Dec 2017

Model Based Architecture Design

基于模型的架构设计

Ming X Jin 金新明 – Chief Risk Architect

DDD Beijing, 9th Dec 2017

HSBC

PUBLIC

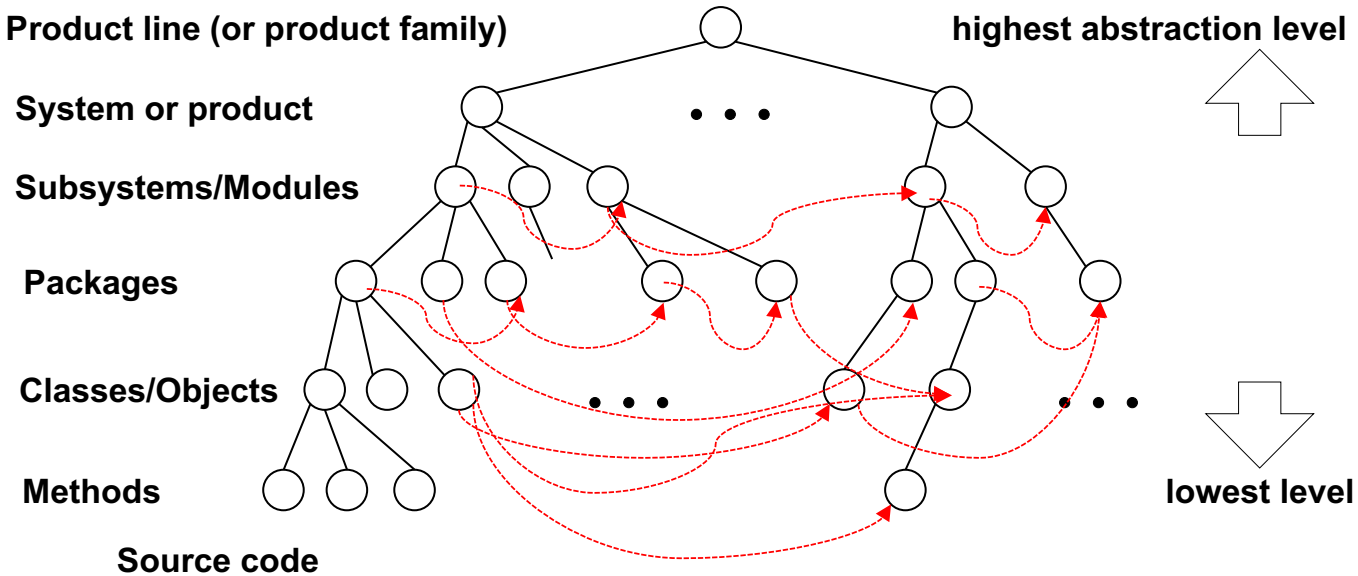
About me …

- Name: Ming X Jin 金新明 Current Role: Chief Risk Architect in HSBC
- Education Background: Automatic Control, Computer Science (Ph.D.)
- PhD topic: Model based business process simulation
- First Job: Cobol programmer for UnionPay POS integration
- Multi Industry Sector experience: Academic, Consultancy, Defense, Manufacturing, Banking
- Multi Architect Roles:
  - Systems Architect – Thales Group (4 years)
  - Technology Architect – Infosys (1 year)
  - Data Architect – RBS (6-month)
  - Process Architect – Barclays Capital (2 years)
  - Solution Architect – RBS (6-month), Barclays Capital (1 year), Standard Chartered (14-month)
  - Enterprise Architect – NOW

Ubiquitous Models

- Model is always there, intentionally or not intentionally;
- Model has a wide context, but a specific purpose;
- There is a right model but no complete model;
- There is no universal model but a suitable model;
- Code alignment with models determines the code quality;

Hierarchical Organisation of Software & Complexity



- Application level complexity
  - Complexity of all interfaces that application uses
  - Complexity of all interfaces that application provides
  - Complexity of application source code and data
- Enterprise level complexity
  - Complexity of overall performance optimisation
  - Complexity of integration
  - Complexity of data management
  - Complexity of business/technology alignment

System Decomposition & Domain Driven Design

Why decomposing systems?

- Tackle complexity by “divide and conquer”
- See if some parts already exist & can be reused
- Focus on creative parts and avoid reinventing the wheel
- Support flexibility and future evolution by decoupling unrelated parts, so each can evolve separately (“separation of concerns”)

Why Domain Driven Design?

