LUT University

Software Engineering Models and Modeling

Model-driven engineering and large language models – future of MDE?

Antti Knutas

Some MDE material courtesy of Brambilla et al. (2012), from MDSE teaching materials package.

Today’s lecture

Introduction to the topic of LLM and then an interactive lecture task about where generative content could work (and how), in the context of our software modelling work

- Introduction to the topic
  - Look into one possible futures of MDSE through LLMs
  - Super quick intro into LLMs (you surely know about it by know)
  - A quick demo about creating models with ChatGPTv4 (plus demo of small OSS models)
- Lecture activities: Thinking where this could fit into the whole process

Introduction contents

Two parts:

- **Recap** of “hardcore” model-driven software engineering (MDSE) from the 00s & 2010s
- Look into one possible futures of MDSE through LLMs

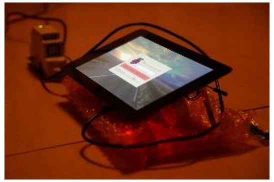
How we have been using models this far

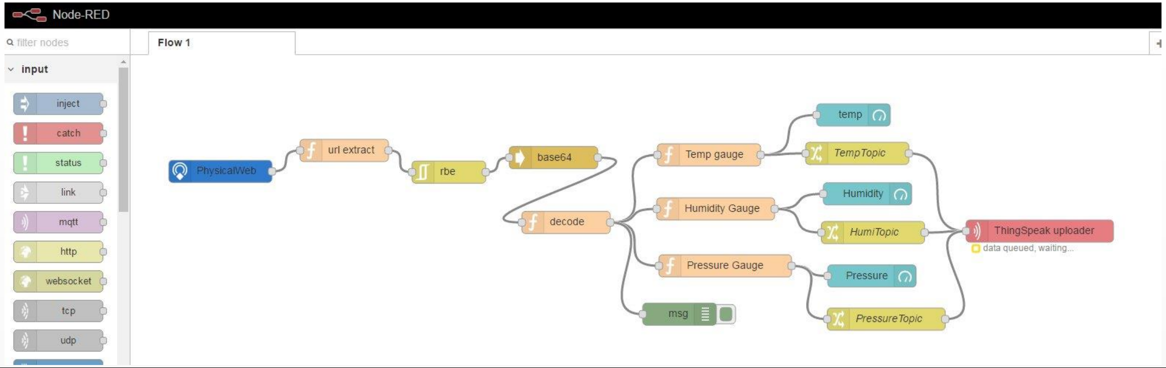
- **Model:** a simplified or partial representation of reality, defined in order to accomplish a task or to reach an agreement

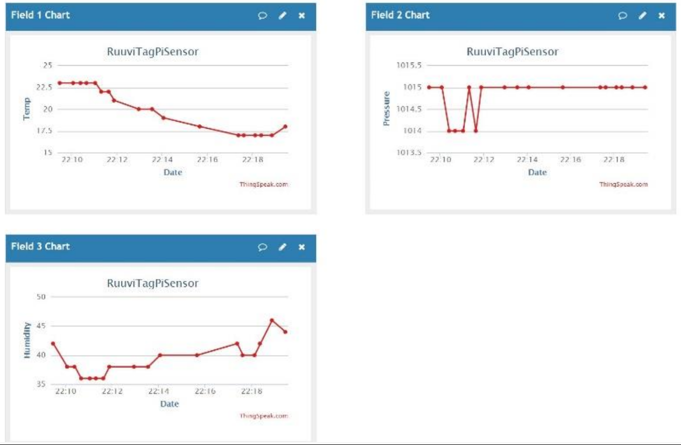
“in software design, considered to be one of the key aspects of software engineering, multiple stakeholders with different technical knowledge and background collaborate on the system design. In this context, shared models are used as ‘a reduced representation of some system that highlights the properties of interest from a given viewpoint.’” (Di Ruscio et al., 2022)

Other uses?

In enterprise systems, executable models and code generation have been used. In "low-code" or "no-code" approaches, visual programming is becoming mainstream again.







Future?

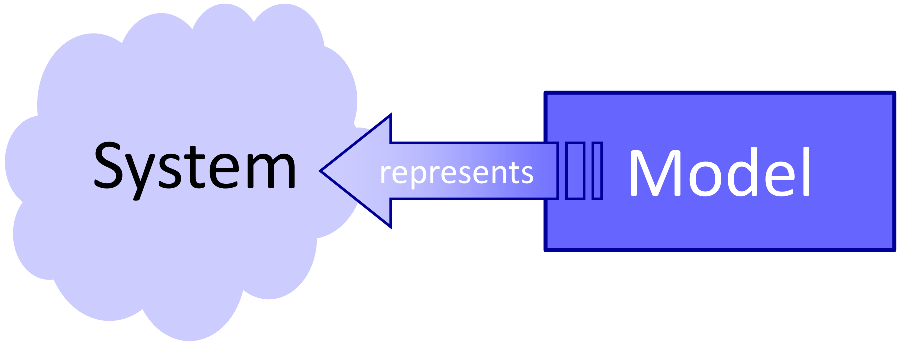
Automated code generation from models, or model generation from free-form text input? Both?

