4DT903 Individual Report

Reflections on Model-Driven Software Engineering

Samuel Fredric Berg (sb224sc)

January 2026

Contents

1 Core Components and Responsibilities

Model-Driven Software Engineering (MDSE) systems are built upon three primary pillars:

- **Models:** High-level abstractions of software that focus on what the system does rather than how it is implemented. They represent specific system instances, such as notebook models containing code and markdown objects.
- **Metamodels (MM):** These define the language of the system by specifying rules, constraints, and elements. They act as the "schema" for models, defining components like folders, files, or notebook elements.
- **Transformations:** These handle the conversion between different levels of abstraction:
  - **Text to Model (T2M):** Parses source information (like notebook files) into a structured model.
  - **Model to Model (M2M):** Combines or transforms models into new models based on specific logic.
  - **Model to Text (M2T):** Generates final artifacts like source code or documentation from a model.

2 Core Benefits of MDSE

The implementation of MDSE offers several technical and industrial advantages:

- **Increased Productivity:** Automation of code generation for repetitive or high-volume tasks saves significant time.
- **Improved Quality:** Code follows a standardized pipeline, ensuring consistency across the project.
- **Platform Independence:** Models are abstract enough to be deployed across different platforms as needed.
- **Maintainability:** System updates are easier because rebuilding the system from models ensures the pipeline remains consistent.
- **Error Reduction:** Automated processes eliminate the risk of manual errors during repetitive coding tasks.

3 Challenges and Limitations

Despite its strengths, MDSE presents several challenges:

- **Development Time:** Creating robust metamodels and transformations is time-consuming and requires specialized expertise.
- **Tool Complexity:** Sophisticated tools like QvTo or Acceleo have steep learning curves; without deep knowledge, developers may use them inefficiently.

- **Pipeline Debugging:** As a pipeline grows, identifying and fixing bugs becomes increasingly difficult.
- **Overhead:** There is a risk that the complexity of building the MDSE infrastructure exceeds the complexity of the task it is meant to solve.

4 Problem Solving and Design Approach

Approaching a problem with MDSE involves shifting focus toward patterns and abstraction:

- **Pattern Analysis:** Analyze the domain to find recurring concepts (e.g., identifying code, comments, and imports in a file) that can be modeled.
- **Metamodel Definition:** Create a Domain Specific Language (DSL) that captures these core concepts.
- **Transformation Rules:** Develop logic to map data into model objects, utilizing tool-specific features like mappings to keep logic simple.
- **Sequential Implementation:** Build and test transformations separately (T2M, M2M, M2T) before integrating them into a full automated pipeline.

5 Adaptation of Software Engineering Practices

Standard engineering practices must be modified to suit an MDSE environment:

- **Version Control:** Git must be managed carefully to track changes within models rather than just raw source code, often requiring customized ignore files.
- **Testing and Validation:** Testing shifts from line-by-line code reviews to model validation, ensuring all models adhere to the constraints defined in the metamodel.
- **CI/CD Integration:** Automated pipelines must handle transformations in a strict sequential order, which can be difficult to integrate into containerized environments like Docker.

6 Points Section

- Samuel Berg (sb224sc): 4
- Emil Ulvagården (eu222dq): 3
- Jesper Wingren (jw223rn): 3