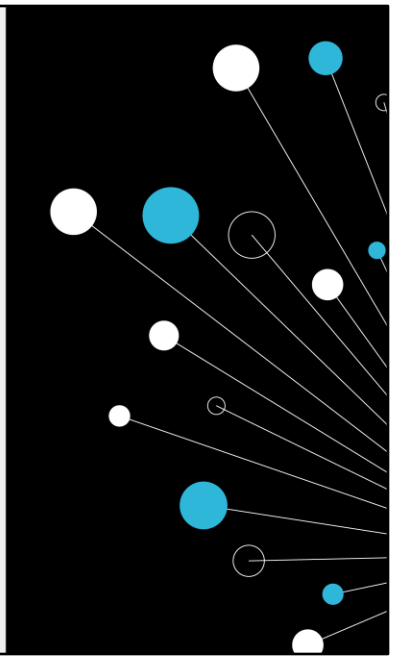
SAMMAN TRAINING

Day 2 – The Creation of content

Programme Learning objectives

- Understand the value of the Samman training hour and how to apply it
- Understand how to apply the 4C training model – Connect, Concepts, Concrete Practice and Conclusions
- Understand how mentoring / coaching can help engineers to learn coding skills
- How to set individual and team technical goals
- Develop facilitation skills to run technical workshops on TDD, Refactoring in ensemble and paired set ups
- Be able to contextualise technical content to the team needs
- Learn how to embed continuous learning and experimentation in engineering work

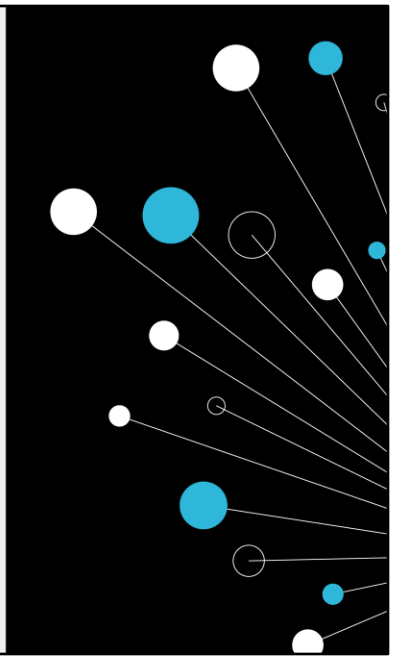
QA



Learning objectives

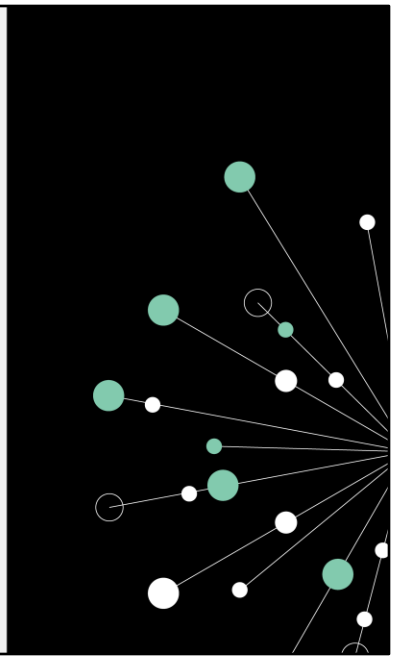
At the end of this session, you will be able to:

- Create Schemes of Work (SOW) for tuition first series
- Create Learning Plan for one SOQ session
- Plan and deliver learning hours to engineers.
- Unpack and predict the challenges engineers will face.
- Adapt challenge along the process in response to engineer needs.
- Apply the 4Cs to the process to ensure maximum progress and engagement.



Phase 2 - Course Outline

- Reflection over the Past Weeks
- Process for Building a Curriculum
- Creating Bespoke Learning Hours

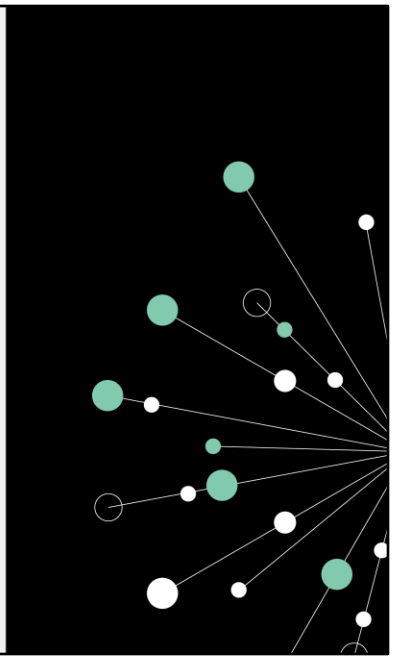


QA

Later...

