iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

DECONSTRUCTING DOMAIN-DRIVEN ARCHITECTURE FOR CO-BRANDED CLOUD BANKING PLATFORMS

Ved Bhat
Senior Technical Business Analyst.
Independent Researcher, United Kingdom

ABSTRACT
Bringing a household consumer brand into a retail bank's card portfolio requires infrastructure that most legacy banking estates were never designed to support. This paper examines how Barclays addressed that challenge in delivering the Next Generation Consumer Banking (NGCB) platform for its Amazon co-branded credit card, using an in-depth case study grounded in the author's direct delivery experience as the Senior Technical Business Analyst responsible for the underlying data-mapping work. The central argument is that a domain-driven approach to data architecture, built on cloud-native infrastructure, is what makes rapid, ecosystem-scale product launches possible inside a bank still running on legacy rails. The paper traces how disparate legacy source systems — the Customer System, Debt Manager and PPE — were mapped into four reusable domains (customer, debt/default, transaction and bureau reporting); how outbound and inbound data exchange with Experian and TransUnion was standardised; and how the resulting affordability and creditworthiness data was made fit for use in Probability of Default (PD), Exposure at Default (EAD) and Loss Given Default (LGD) credit-risk models. These findings are then generalised into a six-step framework that other institutions can adapt, together with the governance and data-lineage disciplines needed to scale it safely. The architecture gave Barclays a reusable route into Amazon's customer base and left the bank better placed to onboard further ecosystem partnerships without repeating the heaviest integration work each time. The paper closes with practical recommendations for institutions planning similar partnerships and an assessment of the approach's future relevance as regulatory scrutiny of data lineage and governance continues to increase.

Keywords:
domain-driven design, cloud-native architecture, retail banking, co-branded credit cards, credit risk modelling (PD/EAD/LGD), data governance

INTRODUCTION
Co-branded credit cards are not a new idea. What has changed is the speed at which banks are now expected to bring a partner's proposition to market, and the amount of data that has to move cleanly between the bank and the partner's ecosystem for that proposition to work day to day. When Barclays took on the Amazon co-branded credit card, the ambition was straightforward to describe and considerably harder to deliver: launch a product that felt native to Amazon's customer experience while meeting every regulatory obligation a UK credit card carries, from affordability assessment through to bureau reporting.

The obstacle was rarely the commercial agreement itself. It was the infrastructure underneath it. Customer records, account and transaction history, and default or delinquency data each lived in separate systems, built at different times, for different purposes, with little consistency between them. Bureau reporting to Experian and TransUnion ran on manual and batch processes that had accumulated over years rather than been designed as a single coherent flow. None of this was unusual for a bank of Barclays' size and history, but all of it stood in the way of a fast, low-friction partnership launch.

This paper describes how that obstacle was addressed on the NGCB platform, and more importantly, why the approach taken — domain-driven architecture on a cloud-native platform — has value well beyond this one project. Any institution weighing up a similar partnership, whether with a retailer, an airline, or a technology platform, is likely to run into the same structural problem: legacy systems that were never built to be shared, extended, or plugged into someone else's ecosystem. The blueprint set out here is offered as a practical starting point for that conversation, not a theoretical one.

This kind of arrangement is becoming more common, not less. Embedded-finance and banking-as-a-service propositions, where a non-bank brand offers a financial product underwritten and operated by a regulated

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

institution, have moved from a niche experiment to a mainstream growth channel for retail banks looking to reach customers who would not otherwise walk into a branch or open a banking app directly. Amazon's card is one visible example of a wider pattern that also includes airline, retail and technology-platform co-branded products across the market. What distinguishes the institutions that execute these partnerships well from those that struggle is rarely the commercial terms; it is almost always whether the underlying data architecture can absorb a new partner's requirements without a bespoke rebuild each time. That distinction is the practical motivation for this paper.

A note on scope: this paper focuses on the data architecture and integration layer of the NGCB/Amazon delivery — the mapping, governance and bureau-reporting work — rather than on commercial deal structuring or customer-facing product design, which sat with other teams. It is also worth being upfront about what this paper is not. It is not a claim that domain-driven architecture solves every problem a bank faces when entering an ecosystem partnership, and it is not a product pitch for any particular vendor or platform. The intention is narrower: to describe, in enough detail to be useful, what one large-scale delivery actually required, and to draw out the parts of it that generalise to other institutions facing a similar decision.

This paper has three objectives. First, to document in operational detail how domain-driven, cloud-native architecture was used to integrate the Amazon co-branded card into Barclays' retail banking infrastructure. Second, to generalise that experience into a repeatable framework that other institutions can adapt for similar ecosystem partnerships. Third, to set out the governance and data-lineage disciplines needed to scale such an architecture safely, and to identify the market gap this paper is intended to fill relative to existing reference-architecture and microservices literature.

This paper is written primarily for architecture, data and technology leaders inside retail banks who are being asked to support a partnership proposition — whether with a retailer, a technology platform, or another consumer brand — and who need a realistic sense of what that support actually involves. It should also be useful to commercial and product teams sponsoring such a partnership, since a good part of the friction described here comes from misaligned expectations between the commercial timeline and the architectural one. It assumes some familiarity with retail banking operations and credit risk concepts, but it does not assume a deep technical background in cloud architecture or domain-driven design; where technical terms are used, they are explained in enough detail to follow the argument, though readers looking for implementation-level detail — API schemas, specific cloud configurations, and so on — will need to look elsewhere, since this paper is deliberately pitched at the level of approach and decision-making rather than code.