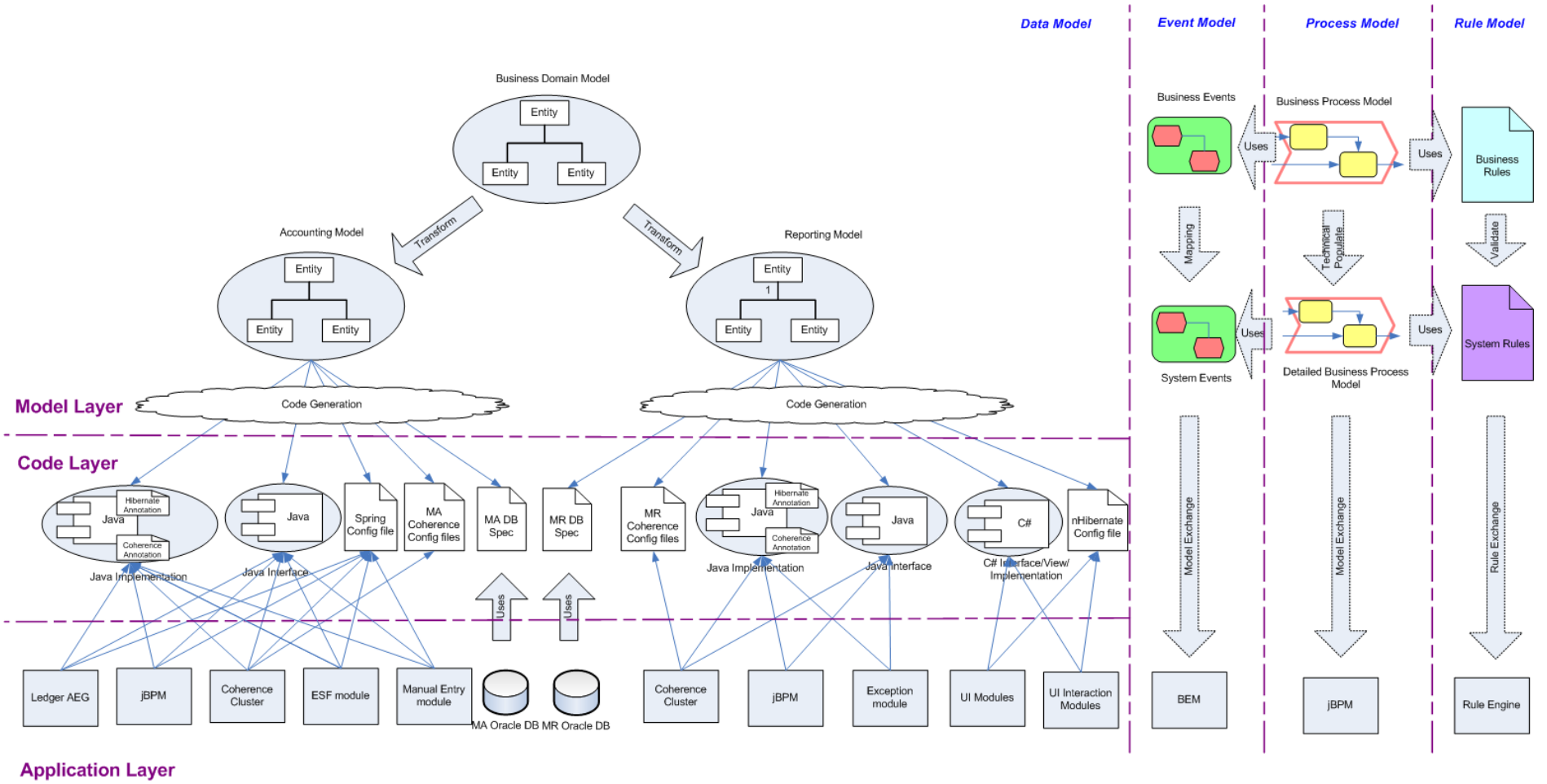
- Disciplined modelling approach
- Bounded context
- Ubiquitous language for easy communication
- Focusing on domain and domain logic instead of interfaces

ONLY for Complexity

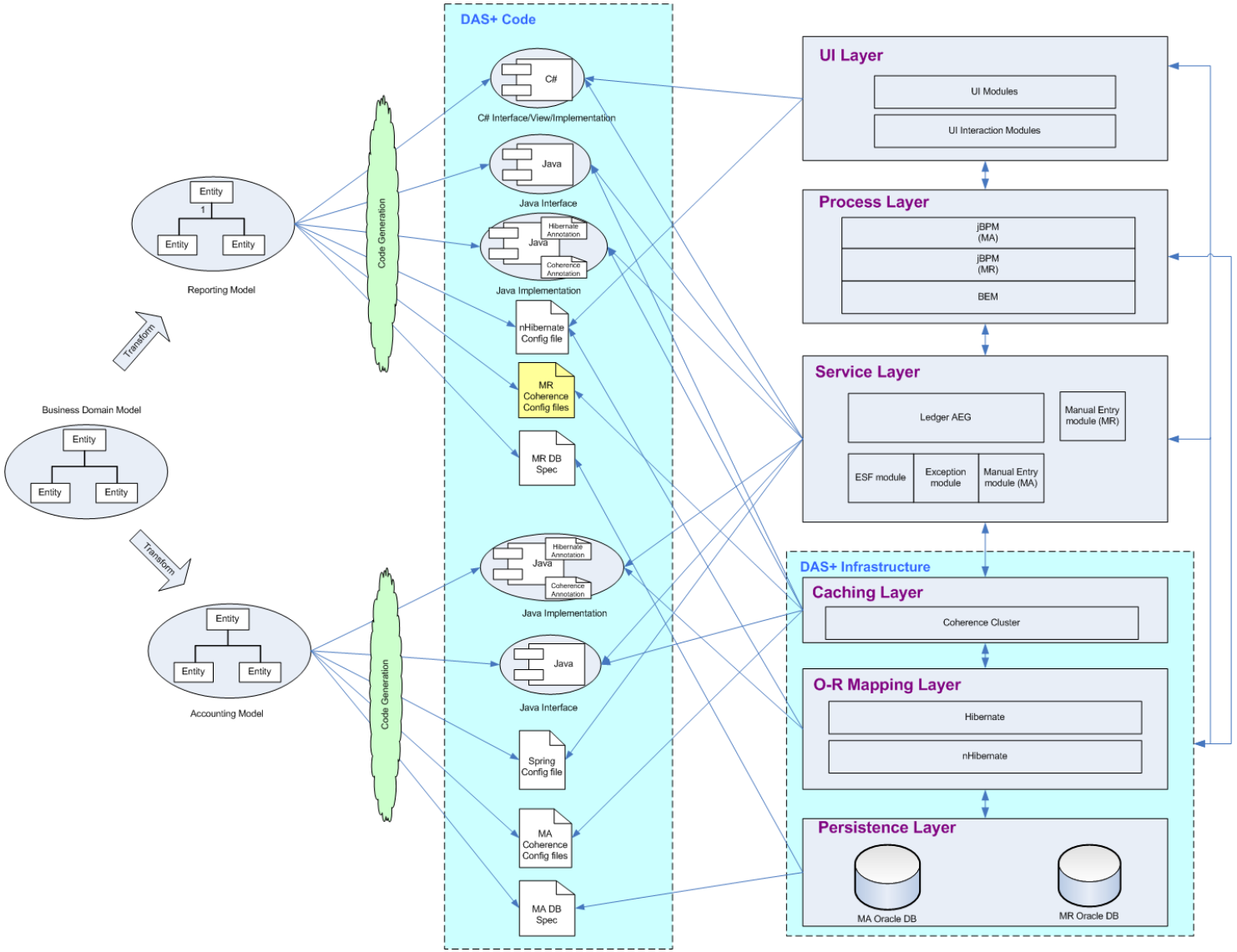
Project Example: Zachman Framework Based Modelling