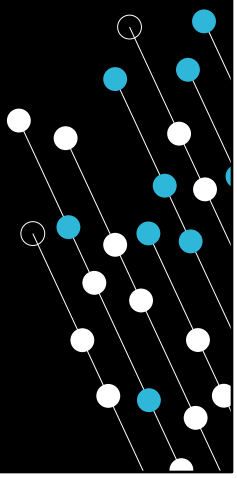
Later, we will be creating schemes of work for your first set of six training cycles. Consequently, you will be asked for:

- A business priority for your team
- A final goal your team will need to achieve in order to meet the business priority



QA

Reflection on the past weeks



QA

How did it go?

Lessons learned from

- Learning Hour
- Ensemble

Stories

Anecdotes

Results

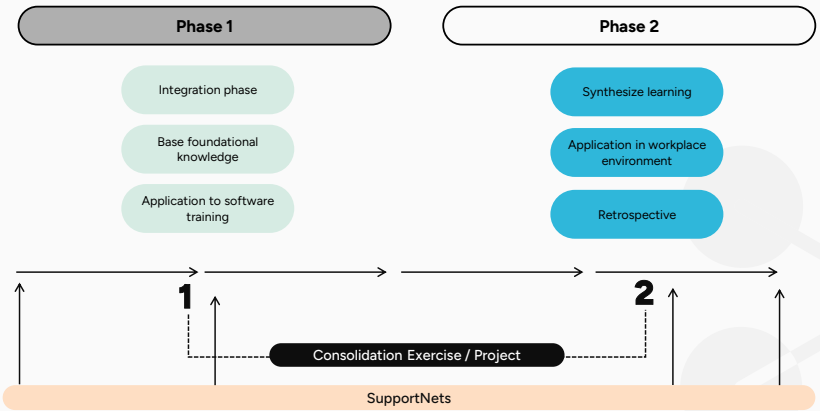
Things that went well

Feelings about the method

Things that didn't go so well

Response of the team

Training Overview



QA

License and Disclaimer

License: CC-BY-SA-4.0. Attribution: sammancoaching.org

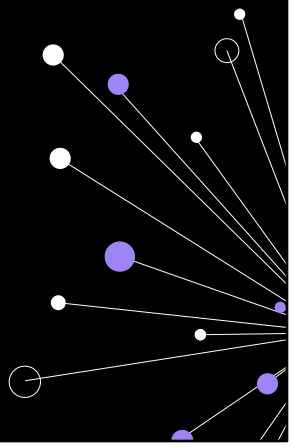
Creative Commons Attribution Share Alike 4.0 International

This material is licensed with CC-BY-SA-4.0.

When you use materials on this site, you must include this licensing information and the details of the attribution. For example most learning hours should be attributed with the author, their affiliation, and a link to sammancoaching.org.

“Samman Coaching” is a trademark owned by the Samman Technical Coaching Society, so you may not use these words in any way that makes it sound like you represent the society or that your training is officially sanctioned by us.

Process for building a curriculum



Start with a plan – Scheme of Work

A SoW (sometimes SoL) is a mid-term plan of how you intend to upskill a group of people.

What are you going to teach, in what order?

What techniques are you going to use to demonstrate and model 'the right way'

What skills will the business need your coding teams to have in 6 months' time?

Are there changes to languages?

What gaps need to be filled?