Domain-driven design is not a new concept in software engineering, but its adoption inside retail banking has been slower than in sectors with fewer legacy constraints. The idea, at its simplest, is to organise systems around business domains — customer, payments, risk, and so on — rather than around whatever technical boundaries happen to exist between old applications. Recent industry analysis of microservices adoption in financial services makes the same point: domain-driven design gives institutions a way of decomposing complex systems into services that are aligned with the business itself, rather than with historical IT boundaries, which lets those services evolve independently of one another (European Journal of Computer Science and Information Technology, 2025). That independence matters more than it might first appear. In a monolithic or tightly coupled environment, adding a new product often means touching several systems that were never meant to be touched together, and each of those touches carries its own risk of delay or error. A domain-driven, cloud-native platform is built the other way round: new capability is added by extending or reusing an existing domain, rather than by re-engineering the whole chain. Several recent studies of banking modernisation describe this same shift from monolithic to microservices-based systems as a genuine change in how banking platforms are designed and operated, not simply a technical refresh (World Journal of Advanced Research and Reviews, n.d.; Zymr, 2026).

The banking sector has also developed its own reference points for this kind of decomposition. The Banking Industry Architecture Network, for instance, has built a standardised approach to defining sub-domains specifically so that banks are not each solving the same decomposition problem from scratch (Banking Industry Architecture Network, n.d.). NGCB's domain split — customer, debt management, transaction, and bureau reporting — sits comfortably within that broader industry direction, even though it was arrived at through the practical demands of the Amazon proposition rather than through a formal reference model.

None of this is to say domain-driven architecture is free of difficulty. Establishing clean boundaries between legacy systems and new services usually calls for some kind of translation layer, so the new architecture is not compromised by the quirks of what it is replacing, and getting that boundary right is as much a matter of judgement as of technical design (GitHub, n.d.). The following sections describe how that judgement played out in practice on the NGCB platform.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

Why cloud-native matters alongside domain-driven design

Domain-driven design describes how data and services should be organised; it does not, by itself, say anything about where that architecture should run. In practice, the two tend to arrive together in modern banking builds, because a cloud-native environment gives a domain-driven platform the elasticity it needs to actually behave the way it was designed to. Consistent environments, automated scaling and better resource utilisation are recurring themes in the literature on cloud-native adoption in financial services, and they are the practical reason a bank can add a new domain, or extend an existing one, without a lengthy hardware or capacity planning exercise getting in the way (World Journal of Advanced Research and Reviews, n.d.; Zymr, 2026).

For NGCB specifically, the cloud-native element is what made 'reusable' a meaningful word rather than an aspiration. A domain that has been designed cleanly but is still tied to fixed, on-premises capacity will always run into a scaling conversation before the next product can be onboarded. Removing that constraint was as much a part of the platform's value as the domain mapping itself.

It is worth being specific about what this elasticity bought in practice. Onboarding a partner the size of Amazon means the platform has to absorb a large, front-loaded surge in account creation and bureau reporting volume during launch, followed by a steadier but still unpredictable ongoing load as adoption grows. A fixed-capacity estate forces a bank to provision for the peak well in advance, at real infrastructure cost, and still risks being caught out if uptake exceeds the plan. A cloud-native, domain-driven platform instead lets each domain scale to the load it is actually seeing, which is what allowed the NGCB build to treat the Amazon launch volumes as an operational question rather than a capacity-planning risk to be negotiated months in advance.

Legacy migration is a process, not an event

One point that tends to get lost in high-level descriptions of domain-driven transformation is that a bank rarely gets to build the new architecture on a clean slate. The legacy estate keeps running throughout, often for years, and the new services have to coexist with it. Industry commentary on this transition tends to describe a phased migration, carried out with engineering discipline and clear domain ownership, as the difference between microservices becoming a genuine competitive advantage and becoming just another modernisation initiative that stalls halfway through (Oracle Corporation, n.d.). NGCB's own rollout followed that pattern: the domains were introduced progressively, alongside the legacy systems they were eventually intended to reduce reliance on, rather than through a single cut-over.

In practice this meant NGCB and the legacy estate had to be treated as two systems in active conversation with each other for the duration of the build, rather than a source and a destination in a one-off migration. Interfaces between the new domains and the legacy systems they were drawing from had to be built with the expectation that the legacy side would keep changing underneath them, which shaped decisions as basic as how tightly a domain's internal schema was allowed to mirror the legacy source schema it was reading from. Domains that copied the legacy shape too closely, for convenience, tended to need rework when the legacy system changed; domains that abstracted away from the legacy shape from the outset generally absorbed those changes more gracefully, at the cost of more up-front design effort.

Data lineage as a design principle, not a compliance afterthought

A theme that recurs throughout the technical literature on financial services modernisation, and one that matters just as much in practice as it does on paper, is that data lineage needs to be built into a domain-driven architecture from the outset rather than added once a regulator asks for it. This is partly a compliance point, since UK and EU reporting obligations increasingly expect institutions to be able to trace a reported figure back to its source, but it is also a practical one. A domain that cannot explain where its data came from is much harder to trust when something in a report looks wrong, and much slower to debug when it does.

The domain boundaries chosen for NGCB were shaped by this consideration as much as by any other. Separating bureau reporting into its own domain, distinct from the customer and transaction domains it draws on, made it considerably easier to trace a given reported data point back through the mapping logic to its originating system, which mattered both for internal assurance and for the bank's ongoing regulatory obligations.

There is also a practical audit dimension worth noting. When a regulator or internal audit function asks how a specific figure in a bureau report was derived, the answer is only useful if it can be produced quickly and traced end to end without engaging several system owners to reconstruct the path by hand. Building that traceability into the domain boundaries themselves, rather than bolting it onto a data warehouse afterwards, is what turns lineage from a slow, manual exercise into something close to a standing capability of the platform.

