata:** A good candidate for discussing seams, decoupling, and wrapping static/global calls.

- **Mars Rover Kata:** Control logic with spatial movement — ideal for mutation testing of branching logic.