SoL - Scheme of Learning
See "What is a Scheme of Work.pdf" for more information

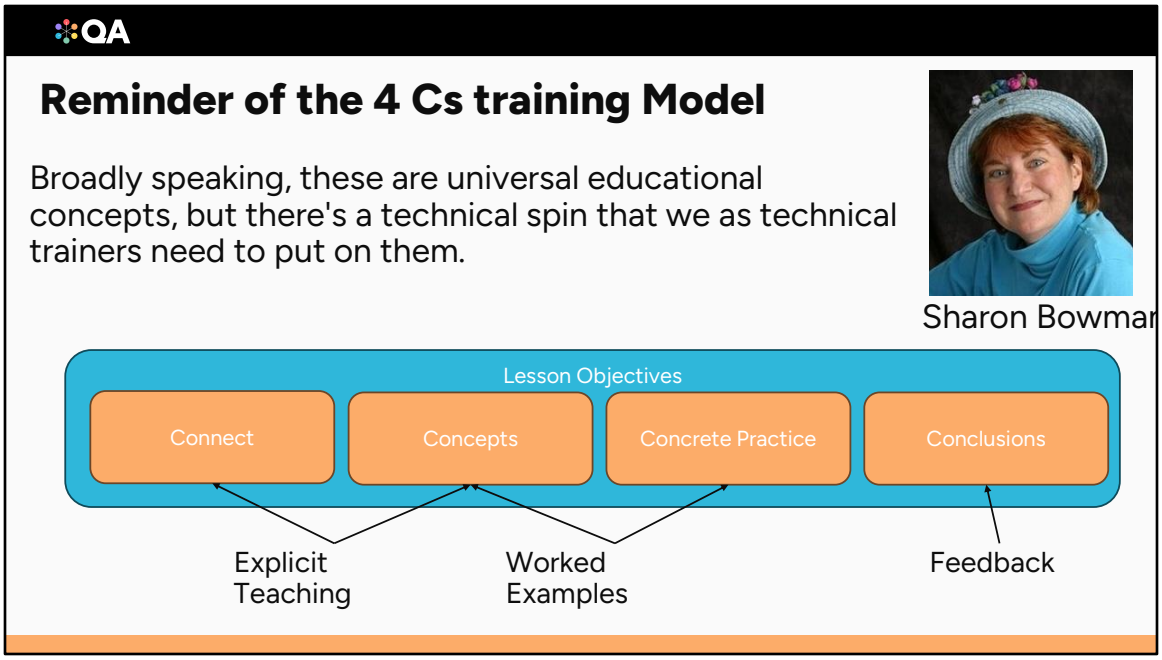
QA

Scheme of Work

React Reskilling Curriculum

Cycle	Activity	Focus	Connect Activity	Concepts	Concrete Practice	Conclusion
1	Samman Hour	Pure components	Links to other languages, what is a pure function? Examples of where they have been used before.	Demonstration on how to create a pure component with Props in React, including return values and callbacks	Delegates create a component which accepts user profile props. It must deconstruct them, then pass it to a child component which renders the JSX for a list of employees	Does a component have to be pure? What is considered good industry practice?
	Ensemble Hour	Provide delegates with the Estate Agent kata. This requires them to build a website where they can register (JSON is fine) to list properties on an estate agents website.				
2	Samman Hour	useEffect() to call in external data	Where does our data live? How do we access it	Demonstrate the creation of a native React API using fetch, console the result and link back to previous learning about deconstruction	Delegates connect to external sources and understand how to use the information that comes in. How do they pass it on to other components? What can be ignored?	What are the potential issues with hosting data externally?
	Ensemble Hour	Continue with the next phase of the Estate agent challenge. How can we read the data and now edit it to include more houses. How do people view the houses that are available on the website.				
3						

Conclusion Is a plenary session (short summary) that ensures attendees understand what they should have got out of the session



Samman and the 4Cs Training Model:
4C Training model
https://sammancoaching.org/activities/4C_model.html

Sharon L Bowman's 4Cs
The4CsMap2016
https://bowperson.com/images/resources/quick-guide-to-four-c-map.pdf

https://bowperson.com/images/resources/the-four-c-map.pdf

Note: Explicit Teaching, Worked Examples and Feedback are 3 of the high impact techniques specified by the EEF

QA

Connect – 5 or 10 minutes at the start

Connect

In a Connect activity, learners make connections with

- What they already know (or think they know) about the training topic
- What they will learn
- What they *want* to learn
- Each other

It's about getting into a state of mind where people are ready to begin learning something new. New knowledge needs to be integrated with existing knowledge, and perhaps existing knowledge needs to be challenged. That last connect - to one another - is particularly important in a software team. Learning is social, that is, people learn from each another as well as from the trainer/coach. Ideally you want a team environment where people feel safe to ask questions and aren't afraid to reveal what they don't know or can't do yet.

QA

Concepts – 10 to 20 Minutes

Concepts

This is as close to traditional teaching as we come. In Software Development, this is generally done through some kind of demonstration from the coach

It's important to remember that people learn through discussing, reading, writing and explaining new concepts to others. Teaching something to another person, and doing the preparation needed for that, is one of the best ways to learn a new concept for yourself. The more interactive and engaging and varied ways you can present a new concept, the better.

QA

Concrete Practice – 20 minutes

Concrete Practice

Usually in a Learning Hour, this will be a coding exercise, although not exclusively.

Often it will be a code kata with a clear desired outcome, like a refactoring kata or a test design kata.

If you're coding from scratch doing TDD you might:

- Introduce rules and restrictions, starting code and/or checklists.
  - This is to make it less challenging and open than full-on TDD.
- Narrow the focus of the session to a particular aspect you want to learn about.
  - You only have an hour.

QA

Conclusions – 5 minutes at the end

Conclusions

In order to begin using the new knowledge in the rest of their work, learners benefit from a "conclusions" activity. You may ask them to:

- **Summarize** what they have learned
- **Evaluate** it
- **Celebrate** it
- Create **action plans** for how they will use the new knowledge or skills afterwards

The conclusions activity is (usually) held at the end of the learning hour, and it's important to save time for it.

