FACULDADE ENGENHARIA
Departamento de Informática

Engenharia de Software
(14341, 16230, 15386)

Model-Driven Requirements Engineering

(adapted from *Software Engineering: International Version (10th Edition)*, Ian Sommerville, Pearson, 2015)

Topics covered

- ✧ Model-driven engineering (MDE)
- ✧ Model-driven requirements engineering (MDRE)
- ✧ MDRE in practice

Soft Skill of the week: Creativity

Model-driven engineering

Model-driven engineering

- Model-driven engineering (MDE) is an approach to software development where models rather than programs are the principal outputs of the development process.
- The programs that execute on a hardware/software platform are then generated automatically from the models.
- Proponents of MDE argue that this raises the level of abstraction in software engineering so that engineers no longer have to be concerned with programming language details or the specifics of execution platforms.

Usage of model-driven engineering

- Model-driven engineering is still at an early stage of development, and it is unclear whether or not it will have a significant effect on software engineering practice.
- Pros
  - Allows systems to be considered at higher levels of abstraction
  - Generating code automatically means that it is cheaper to adapt systems to new platforms.
- Cons
  - Models for abstraction and not necessarily right for implementation.
  - Savings from generating code may be outweighed by the costs of developing translators for new platforms.

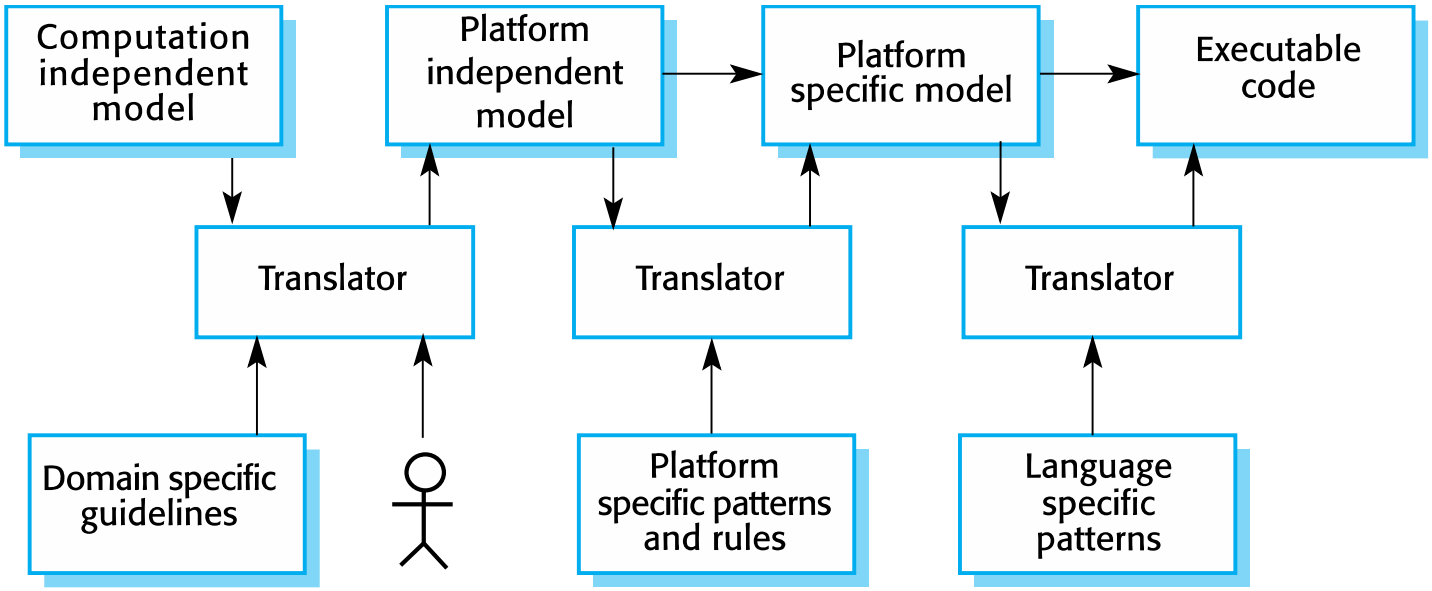
Model driven architecture

- Model-driven architecture (MDA) was the precursor of more general model-driven engineering.
- MDA is a model-focused approach to software design and implementation that uses a subset of UML models to describe a system.
- Models at different levels of abstraction are created. From a high-level, platform independent model, it is possible, in principle, to generate a working program without manual intervention.