In essence, automated MDSE transformations.

Recap of Model-Driven Software Engineering

Models

What is a model?



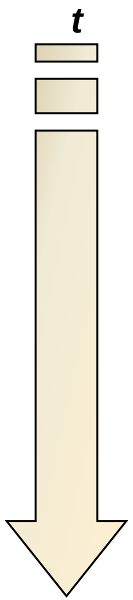
Mapping Feature	A model is based on an original (=system)
Reduction Feature	A model only reflects a (relevant) selection of the original's properties
Pragmatic Feature	A model needs to be usable in place of an original with respect to some purpose

Purposes:

- descriptive purposes
- prescriptive purposes

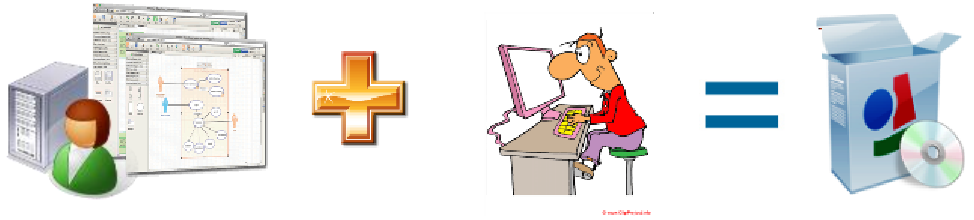
Motivation

Application area of modeling



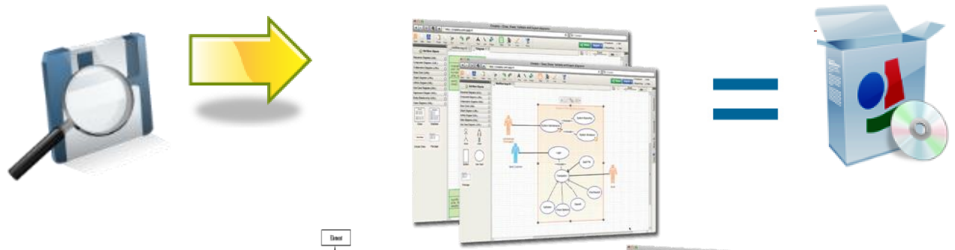
- ***Models as drafts***
  - Communication of ideas and alternatives
  - Objective: modeling per se
- ***Models as guidelines***
  - Design decisions are documented
  - Objective: instructions for implementation
- ***Models as programs***
  - Applications are generated automatically
  - Objective: models are source code and vice versa

Three killer MDSE applications



Code

Generatio
n



Software

Modernizatio
n

Systems
interoperability

View into 00s / 2010s MDSE process

USE CASE1 – MODEL DRIVEN DEVEOPMENT (BY BRAMBILLA ET AL.)

MORGAN & CLAYPOOL PUBLISHERS
Model-Driven Software Engineering in Practice
Marco Brambilla
Jordi Cabot
Manuel Wimmer
Synthesis Lectures on Software Engineering

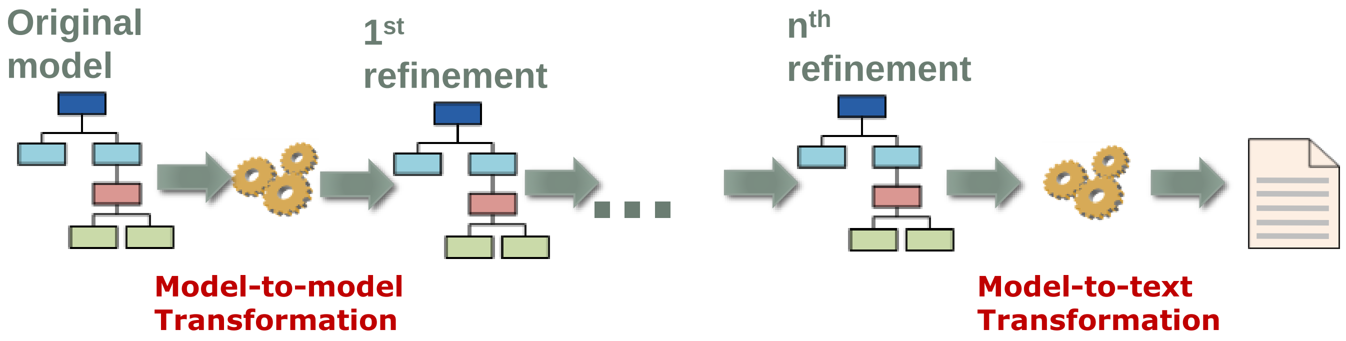
MDD contribution: Communication

- Models capture and organize the understanding of the system within a group of people
- Models as *lingua franca* between actors from business and IT divisions

