Day 1 Resource Pack – CPSA® Advanced Level Module DDD (iSAQB®)
Volkswagen Group India | 22-Dec-2025 | 09:00–13:00 IST

Complete notes + learning guide + exercises + tools + mini-project brief (Day 1 only)

Audience	Mode	Duration	Prereq	iSAQB Focus (Day 1)	Outputs Today
Senior + mid-level engineers/architects (VW Group India)	Online, highly interactive	4 hours	CPSA-F completed	LG 1-1 to LG 1-4	Working Ubiquitous Language + initial Domain Model skeleton

1) Why this module matters in automotive software (without the fluff)

Software-defined vehicles push teams into a reality where multiple domains evolve in parallel (vehicle functions, manufacturing, diagnostics, connected services) and models drift unless the language and boundaries are managed intentionally. DDD is a disciplined way to keep domain knowledge, architecture decisions, and code aligned.

Industry signals (competitors):

- BMW talks about SDV-scale software architecture for its Neue Klasse with concentrated compute ("four superbrains") and zonal wiring. [S1]
- Toyota positions Arene as a software development platform and on-vehicle software services targeting deployment on vehicles starting 2025. [S2][S3]
- Mercedes positions MB.OS as a chip-to-cloud architecture rolled out across its portfolio. [S4][S5]

2) Day 1 learning goals (explicit mapping to iSAQB curriculum)

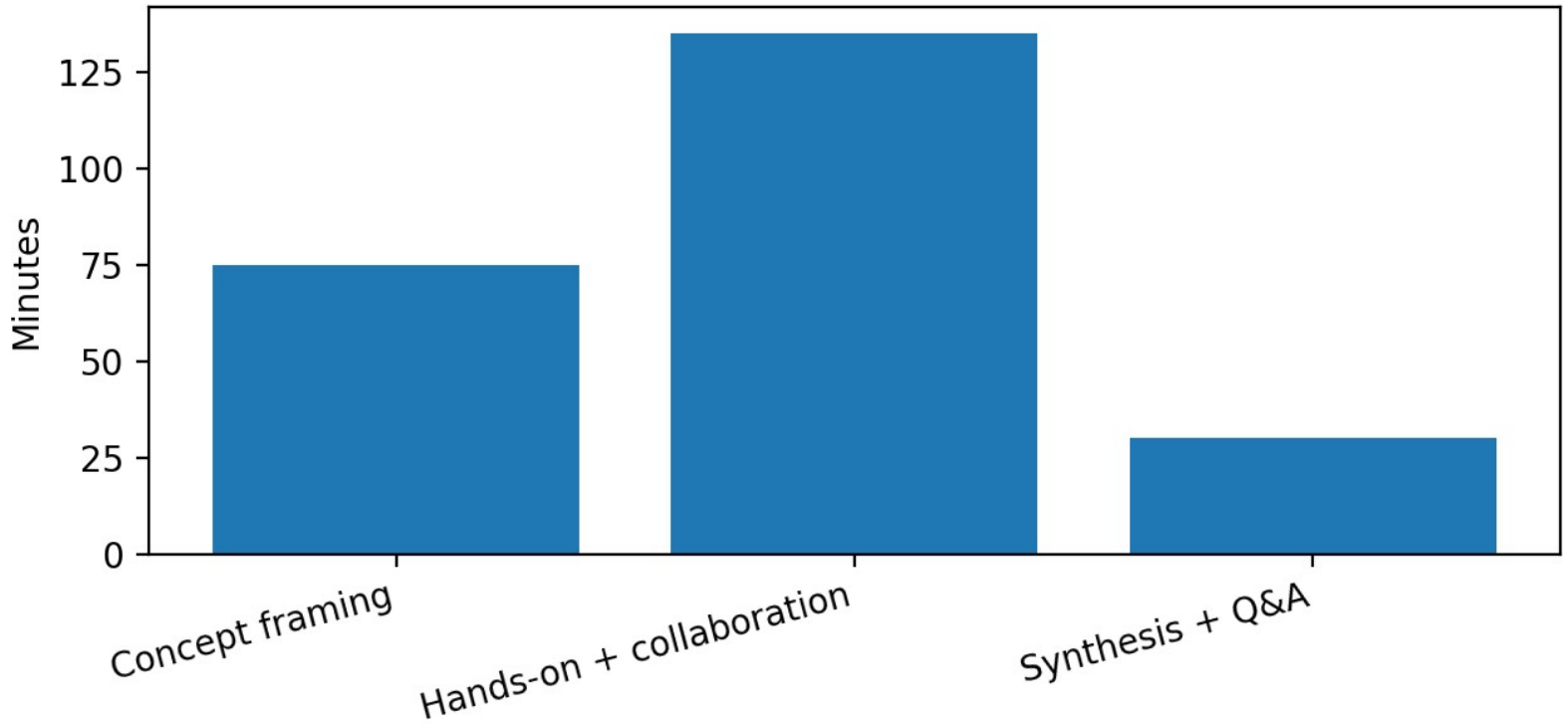
Today we cover the curriculum chapter: "Domain, model and ubiquitous language" and meet LG 1-1 to LG 1-4. (iSAQB CPSA-A DDD curriculum, 2023.1).