	Data (What) (Structural)	Process (How) (Dynamic)	Location (Where)	People and Organisations (Who)	Events (When)	Motivation (Why)
Strategic View **(Scope and User Requirements)**	List of things important to the programme/ strategic initiative/ enterprise system (Identifies the Business Domain Objects); Glossary	Lists, or diagrams, of Business Capabilities and their interdependencies.	List of locations where the enterprise operates	List of organisational units, partners, suppliers etc. Org chart, with roles; skill sets, etc	List of business events / cycles	Overarching Strategic Requirements including Business goals, strategies, and regulatory constraints
Business Process View	Business Domain Model (business objects, attributes and relationships) Business States Model showing states of key objects	Business Use Cases; Business Process Models (BPMN); Business Context Diagrams	Business Distribution Model. Business locations.	Business Participation (or Role) Model (customers of, and workers in, the major activities including participating organisations)	Business Event Catalogue; Business Exception Catalogue; Business Event Model (BEMN)	Business Requirements including Functional Requirements, Non-Functional Requirements and Business Rules
Functional Specification View	Mapping of Business Domain Model to Logical Data Model. System State-charts	System Use Cases; Business Process Models (Executable BPMN); System Context Diagram	System Communications Model. How business locations communicate with each other	List of System Actors; Use Case Context Diagram to show actors access to use cases	System Event Catalogue; System Exception Catalogue; System Event Model (BEMN)	System Requirements including Functional Requirements, Non-Functional Requirements and Rules
Platform Independent Models View	Architectural descriptions (e.g. Presentation, Control, Delegation, Entity, Service Components and Interfaces including message structure); Logical data model with normalisation, generalisation etc.	Component Interaction Diagrams; Service Choreography Models	System architecture (hardware, network, software, data)	User interface design. Web page wireframes and layouts (how the presentation layer will behave)	Major events (timing for batch process, system interactions, consolidation, up/downloads etc)	Business rule design (Depending on Rule Engine capabilities)
Platform Specific Models View	Platform architecture models; Data schemas and object definitions; Mapping to legacy data; Message definitions	Detailed Program Design. Language/Platform applications and source code.	Data Centre architecture Hosting services Communications facilities	Screen designs, XAML/ASP/JSP security architecture (who can see what?)	Run books Timing definitions	Rulesets for specific rule engines; Rule specification in program logic

Project Example: Various Model Usage



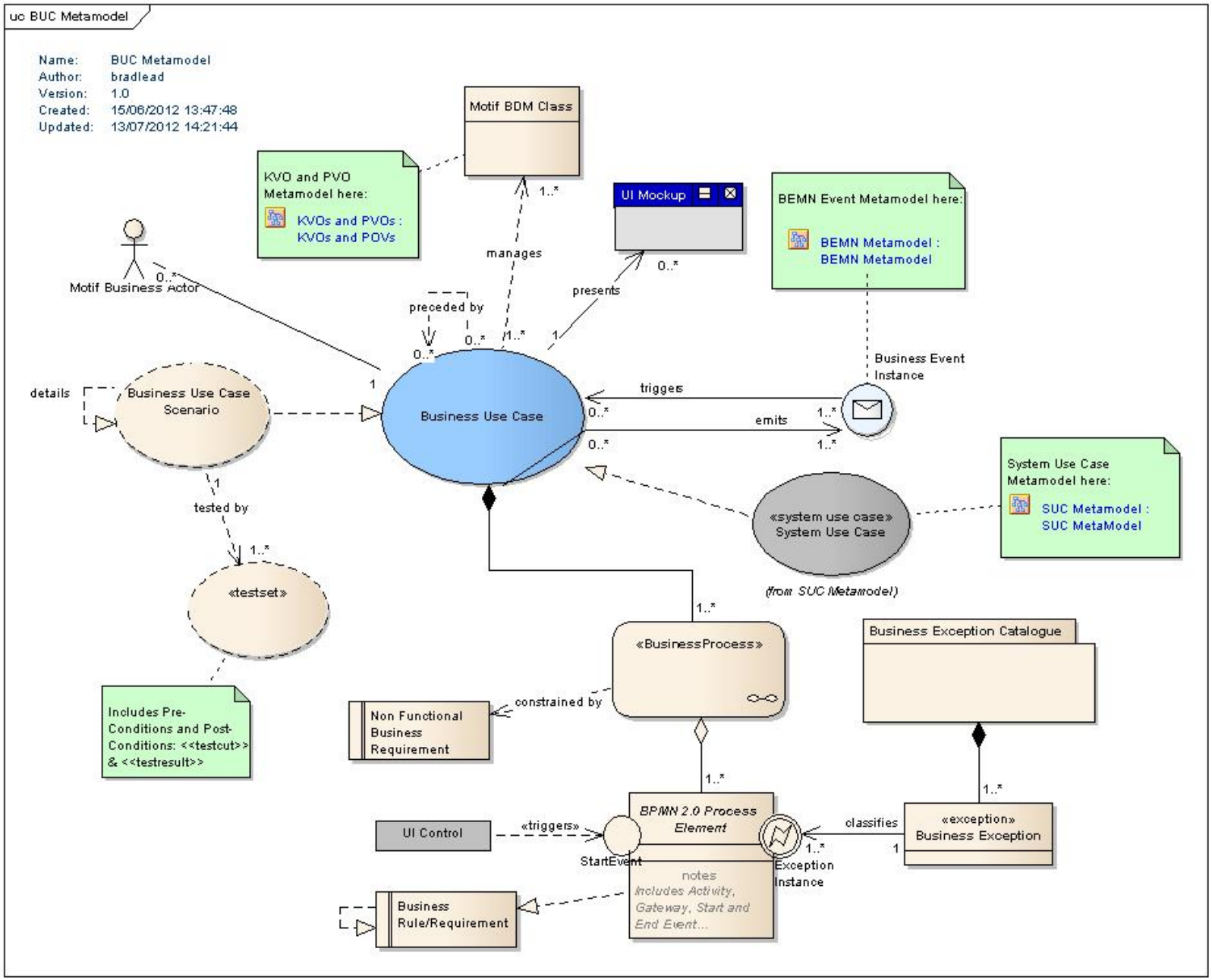
Project Example: Model Based Code Generation