The market gap this paper addresses

It is worth being precise about what is already well covered elsewhere, so the contribution of this paper is not overstated. The Banking Industry Architecture Network already provides a mature, industry-agreed vocabulary for what a domain-decomposed bank should look like at rest, and the broader microservices-in-financial-services

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

literature is reasonably thorough on the destination architecture: cloud-native infrastructure, service boundaries aligned to business capability, and the technical case for moving away from monolithic cores (European Journal of Computer Science and Information Technology, 2025; World Journal of Advanced Research and Reviews, n.d.; Zymr, 2026; Banking Industry Architecture Network, n.d.). An institution asking 'what should our target domain model look like' already has good places to look.

The gap sits one level down, in the sequencing and governance of getting a live, legacy-encumbered estate from where it actually is to that target model, under a commercial deadline set by an external partner rather than by the bank's own modernisation roadmap. Reference architectures and vendor material are generally written at the pattern level, and are largely silent on the questions that actually determined the pace of the NGCB delivery: who owns a data point once it moves out of the system that has always held it, how to keep new domain mappings in step with a legacy estate that keeps changing underneath them, how much interpretive judgement bureau reporting standards really leave once you get into the detail, and how to design reusability into a first product rather than assuming it follows automatically — each examined in the Challenges and Lessons Learned discussion below.

This paper's contribution is narrower than a reference model but more operational: a sequenced framework, grounded in one live delivery, for the order in which those governance and ownership questions have to be resolved before the domain-driven architecture itself can be trusted at scale.

This paper is therefore best read as complementary to, rather than competing with, the pattern-level literature cited above. A reader who wants to know what a target domain model should contain will find little new here; a reader who wants to know what actually happens in the eighteen months between deciding to pursue that target model and having a live, regulator-facing platform running on it — who has to agree what, in what order, and what tends to go wrong along the way — is the intended audience for the account that follows.

METHODOLOGY

This paper adopts a single, in-depth case-study methodology, a design well suited to research questions that ask 'how' and 'why' a real-world intervention unfolded, particularly where the phenomenon and its organisational context cannot easily be separated. The case examined is the delivery of the Next Generation Consumer Banking (NGCB) platform for Barclays' Amazon co-branded credit card, selected because it is a live, completed delivery in which domain-driven architecture decisions had directly observable commercial and regulatory consequences, rather than a theoretical or simulated exercise.

Data sources

The account presented here draws on the author's direct, first-hand involvement in the NGCB/Amazon delivery as Senior Technical Business Analyst, supplemented by delivery documentation, source-to-target mapping artefacts, and structured recollection of stakeholder workshops conducted across engineering, architecture, risk, compliance and commercial functions during the build. No customer-level, transactional or otherwise commercially sensitive data is reproduced in this paper; all descriptions are given at the level of process, domain structure and outcome.

Analytic approach

The analysis follows a structured, thematic account of the delivery: (i) documenting the legacy starting point and its practical consequences, (ii) tracing how source systems were mapped and decoupled into domains, (iii) examining how outbound and inbound bureau-reporting data was standardised, and (iv) assessing how that data was made usable for downstream credit-risk models. From this account, the paper inductively generalises a repeatable framework (Section on 'A Repeatable Framework for Ecosystem Partnerships') and a set of governance and lessons-learned themes, checked against the author's experience of what did and did not transfer cleanly beyond the specific Amazon proposition.

Researcher's role and positionality

Ved Bhat is a Senior Technical Business Analyst with 20 years of experience across banking, financial services and insurance (BFSI) digital transformation programmes, and a Certified Business Analysis Professional (CBAP). On the NGCB/Amazon delivery discussed in this paper, his role centred on the data-mapping work that underpinned the domain-driven build: mapping customer master data from the Customer System, default and delinquency data from Debt Manager, and account and transaction data from PPE into NGCB's domain-driven structure, working closely with engineering and architecture colleagues on the source-to-target mappings, transformation rules and validation logic described in the case-study section below. He writes here as an independent researcher, drawing on that delivery experience rather than on a formal architectural mandate; this practitioner vantage point is a source of first-hand operational detail, but it is also a limitation, discussed further in the Conclusion.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

Trustworthiness and rigor

Case-study research of this kind is judged less on statistical validity and more on whether the account given is credible, well-evidenced and transparent about its own vantage point. Three practical steps were taken to support that here. First, the sequence of events described in the case-study section below was cross-checked against contemporaneous delivery artefacts — mapping documents, workshop outputs and validation logs — rather than reconstructed purely from memory some time after the fact. Second, wherever a claim in this paper concerns another function's perspective (risk, compliance, engineering), it is presented at the level of what that function needed and received, rather than as an interpretation of that function's internal reasoning, which the author was not positioned to observe directly. Third, the limitations section above is intended as a genuine constraint on how the findings should be read, not a formality: a single-author, single-case account of a live commercial delivery cannot be independently replicated in the way a controlled study can, and the framework offered in this paper should be treated as a starting point for adaptation rather than a validated prescription.

Scope and limitations

Consistent with the case-study design, findings are drawn from a single delivery at a single institution and are not statistically generalisable; they are offered as an analytically generalisable framework for institutions facing structurally similar problems, to be adapted rather than applied mechanically.

CASE STUDY: THE NGCB / AMAZON CO-BRANDED CREDIT CARD PLATFORM

NGCB is Barclays' cloud-based platform for onboarding new consumer credit products, and the Amazon co-branded card was the first major proposition delivered on it. The remainder of this section works through the legacy starting point, the domain-mapping approach taken, the bureau-integration work, and the way affordability and credit-risk data was brought into the bank's PD, EAD and LGD models.