QA

Scheme of Work

There is no requirement to create any of the resources at this stage. It is purely about a broad stroke understanding of where you need to be at the end of the learning cycle.

Then, after a review and some consideration of how well that block of learning worked, you adapt and send it out again.

Having run a session you may well review the SoW and update the content and structure of later sessions.

QA

The Process

Start with the ultimate goal and reverse engineer the requirements

Every business has a 3-year plan! Can you link your long-term plans to the overall business goals.

Even with a plan, you need to be flexible with it. What happens if a new UI standard pops up? How do you keep up?

As you get better you can project longer term change.

QA

Challenge 1 – Make Plans for a Scheme of Work

Using your knowledge of:

- The current market
- Upcoming changes to the core languages (don't forget dependencies and libraries)
- Changes in the business
- Make up of your team
- Particular challenges

Gather information and plan to create a SoW which:

- Spans 6 Samman Hours and Ensemble coding sessions
- Addresses the challenges and prepares your team to meet them

Challenge 1

Document:

- What business priorities you are addressing
- Resources needed
- Justification (Sorry, but you will be asked) for why this is the best internal training for your team
- How you will know that the team have understood and made progress

QA

Challenge 1

Some helpful steps:

- Decide the final goal
- Split the learning into the major dependencies
- Decide what order the learning needs to go in
- Are there any challenges or major gaps in knowledge before you start? How do you fix/fill these, so everyone is at the same starting point before the first hour?

Mind map these decisions above for 10 - 15 minutes

Mind Map - Review

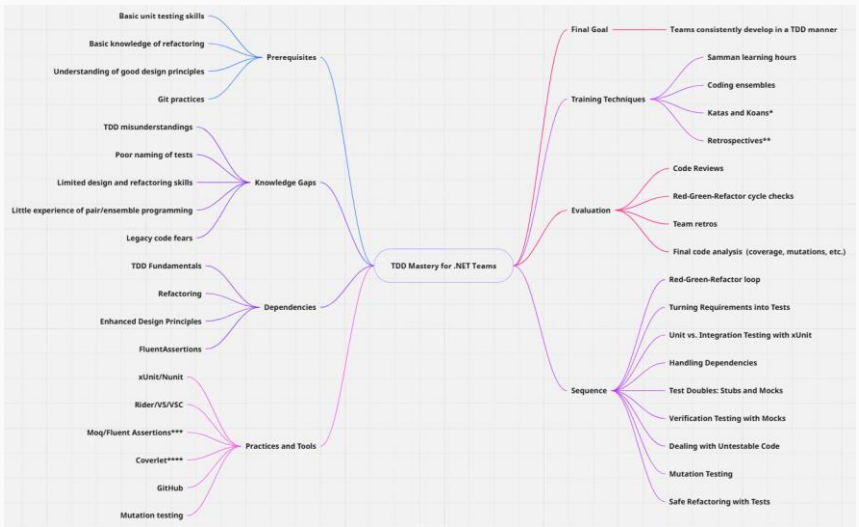
- Mind Maps and SoW's will be team centric, but can be useful to share with others for:
  - Knowledge Transfer Across Domains
  - Alignment and Standardization
  - Cross-Pollination of Ideas
  - Feedback and Improvement
  - Scalability and Future Reuse
  - Supports Mentorship and Coaching

Even if the scheme of work is specific (e.g. backend Java development), the **structure, pedagogical approach, sequencing of concepts, and delivery methods** may be broadly applicable to other domains.

- Can help to **maintain consistency** across training programs because sharing helps identify commonalities and gaps between how different teams are trained. It can also encourage alignment in learning goals, tooling, and terminology across domains.
- A detailed scheme of work can inspire **innovations in curriculum design** in other areas. Coaches may adopt or modify specific exercises, assessments, or learning models (e.g. pair programming, project-based learning). Encourages **interdisciplinary thinking**, especially in areas like DevOps or SRE where multiple skills intersect.
- Coaches bring diverse perspectives and may offer critical feedback. This can help improve the scheme itself. Peer review can reveal blind spots or assumptions that a team-specific plan may have overlooked.
- Ideas can be repurposed for future teams with minor adjustments and reduce effort duplication.
- Technical coaches mentoring new teams benefit from seeing how others guide skill development.

QA

Example Mind Map – TDD Mastery for .NET Teams



***Katas** are short exercises like the "Bowling Game" kata or "String Calculator" kata, where developers practice writing code to solve a specific problem or implement a particular algorithm.
Koans are exercises that present a series of failing tests, and the student's task is to make them pass by understanding the underlying language features.