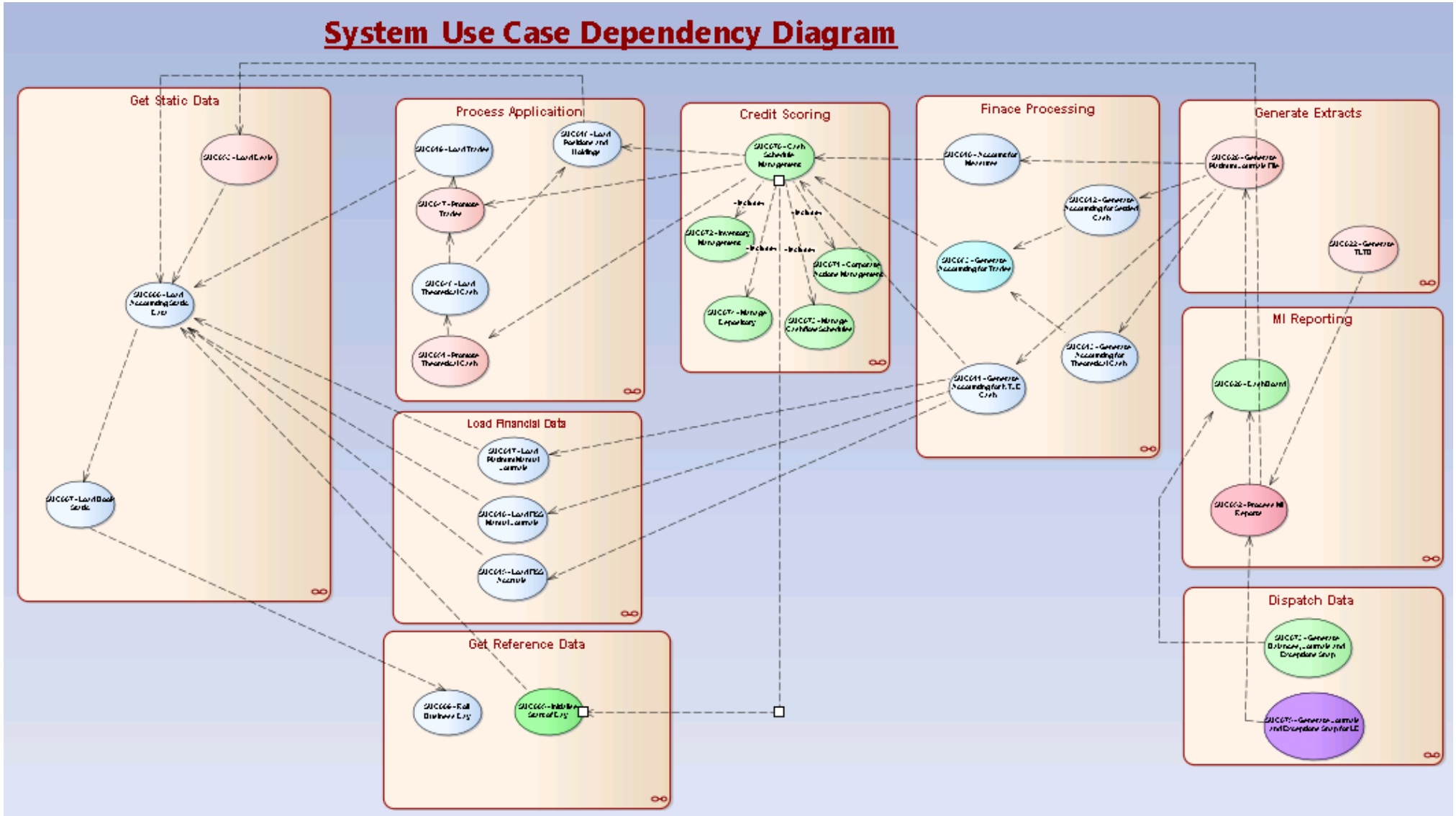
Project Example: What & How delivered?