Learning Goal	What competence looks like at work	Evidence we produce today	Common failure mode (what we avoid)
LG 1-1 Connections: domain ↔ software ↔ model	You can explain why software is not an end-in-itself, and how the model abstracts expert knowledge into design decisions.	A 1-slide "domain dependency statement" + model purpose statement for one VW-relevant domain slice.	Treating the domain model as a database schema or a "class diagram deliverable".
LG 1-2 Ubiquitous Language (UL) and terminology	You can construct and enforce a shared language inside a clear boundary; terms show up in	A UL glossary v0: 20–30 terms, with definitions + synonyms to ban.	Letting multiple teams use the same word with different meanings (semantic drift).

	conversations, diagrams, docs, and code.		
LG 1-3 DDD building blocks	You can distinguish Entities vs Value Objects, Aggregates, Services, Factories, Repositories, Domain Events.	A candidate model skeleton with 6–10 building blocks identified and justified.	Overusing “Service” as a dumping ground; ignoring invariants and identity.
LG 1-4 Connections between building blocks	You can reason about how blocks interact and why (aggregate boundaries, invariants, event publishing).	A small interaction sketch: Aggregate Root commands -> invariant checks -> Domain Events.	Designing everything as an aggregate; or allowing external references into aggregate internals.

3) 4-hour session plan (how the time is spent)

We will trade breadth for depth. You will leave with tangible artifacts you can reuse in your projects.



Time	Activity
09:00–09:20	Kickoff + expectations + choose a shared automotive domain slice for exercises
09:20–10:05	Domain, domain model, UL – why language is architecture
10:05–10:55	Tactical building blocks: Entity, Value Object, Aggregate, Service, Factory, Repository, Domain Event
10:55–11:05	Break
11:05–12:05	Exercise 1: Ubiquitous Language + term conflicts + glossary v0
12:05–12:45	Exercise 2: Entity vs Value Object + Aggregate boundaries + invariants
12:45–13:00	Synthesis: artifacts review + how to carry these into Day 2 knowledge crunching

4) Core concepts and terms (Day 1) – sharp definitions + automotive examples

These are the terms you must be able to use precisely by end of Day 1. Definitions align with standard DDD literature (Evans; Vernon) and iSAQB curriculum terminology.

Term	Working definition	Automotive anchoring / nuance
Domain	The problem space your software serves. In automotive: e.g., vehicle configuration, diagnostics, manufacturing execution, OTA updates.	If you cannot explain the domain, you cannot design good software for it.
Domain Model	An abstraction of domain knowledge that captures the key concepts, constraints, and relationships.	Not a DB schema. Not only a UML diagram. It must drive code and decisions.
Ubiquitous Language (UL)	A shared, intentionally curated vocabulary used consistently within a boundary (bounded context) across speech, docs, diagrams, and code.	UL is a design choice. Synonyms and overloaded words are technical debt.
Entity	A thing with identity and lifecycle. Identity stays stable even if attributes change.	Vehicle, CustomerOrder, DiagnosticSession.
Value Object	Describes a concept by its attributes only; no identity; typically immutable and comparable.	VIN, OdometerReading, TorqueNm, Money, GeoCoordinate.
Aggregate	A cluster of associated objects treated as a unit for data changes, with one Aggregate Root controlling invariants.	Vehicle aggregate containing VehicleSpec, SoftwarePackageSet, FeatureFlags.
Aggregate Root	The only object outside code can reference to modify the aggregate; it enforces invariants.	Vehicle root ensures “VIN immutable” and “software compatibility constraints”.
Domain Service	Domain operation that does not naturally belong to a single entity/value object; should be stateless.	PricingPolicy, EligibilityCheck for feature activation.
Factory	Encapsulates creation logic that is complex or business-rule heavy.	VehicleOrderFactory applying mandatory defaults + compliance rules.
Repository	Provides access to aggregates by identity and hides persistence details.	VehicleRepository loads/saves Vehicle aggregate without leaking DB concerns.
Domain Event	A domain-significant thing that happened (past tense) that other parts care about.	VehicleRegistered, SoftwarePackageInstalled, WarrantyClaimOpened.

