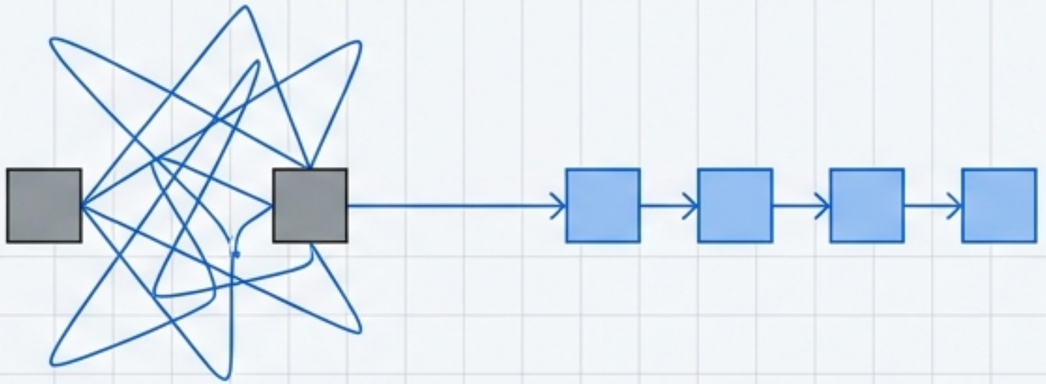
Refactoring the Export Service Architecture

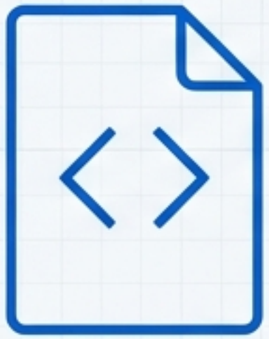
A Blueprint for a Modular, Extensible, and Maintainable System



Project Version: v1.4.0 | Based on Brief v1.3.2 | Date: December 2025

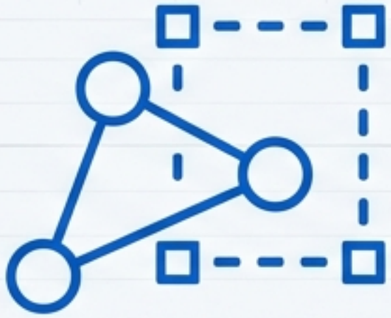
Our Objective: Achieve a Clean Separation of Concerns

The current `Html_Graph__Export__Service` has grown organically and now mixes several distinct responsibilities. This refactoring will isolate these functions into a clear, logical pipeline. Our goal is to decouple four key stages:



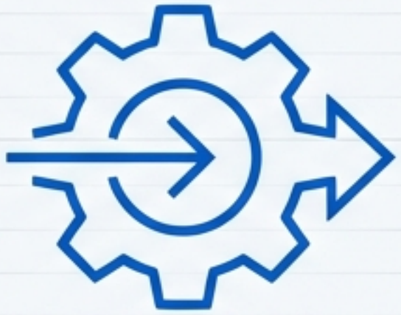
1. HTML Parsing

The creation of the initial `Html_MGraph`.



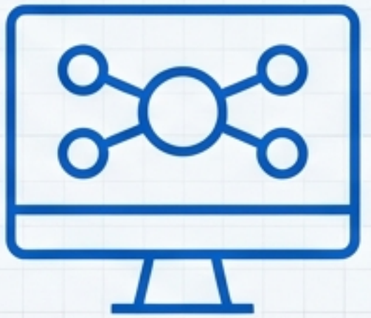
2. Graph Selection/Assembly

Choosing the correct source graph for rendering.