	Data (What)	Events (When)	Process (How)	Drivers (Why)	Resource Type	% of Overall Effort
Business Requirements (Level 2)	Business Domain Model (BDM)	Life Cycle Events	Business Process Model (BPMN)	Business Requirements	Business Analyst	30%
Functional Requirements (Level 3)	System Domain Model (SDM)	System Events (BEMN)	Functional Process Model (BPMN)	System Requirements	Systems Analyst	30%
Platform Independent Model (Level 4)	Data Object Model	Event Execution Language (EVEL)	Enriched Business Processes	Formalised Rules	Solution Architect	15%
Platform Specific Model (Level 5)	Platform Specific Generated Code	Platform Specific Generated Code	Platform Specific Generated Code	Platform Specific Generated Rules	Model Developer	5%
Code (Level 6)	DAS+	BEM	jBPM	Hand-Built Application Code	Developer	20%

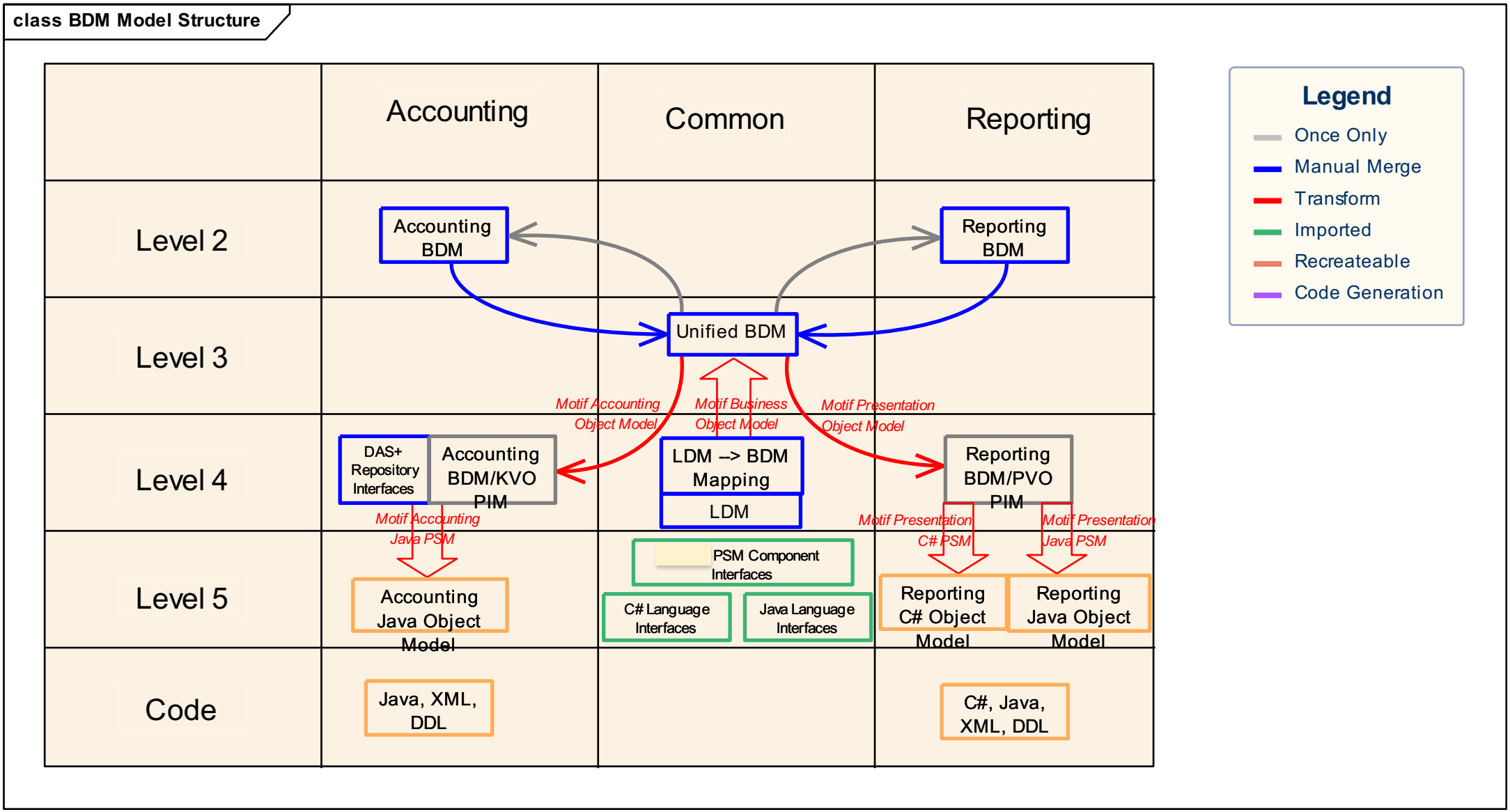
Project Example:
Defining Context - Business Use Case Meta Model



