Title:

Model Version Management Benchmark: Questionnaire for Assessing MVMS Capabilities

A Structured Assessment Based on Functional Features of Model Versioning and Branching

Author:

[redacted]

[redacted]

[redacted]

Abstract:

This document presents a structured questionnaire designed to evaluate the functional capabilities of Model Version Management Systems (MVMSs). Inspired by the principles of Function Point Analysis, the questionnaire assesses 13 key features of MVMSs, including artifact representation, versioned storage, change detection, indexing, versioning mechanisms, locking, versioning architecture, collaboration type, and branching characteristics. Each question includes scoring guidelines to quantify system capabilities, enabling technology-independent benchmarking and comparison across collaborative modeling environments.

Document Purpose:

This questionnaire is intended to:

1. Provide a systematic means of evaluating MVMS functionality.
2. Enable scoring of features to compute the Model Version Management Capability (MVM-Capability) metric.
3. Serve as a reproducible tool for researchers and practitioners assessing collaborative modeling systems.

Usage Instructions:

- Each question includes scoring guidelines with a maximum score of 5 points.
- Scores are assigned based on the presence, type, and combination of features supported by the system.
- Questions marked as Alt represent mutually exclusive alternatives; Or indicates features that may be combined.
- Results can be used to compute the Unadjusted and Adjusted MVM-Capability as defined in the benchmark methodology.

Question 1: Artifact Representation in Versioning

Question:
Which type of artifact representation does the system use for versioning? Choose only one option.

Scoring Guidelines:

Score	Description
0	The system does not support artifact versioning.
2	File-based versioning only (syntactic representation, serialized data; basic integration with tools like Git; no semantic understanding).
5	Model-based versioning only (semantic-aware, fine-grained change control, conflict detection, and resolution; tailored to modeling domain).

Rationale:

- **Model-based versioning** provides higher capability for collaborative modeling due to semantic awareness and domain-specific operations.
- **File-based versioning** is simpler, focusing on syntactic changes, with basic repository support.
- Since this is an **Alt group**, the system cannot support both types simultaneously; scoring reflects the relative capability.

Question 2: Artifact Storage Strategy

Question:
Which versioned storage strategies does the system support for archiving artifact versions?

Scoring Guidelines:

Score	Description
0	No versioned storage supported.
2	Supports snapshots only (entire model saved per commit; simple but storage-intensive).
3	Supports deltas only (records only differences between versions; efficient storage).
5	Supports both snapshots and deltas (flexible storage strategy combining simplicity and efficiency).

Rationale:

- **Snapshots** provide complete historical states but consume more storage.
- **Deltas** are storage-efficient and track fine-grained changes.
- Supporting **both** gives maximum flexibility, balancing storage efficiency with historical completeness.

Question 3: Change Detection Strategy

Question:
Which change detection strategy does the system employ to manage versioning?

Scoring Guidelines:

Score	Description
0	No change detection mechanism supported.
2	Log-based detection only (records discrete operations; fine-grained edits; detailed tracking; precise conflict resolution).
3	Comparison-based detection only (state-based; identifies changes by comparing full snapshots; simpler retrieval).
5	Both log-based and comparison-based detection supported (combines fine-grained tracking with simplified retrieval).

Rationale:

- **Log-based** allows precise conflict resolution and detailed change tracking.
- **Comparison-based** simplifies version retrieval and is easier to implement.
- Supporting **both strategies** provides maximal flexibility and capability in collaborative version management.

Question 4: Indexing Strategy for Versioned Models

Question:
Which indexing strategy does the system use to organize versioned models? Choose only one option.

Scoring Guidelines:

Score	Description
0	No indexing mechanism provided.
1	Unstructured indexing (simple text-based listing of versions; easy to implement but limited retrieval and scalability).
2	Table-based indexing (structured, records version identifiers and relationships; moderate retrieval efficiency).
3	Tree-based indexing (hierarchical organization; supports parent-child relationships and branching).
4	Graph-based indexing (flexible structure; captures complex version dependencies, merges, and alternative development paths).
5	Custom indexing (innovative approaches, e.g., blockchain-based, ensuring immutability, traceability, and advanced retrieval capabilities).

Rationale:

- **Unstructured indexing** is simple but lacks scalability and efficiency.
- **Structured methods** (table, tree, graph) improve retrieval, traceability, and management of complex version histories.
- **Custom indexing** offers maximal flexibility, security, and advanced features tailored to collaborative modeling environments.
- Since this is an **Alt group**, only one indexing method is chosen, so scoring reflects the **relative capability** of the chosen approach.

Question 5: Versioning Mechanism

Question:
Which versioning mechanism does the system employ to track the evolution of models? Choose only one option.

Scoring Guidelines:

Score	Description
0	No versioning mechanism is provided.
2	External versioning only (relies on existing version control systems, e.g., Git or file-based systems; may track models as files without model-specific semantic support).
5	Internal versioning only (custom mechanism integrated with the modeling domain; supports model-specific concepts, semantic-aware version tracking, and enhanced operations).

Rationale:

- **External versioning** leverages existing tools but may be limited to file-based operations and less semantic awareness.
- **Internal versioning** provides maximal capability, with tight integration to the modeling domain and semantic tracking.
- Since this is an **Alt group**, only one mechanism is selected, and scoring reflects the **relative functional capability** of that choice.

Question 6: Locking Strategy

Question:
Which locking mechanism(s) does the system provide to manage concurrent edits on models?

Scoring Guidelines:

Score	Description
0	No locking mechanism supported; concurrent edits may cause conflicts.
2	Pessimistic locking only (restricts multiple users from editing simultaneously; prevents overwrites but reduces collaboration flexibility).
3	Optimistic locking only (allows multiple users to edit simultaneously; conflicts resolved at commit time; higher collaboration flexibility).
5	Both pessimistic and optimistic locking supported (flexible concurrency control; supports strict conflict prevention and collaborative freedom).