3. Graph Transformation

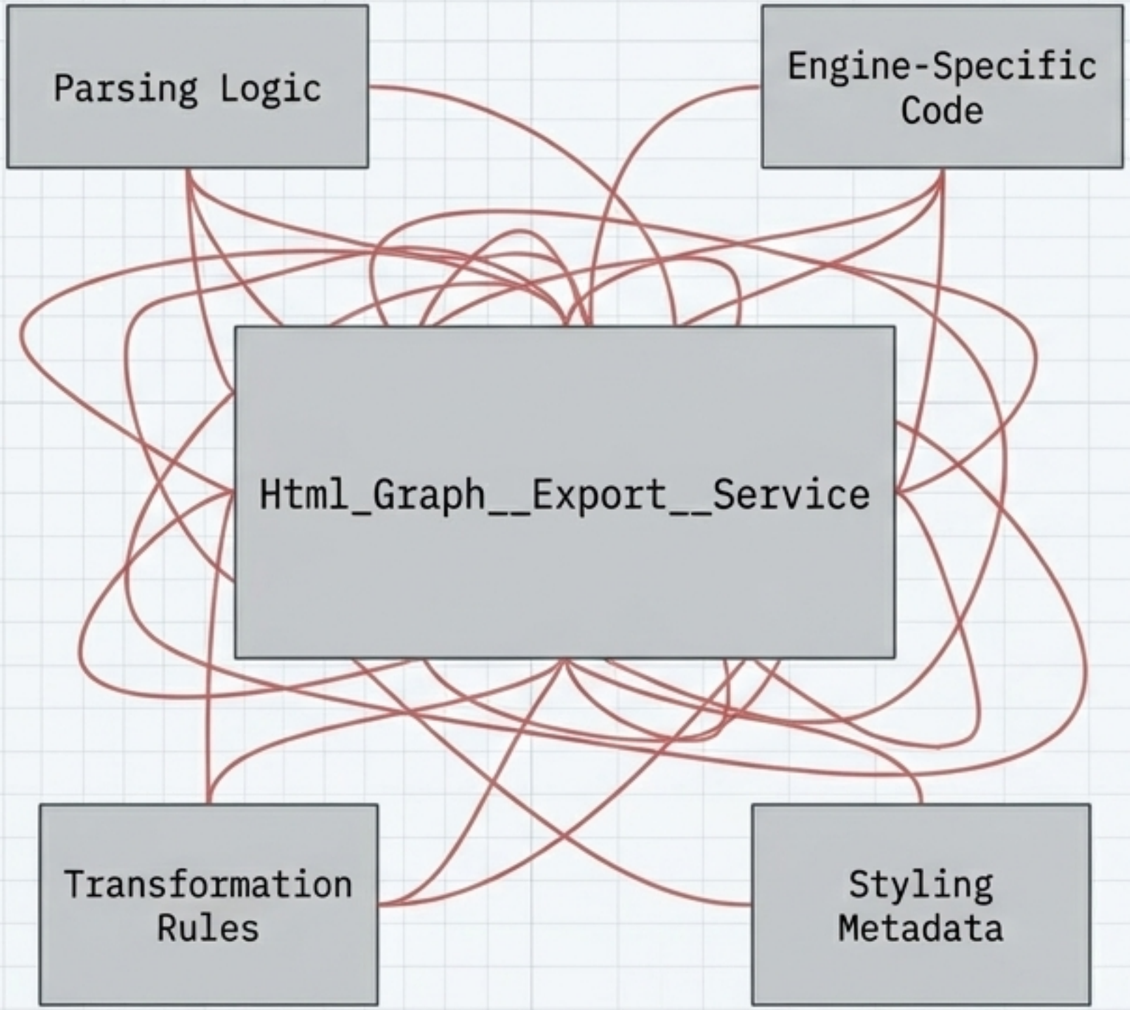
Applying filtering, styling, and structural changes.



4. Engine Rendering

Generating the final, format-specific output.

