# Report on the State of Model Driven Development 28 Years After the Completion of the SISU Project

Stein Erik Ellevseth
*Retired*
Oslo, Norway
Email: steinee@eunet.no

Peter Herrmann
*Norwegian University of Science and Technology (NTNU)*
Trondheim, Norway
Email: herrmann@ntnu.no

Ole Lehrmann Madsen
*Aarhus University*
Aarhus, Denmark
Email: ole.l.madsen@cs.au.dk

Emmanuel Gaudin
*PragmaDev*
Paris, France
Email: emmanuel.gaudin@pragmadev.com

Juergen Dingel
*Queen's University*
Kingston, Ontario, Canada
Email: dingel@queensu.ca

Hessa Alfraihi
*Princess Nourah bint Abdulrahman University*
Riyadh, Saudi Arabia
Email: haalraihi@pnu.edu.sa

***Abstract*—In October 2023, a special session of the 15th System Analysis and Modelling Conference (SAM) in Västerås, Sweden, was dedicated to the development of Model-Driven Development (MDD) in the last 20 to 30 years. It consisted of the presentation of a survey article followed by a panel discussion. Most of the authors of this article acted as the panelists guided by Stein Erik Ellevseth. The starting point for the session was a look at the results of the Norwegian project Support to Integrated System Development (SISU), in which 19 companies and several research institutions worked together in the 1990s to build software systems using the Specification and Description Language (SDL), a model-based engineering technique and language. The panelists agreed that, in spite of the fact that the quality of the systems produced with MDD was usually very high, MDD did not progress as expected by the SISU participants. This article summarizes the arguments brought forward in the panel session and gives some further insights into the topic. In particular, the reasons for the reluctance of many software developers and their companies to use MDD are discussed. Moreover, the paper presents a number of recommendations, that might be helpful to change course towards a more prominent use of MDD in software engineering in the future.**

## I. Introductory Summary

Model-Driven Development (MDD) is an important subset of the Model-Driven Engineering[1] (MDE) paradigm. Both, MDE and MDD support the use of higher abstraction levels, as well as simulation, system analysis, verification, and automatic code generation in software development [1]. The general perception of MDD-based methods and tools is that they improve profitability, lower the development cost, enhance quality, shorten time to market, and increase controllability.

As a step to encourage the use of MDD in the Norwegian embedded and telecom industry, the project Support to Integrated System Development (SISU[2]) was launched. It lasted from 1989 to 1996 and brought 19 companies and four research institutions together to use tools supporting the Specification and Description Language (SDL) in practice [2]. An assessment of the impact of SISU was conducted in 2016, i.e., 20 years after the SISU project was terminated, see [3].

To explore this issue further, a panel was formed at the 15th System Modelling and Analysis (SAM) conference, that took place in Västerås, Sweden, in October 2023. Its title was "Model-Driven Development: 20 Years Later". Important topics were programming, graphical modeling, tool support, and agility. The panel session also included the presentation of surveys about MDD in general [4] and SISU in particular.

The following panelists presented their views: Ole Lehrman Madsen from Aarhus University discussed highlights from his KeyNote presentation about early and future days of modeling and programming [5]. Emmanuel Gaudin, CEO of PragmaDev and the chairman of the SDL Forum Society, shared his views on tool support and agility. Juergen Dingel from Queen's University presented his views on using MDD for graphical modeling, open model repositories and open source tooling. Hessa Alfraihi from Princess Nourah bint Abdulrahman University presented highlights from her paper "A Survey on Trends and Insights into the Use of Model-Driven Engineering" presented at the conference [4]. Stein Erik Ellevseth, formerly ABB AS, Ericsson AS, and Alcatel STK, introduced outcomes from the SISU project and its assessment 20 years later [3].

The main conclusion of the discussion was that the benefits of MDD are still valid, and that it will also have a place in the future. However, there are also threats and obstacles that need to be addressed:

- **Methodology enthusiasts and improvement agents** in the companies went missing, moved away, were retired, or lost interest due to low management support during the time after the termination of SISU in 1996 facing comments like "We spent a lot of money on SISU, now we expect payback". The national industrial network that was created during the SISU cooperation was neither

[1]Both terms Model-Driven Engineering (MDE) and Model-Driven Development (MDD) were used by the contributors but, for clarity, we decided to restrict us to the term MDD throughout this text.