Rationale:

- **Pessimistic locking** prevents accidental overwrites but limits parallel work.
- **Optimistic locking** allows concurrent work but may require conflict resolution.
- Supporting **both mechanisms** maximizes collaborative capability and flexibility in different team workflows.

Question 7: Versioning Architecture

Question:
Which versioning architecture does the system use to manage artifact versions? Choose only one option.

Scoring Guidelines:

Score	Description
0	No versioning architecture supported.
2	Centralized architecture (single master repository; simpler management but limited flexibility and scalability).
5	Distributed architecture (each user maintains a local repository; flexible, resilient, supports parallel collaboration and decentralized version history).

Rationale:

- **Centralized architecture** is simpler to manage but restricts collaboration and scalability.
- **Distributed architecture** provides maximal flexibility, resilience, and parallel collaboration, making it better suited for modern collaborative modeling.
- Since this is an **Alt group**, only one architecture is selected, so scoring reflects the **relative functional capability**.

Question 8: Collaboration Type

Question:
Which type of collaboration does the system support for model version management? Choose only one option.

Scoring Guidelines:

Score	Description
0	No collaborative support provided.
2	Offline (non-real-time) collaboration only (changes are persisted locally; updates visible after commit/synchronization; low interactivity).
4	Real-time collaboration only (simultaneous editing; changes visible immediately; high interactivity and team responsiveness).
5	Hybrid collaboration (users can switch between offline and real-time modes; maximizes flexibility and team efficiency).

Rationale:

- **Offline collaboration** provides basic support but has delayed synchronization.
- **Real-time collaboration** allows immediate feedback and interactive teamwork.
- **Hybrid collaboration** provides the best flexibility, enabling users to choose the mode based on workflow needs.
- As this is an **Alt group**, only one option is selected, so the score reflects the **relative capability** for collaboration.

Question 9: Branching Multiplicity

Question:
Which branching multiplicity does the system support for managing model versions?

Scoring Guidelines:

Score	Description
0	No branching supported.
2	Single branching only (all branches originate from the mainline; simple linear model; limited flexibility).
5	Multi-branching supported (branches can originate from the mainline or existing branches; supports hierarchical development and parallel workflows).

Rationale:

- **Single branching** simplifies version control but limits parallel development and experimentation.
- **Multi-branching** allows flexible development workflows, hierarchical structures, and better support for collaborative teams.
- Since this is an **Or group**, the system may technically support multiple modes, but scoring emphasizes the **maximum branching flexibility** provided.

Question 10: Branching Mechanism

Question:
Which branching mechanism does the system use to manage branches of model versions?

Scoring Guidelines:

Score	Description
0	No branching mechanism supported.
2	Implicit branching only (branches emerge naturally by duplicating data; flexible but less organized; may increase storage requirements).
4	Explicit branching only (branches formally defined using labels or metadata; provides clear structure and improved traceability).
5	Both implicit and explicit branching supported (maximizes flexibility, clarity, and traceability for complex workflows).

Rationale:

- **Implicit branching** is easy to create but can be disorganized and storage-heavy.
- **Explicit branching** improves traceability, structure, and management of branches.
- Supporting **both** provides the highest capability for collaborative model versioning and workflow flexibility.

Question 11: Branching Accessibility

Question:
Which accessibility levels does the system provide for branches?

Scoring Guidelines:

Score	Description
0	No accessibility control; all branches are equally accessible.
2	Private branches only (restricted to creator; limits collaboration).
3	Public branches only (accessible by all team members; supports open collaboration but lacks control).
4	Protected branches only (restricted access for modifications; ensures stability but limits flexibility).
5	Combination of private, public, and protected branches supported (flexible collaboration with fine-grained access control and stability management).

Rationale:

- **Private branches** support independent development but limit teamwork.
- **Public branches** enable open collaboration.
- **Protected branches** ensure controlled modifications for stability.
- Supporting **all three types** maximizes **flexibility, collaboration, and control**, aligning with best practices in collaborative model versioning.

Question 12: Branching Pattern

Question:
Which branching pattern does the system support for model version management?

Scoring Guidelines:

Score	Description
0	No branching pattern supported.
2	Short-lived branching only (temporary feature branches merged quickly; ideal for small, incremental updates; minimizes conflicts but limited for large-scale or long-term development).
3	Long-lived branching only (branches remain independent for extended periods; supports complex features and distributed teams; requires frequent synchronization).
5	Both short-lived and long-lived branching supported (maximizes flexibility, supports parallel development, large-scale features, and efficient collaboration).

Rationale:

- **Short-lived branches** improve agility and minimize conflicts for small, incremental changes.
- **Long-lived branches** provide flexibility for complex or large-scale development but may introduce integration challenges.
- Supporting **both** ensures the system can adapt to different workflows, team sizes, and collaboration needs, maximizing MVM-Capability.

Question 13: Branching Scope

Question:
Which branching scope does the system support for managing model versions?

Scoring Guidelines:

Score	Description
0	No branching scope supported.
2	Partial branching only (selective branching of specific components; efficient for multi-view modeling; reduces storage and processing overhead; limited traceability).
3	Full branching only (all elements of a version are branched; ensures complete traceability and consistency; higher storage/processing cost).
5	Both full and partial branching supported (maximizes flexibility; allows targeted modifications without affecting entire versions; supports multi-view modeling effectively).

Rationale:

- **Partial branching** reduces overhead and supports selective modifications.
- **Full branching** ensures comprehensive traceability and rollback capabilities.
- Supporting **both** maximizes flexibility, efficiency, and collaboration in multi-view or complex modeling scenarios.