The legacy starting point

Before the NGCB work began, Barclays' card data sat across a set of systems that had grown up independently of one another. Customer master data lived in the Customer System. Default and delinquency information sat in the Debt Manager platform. Account and transaction activity was held in PPE. None of these had been designed to share a common schema, and there was no unified data model tying them together.

This created two distinct problems. The first was reporting: sending accurate, complete data to Experian and TransUnion depended on manual reconciliation and batch file preparation, which is slow by nature and carries a higher risk of error than an automated flow. The second, and arguably the more limiting, was scalability. A tightly coupled set of systems is workable when a bank only ever needs to onboard its own products at its own pace, but it becomes a serious constraint the moment a partnership like Amazon's requires fast, repeatable onboarding.

It is worth being concrete about scale, since the practical difficulty of a legacy estate is often a function of size and history as much as of technical design. Systems such as the Customer System, Debt Manager and PPE had each accumulated years of incremental change, localised fixes and system-specific conventions that made sense in isolation but did not compose cleanly when three systems needed to agree on a shared view of the same customer. Untangling that history — rather than the initial act of connecting the systems together — was consistently the more time-consuming part of the legacy-mapping work.

Legacy characteristic	Practical consequence
Disparate source systems, no unified data model	Customer, debt, and transaction data could not be reconciled without manual effort
Manual, batch-driven bureau reporting	Slower turnaround and higher exposure to reporting error at Experian and TransUnion
Tightly coupled architecture	Every new product meant a fresh integration effort rather than reuse
Inconsistent mapping of bureau scores	Affordability and creditworthiness data was hard to feed cleanly into PD, EAD and LGD models
Weak domain separation	Ownership, traceability and governance of data were difficult to establish

Table 1. Key limitations of the pre-NGCB legacy environment

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

Mapping and decoupling the domains

The first substantive piece of work was to define what a sensible domain split should look like for this platform, and then map the existing data into it. Customer master data was drawn from the Customer System, default and delinquency data from Debt Manager, and account and transaction data from PPE, each mapped into a standardised structure within the NGCB domain-driven architecture rather than left in its original, system-specific shape.

This was not simply a relabelling exercise. Each domain needed its own clear ownership, its own definition of what data belonged inside it, and its own set of rules for how it would interact with the others. Getting that boundary right took close collaboration with engineering and architecture colleagues, working through source-to-target mappings, transformation rules, validation logic, and the data lineage requirements that would let the bank trace any given data point back to where it originated.

The payoff for this groundwork was reusability. Because the domains were defined around business concepts rather than around the quirks of any one legacy system, the same structures could, in principle, support the next product onboarded onto NGCB without a repeat of the mapping exercise from scratch. That is the practical meaning of a 'repeatable architectural framework' in this context: not that every product is identical, but that the underlying domain scaffolding does not need reinventing each time.

Ownership rules were made concrete rather than left as a principle on a slide. For each domain, a single accountable owner was named, along with a documented definition of which fields belonged inside that domain, which system was treated as the system of record for each field, and which downstream consumers were permitted to read from it. Where a field could plausibly sit in more than one domain — account status is a good example, since it touches both the customer and transaction domains — the resolution was agreed explicitly and documented, rather than left for each new consuming team to decide for itself. This level of specificity is what made the domain boundaries defensible later, when questions of ownership came under scrutiny during the governance and audit conversations described in the Results and Discussion section.

Standardising bureau integration with Experian and TransUnion

Credit card issuers in the UK are required to report a defined set of customer and account data to the major credit reference agencies, and Barclays' obligations to Experian and TransUnion were no exception. What NGCB changed was how that reporting happened. The exact fields required for outbound transmission were identified and documented in detail: personal details, account balances, payment history, delinquency and default indicators, credit account status, and affordability-related attributes.

Rather than continuing with the batch-and-reconcile approach inherited from the legacy environment, these outbound flows were rebuilt around standardised, cloud-based integration patterns. The intent was consistency as much as speed: a repeatable outbound structure reduces the room for the kind of reconciliation error that manual processes tend to accumulate over time, which matters both for regulatory reporting integrity and for the bank's own confidence in the data it holds.

The inbound side of the relationship needed equal attention. Bureau-sourced creditworthiness and affordability scores had to be ingested and aligned in a form the bank's own risk models could actually use, which is a different problem from simply receiving the data. Getting the ingestion and mapping logic right here was what allowed those scores to flow cleanly into downstream capital and strategy processes, discussed next.

It is worth noting what standardisation did not mean in this context. It did not mean forcing Experian and TransUnion onto an identical technical interface, since each bureau retained its own specification and its own change-management process. What was standardised was the bank's own side of the relationship: a single, domain-owned pipeline for preparing and validating outbound data before it was adapted to each bureau's particular format, rather than two parallel, independently maintained processes that could drift apart over time. That distinction — standardising the bank's internal handling while still accommodating each bureau's external requirements — turned out to matter a great deal in practice, and is picked up again in the discussion of bureau standards under Challenges and Lessons Learned.