[2]The project title in Norwegian was "Støtte til Integrert SystemUtvikling".

TABLE I
THE SISU MATURITY MODEL, TAKEN FROM CMM.

| | System Description | Verification & Validation | Trans-formation | Product Management | Reuse | Process (CMM) |
|---|---|---|---|---|---|---|
| 1 | Realization-oriented | Test-oriented | Compiled | Initial | Initial | Initial |
| 2 | Partially Design-oriented | Inspection | Partially Generated | Version-oriented | Product-oriented | Repeatable |
| 3 | Design-oriented | Animated | Generated | Product-oriented | Domain-oriented | Defined |
| 4 | Product-oriented | Analyzed | System-generated | Integrated | Integrated | Managed |
| 5 | Required | Synthesized | Design transformed | Required | Global | Optimized |

sufficiently nourished on the national level nor inside the companies.

- Initially, the Object-Oriented languages Simula and Beta were seen as suited to support both **modeling and programming**. However, modeling and coding were soon regarded as two completely separate tasks, that had to be treated independently from each other. For instance, while program execution can be seen as a model, most state-of-the-art modeling tools are limited with respect to debugging the execution of a system. Further, **graphical descriptions** are seen as important in the initial develop phases, but in later stages manually written code is preferred.
- The **Agile development approach** is getting more popular in system development. Unfortunately, however, it has reduced significant benefits of verifying abstract models early. For instance, deeper analyses of the impacts that design decisions have on certain system requirements, are omitted, since agile is relying on incrementally verification through iterations (sprints). This leads to defects that have to be corrected in later iterations (sprints) that can cause significant costs.
- Nowadays, licenses for training, prototyping and proof-of-concept tend to be free. Thus, the **revenues from selling modeling tools** significantly decreased, which led to a declined economy of the tool vendors. Therefore, many tool producers had to reduce their support and investment into tool improvements.
- **Model interchange** is lacking since comprehensive, standardized and universally supported formats are missing. Moreover, open model repositories specifying recurrent system functionality might help to reuse parts of the models. This would lead to shorter time to market and lower development costs.
- Practitioners expect the availability of **long-term vendor support**, **training**, and **education**. This may, however, be sometimes difficult to achieve in an open-source community, in which the tools are expected to be free. Further, companies that depend on these tools, are required to engage in agreements with organizations supporting technology like Apache, Red Hat, TASTE, or Fedora, that can give support.

In the following sections, we sketch the topics named by the participants in the panel discussion and add a few others. First, we introduce the SISU project and sketch the development in the member companies after its completion in Sect. II. In Sect. III, we discuss the relationship between modeling and programming in general and agile computing in particular. This is followed by a look at modeling tools in Sect. IV and at the perspective of the users in Sect. V. Thereafter, some other approaches are introduced in Sect. VI followed by a discussion about formal methods in Sect. VII. Next, we evaluate the SISU goals in Sect. VIII. The paper is completed by giving some recommendations that might help to increase the acceptance of MDD in Sect. IX. The numerous abbreviations used in this text are explained in a dictionary at the end.

## II. SISU, SDL AND THE NORDIC INITIATIVE

The SISU project was a Norwegian national project carried out in two phases [2]. The first one lasted from 1989 to 1992 and was initially funded with 28 million Norwegian kroner (3.5 million Euros) by the Research Council of Norway. It focused on the use of tools based on SDL, SA/SD, Simula, ER, and OSDL [6], that were integrated into the software generation process of the project participants. Of particular interest were early verification models as well as simulators for these artifacts and technology to generate code from them. The second phase was SISU II, which was carried out between 1993 and 1996 and funded by 51 million NOK (6.4 million Euros). It mainly focused on business improvements using tools from SISU I.

Four research organizations and at least 19 companies including two tool vendors were involved in the first phase. 15 companies completed also the second phase. In addition, there were contributions by the ESPRIT projects Proteus, Reboot, and Perfect. The 19 companies were furnished with SDL support, simulation, code generation, and runtime support for SDL systems, while the four research institutes acted as technology providers, trainers and mentors for the participating companies.