$$\begin{tikzcd}
\boxed{\text{Requirements}} \arrow[r, "\text{M2M}"] & \boxed{\text{Analysis}} \arrow[r, "\text{M2M}"] & \boxed{\text{Design}} \arrow[r, "\text{M2T}"] & \boxed{\text{Implementation}}
\end{tikzcd}$$

MDD contribution: Productivity

- MDD (semi)automates software development
- In MDD, software is derived through a series of model-to-model transformations (possibly) ending with a model-to-text transformations that produces the final code

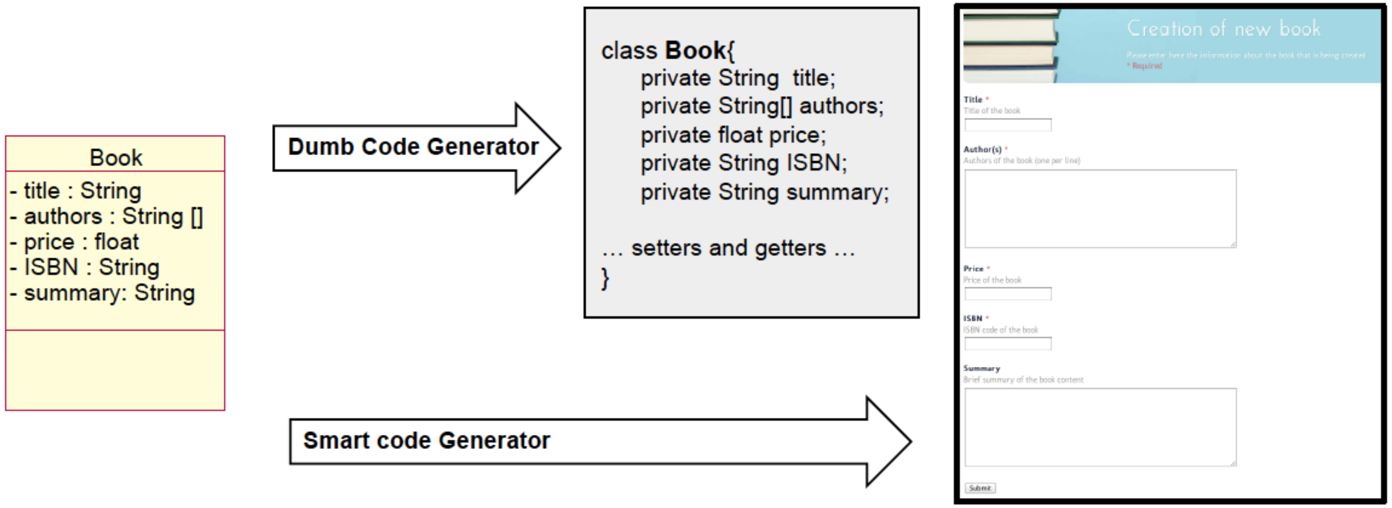


Executable models

- An executable model is a model complete enough to be executable
- From a theoretical point of view, a model is executable when **its operational semantics are fully specified**
- In practice, the executability of a model may depend on the adopted execution engine
  - models which are not entirely specified but that can be executed by some advanced tools that are able to fill the gaps
  - Completely formalized models that cannot be executed because an appropriate execution engine is missing.

Smart vs dumb execution engines

- CRUD operation typically account for 80% of the overall software functionality
- Huge spared effort through simple generation rules



```
class Book{
    private String  title;
    private String[] authors;
    private float price;
    private String ISBN;
    private String summary;

… setters and getters …
}
```

Creation of new book

Please enter here the information about the book that is being created

* Required

Title *
Title of the book

Author(s) *
Authors of the book (one per line)