Project Example:
Defining Context - Use Case Interaction Model



Project Example: Model Transformation



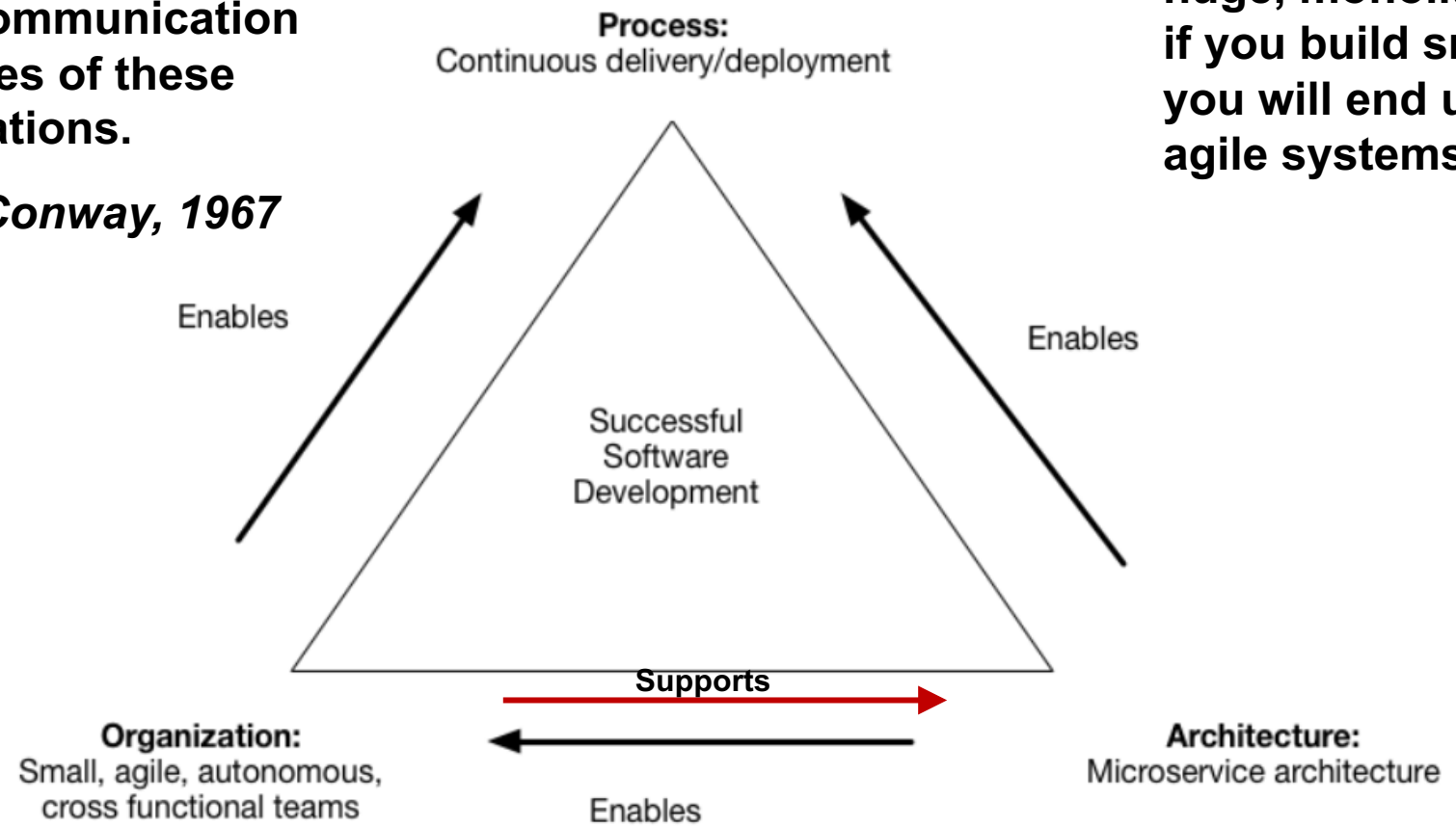
Project Example:
Model Based Design - Cons Vs. Pros

- **Pros:**
  - Separate concerns at different levels from different views;
  - Provide a single version of truth for all design artefacts across different teams and releases in the project life cycle;
  - Ensure a holistic, consistent, and integrated Model;
  - Build traceability between various artefacts of the development process from high level business requirements to detailed implementation further extend to testing;
  - Reusable and extensible;
  - Supports all development teams with different artefacts originated from the same source model.
- **Cons:**
  - Long release cycle;
  - Model becomes the bottleneck for fast delivery;
  - High front investment on modelling;
  - Need domain experts;
  - Isolation between business and dev teams.

What would I do differently ? - Conway's Law

Organizations which design systems . . . are constrained to produce designs which are copies of the communication structures of these organizations.