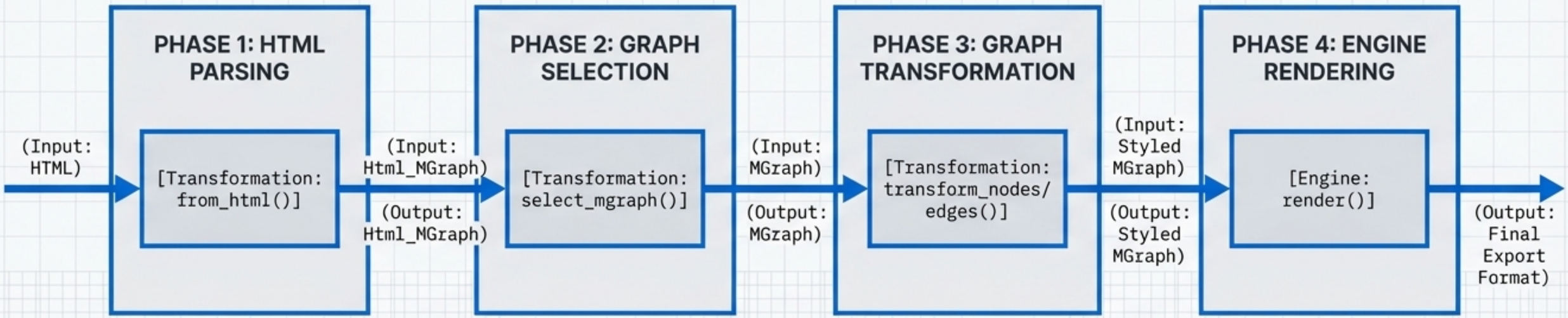
The Current Architecture Has Reached Its Limits



We have identified five core issues that inhibit scalability and maintainability:

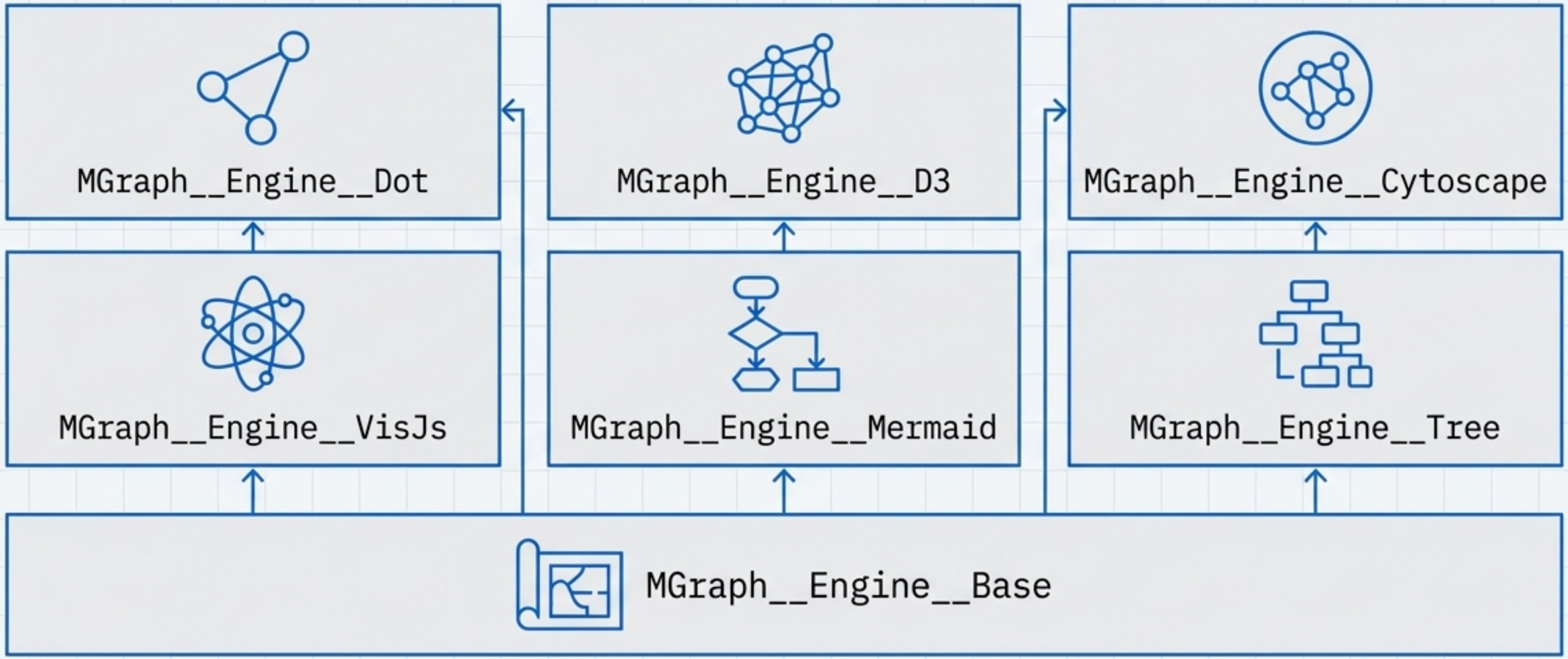
1. **Incorrect Return Types**: Transformations return `Html_MGraph`, but engines need the simpler `MGraph`.
2. **Limited Transformation Control**: Parsing happens *before** a transformation is called, preventing customisation.
3. **Scattered Engine Logic**: Duplicated rendering code across multiple `Html_MGraph__To__*` classes.
4. **No Engine Configuration**: Transformations cannot influence engine-level settings like `rankdir` or `splines`.
5. **Ambiguous Styling Pipeline**: It's unclear when and how styling metadata from `Html_MGraph__Data__Extractor` is applied.