5) Visuals you can reuse (text-rendered)

Ubiquitous Language lifecycle (inside a boundary)

Domain expert terms --> Workshop conversation --> Glossary v0

| | |

```
      v               v               v
Edge-case stories --> Model sketching --> Code names + tests
      |               |               |
      +------ feedback loop: rename, split, deprecate terms ------+
```

Entity vs Value Object quick test

Does it have a stable identity over time?

- Yes --> Entity (identity matters more than attributes)
- No --> Value Object (attributes define it; prefer immutability)

Aggregate boundary heuristic (first-pass)

Pick a candidate root that:

1) Owns invariants that must be consistent at change time
2) Is the ONLY entry point for modifications
3) Can be loaded/saved as a transaction boundary (often)

6) Hands-on exercises (Day 1) – designed to satisfy LG 1-1 to LG 1-4

Exercise 1 (60 min): Build the Ubiquitous Language v0 for a chosen automotive slice

Goal mapping: LG 1-1, LG 1-2

Step	What you do	Output artifact	Facilitator nudge (what I will ask)
1	Pick one shared slice (example): Vehicle Configuration + Feature Activation (OTA).	Named domain slice + boundary assumption.	What is the business outcome? Who are the domain experts?
2	List 25 candidate terms from real conversations/docs (no abstractions yet).	Raw term list.	Which terms are overloaded? Which are synonyms?
3	For each term: definition, example, and 'do-not-use' synonym.	Glossary v0 table.	If two teams disagree on a term, what happens in code and integration?
4	Identify 3–5 term conflicts and resolve: split, rename, or scope to boundary.	Conflict log + resolution decisions.	Where does 'ubiquity' apply? What is outside the boundary?
5	Translate 5 key terms into code-level names (classes, methods) and test phrases.	Naming sheet + test sentence list.	Can a new joiner infer meaning from names alone?

Deliverable template (copy/paste into Miro/Sheet):

Term	Definition (1 line)	Example	Synonyms to ban	Boundary notes

Exercise 2 (40 min): Identify Entities, Value Objects, Aggregates and invariants

Goal mapping: LG 1-3, LG 1-4

Scenario seed (you can swap with your real project): 'Customer configures a vehicle, selects packages, order is validated, software features are activated post-delivery via OTA'.

Task	Hint (what to watch for)	Output
A) List candidate Entities and justify identity.	Identity survives attribute change. Think lifecycle and references.	Entity list with identity key.
B) List candidate Value Objects and justify immutability.	If it can change, prefer replacing the whole object.	Value Object list + equality rule.
C) Propose 1–2 Aggregates and define invariants.	Invariants are business rules that must always	Aggregate sketch + invariants.

	hold.	
D) Draft 2 Domain Events (past tense).	Events are integration-friendly facts, not commands.	Event names + payload fields.

Event examples (pattern):

VehicleRegistered { vin, modelCode, plantCode, timestamp }
SoftwareFeatureActivated { vin, featureCode, version, activationReason, timestamp }

7) Online platforms and free tools (practical, not aspirational)

We will use tools that work well in online workshops and have free tiers suitable for practice. Tool choice is optional; outputs matter more than tooling.