Melvin Conway, 1967

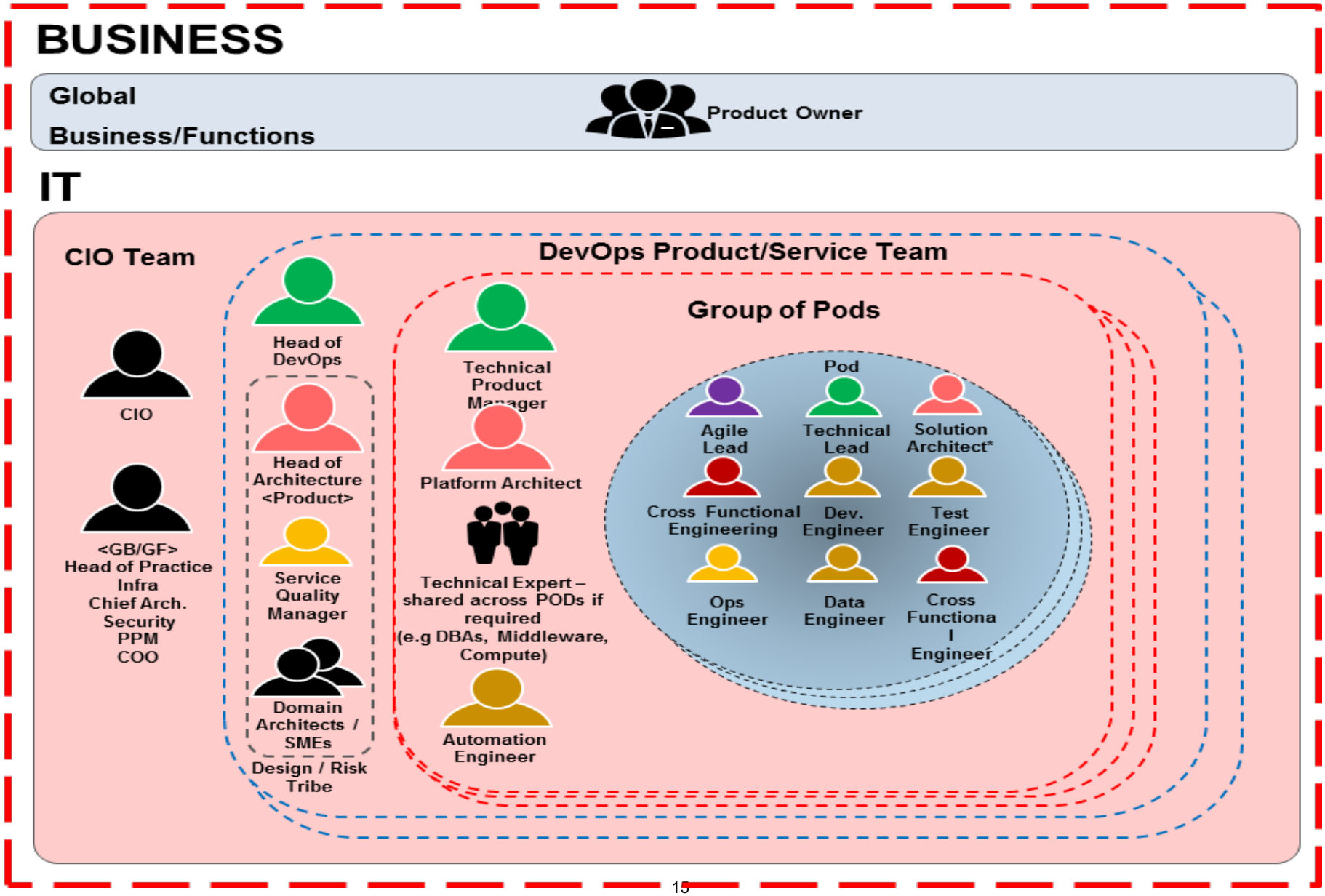


OR

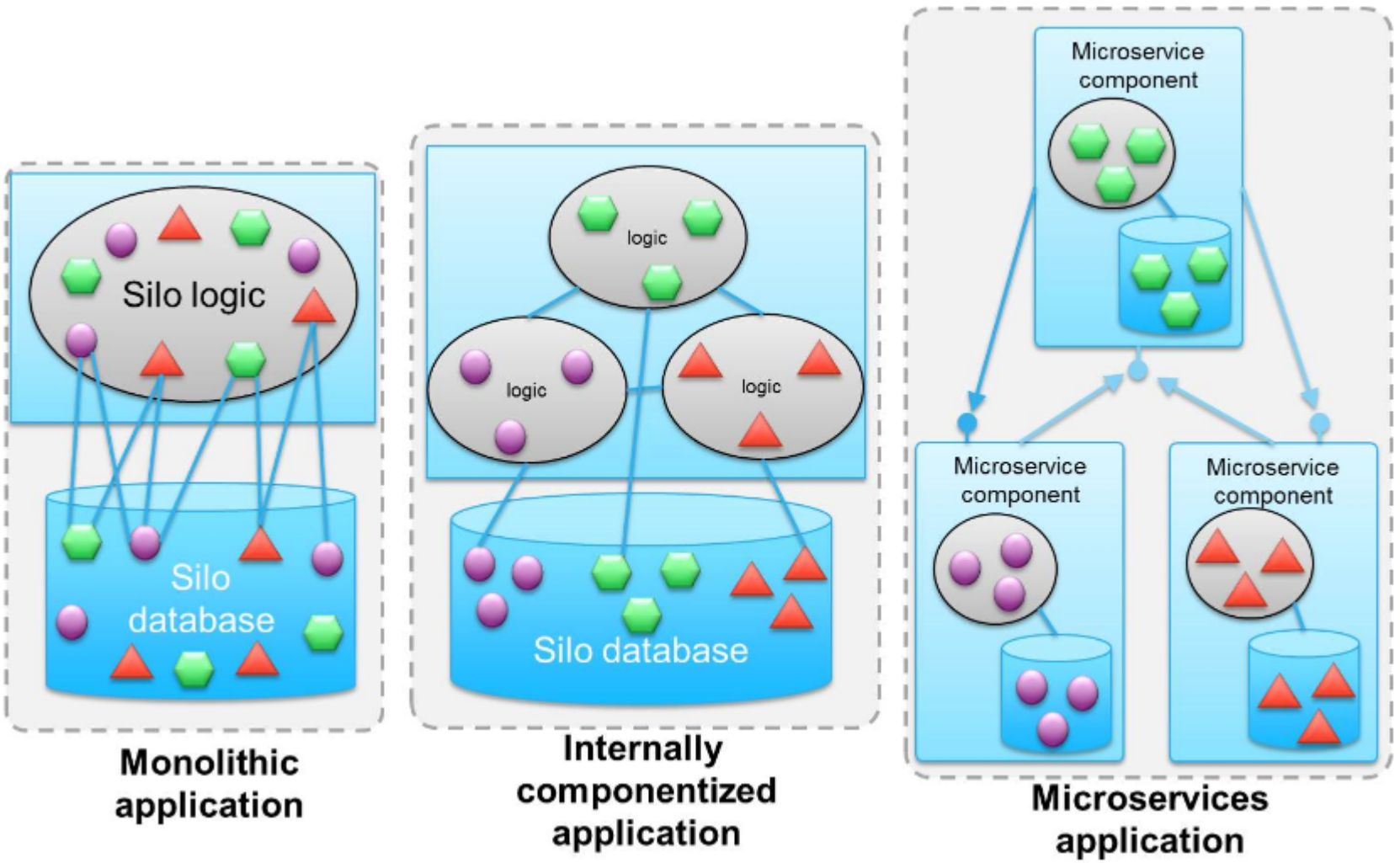
If you build huge, monolithic teams, you will end up with huge, monolithic systems, but if you build small, agile teams, you will end up with small, agile systems.