Price *
Price of the book

ISBN *
ISBN code of the book

Summary
Brief summary of the book content

Submit

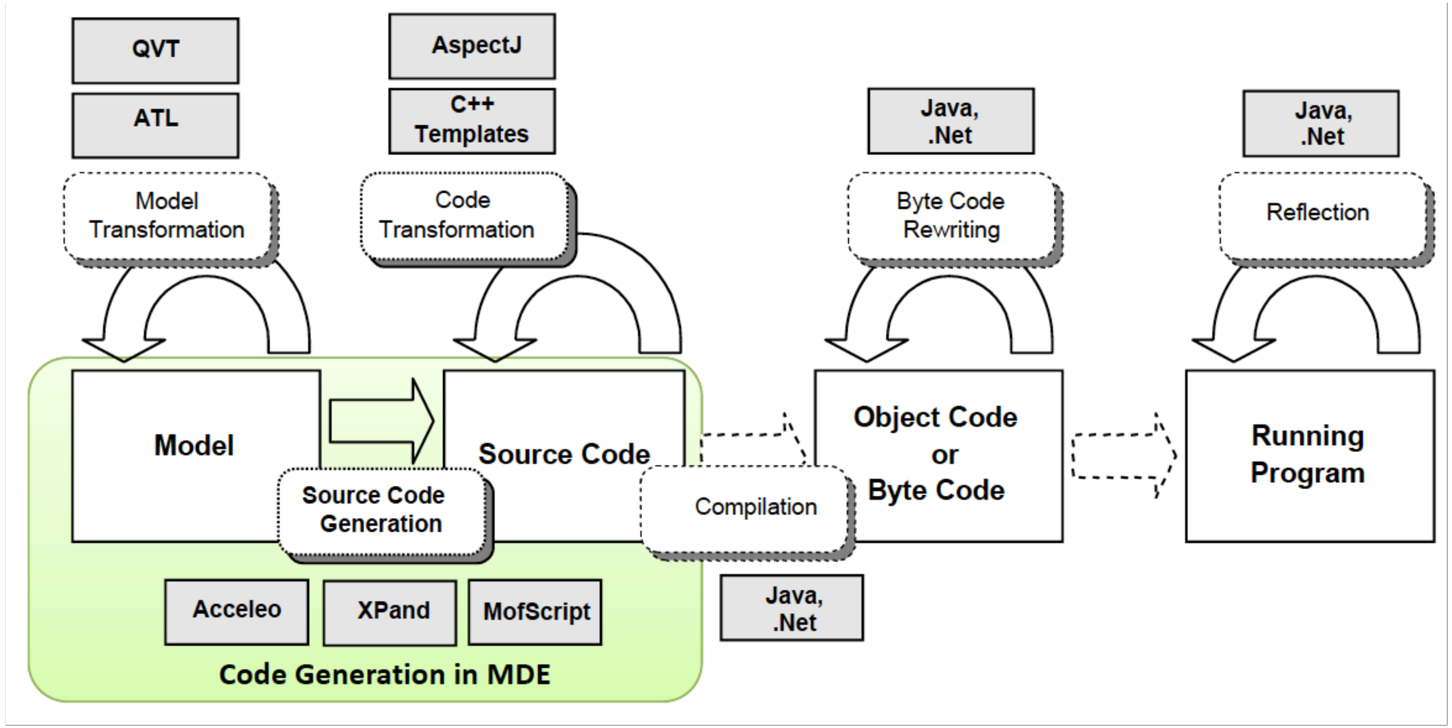
Executable models: 2 main approaches

- **Code generation**: generating running code from a higher level model in order to create a working application
  - by means of a rule-based template engine
  - common IDE tools can be used to render the source code produced

- **Model interpretation**: interpreting the models and making them run

- Non-empty intersection between the two options

Code Generation: Scope



Code Generation

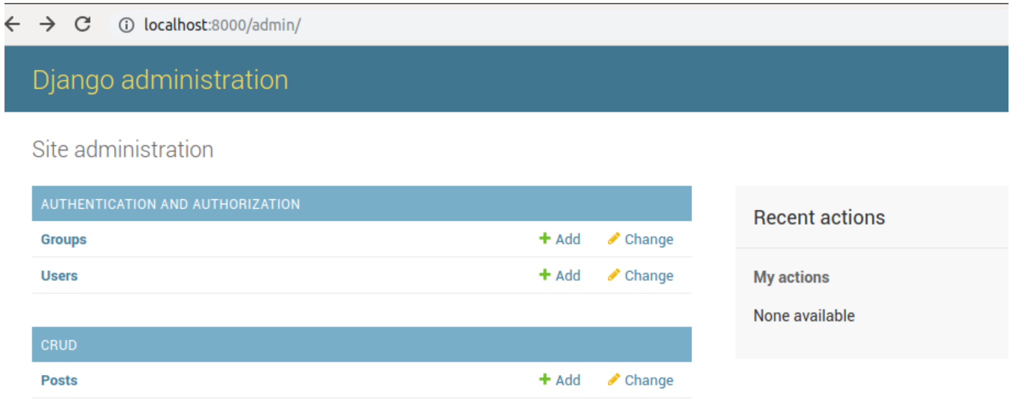
- Goal: generating running code from higher level models
  - Like compilers producing executable binary files from source code
  - Also known as model compilers