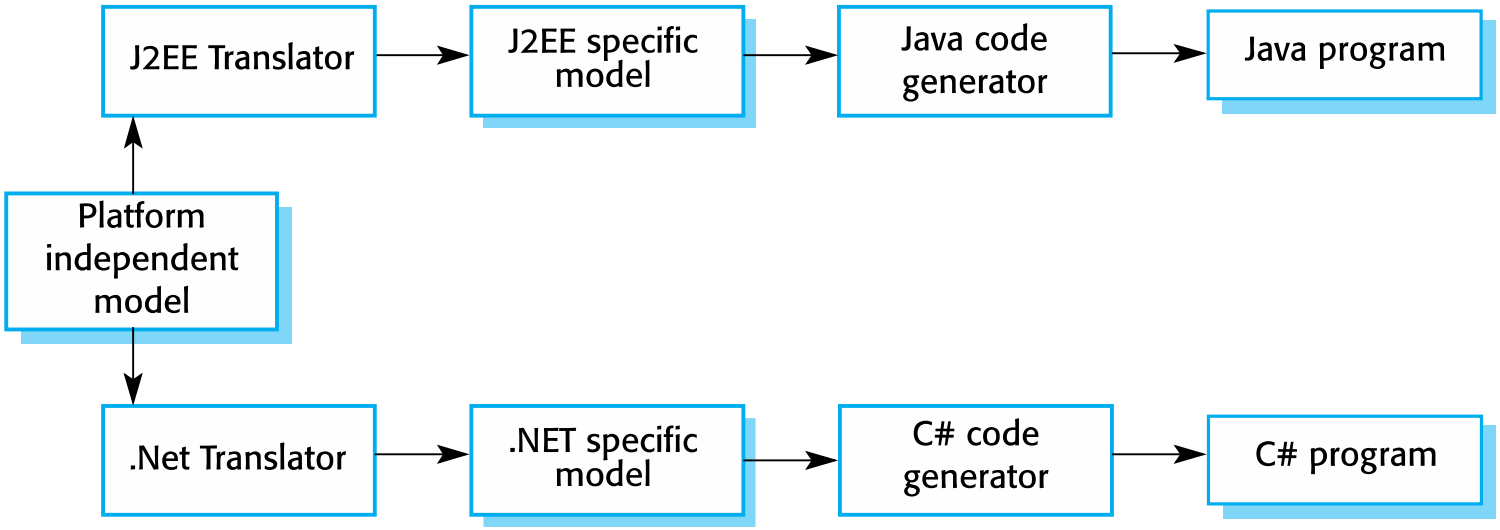
Types of model

- A computation independent model (CIM)
  - These model the important domain abstractions used in a system. CIMs are sometimes called domain models.
- A platform independent model (PIM)
  - These model the operation of the system without reference to its implementation. The PIM is usually described using UML models that show the static system structure and how it responds to external and internal events.
- Platform specific models (PSM)
  - These are transformations of the platform-independent model with a separate PSM for each application platform. In principle, there may be layers of PSM, with each layer adding some platform-specific detail.

MDA transformations



Multiple platform-specific models



Agile methods and MDA

- The developers of MDA claim that it is intended to support an iterative approach to development and so can be used within agile methods.
- The notion of extensive up-front modeling contradicts the fundamental ideas in the agile manifesto…
- If transformations can be completely automated and a complete program generated from a PIM, then, in principle, MDA could be used in an agile development process as no separate coding would be required.

Adoption of MDA

- A range of factors has limited the adoption of MDE/MDA.
- Specialized tool support is required to convert models from one level to another.
- There is limited tool availability and organizations may require tool adaptation and customization to their environment.
- For the long-lifetime systems developed using MDA, companies are reluctant to develop their own tools or rely on small companies that may go out of business.

Adoption of MDA

- Models are a good way of facilitating discussions about a software design. However the abstractions that are useful for discussions may not be the right abstractions for implementation.
- For most complex systems, implementation is not the major problem – requirements engineering, security and dependability, integration with legacy systems and testing are all more significant.

Adoption of MDA

- The arguments for platform-independence are only valid for large, long-lifetime systems. For software products and information systems, the savings from the use of MDA are likely to be outweighed by the costs of its introduction and tooling.
- The widespread adoption of agile methods over the same period that MDA was evolving has diverted attention away from model-driven approaches.

Model-driven requirements engineering

Why Not Only Use Text?

- ✧ Natural language can be ambiguous.
- ✧ Example: 'The system shall show the nearest hospitals' — unclear radius, ranking.
- ✧ Models provide structure and reduce misunderstandings.

Definition of MDRE

- ✧ Capturing, analyzing, and validating requirements.
- ✧ Using visual models as the main communication and/or documentation method.
- ✧ Improves requirements traceability, enabling seamless tracking of requirements throughout the development lifecycle.

Objectives of MDRE

✧ Reduce ambiguity

- Replace vague natural-language requirements with precise, visual representations.
- Minimize misinterpretations by using standardized modeling elements.
- Example: Instead of “fast system response,” specify “system shall respond within 2 seconds under peak load.”

✧ Improve shared understanding

- Ensure all stakeholders (technical and non-technical) interpret requirements the same way.
- Use diagrams and models as a “common language” to bridge communication gaps.
- Foster early consensus on system functionality and constraints.

Objectives of MDRE

✧ Link requirements to downstream artifacts

- Maintain direct traceability between requirements and their implementation.
- Enable impact analysis — when a requirement changes, see which design elements, code modules, and test cases are affected.
- Improve consistency across development stages.

✧ Enable early validation and analysis

- Models can be validated for completeness, consistency, and correctness using tools.
- Automated generation of design artifacts or test cases is possible.
- Early detection of conflicts and missing elements reduces costly rework later.