The New Blueprint: **A 4-Phase Export Pipeline**



A New Toolkit of Rendering Engines

We are replacing scattered rendering logic with a dedicated, object-oriented engine layer. All engines inherit from a common `MGraph__Engine__Base` and are located in `service/mgraph__engines/`. This provides a consistent interface and isolates format-specific logic.



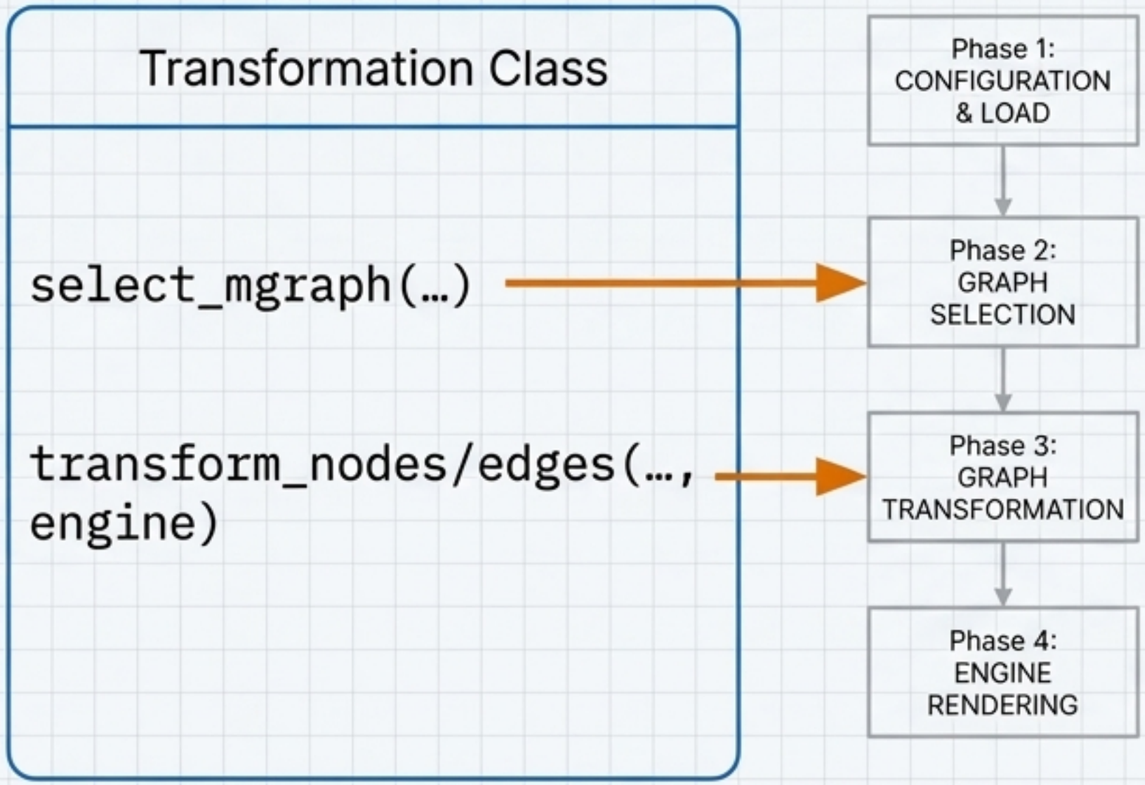
The Engine Contract: Typed Configs and Standardised Outputs