- Once the source code is generated state-of-the-art IDEs can be used to manipulate the code

Code Generation: Partial Generation

- Input models are not complete & code generator is not smart enough to derive or guess the missing information
- Programmers will need to complete the code manually

Some examples: Django, Sails.js


localhost:8000/admin/

Django administration

Site administration

AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Users	+ Add	Change

CRUD		
Posts	+ Add	Change

Recent actions

My actions

None available

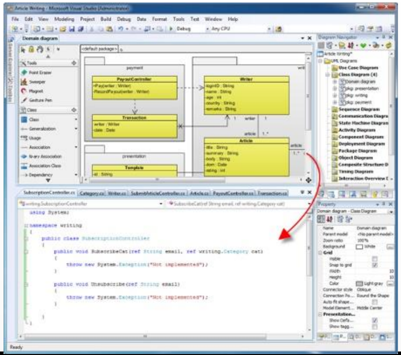
What are the advantages?

- It could make SWE much faster and reliable, with studies documenting benefits in **productivity** and **quality** through **abstraction** and **automation**
  - Software specification
  - Testing
  - Maintenance
  - Code generation

What are the disadvantages?

- Latest 2020 software technologies are rarely 100% MDSE compatible (from specification to code generation)
  - Visual paradigm supports UML to C#, Java, DLL, .NET, VB.NET, PHP, Python, XSD, XML, ActionScript, Objective-C, Ada95, ODL, IDL, Delphi, PERL & Ruby

Solutions?


```
using System;

namespace writing
{
    public class SubscriptionController
    {
        public void SubscribeCat(ref String email, ref writing.Category cat)
        {
            throw new System.Exception("Not implemented");
        }

        public void Unsubscribe(ref String email)
        {
            throw new System.Exception("Not implemented");
        }

    }
}
```

Large language models

Disclaimer: Some materials are speculative and might change fast. References include Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., ... & Wang, H. (2023). Large Language Models for Software Engineering: A Systematic Literature Review. *arXiv preprint arXiv:2308.10620*. and Vaithilingam, P., Zhang, T., & Glassman, E. L. (2022, April). Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *Chi conference on human factors in computing systems extended abstracts* (pp. 1-7).

Low code development platforms or LLM - based code generation, motivation

First: Why?

⇒ Major part of modern programming is defining data flows, putting data on screen and Create / Read / Update / Delete operations.

⇒ One hardly needs a software engineering with a five-year degree to connect input-outputs with predefined graphics or manually program yet another form field.

Can environments be specified in a manner that allow software engineering to move even to higher level planning and processes, such as quality control or specification?

Large language models, defined

“Large language models (LLMs) are deep learning algorithms that can recognize, summarize, translate, predict, and generate content using very large datasets.”

(Gartner, nVidia)

Made possible by past advances in neural network architecture and recent availability of computational power.

Incredibly advanced, but no different in operating principle than predictive text input in your mobile phone.

LLMs, current ways to execute (will be outdated eventually)

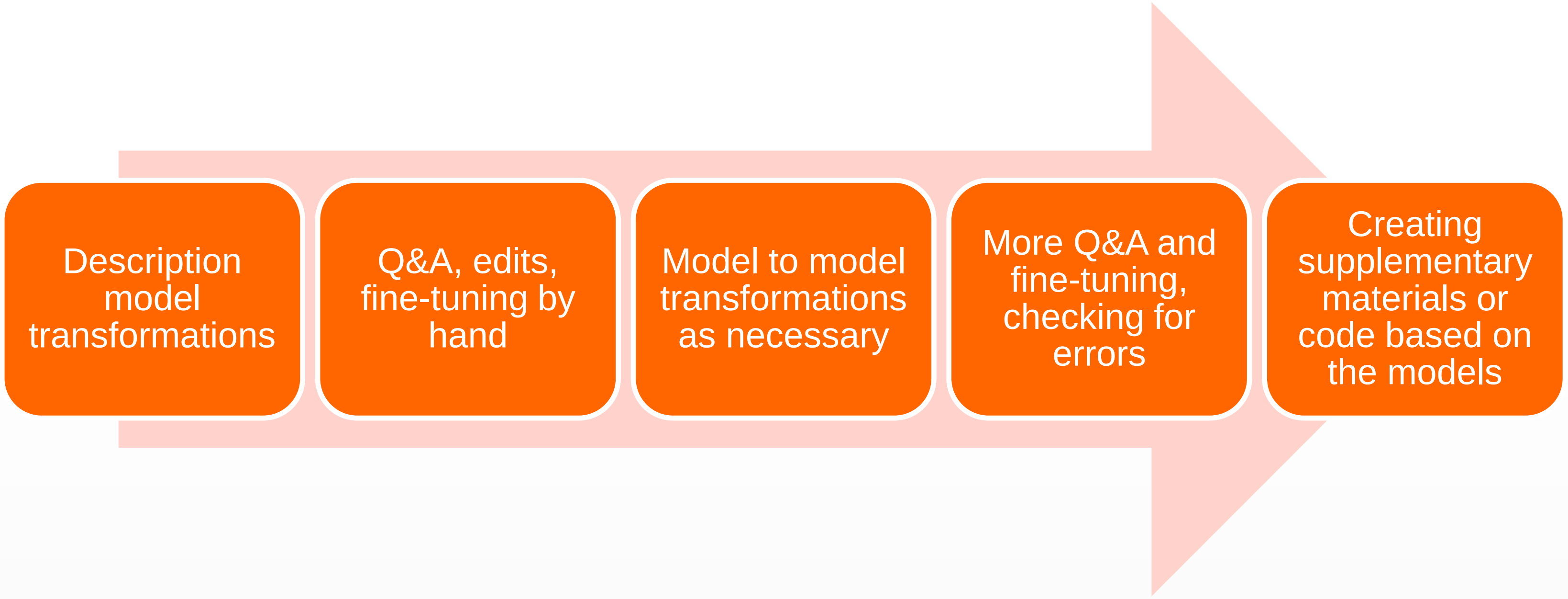
- Open-source models, such as Llama and Falcon
  - => can be run locally with 32GB+ of RAM and fast multicore, or extremely fast GPUs
  - E.g. with LMStudio
- Commercial models, such as Google Bard or ChatGPT
  - => available through their website