Timing also mattered more than it might seem from a purely technical description. Bureau reporting obligations do not pause for a migration, so the cut-over from the legacy batch process to the new standardised pipeline had to be planned as a parallel-running exercise: both paths operating simultaneously for a defined period, with outputs reconciled against each other before the legacy path was retired. This added time to the schedule that a simple like-for-like replacement would not have needed, but it also meant the bank never had a period where its regulatory reporting obligations depended on an unproven pipeline with no fallback, which was judged the right trade-off given the consequences of a reporting failure to a credit bureau.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

Feeding standardised data into PD, EAD and LGD models

Probability of Default, Exposure at Default and Loss Given Default are the three parameters that sit at the heart of credit risk modelling under the Basel framework, and together they determine both the expected loss a lender anticipates on an exposure and the regulatory capital it must hold against that risk (Bank for International Settlements / Basel Committee on Banking Supervision, n.d.; LearnSignal, 2026; Office of the Superintendent of Financial Institutions (Canada), 2026). Because these parameters feed directly into pricing, provisioning and capital allocation decisions, the quality of the data going into them is not a peripheral concern; it is close to the whole point of the exercise (Office of the Superintendent of Financial Institutions (Canada), 2026).

On the NGCB platform, inbound affordability and creditworthiness scores from Experian and TransUnion were aligned specifically for this downstream use, meaning the mapping logic was designed with the PD, EAD and LGD models' data requirements in mind from the outset, rather than treating bureau data as a generic feed to be reshaped later. This is a subtle but important distinction: data that is merely accurate is not automatically data that is usable, and a good deal of the value in this work came from closing that gap.

The practical effect was a more reliable input to the bank's capital and strategy models for this product line, supporting better credit-risk assessment without requiring the risk teams themselves to adapt their models around inconsistent inputs. In a partnership like this one, where a new, high-volume customer base is being brought on board quickly, that reliability is not a nice-to-have; it is what allows the credit-risk function to keep pace with commercial ambition rather than becoming the bottleneck.

It is also worth being precise about the limits of what the architecture itself could guarantee. Aligning inbound bureau data with the PD, EAD and LGD models' requirements improved the reliability and usability of the inputs those models received; it did not, and was never intended to, change the statistical methodology of the models themselves, which remained the responsibility of the credit risk and capital modelling teams. The contribution of the domain-driven architecture was to remove data quality and format as a source of model error, so that whatever performance the models achieved could be attributed to the modelling approach itself rather than to noise introduced upstream.

Working across teams, not just across systems

A change of this scale was never going to be a purely technical exercise, and it is worth being explicit about how many parts of the bank had a stake in it. Credit risk and capital modelling teams depended on the inbound affordability and creditworthiness scores for their PD, EAD and LGD calculations. Customer and servicing platforms needed the customer master data, account balances and transaction activity integrated cleanly into the new architecture. Debt management and collections functions supplied the default and delinquency data that fed into bureau reporting and risk assessment. Regulatory reporting and compliance teams relied on the standardised outbound processes to meet ongoing credit-reporting obligations. Technology, data engineering and architecture teams were the ones adopting and maintaining the reusable integration patterns for future onboarding. And commercial and card product functions were, ultimately, the ones leaning on the scalable platform to support the Amazon launch and whatever came after it.

Coordinating across that many functions is, in some ways, harder than the data mapping itself. Each team had its own priorities and its own definition of what 'done' looked like, and a good part of the delivery effort went into keeping those definitions aligned rather than letting each function optimise for its own view of the platform in isolation.

A practical mechanism that helped here was to give each domain a named product-style owner who sat across the functional interests touching that domain, rather than leaving coordination to ad hoc meetings convened whenever a conflict surfaced. That owner was not expected to resolve every cross-functional disagreement personally, but was accountable for making sure a disagreement was surfaced and routed to the right people quickly rather than left to fester until it blocked a delivery milestone. This is a small structural point, but it is one of the more transferable lessons from the delivery: the technical domain boundaries and the organisational accountability for those domains need to be designed together, not treated as separate exercises.

A REPEATABLE FRAMEWORK FOR ECOSYSTEM PARTNERSHIPS

The NGCB experience points towards a framework that other institutions can adapt when planning a similar co-branded or ecosystem partnership. It is not a rigid methodology, since every bank's legacy estate is different, but the sequence below reflects the order in which the real decisions had to be made.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

Step 1 — Map the legacy estate honestly
Before any architecture can be designed, someone has to produce an honest account of where customer, transaction and default data actually live, what format they are in, and where the inconsistencies are. Skipping this step, or doing it superficially, is the most common reason integration timelines slip later on.
Step 2 — Define domains around the business, not the systems
Resist the temptation to mirror existing system boundaries when defining domains. The domains that matter are customer, transaction, debt/default, and reporting, regardless of how many legacy applications currently sit across those lines.
Step 3 — Agree data contracts before writing integration code
Source-to-target mappings, transformation rules and validation logic should be agreed and documented before development starts, not discovered midway through it. This is where architecture and engineering teams need to work closely together, and where most of the later rework is avoided.
Step 4 — Standardise outbound regulatory reporting first
Bureau or regulatory reporting obligations tend to be the least flexible part of the build, so bringing them onto a standardised, automated pattern early removes a major source of risk before the more customer-facing elements of the partnership are finished.
Step 5 — Design inbound data for its eventual use, not just its arrival
Inbound data, such as bureau scores, should be mapped with its downstream destination in mind — in this case, PD, EAD and LGD models — rather than treated as a generic feed to be reshaped later by whoever happens to consume it.
Step 6 — Build for the next product, not just this one
The final test of a domain-driven build is whether the next partnership can reuse it. If every new proposition still requires a bespoke integration, the architecture has not really achieved what it set out to.