A **retrospective (or retro) is a structured meeting where a team reflects on a completed period of work (like a sprint in Agile development) to identify what went well, what could be improved, and to create action items for future iterations.

****FluentAssertions** is a popular assertion framework for .NET that enhances the readability and expressiveness of unit tests. It provides a set of extension methods that allow you to write assertions in a natural, English-like style, making your tests easier to understand and maintain.

*****Coverlet** is a cross-platform code coverage framework for .NET, with support for line, branch and method coverage.

Challenge 2 – Create a Scheme of Work

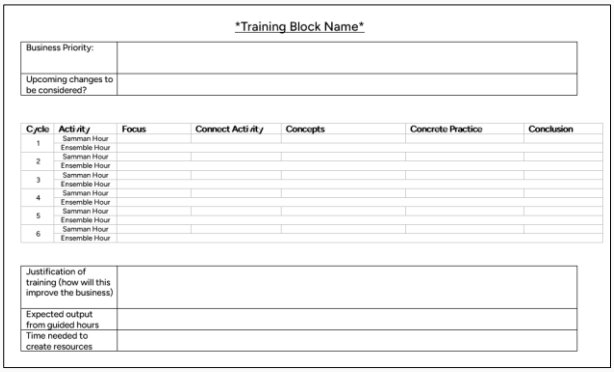
Now you have some ideas in place:

Create a SoW which spans over 6 Samman Hours and Ensemble coding sessions, which addresses the challenges and prepares your team to meet them.

QA

Use the SOW template (up to 1 hr 30)

- Take your mind map and put the ideas into the table



Training Block Name

Business Priority:	
Upcoming changes to be considered?	

Cycle	Activity	Focus	Connect Activity	Concepts	Concrete Practice	Conclusion
1	Samman Hour					
	Ensemble Hour					
2	Samman Hour					
	Ensemble Hour					
3	Samman Hour					
	Ensemble Hour					
4	Samman Hour					
	Ensemble Hour					
5	Samman Hour					
	Ensemble Hour					
6	Samman Hour					
	Ensemble Hour					

Justification of training (how will this improve the business)	
Expected output from guided hours	
Time needed to create resources	

QA

Review

Let's have some of you talk us through the SoW you have built and get some feedback on how it might work.

You would all have faced similar problems with this task and it’s good to see how others have tackled the issues

QA

SOW for TDD Mastery for .NET Teams - 1

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?" "Was it caught by a unit test? Integration test? Neither?"	Compare test scopes in xUnit	**Supermarket Receipt Kata** — blend of unit and integration testing (discount engines, item totals)	"Where did you struggle drawing the line between unit and integration?"
	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	–

QA

SOW for TDD Mastery for .NET Teams - 1

Cycle	Activity	Focus/Topic	Connect Activity	Concepts	Concrete Practice	Conclusion
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 behaviour 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 *never* 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	–

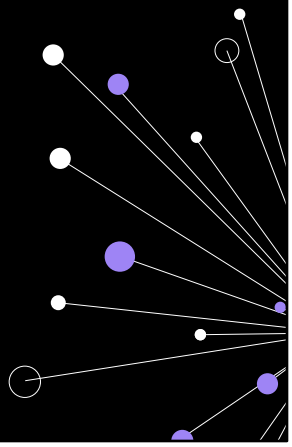
SAMMAN TRAINING

Day 3 – The Creation of content

Recap on yesterday's challenges

We looked at the process of planning an extended period of learning, splitting it up to a logical process

Creating Bespoke Learning Hours



QA

Now you know what you want your Devs to learn

You have a solid basis of where you need your dev to be and how to get them there.

The next steps are to create the individual Learning hours and ensemble hours content

We will investigate this process together and you will have a go at cycle one

QA

Lesson Plans

A huge caveat to this process before we start!

Having a plan of what you want to do for the hour is essential

BUT

Don’t be afraid to change it dynamically:

- If it all goes wrong
- A tangent seems interesting and relevant

QA

The EEF (Education Endowment Foundation)

Identify these ~~10~~9 techniques as having high impact in the classroom

Setting Goals	Structuring Learning	Explicit Teaching
Worked Examples	Collaborative Learning	Multiple Exposures
Questioning	Feedback	MetaCognition
	~~Differentiation~~	

Lesson Plans

At the very least you need to plan and record:

- Learning Outcomes – Why are we doing this? Share them with the attendees
- Timings – what, when, how long
- Demonstrations – how will you 'teach' the delegates? This should be no more than 20 minutes
- Activities – how will the delegates implement or apply what you have just shown them? This is where you can probe understanding with questions
- Conclusions – Link their progress back to the outcomes.

QA

Remember the 4Cs

- Connect
- Concepts
- Concrete Practice
- Conclusions

Good Example from Samman?