Ways to utilize LLMs in software modelling?

- Model generation (case => model => model => code)
- Model to model transformations
- Evaluating model quality or giving feedback
- Code generation from user-created models
- Automating repetitive tasks and GUI boilerplate generation once the engineer has been able to define the idea

These are at experimental, alpha stage with no literature on reliability or validity yet, in 2023. 2024 or 2025 will be a different world.

My current idea for a workflow (disclaimer, experimental, but based on MDSE processes)


Description model transformations
Q&A, edits, fine-tuning by hand
Model to model transformations as necessary
More Q&A and fine-tuning, checking for errors
Creating supplementary materials or code based on the models

Potential pitfalls?

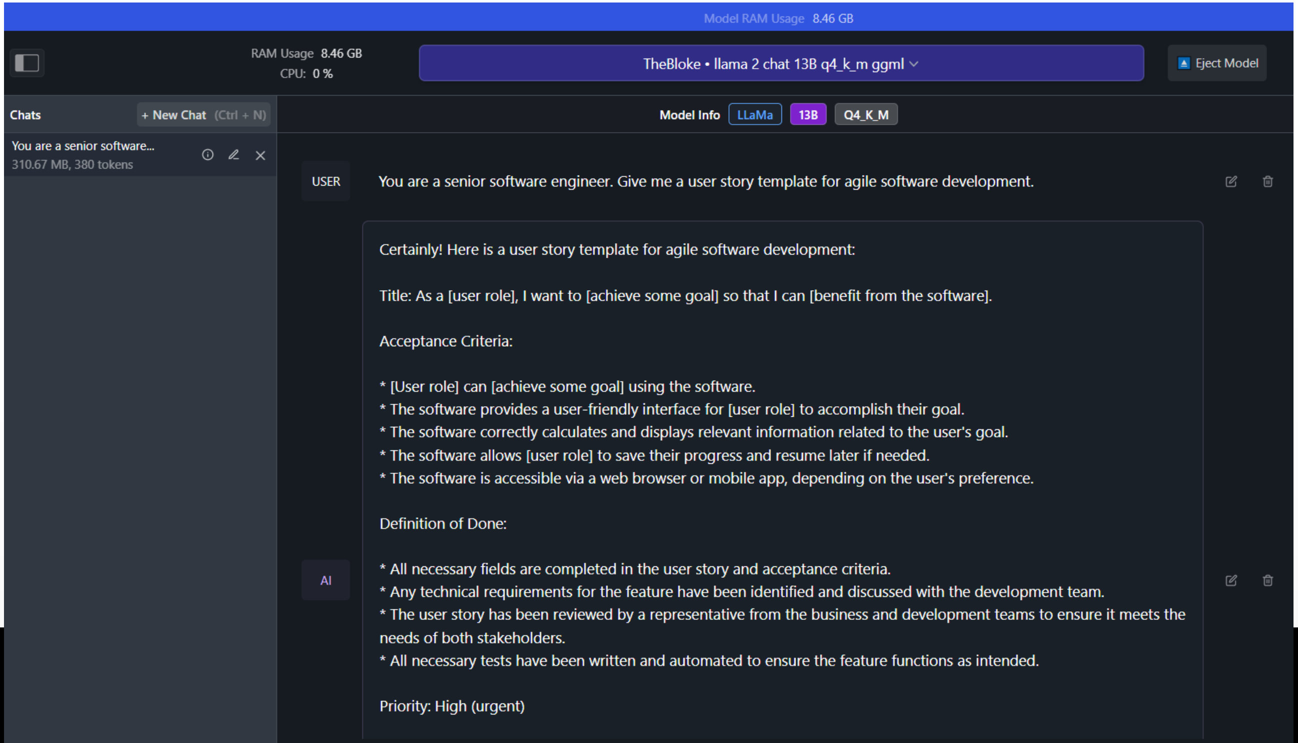
- Hallucinations and generating false information
- Misunderstanding the source material
- Difficulty of QA
- Who signs off the product and who is responsible in case of product errors?
- People treating the computing as an authority (see: “the computer says no”)