Step	Primary owner	Typical output
1. Map the legacy estate	Data / architecture teams	Source system inventory, known inconsistencies log
2. Define domains	Enterprise architecture	Agreed domain boundaries and ownership model
3. Agree data contracts	Architecture and engineering jointly	Source-to-target mappings, transformation and validation rules
4. Standardise outbound reporting	Regulatory reporting / compliance	Automated, standardised bureau or regulatory feeds
5. Design inbound data for its use	Risk / data engineering jointly	Model-ready inbound data mappings
6. Build for reuse	Architecture leadership	Domain scaffolding validated against a second, hypothetical product

Table 2. Framework steps, ownership, and expected outputs

RESULTS AND DISCUSSION

Governance, data lineage and risk considerations
None of the above is safe to scale without proper governance sitting alongside it. A domain-driven architecture makes data lineage easier to establish, but it does not establish it automatically; someone still has to define who owns each domain, how changes to it are approved, and how a given data point can be traced back to its source when a regulator or an internal audit asks.
On the NGCB platform, this meant clear separation of business domains — customer, debt, transaction and bureau reporting — specifically to improve traceability and scalability, alongside the reusability of integrations across future products. Multiple functions across the bank had a stake in getting this right: credit risk and capital

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

modelling teams consuming the inbound scores, customer and servicing platforms handling the master data, debt management and collections functions supplying default information, regulatory reporting and compliance teams relying on the outbound flows, and technology and architecture teams maintaining the patterns themselves.

A point worth stressing for any institution considering a similar build: the governance conversation is easier to have before the architecture is finished than after. Retrofitting ownership and lineage rules onto a platform that is already live tends to be considerably more disruptive than agreeing them as part of the initial domain design.

A practical governance artefact that proved useful was a simple domain ownership register: one place that recorded, per domain, the accountable owner, the fields in scope, the system of record for each field, and the approved downstream consumers. This is not a sophisticated tool, and several teams initially treated it as bureaucratic overhead rather than something worth maintaining. Its value became obvious the first time an audit question required tracing a reported figure back to its source under time pressure: the register turned what could have been a multi-day investigation across several system owners into something that could be answered from a single document.

Challenges and lessons learned

It would be misleading to present this delivery as a smooth, linear build, so it is worth setting out honestly where the friction actually sat.

Reconciling ownership before reconciling data

The single biggest source of delay was not technical. It was agreeing who actually owned a given data point once it sat inside a shared domain, rather than inside the system that had historically held it. Customer data, for instance, touched several functions that had each grown used to treating their own copy as the master record. Getting agreement on a single owner per domain took longer than mapping the data itself, and it is a step that is easy to underestimate when planning a similar build. What eventually resolved most of these disputes was not a technical argument but a simple reframing: rather than asking which function should own a domain, the more productive question was which function's definition of the data the rest of the bank could most safely depend on, which usually pointed to a clear answer even when the ownership question itself felt contested.

Legacy systems do not wait politely

Because the legacy estate kept running throughout the build, changes on the old side of the fence — a schema tweak in Debt Manager, say, or a change to how PPE stored transaction status — could ripple into the new domain mappings without much warning. Keeping the two in step required more ongoing coordination with the legacy application owners than the original plan accounted for.

Bureau standards leave less room for interpretation than they first appear to

The outbound data fields required by Experian and TransUnion look, at a glance, like a fixed specification that simply needs to be met. In practice, translating the bank's own internal definitions of account status, delinquency and affordability into the bureaus' expected format threw up more edge cases than expected, particularly around part-way and disputed accounts. Resolving these took close, iterative dialogue with the bureaus rather than a one-off mapping exercise. A disputed account under investigation, for example, sits in a genuinely ambiguous state that the bank's internal systems and the bureaus' reporting categories did not always describe in the same terms, and agreeing a shared interpretation took longer, in a few cases, than building the technical integration itself.

Reusability has to be designed for, not assumed

It is tempting to assume that once one product has been onboarded onto a domain-driven platform, the next one follows automatically. That has not proven entirely true in practice. Reuse only works where the domain was deliberately designed with a second, not-yet-known product in mind, which means resisting the urge to over-fit the initial build to the specific requirements of the first partnership it happens to serve.

Business impact

The commercial case for this kind of investment is straightforward, even if the engineering underneath it is not. By replacing fragmented, manual bureau-reporting processes with standardised, cloud-based integration patterns, the platform reduced the onboarding effort required for new financial products and supported a faster path to launch for the Amazon proposition specifically.

Data quality improved as a direct consequence of the mapping work: standardised handling of customer, debt and transaction data across the Customer System, Debt Manager and PPE reduced reconciliation issues and reporting defects in what was being sent to Experian and TransUnion. That, in turn, strengthened the reliability of the affordability and creditworthiness data feeding into PD, EAD and LGD models, supporting more consistent credit-risk assessment for the new card book.

The wider commercial outcome was access. The scalable, domain-driven NGCB platform gave Barclays a practical route into Amazon's existing customer ecosystem, supporting increased cross-selling opportunity, stronger customer engagement, and higher card adoption and spend. Because the underlying architecture was

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

designed to be reused rather than rebuilt each time, the same foundation also left the platform better placed to onboard further propositions without repeating the heaviest of the integration work.

Synthesis: what the case study suggests more broadly

Read together, the governance, challenges and business-impact findings above point to a single underlying idea: domain-driven architecture is less a technology choice than an ownership and sequencing discipline that happens to be implemented in technology. The technical work of mapping systems into domains was substantial, but it was rarely the reason things slowed down. The delays traced consistently back to unresolved questions of ownership, sequencing and interpretation — who decides, in what order, and against what standard — that no amount of additional engineering effort could shortcut. This has a direct implication for how institutions should staff and sponsor similar programmes: the governance and ownership conversations deserve the same seniority and the same early attention as the architecture design itself, rather than being treated as a downstream administrative task to be resolved once the technical build is under way.