Design for Approval Testing

Extra challenge with dates, process ids, things in random order.

- 2 min connect: is testability an architectural concern?
- 10 min concrete: analyse the problem
- 10 min concrete: Demo pre-comparison processing
- 20 min concrete: fix the problem your own way
- 10 min concept: Pre-comparison processing
- 5 min conclusions: Strategies for testability

Connect - is testability an architectural concern?

Is it acceptable to re-design your production code to make it more testable? What kinds of things might you want to change in your architecture? Have you changed anything in your architecture purely for the sake of testing?

Hopefully they will mention controlling the current time.

Concrete - analyse why the tests are failing and how to fix them

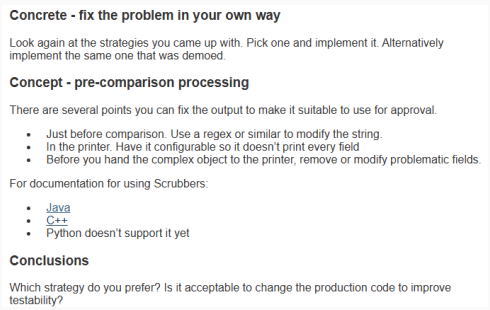
Use the java, C++ or python version of Supermarket Receipt starting on the 'with_date' branch. All the tests are currently failing.

- Why are the tests failing?
- Come up with two or three strategies for how to make them pass again.

Don't implement the strategy straight away. Discuss in a group the best solution.

Demo - pre-comparison processing

Demo a solution that fixes the problem by setting the date in receipt to be a fixed, known date.



Concrete - fix the problem in your own way

Look again at the strategies you came up with. Pick one and implement it. Alternatively implement the same one that was demoed.

Concept - pre-comparison processing

There are several points you can fix the output to make it suitable to use for approval.

- Just before comparison. Use a regex or similar to modify the string.
- In the printer. Have it configurable so it doesn't print every field
- Before you hand the complex object to the printer, remove or modify problematic fields.

For documentation for using Scrubbers:

- Java
- C++
- Python doesn't support it yet

Conclusions

Which strategy do you prefer? Is it acceptable to change the production code to improve testability?

What makes this a good example of a Learning Hour?

Link

Design for Approval Testing
https://sammancoaching.org/learning_hours/legacy/handle_dates.html

QA

Good Example from Samman? - Justifications

- Strong Alignment with Learning Principles
  - Connect–Concrete–Concept–Conclusion (4C) structure
  - Variety of learning modes
  - Active learning
- Relevance to Real-World Testability Challenges
- Drives Critical Thinking About Architecture
- Practical, Tool-Specific Instruction with Language Support
- Clear Learning Objectives & Measurable Outcomes
- Promotes Team Dynamics & Ensemble Learning

See "What makes the Design for Approval Testing Samman Learning Hour a good exemplar of a lesson plan.pdf" for further breakdown of justifications.

QA

Another Good Example of a Learning Hour?

Learning Hour			
Hour Title	Unit vs. Integration Testing	**Position**: 3	
Pre / prior learning	Previous Session	**Created** by:	
Timings	**phase**	**Activity**	**Link to Resource**
0 - 20	Connect	Ask: "What's the riskiest kind of bug you've seen recently?"	
10 - 25	Concepts	Look at xUnit docs and discuss	https://xunit.net/docs/getting-started
25 - 55	Concrete Practice	Tackle the "Supermarket Kata"	Kata01: Supermarket Pricing - CodeKata
55 - 60	Conclusion	Ask "What surprised you?"	

QA

No and Here's Why

- **What Makes This a Poor Learning Hour**
- **Timing:** 20-minute connect phase is too long and off-topic, leaving little time for meaningful content. Other timings don't add up (easy mistake to make when adapting an existing learning hour for a new purpose).
- **Focus:** The main topic (unit vs integration) is barely explored — no real explanation, no visual aid.
- **Structure:** No scaffolded practice or test-usage. No mention of a hands-on demo or walkthrough. Katas often have multiple uses so, activity column needs to clearly specify how the author wants the Kata to be used so the instructions can be passed on to the learners. You/others may not remember/know your original intentions when you/they come to deliver the hour.
- **Engagement:** No energy or direction; the facilitator fails to engage or model expectations.
- **Resources:** The xUnit link is just dropped in with no context. The choice of Kata link is strange because it involves no coding! There are many variations of the same Kata, make sure you point at the correct one.
- **Reflection:** The closing prompt and lacks relevance/takeaways.