Each engine is paired with a typed configuration class, allowing transformations to influence rendering in a structured way. The pattern established by the D3 engine is followed by most others.

Engine	Config Class	Export Return Type	Key Config Fields
Cytoscape	`MGraph__Engine__Config__Cytoscape`	`dict` (elements.nodes/edges)	`layout`, `spacing`, `fit`
VisJs	`MGraph__Engine__Config__VisJs`	`dict` (nodes/edges)	`physics`, `layout`, `interaction`
Mermaid	`MGraph__Engine__Config__Mermaid`	`str` (flowchart syntax)	`direction`, `theme`
Tree	`MGraph__Engine__Config__Tree`	`dict` or `str`	`indent`, `format`, `max_depth`

Evolving Transformations for the New Pipeline

Existing transformations will be refactored to align with the new pipeline. Instead of a single, monolithic method, they will use dedicated methods and per-engine callbacks to operate at the correct stage.



Transformation	Key Phase & Action
`Base / Body_Only`	**Phase 2**: Select `body_graph.mgraph`
`Structure_Only`	**Phase 3**: Filter to element nodes only
`Attributes_View`	**Phase 2**: Select `attrs_graph.mgraph`
`Clean`	**Phase 3**: Filter tag/attr nodes
`Semantic`	**Phase 3**: Collapse single children, merge text

Transformation in Practice: From Post-Processing to Direct Control

The new design enables transformations to apply engine-specific styling. For example, `Graph_Transform__Structure_Only` can now directly set attributes for the DOT engine.

BEFORE: Limited Control

```
# old Graph_Transform__Structure_Only
def transform_mgraph(graph: Html_MGraph) -> Html_MGraph:
    # Filters nodes...
    # ...but cannot influence engine-specific rendering.
    return filtered_graph
```