Category	Free options	How we use it in this training
Collaborative whiteboarding	Miro (free tier), FigJam (free tier), Mural (trial)	EventStorming-like mapping, glossary tables, context maps
Diagramming	diagrams.net (draw.io), Excalidraw, Mermaid Live Editor	Quick domain sketches, aggregate diagrams, architecture diagrams
Text + glossary	Google Sheets / Docs, Notion (free), Obsidian (local)	Ubiquitous Language glossary, conflict log
Code + modeling	VS Code + Markdown, GitHub repo, PlantUML server (optional)	Keep UL close to code; version the model decisions

Recommended lightweight convention (so artifacts don't die after the call)

Repository structure (suggested)

```
/ddd-workshop/
  /day1/
    ubiquitous-language.md
    glossary.csv
    model-skeleton.drawio
    decisions.md  (why we named/split/defined boundaries)
```

8) Case study intel (competitor context) – what to learn without copying

We are not copying competitors. We are reading signals about SDV complexity: compute centralization, platform thinking, and rapid feature evolution. DDD's value is in keeping domain meaning stable while the technical substrate changes.

Competitor signal	What was publicly stated	DDD takeaway (Day 1 lens)
BMW (Neue Klasse)	BMW describes SDV-scale changes: key functions concentrated in four compute units ('superbrains') and a zonal wiring harness approach.	When the platform evolves fast, language and boundaries become the stabilizer. Model drift becomes expensive. [S1]
Toyota (Arene)	Toyota positions Arene as a software development platform and on-vehicle software services; public statements target deployment beginning 2025.	Platformization increases cross-domain interactions; UL discipline prevents "one model to rule them all". [S2][S3]
Mercedes (MB.OS)	Mercedes positions MB.OS as a chip-to-cloud architecture for future products and describes	Chip-to-cloud implies many bounded contexts: in-vehicle, cloud, services, data. Boundaries and

	rollout across portfolio.	translation patterns matter. [S4][S5]

9) Mini-project (starts Day 1, continues across the week)

Mini-project theme: Vehicle Configuration + Feature Activation (OTA) – from language to model.

Day 1 deliverable: UL glossary v0 + model skeleton with Entities/VOs/Aggregates + 2 Domain Events.

Part	Description	Day 1 scope	How we will extend later
P1: Language	Build UL and conflict log inside an explicit boundary.	Glossary v0 (20–30 terms) + 3 resolved conflicts.	Day 2: validate via interviews/observation; evolve terms through stories.
P2: Model skeleton	Identify Entities/VOs and propose Aggregates + invariants.	One aggregate boundary + invariants + root commands.	Day 3: map to code structures and repositories/factories.
P3: Events	Define domain events that matter and are integration-friendly.	Two events with payload and meaning.	Day 5–6: events as cross-context communication; risks and event store.
P4: Architecture link	Explain why model matters to architecture boundaries.	Short statement only.	Day 4–6: bounded contexts + context map + integration patterns.

10) Self-study + recap checklist (before Day 2)

Day 2 is about extracting knowledge with domain experts. So you need clean Day 1 artifacts, not perfect diagrams.

- Refine glossary: remove duplicates, mark ambiguous terms, add example sentences.
- Bring 2 real VW project terms that are overloaded (same word, different recap meanings).
- Pick 1 domain expert persona relevant to your slice (manufacturing, diagnostics, sales, etc.) and list 5 interview questions.

Day 2 preview: Knowledge Crunching and Collaborative Modeling (EventStorming / Domain Storytelling / interviews). We will stress-test Day 1 UL against real scenarios and uncover hidden knowledge.

References (public sources used for Day 1 pack)

[S0] iSAQB. (n.d.). Domain Driven Design - CPSA Advanced Level Module (DDD). iSAQB.org.

[S1] BMW Group. (2025, Mar 11). Four Superbrains for the Neue Klasse: More intelligent, more efficient, more powerful. Press release.

[S2] Woven by Toyota. (n.d.). Arene - Software development platform. woven.toyota.

[S3] Toyota Global Newsroom. (2023, Apr 11). Woven by Toyota to Accelerate Toyota's Vision for Mobility (Arene).

[S4] Mercedes-Benz Group. (n.d.). Mercedes-Benz Operating System (MB.OS).

[S5] Business Wire. (2023, Feb 22). Mercedes-Benz previews its operating system MB.OS (chip-to-cloud).

[S6] Evans, E. (2003). Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley.

[S7] Vernon, V. (2013). Implementing Domain-Driven Design. Addison-Wesley (sample pages).