* **Model-driven Engineering**

Model-driven software engineering provides an approach such that one can design complex software or system in the form of a model and use that model to generate a code for the system automatically. As the model is a preferred output of model-driven software engineering (MDSE) there is no need for engineers to be concerned with the actual specifics of the execution platform or which programming language has to be used, there is no need for programming language details. It allows engineers to think about software at a high level of abstraction, without being concerned about the implementation.

To overcome all these, Model-Driven Engineering (MDE) provides an approach that leads to platform-independent implementation and reduces the complexities and complications involved.

In MDE, the aim is to develop models rather than the program which will generate the code automatically. With the use of MDE level of abstraction is raised from program to model and the tiring process to write a code can be dropped.

**Historical Overview**

In the actual development process, abstraction plays an important role. From assembly language which was used to write machine code to a c programming language to object-oriented programming languages like java and python, there is a continuous rise in levels of abstraction. In the 80s prominent efforts began to raise abstraction levels and Computer-aided software engineering (CASE) was a result of it. It stressed on developing methods and tools that helped developers to represent their styles in terms of general-purpose graphical programming representations, including state machines, structure diagrams, and dataflow diagrams. But due to some reasons, it failed miserably. And now we are stepping forward to level up abstraction to the modeling. Lots of work has been done in this field.

**Model-driven engineering in software**

In simple language, MDE can be defined in two parts:

1. Defining the correct model for a problem statement that makes use of proper abstractions and provides an accurate solution so that people will be more concerned about major key aspects of the problem statement rather than focusing on programming.
2. Providing an accurate solution as a code that will be generated from developed models automatically.

**Models and abstraction levels**

*Models can be considered as a key in MDE as it plays an important role in developing complex systems. While creating of model one must be concerned with the abstraction levels.*

Some of the abstraction types as described as follows:

* ***Structure*** — In this type of abstraction only structural components such as classes, subclasses, modules, interfaces which includes inputs and outputs, etc. are shown and all other details are hidden.
* ***Behavior***— Main functionality of software to be developed i.e behavioral details are shown in this type.
* ***Timing***— All the time-related constraints are addressed.
* ***Resources*** — Models may represent the environment of the software.
* ***Metamodels***— Just like metadata which can be considered as data about the data, meta-models can be considered as models about the models. They describe how the model should be.

**Generation of code**

We can design and develop code generators in many different ways some of the important ones as explained below:

* ***Templates and filtering***

This is one of the simplest ways of code generation. Specifications are given in the textual format. Initially, we have a source model which has different types of models, after filtering some of the specifications we get the subset of the source model, and code is embedded in the templates using a resultant subset of the source model. Thus the resultant code is generated.

* ***Templates and metamodel***

This is an extension of templates and filtering. We don’t directly apply patterns to the model instead of that we instantiate metamodel from the specification first. Templates are applied to this metamodel. Thus the resultant code is generated.

* ***API based generators***

These generators provide an API against which code-generating programs are written. A target language or programming language is focused while developing the API.

* ***Inline code generation***

In inline code generation, the final code is generated at the time of compilation of the non-generated program or using a precompiler.

*Now as we know the basics of Model-driven engineering let’s take a look at the pros and cons of MDE.*

**Advantages of Model-driven engineering**

* ***Higher Productivity*** — As code is generated automatically by the developed model tiring process of programming is done autonomously by the model.
* ***Abstraction*** — Abstraction is raised to model level from program level with the help of Model-driven software engineering.
* ***Consistency*** — As code is auto-generated it’s well maintained and consistent in nature.
* ***Time*** — More time can be given to the core problem statement and its key concepts as the time is reduced in coding.
* ***Improved quality*** — Handwritten code comes with variations and contrasts as work is done in a distributed manner. Hence these improvisations have to be made manually to make code more consistent but in the case of generators, incompetency is reduced significantly thus the overall quality of code is improved.

**Disadvantages**

* To build a model itself is a tedious and quite time-consuming task.
* Even after building a code generator, there are chances that it might not be sufficient enough and work in all the cases.
* In each system, there might be a type of code that the model cant generate and has to be generated manually.