LLMs, current ways to execute (will be outdated eventually)

- Open source models, such as Llama and Falcon
  - => can be run locally with 32GB+ of RAM and fast multicore, or extremely fast GPUs
  - E.g. with LMStudio
- Commercial models, such as Google Bard or ChatGPT
  - => available through their website

LLMs & SEMM, current workflows?

- Text generation and then visualization



You are a senior software engineer. Give me a user story template for agile software development.

Certainly! Here is a user story template for agile software development:

Title: As a [user role], I want to [achieve some goal] so that I can [benefit from the software].

Acceptance Criteria:

* [User role] can [achieve some goal] using the software.
* The software provides a user-friendly interface for [user role] to accomplish their goal.
* The software correctly calculates and displays relevant information related to the user's goal.
* The software allows [user role] to save their progress and resume later if needed.
* The software is accessible via a web browser or mobile app, depending on the user's preference.

Definition of Done:

* All necessary fields are completed in the user story and acceptance criteria.
* Any technical requirements for the feature have been identified and discussed with the development team.
* The user story has been reviewed by a representative from the business and development teams to ensure it meets the needs of both stakeholders.
* All necessary tests have been written and automated to ensure the feature functions as intended.

Priority: High (urgent)

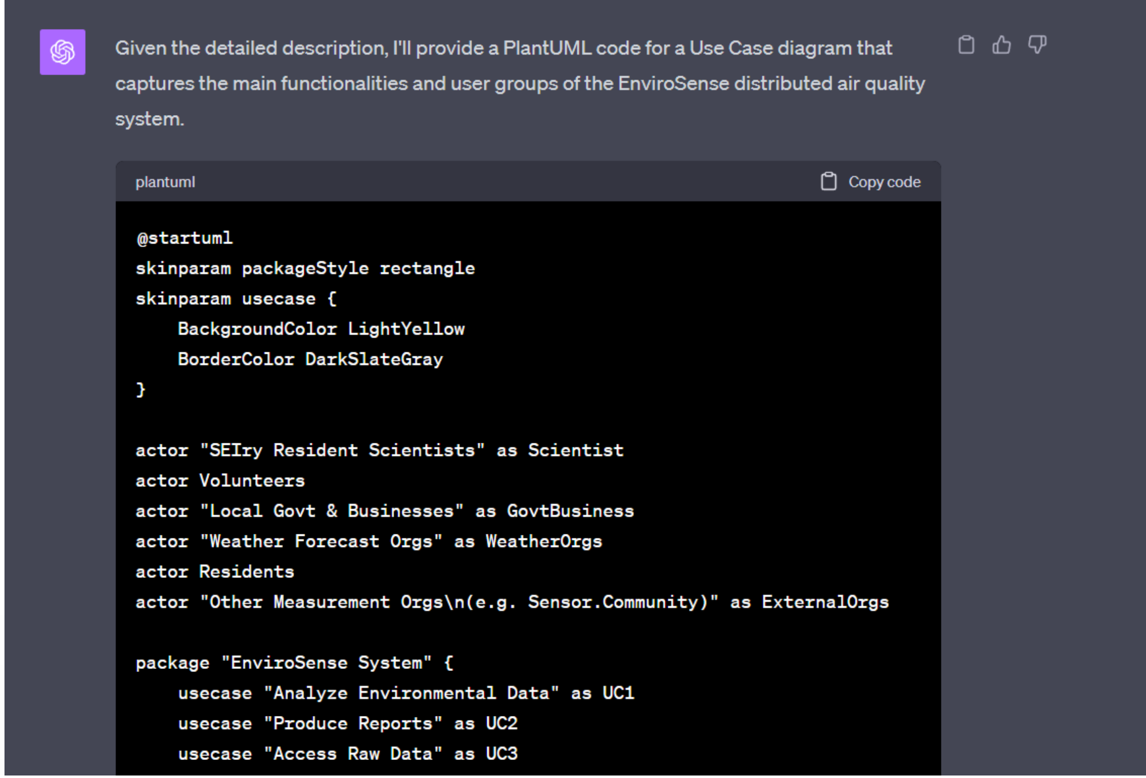
LLMs & SEMM, short demo (Llama 2 & ChatGPT)