Learning Hour Template

Learning hour

Hour Title		Position	
Pre / prior learning		Created by	

Timings	Phase	Activity	Link to Resource

QA

Ensemble Hour Template

Associated Ensemble Hour	
Scenario	
Feedback / Adjustments for next delivery	

QA

Your challenge: 60 – 90 mins

Take one of the lessons from your SoW yesterday and create the resources that would allow you to deliver the session and to run the Ensemble hour follow up.

You need to:

- Update the Lesson plan step by step
- Include the timings and the activities you will carry out
- Write the code used for demo and write instructions on how to run that demo
- Write instructions for the Activity (this can be the same code your demo started with)
- Put links to the resources (GitHub etc)
- Write a scenario that supports the learning done in this session for the ensemble hour

QA

Peer Review of Plans

Let's spend some time looking at the plans of a lucky few volunteers.

What do we think about (as a group) about:

- The timings
- The activities
- The quality of the codebase
- How effective it will be as a lesson in achieving the SoL outcomes

QA

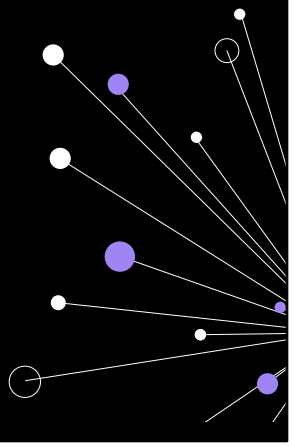
Use of AI to Create SoW's, Learning Hours and Ensemble Sessions

- If the requests are carefully drafted AI can be a helpful tool
- The materials it creates *initially* look great and convincing
- However, trust **NOTHING** it produces. Watch out for:
  - Hyper links that don't work
  - Hyper links that take you to places that have nothing to do with the topic
  - Entries that completely differ from the suggestions given in the SoW
  - Entries that miss the point
  - Activities that lack depth
- Using it can feel like playing a game of Whack-A-Mole
- Refinement is key

See: 01 First Draft AI LearningHour - TDD Mastery for .NET Teams - TDD Workflow (Red Green Refactor).pdf for things to watch out for.

See: 01 LearningHour - TDD Mastery for .NET Teams - CHATGPT Discussion.docx for requests that got to the first draft

Creating Bespoke Learning Hours Further Considerations

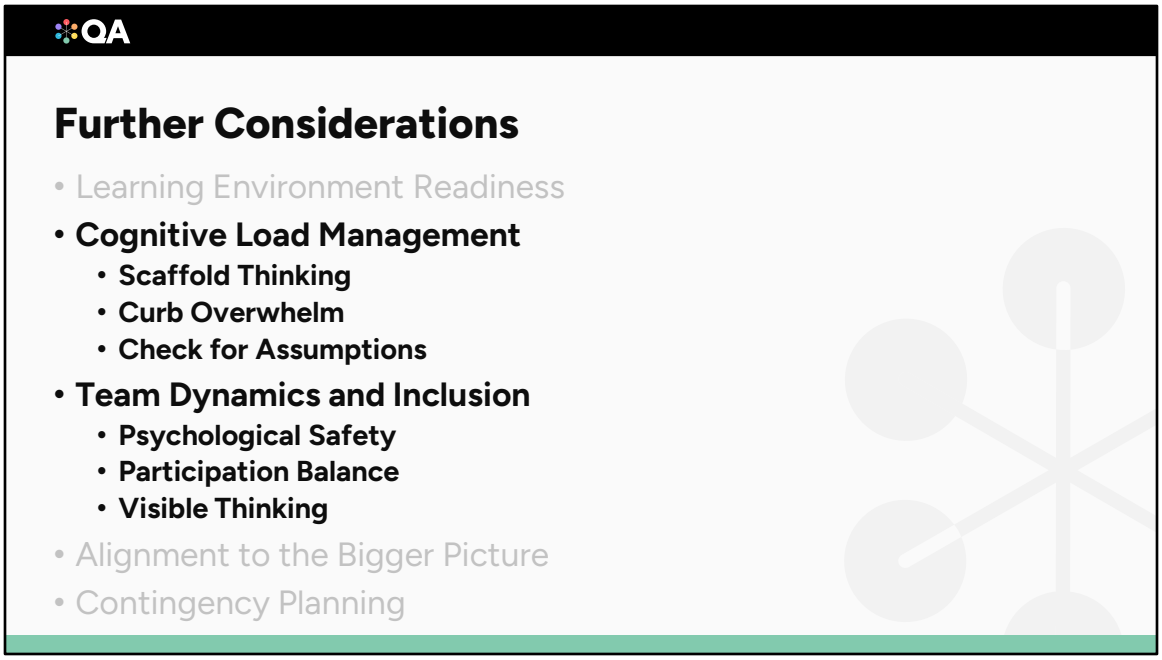


QA

Further Considerations

- **Learning Environment Readiness**
  - **Physical/Digital Setup**
  - **Pairing/Grouping Strategy**
  - **Timekeeping**
  - **Test Project Bootstrapping**
- Cognitive Load Management
- Team Dynamics and Inclusion
- Alignment to the Bigger Picture
- Contingency Planning