RECOMMENDATIONS FOR INSTITUTIONS PURSUING SIMILAR PARTNERSHIPS

Drawing the threads of the previous sections together, six recommendations stand out as worth taking seriously before committing to a similar partnership.

- Treat the legacy data audit as a distinct, resourced phase of the project, rather than folding it informally into early design work. Underestimating this stage is the single most common reason integration timelines slip, and it is far cheaper to resource properly at the outset than to revisit once development is under way.
- Define domain boundaries around business concepts — customer, transaction, debt, reporting — and hold that line even when it is more convenient to mirror existing system boundaries. The short-term convenience of following legacy lines tends to be repaid with long-term inflexibility.
- Bring regulatory and bureau reporting onto a standardised pattern early, since this tends to be the least negotiable part of the delivery and the one most likely to delay a launch if left until later. Bureau standards, in particular, are worth engaging with iteratively rather than assuming a single mapping exercise will cover every edge case.
- Design inbound data mappings with their downstream model in mind, particularly where the data will feed credit-risk parameters such as PD, EAD and LGD. Accurate data that arrives in the wrong shape for its intended model is not much more useful than inaccurate data.
- Agree data ownership and lineage rules as part of the architecture design, not as a governance exercise bolted on afterwards. Resolving ownership disputes before the domains go live is considerably less disruptive than resolving them once other functions are already depending on the platform.
- Judge the success of the build by whether the next partnership can reuse it, not only by whether the current one launches on time. A platform that only ever serves the first product it was built for has not achieved the reusability a domain-driven approach is meant to deliver.

None of these six recommendations is individually surprising; taken together, however, they amount to a case for sequencing the work differently from how a purely technical delivery plan would naturally order it. Legacy audit and domain definition have to happen before contracts and reporting standards are agreed, which in turn have to be stable before inbound model-ready mappings and reuse can be tested properly. Institutions that compress this sequence to meet a commercial deadline tend to find the compressed steps resurfacing later, in the form of rework, disputed ownership or a platform that turns out to serve only the first partnership it was built for.

FUTURE WORK

The NGCB platform was built with the Amazon proposition as its first product, but not, by design, as its only one. As more banks look to ecosystem partnerships, whether with retail, travel or technology brands, the pressure to onboard those partnerships quickly is only going to increase, and institutions that have already done the harder work of establishing domain-driven, cloud-native foundations will be better placed to respond to that pressure than those still working through it product by product.

There is also a reasonable expectation that regulatory scrutiny of data lineage and governance in banking will keep tightening rather than easing off, given the direction of travel of both UK and EU reporting requirements. A domain-driven architecture that was designed with lineage in mind from the outset, as NGCB's was, is in a considerably stronger position to absorb that scrutiny than an architecture where lineage has to be reconstructed retrospectively.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

None of this suggests the work is ever finished. Each new partnership will surface its own edge cases, its own legacy quirks, and its own governance questions, much as the Amazon proposition did. What a domain-driven foundation offers is not an end to that work, but a considerably better starting point for it each time. Future research could usefully extend this single-case account into a comparative study across multiple institutions and partnership types, to test how far the six-step framework proposed here travels beyond the specific conditions of the NGCB/Amazon delivery.

One direction that seems particularly worth pursuing is how far a domain-driven foundation of this kind can support not just faster onboarding of similar co-branded card products, but genuinely different product types — lending, savings, or embedded insurance offered through the same ecosystem partner, for instance. The customer and transaction domains established for the Amazon proposition were deliberately generic enough that they should, in principle, extend to those adjacent product types without a redesign, but that claim has not yet been tested against a live second product at the time of writing, and doing so would be a natural next step for validating the reusability argument made throughout this paper.

CONCLUSION

The lesson from the NGCB/Amazon delivery is not really about Amazon, or even about co-branded credit cards specifically. It is about what has to be true of a bank's data architecture before it can take on ecosystem partnerships at the pace the market now expects. Disparate legacy systems, manual bureau reporting and inconsistent data mapping are common to most established banks, and they will resurface on the next partnership just as they did on this one, unless the underlying architecture is addressed rather than worked around.

A domain-driven, cloud-native approach will not remove the effort involved in that work, but it does change its shape: from a bespoke integration exercise repeated for every new partner, to a reusable set of domains that each new proposition can plug into. For Barclays, that shift turned the Amazon proposition into more than a single successful launch; it left the bank with a platform genuinely capable of supporting the next one. For architecture, data and technology leaders weighing a similar partnership, the practical starting point is the honest legacy-estate audit described in Step 1 of the framework above: mapping where customer, transaction and default data actually sit today is the piece of work that most reliably determines whether the rest of this framework can be applied, and it can be started well before any partnership agreement is signed.

As with any single-case account, these findings are offered for their analytical transferability rather than statistical generalisability, and readers applying this framework to their own institutions should expect to adapt it to their own legacy estate and regulatory context rather than apply it unchanged.

None of this required exotic technology. What it required was the discipline to define domains around the business rather than the systems, to settle ownership before scale made ownership disputes expensive, and to build regulatory reporting on a standardised pattern before the more visible, customer-facing parts of the partnership took the spotlight. Those are organisational and sequencing disciplines as much as they are architectural ones, and they are available to any institution willing to resource the unglamorous early work — the legacy audit, the ownership register, the data contracts — that this paper has tried to describe in enough detail to be actionable rather than merely aspirational.

ACKNOWLEDGEMENT

The author thanks the engineering, architecture, risk, compliance and commercial colleagues involved in the NGCB/Amazon delivery for the collaborative work this paper draws on. All descriptions are given at a general, de-identified level and do not disclose confidential commercial or customer information.