Source: http://microservices.io/patterns/decomposition/decompose-by-business-capability.html

What would do differently ? – Move to DevOps Model



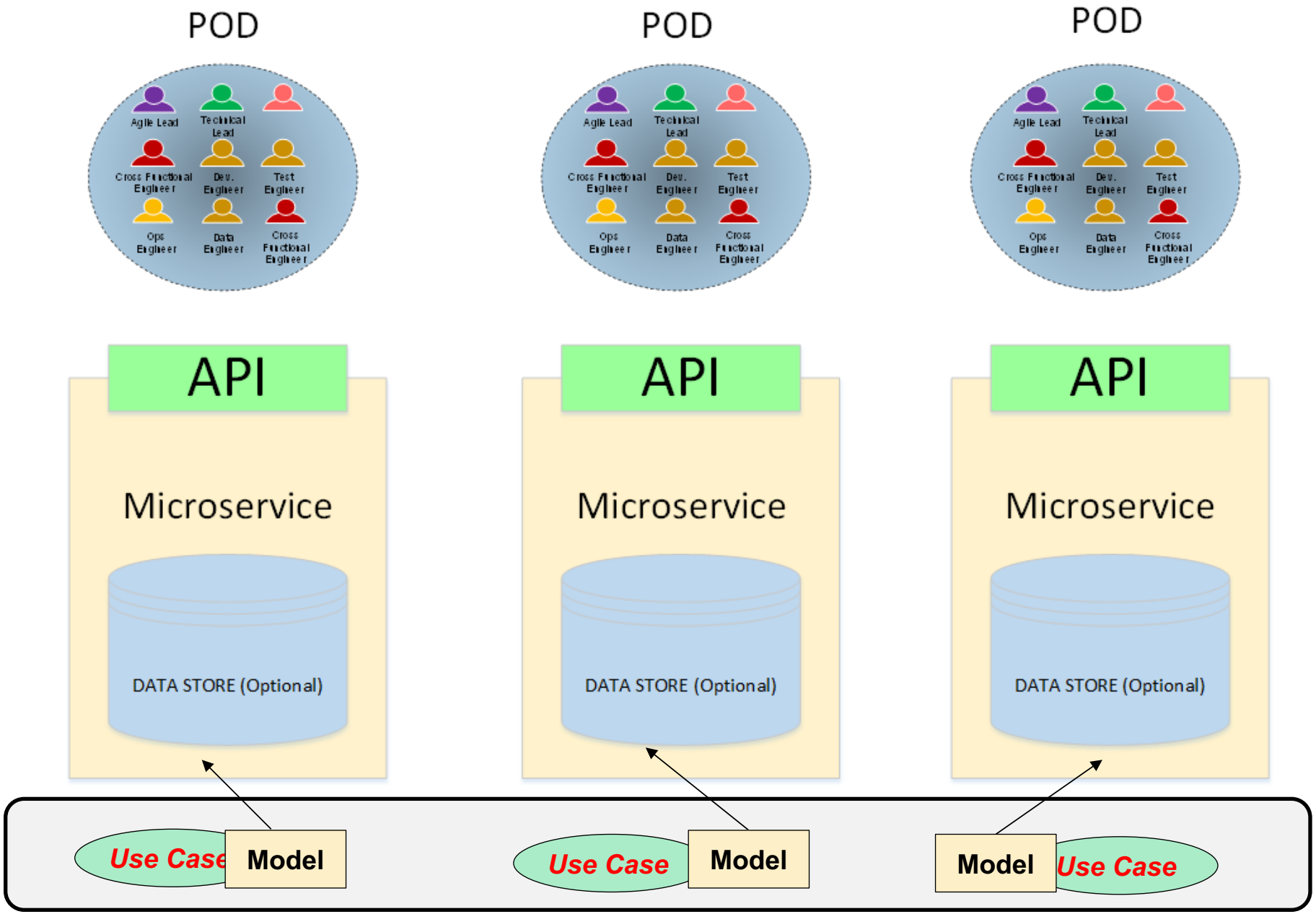
What would do differently ? – Move to Microservices (1)



Source: https://www.ibm.com/developerworks/websphere/library/techarticles/1601_clark-trs/1601_clark.html

What would do differently ? – Move to Microservices (2)

Enabling Pods to deliver business value independently



Looking Forward – HSBC's Vision and How DDD May Help

- Our Group CTO has got a vision of 100% API, 100% Cloud and 100% Services;
- The complexity of system landscape incorporated with domain knowledge in HSBC needs a full picture and understanding to allow fast changes;
- We are adopting DevOps to compete with FinTech and InsureTech firms who are constantly releasing new and great products. DDD goes hand by hand with DevOps;
- Microservices are best used in a Cloud architecture, where they can scale horizontally. However, we need start from the models instead of services;

Thank you!