Benefits of MDRE

✧ Clarity — better than text alone

- Visual models present relationships, processes, and structures in a way that's easier to understand than long paragraphs.
- Reduces ambiguity by standardizing how requirements are represented.
- Helps stakeholders with varying technical backgrounds grasp the same concept quickly.

✧ Collaboration — aligns teams

- Creates a shared visual language between developers, testers, product managers, and clients.
- Encourages discussion and validation early in the process.
- Improves stakeholder engagement in reviewing requirements.

Benefits of MDRE

✧ Change management — faster updates

- Models can be updated quickly when requirements change.
- Avoids re-writing large text documents.
- Supports impact analysis — quickly see what elements are affected.

✧ Traceability — direct links to implementation

- Each model element can be mapped directly to system components, test cases, and documentation.
- Facilitates compliance with standards (e.g., ISO, CMMI).
- Makes auditing and verification easier.

✧ Scalability — works for complex systems

- Supports modular decomposition for large projects.
- Can represent different levels of abstraction (system, sub-system, component).
- Helps maintain consistency across distributed teams and large-scale developments.

Traditional RE vs MDRE

Aspect	Traditional Requirements Engineering	Model-Driven Requirements Engineering (MDRE)
Representation	Text-based documentation	Visual and formalized models
Clarity	Prone to ambiguities and misinterpretations	Precise, unambiguous representation of requirements
Traceability	Manual, limited, and error-prone	Automated and comprehensive traceability
Validation and Verification	Relies on manual reviews and testing	Automated validation through modeling tools
Scalability	Challenging for complex systems	Scalable for large and multidisciplinary systems
Change Management	Time-consuming and error-prone	Streamlined with model updates and real-time tracking

MDRE in practice

✧ Industries

- **Automotive** – Designing advanced driver-assistance systems (ADAS), infotainment, and vehicle control software where functional safety (ISO 26262) is mandatory.
- **Aerospace** – Avionics and flight control systems requiring strict compliance with standards.
- **Healthcare** – Medical device software and hospital information systems where traceability and regulatory compliance (e.g., FDA, ISO 13485) are critical.

MDRE in practice

✧ Large distributed teams

- Helps maintain a **single source of truth** across geographically dispersed teams.
- Reduces miscommunication when development, testing, and design are done in different countries.
- Supports asynchronous collaboration through shared, version-controlled models.

✧ Safety-critical and regulated projects

- Ensures **traceability from requirements to implementation** for audits.
- Supports **impact analysis** when requirements or regulations change.
- Minimizes risk of non-compliance through rigorous modeling and validation.

Types of requirement models

- ✧ Functional — define system features and behaviors.
- ✧ Non-functional — specify performance, constraints, and quality attributes.
- ✧ Informal visuals — use block diagrams, flowcharts, or mind maps.

Agile methods and MDRE

- **Visual representation of requirements**
  - MDRE leverages **visual and formalized models** to represent requirements clearly and concisely, minimizing the ambiguity often associated with purely text-based Agile user stories.
  - Visual models allow teams to quickly understand requirements, spot inconsistencies, and adapt to shifting priorities without losing the overall system perspective.

Agile methods and MDRE

- **Real-Time updates and adaptability**
  - In Agile projects, requirements evolve throughout the development lifecycle.
  - MDRE ensures models are **dynamically updated** to reflect ongoing changes, ensuring the entire team remains aligned on the latest scope and objectives.
  - This reduces miscommunication and prevents costly delays caused by unclear or outdated requirements.

Agile methods and MDRE

✧ Enhanced Cross-Team Collaboration

- MDRE tools (e.g. Visure Requirements ALM Platform) provide a **shared, model-driven workspace** for cross-functional teams.
- Designers, developers, testers, and stakeholders can all interact with the same models, enabling transparent discussions and faster decision-making.

Practice: MDRE

Based on the video you watched:

✧ **Identify** 4–6 core requirements from the scenario.

✧ **Create** a simple diagram (boxes + arrows).

✧ **Use** an online tool to produce the schematic.

Key points (1 of 2)

- A model is an abstract view of a system that ignores system details. Complementary system models can be developed to show the system's context, interactions, structure and behavior.
- Context models show how a system that is being modeled is positioned in an environment with other systems and processes.
- Use case diagrams and sequence diagrams are used to describe the interactions between users and systems in the system being designed. Use cases describe interactions between a system and external actors; sequence diagrams add more information to these by showing interactions between system objects.
- Structural models show the organization and architecture of a system. Class diagrams are used to define the static structure of classes in a system and their associations.

Key points (2 of 2)

- Behavioral models are used to describe the dynamic behavior of an executing system. This behavior can be modeled from the perspective of the data processed by the system, or by the events that stimulate responses from a system.
- Activity diagrams may be used to model the processing of data, where each activity represents one process step.
- State diagrams are used to model a system's behavior in response to internal or external events.
- Model-driven engineering is an approach to software development in which a system is represented as a set of models that can be automatically transformed to executable code.


WHY
WHERE
WHEN
WHERE
WHY
WHO
WHEN
WHAT
WHAT
WHY
WHEN
WHAT
WHO
WHERE
WHY
HOW