1. “You are a senior software engineer. Give me a user story template for agile software development.”
2. Branching
   A) Create a PlantUML Use Case Code based on the following case description: “(paste case text)”
   B) Select and create a report template for the Produce Reports use case, using the agile user story template. Pre-fill it with information that can be extracted unambiguously from the case description. Leave blank (denote with "TODO") parts that would need guesswork.

Q: Why did I specify optional branching, instead of continuing?

(demo sample link at https://chat.openai.com/share/748bdec1-0421-44c4-9903-1312b4f204b3)

LLMs & SEMM, outcomes

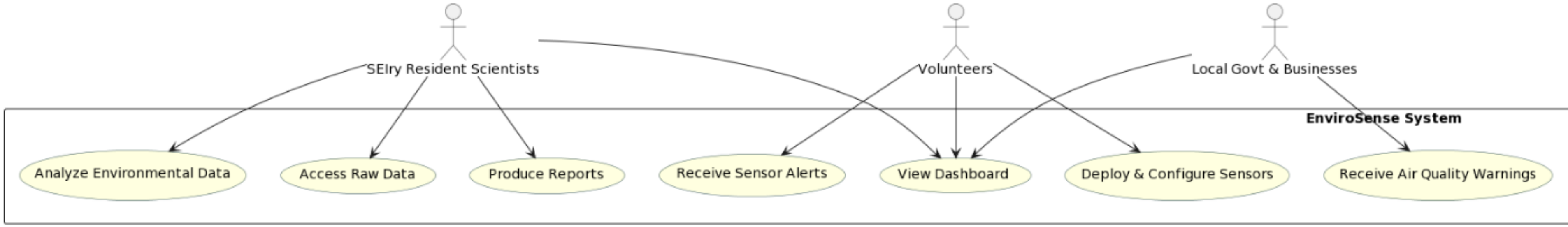

Given the detailed description, I'll provide a PlantUML code for a Use Case diagram that captures the main functionalities and user groups of the EnviroSense distributed air quality system.

plantuml

```
@startuml
skinparam packageStyle rectangle
skinparam usecase {
    BackgroundColor LightYellow
    BorderColor DarkSlateGray
}

actor "SEIry Resident Scientists" as Scientist
actor Volunteers
actor "Local Govt & Businesses" as GovtBusiness
actor "Weather Forecast Orgs" as WeatherOrgs
actor Residents
actor "Other Measurement Orgs\n(e.g. Sensor.Community)" as ExternalOrgs

package "EnviroSense System" {
    usecase "Analyze Environmental Data" as UC1
    usecase "Produce Reports" as UC2
    usecase "Access Raw Data" as UC3
```



Further resources and selection of discovered online tutorials

- Online modelling tools that will visualize LLM text input
  - https://www.gituml.com/plantuml
  - https://mermaid.live/ or https://mermaid.js.org/#/
- Few tutorials (login wall)
  - https://blog.devgenius.io/chatgpt-how-to-generate-uml-diagram-cb745ecba83e (PlantUML)
  - https://atharvashah.netlify.app/posts/tech/uml-diagrams-with-chatgpt/ (PlantUML)
  - https://machinelearningmastery.com/generating-graphics-with-chatgpt/ (Mermaid)
  - https://betterprogramming.pub/how-to-create-diagrams-with-chatgpt-19099de795ce?gi=2a6acedcec45 (Merpaid)

In summary

- Model-driven engineering seeks to improve software development
  - by raising abstraction
  - and hiding implementation details
- LLMs can be a new way to create and transform models
- Advantages: Increased automation, accepts “messy” and incomplete input (previous model transformers and generators required specific input-output formats)
- Disadvantages: Potential for errors, hallucinations, ownership of text & QA, mistaking generated boilerplate for authoritative material

Lastly, LUT LLM policy

“Students are always responsible for the assignments they submit and the independent work they do to achieve learning outcomes. Academic writing rules, including accurate citations and references, continue to apply. Students must disclose if and how they have used AI. …students may not claim AI-generated texts as their own.”

=> My take *in this course*: Document its use clearly. If you can create high quality materials faster that you can sell to the client, I’m fine with it!

=> What not to do: Ask the system to make decisions for you. (e.g. ethics, priorisation)

https://elut.lut.fi/en/completing-studies/rules-and-regulations/ai-based-tools-policies