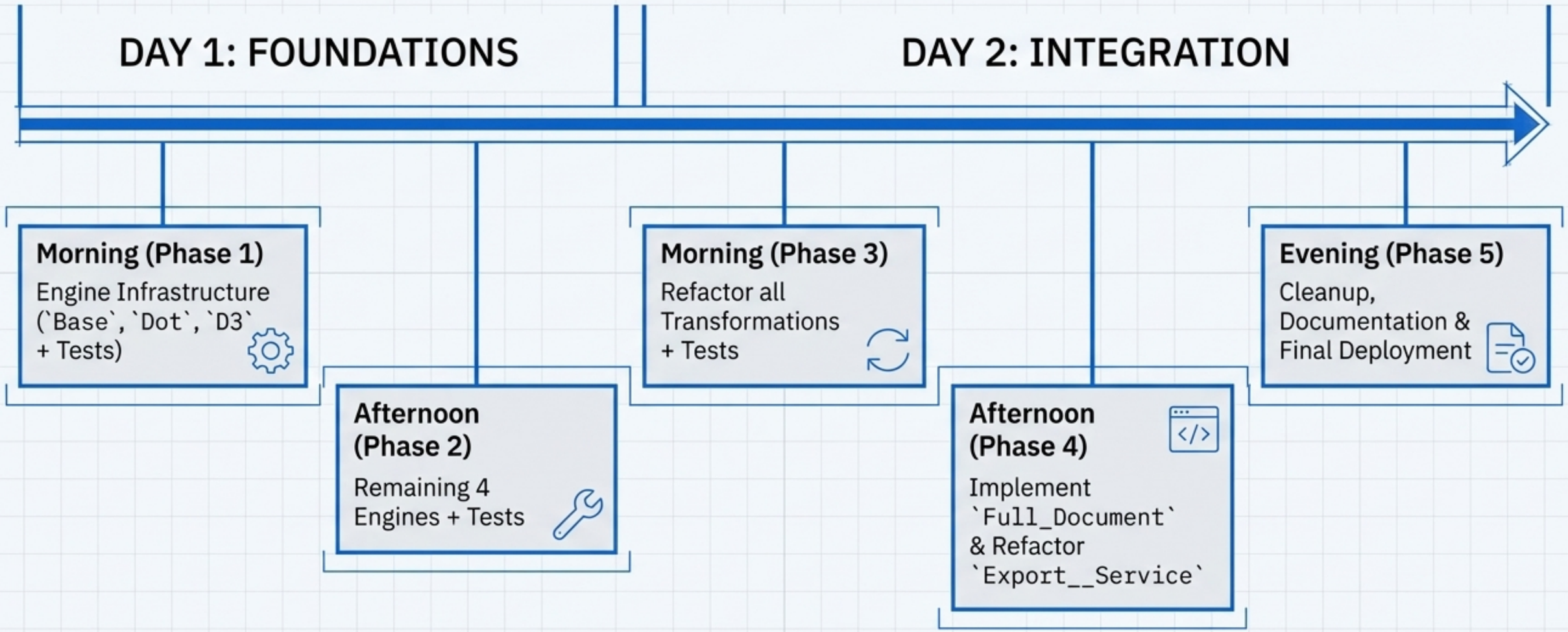
AFTER: Per-Engine Callbacks

```
# new Graph_Transform__Structure_Only
def transform_nodes(nodes: list, engine: str) -> list:
    for node in nodes:
        if engine == 'dot':
            node.attributes['shape'] = 'box' # Direct control!
    return nodes
```

Scope of Work: Key Deliverables

Component	Action	Location
MGraph__Engine__Base & 6 Engines	New	service/mgraph__engines/
Graph_Transformation__Base	Refactor	service/html_graph__transformations/
All 6 existing transformations	Refactor	service/html_graph__transformations/
Graph_Transform__Full_Document	New	service/html_graph__transformations/
Html_Graph__Export__Service	Refactor	service/html_graph__export/

The Implementation Roadmap: A 2-Day Plan



Our Commitment to Stability: Backward Compatibility

This is a purely internal refactoring. There will be no breaking changes to the public API.

API Contract Unchanged

- `Html_Graph__Export__Service` method signatures remain the same.
- Transformation names and output formats are stable.

Clear Deprecation Path

- Old `Html_MGraph__To__*` classes will be marked with deprecation comments.
- They will be removed in a future version (v1.5.0+).

The Definition of Done: Success Criteria

- [x] All **978+** existing tests pass.
- [x] `FastAPI routes` are functionally unchanged.
- [x] The **4-phase pipeline** is implemented for all transformations.
- [x] All **7 engine classes** are created with typed configs.
- [x] **Per-engine callbacks** are available in transformations.
- [x] The new `` `Full_Document` `` **transformation** works correctly.
- [x] A **clean separation of concerns** is achieved.
- [x] There is **no performance regression**.

The Path Forward: Our First Design Decisions

The blueprint is complete, but several implementation details require your expertise. These are the first design decisions for the team to address as we begin implementation.

1. **Styling Metadata:** How should we store style hints (colours, shapes) on `MGraph` nodes?

   Options: `node_data` *dict, separate metadata graph, or extend* `Schema__MGraph__Node`*?*

2. **DOT Cluster Support:** How should the DOT engine handle subgraph clusters for the `Full_Document` view?

   Options: Explicit cluster config or derive from node metadata (e.g., a `cluster_id`*)?*

3. **Tree Engine Variants:** Should `to_tree` and `to_tree_text` be a single engine with a format option, or two separate engines?