Further, the progress of the project participants over the project duration was determined. To achieve that, a model to score the companies was created, that was inspired by the Capability Maturity Model (CMM) [7]. As depicted in Tab. I,

TABLE II
SISU MATURITY LEVELS 1992, 1996, AND 2016.

| Discipline | Year | Company | | | | | | | | | | | | | | | Sum | Average |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | | |
| System Description | 1992 | 3 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 1 | 1 | 3 | 2 | 2 | 2 | 1 | 26 | 1.7 |
| | 1996 | 4 | 2 | 2 | 3 | 2 | 3 | 3 | 3 | 2 | 2 | 4 | 3 | 3 | 3 | 3 | 42 | 2.8 |
| | 2016 | 2 | 2 | 2 | 3 | 1 | 2 | 3 | 1 | 1 | 1 | 2 | 1 | 2 | 3 | 1 | 27 | 1.8 |
| Verification & Validation | 1992 | 2 | 1 | 1 | 1 | 1 | 1 | 2 | 1 | 1 | 1 | 2 | 3 | | 1 | 1 | 19 | 1.4 |
| | 1996 | 3 | 2 | 2 | 2 | 2 | 2 | 3 | 2 | 2 | 2 | 3 | 3 | | 3 | 2 | 33 | 2.4 |
| | 2016 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | | 3 | 1 | 29 | 2.1 |
| Transfor-mations | 1992 | 3 | 1 | 1 | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 3 | 2 | 1 | 1 | 21 | 1.4 |
| | 1996 | 3 | 2 | 2 | 3 | 2 | 3 | 3 | 1 | 2 | 3 | 3 | 3 | 3 | 3 | 2 | 38 | 2.5 |
| | 2016 | 2 | 2 | 2 | 3 | 1 | 3 | 3 | 1 | 1 | 2 | 2 | 1 | 1 | 3 | 1 | 28 | 1.9 |
| Product Management | 1992 | 2 | 2 | 1 | 1 | 2 | 2 | 3 | 2 | 1 | 1 | 2 | 2 | | 1 | 2 | 24 | 1.7 |
| | 1996 | 3 | 3 | 2 | 3 | 3 | 2 | 3 | 2 | 3 | 2 | 3 | 3 | | 2 | 3 | 37 | 2.6 |
| | 2016 | 3 | 3 | 3 | 3 | 3 | 2 | 3 | 3 | 3 | 3 | 3 | 3 | | 2 | 2 | 39 | 2.8 |
| Reuse | 1992 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | 1 | 1 | 19 | 1.4 |
| | 1996 | 2 | 2 | 3 | 3 | 3 | 1 | 2 | 2 | 3 | 2 | 1 | 2 | | 1 | 2 | 29 | 2.1 |
| | 2016 | 2 | 2 | 3 | 3 | 2 | 1 | 3 | 1 | 3 | 2 | 2 | 2 | | 1 | 1 | 28 | 2.0 |
| Process | 1992 | 3 | 2 | 2 | 2 | 2 | 2 | 3 | 2 | 1 | 2 | 3 | 2 | | 1 | 2 | 29 | 2.1 |
| | 1996 | 3 | 3 | 3 | 3 | 3 | 3 | 4 | 3 | 3 | 3 | 4 | 4 | | 2 | 3 | 44 | 3.1 |
| | 2016 | 4 | 3 | 3 | 3 | 3 | 3 | 4 | 2 | 3 | 2 | 2 | 3 | | 2 | 2 | 39 | 2.8 |
| Changes | 1992 – 1996 | 3 | 5 | 6 | 9 | 4 | 5 | 6 | 4 | 9 | 7 | 6 | 5 | 2 | 7 | 7 | 85 | 5.7 |
| | 1996 – 2016 | -3 | 0 | 1 | 0 | -3 | -1 | 1 | -3 | -2 | -2 | -5 | -6 | -3 | 0 | -7 | -33 | -2.2 |
| | 1992 – 2016 | 0 | 5 | 7 | 9 | 1 | 4 | 7 | 1 | 7 | 5 | 1 | -1 | -1 | 7 | 0 | 52 | 3.5 |