Physical/Digital Setup: Ensure access to coding tools (e.g. IDE with xUnit, access to Tennis Kata repo or prompt), shared screen or whiteboard, and any necessary ensemble programming tools (e.g. mobti.me, Live Share, Zoom).
Pairing/Grouping Strategy: Pre-decide if you'll use pairs, trios, or small groups. Random assignment is fine but consider experience mix (pair juniors with mentors). Have a quick method for forming groups.
Timekeeping: Use visible timers for segments (especially the split between demo and coding). This prevents overrun and models disciplined facilitation.
Test Project Bootstrapping: Ensure a starter code structure is available (e.g. empty test project in C# with xUnit and a placeholder class under test) to avoid spending learning hour time on setup.



Scaffold Thinking: For groups unfamiliar with domain rules or test naming, provide early scaffolds (e.g. “Use Given–When–Then to frame test cases” or “What are the inputs/outputs for ‘BOGOF’?”).
Curb Overwhelm: Don’t introduce too many complex rules at once. Consider breaking the task (Kata) into layers (e.g., start with basic BOGOF before moving to applying additional club card member discounts).
Check for Assumptions: Listen for "silent consensus" or misunderstanding of the rules and bring those to the surface: “How are we interpreting 'BOGOF, TFTPOT, Fish Friday'? What exactly triggers them?”. Does BOGOF mean "Binary Operations Generating Optimal Functions" or "Buy One Get One Free"? TFTPOT: "Transfer Files To People On Time", "Tools For Tech People On Task", "Two For The Price Of Three"? Fish Friday: "That weird day when projects mysteriously multiply and everyone is “floundering.”, "An end-of-week event where colleagues “tackle big fish” (major tasks or clients).", "Discounted seafood at the Waitrose fish counter on Fridays"

Psychological Safety: Remind the team this is *practice*, not performance. “We’re here to think out loud and learn by doing.” Invite early questions, model vulnerability (“This rule always trips me up too…”).

Participation Balance: Watch for dominant voices. Use gentle nudges like: "Let's hear from someone who hasn't spoken yet," or reshuffle pairings mid-way.
Visible Thinking: Encourage teams to write test cases on whiteboards or sticky notes *before* code, to externalise their thinking and make discussions visible.

QA

Further Considerations

- Learning Environment Readiness
- Cognitive Load Management
- Team Dynamics and Inclusion
- **Alignment to the Bigger Picture**
  - **Bridge to Ensemble**
  - **Support Broader Learning Goals**
  - **Vocabulary Calibration**
- **Contingency Planning**
  - **People get stuck on rules/requirements**
  - **Participants debate rule/requirement interpretation**
  - **Time runs short**

Bridge to Ensemble: Re-emphasise the reason we're doing this: "In the next session, we'll apply these skills together as a whole team on a more ambiguous story. Think of this as rehearsal for that."
Support Broader Learning Goals: For example, Reinforce the idea that translating requirements into tests is the first step of the TDD cycle (Red → Green → Refactor). Optional: reflect on where today's activity sits in that cycle.
Vocabulary Calibration: Encourage shared terms: "What do *we* mean by 'unit test', 'requirement', or 'testable'?" Shared vocabulary = smoother communication.

People get stuck on rules/requirements: Offer one concrete example test (e.g. SecondItemInBOGOF_ShouldReturn_PriceOfMoreExpensiveItem) and model the structure.
Participants debate rule/requirement interpretation: Let them! Capture both interpretations and encourage tests for each. Point out how tests *surface* ambiguity.
Time runs short: Prioritise objectives. Streamline activities and be prepared to adjust or even skip certain parts. In the TDD example, meaningful test naming and at least one red → green cycle per group. Cut deeper implementation to

preserve debrief time. Can always restructure the SoW and later Learning Hour plans or rely on the associated ensemble hour to draw out examples.

QA

Optional Enhancements

- **Mini Glossary** on the wall/whiteboard: Definitions for terms like “unit test,” “integration test,” “test name clarity,” etc.
- **Shared Test Case Log**: Create a live document or board where groups can add their example test names. Review together in conclusion phase.
- **Homework**: Ask them to refine or extend their Tennis Kata tests individually or in pairs and bring them to the next session.

QA

Summary

There are lots of things to consider when it comes to creating material to help others learn

Specific Role

Starting positions of junior devs

How long it takes to complete tasks

What are your fears?


QA
Learn. To Change.