CONFLICT OF INTEREST

The author declares no conflict of interest.

FUNDING

This research received no external funding and was conducted independently by the author.

DATA AVAILABILITY STATEMENT

The underlying delivery data discussed in this paper is commercially confidential and is not publicly available. This paper presents only de-identified, process-level descriptions of that delivery.

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

REFERENCES

1) Bank for International Settlements / Basel Committee on Banking Supervision. (n.d.). The internal ratings-based approach [Basel II capital framework documentation]. https://www.bis.org/publ/bcbsca05.pdf
2) Banking Industry Architecture Network. (n.d.). Service landscape and reference architecture for the banking industry. https://bian.org
3) Basel Committee on Banking Supervision. (n.d.). Basel II/III internal ratings-based approach framework documentation.
4) Cloud Native Now. (2019). Let's get it right this time: Domain-driven design and microservices. https://cloudnativenow.com
5) European Journal of Computer Science and Information Technology. (2025). Microservices architecture in financial services: Enabling real-time transaction processing and enhanced scalability. https://eajournals.org
6) Federal Deposit Insurance Corporation. (2006). Financial stability and Basel II (FDIC Working Paper).
7) GitHub. (n.d.). Implementing domain-driven design for microservice architecture. https://github.com/ernesen/DDD
8) LearnSignal. (2026). PD, LGD & EAD: The building blocks of credit risk. https://learnsignal.com
9) Office of the Comptroller of the Currency. (2003). Implementation of new Basel Capital Accord.
10) Office of the Superintendent of Financial Institutions (Canada). (2026). Capital adequacy requirements (CAR) 2026 – Chapter 5 – Credit risk – Internal ratings-based approach [drawn from the Basel Framework published by the Bank for International Settlements]. https://osfi-bsif.gc.ca
11) Oracle Corporation. (n.d.). Deploying Oracle banking microservices and OCI Kubernetes engine. https://docs.oracle.com
12) World Journal of Advanced Research and Reviews. (n.d.). Cloud-native microservices in financial services: Architecting for scalability and flexibility. https://wjarr.com
13) XGBoost LGD Study. (2024). Improving realized LGD approximation: A novel framework with XGBoost for handling missing cash-flow data. arXiv. https://arxiv.org
14) Zhao, K., Li, Y., & Chen, X. (2023). Domain-driven design in software development: A systematic literature review on implementation, challenges, and effectiveness [Preprint]. arXiv. https://arxiv.org/abs/2310.01905
15) Zymr. (2026). Microservices in digital banking transformation: Architecture & migration guide. https://zymr.com

APPENDIX A — DOMAIN MAPPING SUMMARY

The table and figure below summarise how the three legacy source systems referenced throughout this paper were mapped into NGCB's domain-driven structure, and which downstream function primarily consumes each domain.

ISSN: 2456-9348 (Online)
SJIF Impact Factor: 8.232

iJETRM
INTERNATIONAL JOURNAL OF ENGINEERING TECHNOLOGY RESEARCH & MANAGEMENT (IJETRM)
https://ijetrm.com/

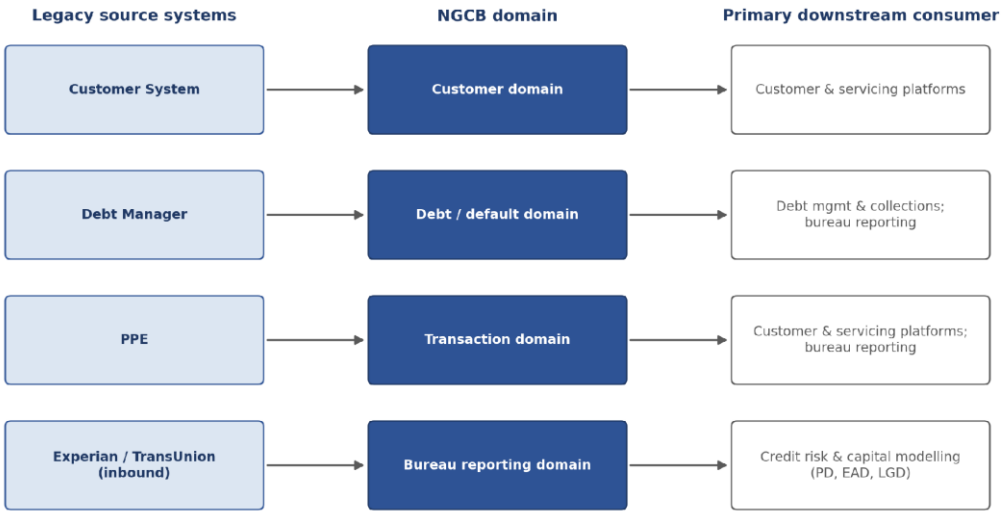

Figure 1. Domain mapping from legacy source systems to NGCB's domain-driven structure and their primary downstream consumers

Figure 1. Domain mapping from legacy source systems to NGCB's domain-driven structure and their primary downstream consumers

Legacy source	NGCB domain	Primary data	Main downstream consumer
Customer System	Customer domain	Personal details, account status	Customer & servicing platforms
Debt Manager	Debt / default domain	Delinquency and default indicators	Debt management & collections; bureau reporting
PPE	Transaction domain	Account balances, payment history	Customer & servicing platforms; bureau reporting
Experian / TransUnion (inbound)	Bureau reporting domain	Affordability & creditworthiness scores	Credit risk & capital modelling (PD, EAD, LGD)

Table 3. Domain mapping summary across the NGCB/Amazon build