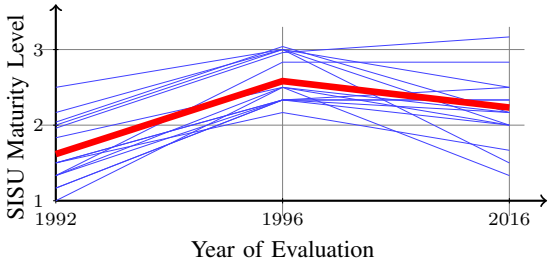


Fig. 1. Development of the Mean Maturity Levels in the Companies and their Average over Time.

this model assessed the six disciplines System Description, Verification and Validation, Transformation, Product Management, Reuse, and Process. For each of these disciplines, each company was given a maturity score between 1 and 5 following the criteria outlined in Tab. I at the start of the project and at its end. Moreover, the scores of the companies 20 years after completing the project were also identified.

The overall results are presented in Tab. II. The rows show the six disciplines as well as the years, in which they were measured. The columns refer to the 15 companies having participated in the examination. Of particular interest are the changes of the companies' maturity levels over time, which is depicted in the three lowermost rows of Tab. II denoted as "Changes". As shown by the comparison between 1992 and 1996, the project led to a significant improvement of the maturity levels by 5.7 score points (levels) per company in average. Not surprisingly, the maturity dropped after finishing the project, as the comparison between 1996 and 2016 discloses. This drop, however, was much smaller than the gain during the project, such that the maturity level in 2016 was still around 0.6 score points higher for each company and discipline than before the start of the project in 1992.

The development of the SISU maturity levels over time is also illustrated in Fig. 1. Here, we use a thin blue line to mark the development of these levels in each of the companies, while the thick red line describes the averages throughout the companies. While this improvement could, at least to some degree, have also happened without SISU, we still believe that the technology harvested from the project has contributed to a more sustainable software development and economic benefits for the participating companies.

The technology used in the project had its root in the partner companies and the Norwegian University of Science and Technology (NTNU) and Sintef Delab (former Elab-NTH) in Trondheim as well as the Simula environment at the Norwegian Computing Center (NCC / Norsk Regnesentral, NR) in Oslo. Two central technologies used in SISU were Beta and OSDL, that both were results of the Mjølner project, a cooperative Nordic project with participants from Sweden, Norway, Finland, and Denmark, see [8], [9]. It is well-known that Object-Orientation started with SIMULA in 1967, but it is little-known that SIMULA also formed the basis for the modeling language Beta-73. Moreover, the Ericsson AXE software structure was one of the foundations for SDL-76 as well as later for composite structures in UML2.

The SISU initiative was regarded as a great success among the participants and in academia thirty years ago. In particular, the project proved that the expectations were achievable in many cases. SISU made major contributions to Object-Orientation to the SDL community through standardization of SDL Z.100, also called SDL-92 and Message Sequence Charts (MSC-96, Z.120). Moreover, also the composite structures, sequence diagrams, and active classes in UML 2.0 (Unified Modeling Language from OMG, Object Management Group)

were influenced by SISU developments [10], [11]. Further, a SISU Textbook was published in 1993 and is still being used in master student education at NTNU [12]. A second book, this time in electronic form, was developed during SISU II [13].

After 28 years, six of the 19 industry participants still use and maintain products created in the project. These products and the tools applied for their development are listed in Tab. III. In some cases, the modeling tools and code generators are not used anymore, but the source code is maintained manually. In this case, the names of the originally used technologies are crossed out in Tab. III. More details on the tools are described in the Dictionary and in the summary from the SISU assessment meeting in 2016 [3].

The people working in the participant companies during SISU proved to be important resources for MDD in their own company and in the companies they later attended. They matured the industry and spread the methodologies, tools, and standards to other companies, to which they later moved. On the other hand, after 1996 some methodology enthusiasts and improvement agents in the companies moved, retired, or lost interest over time. Moreover, the national industrial network created during the SISU cooperation, was neither sufficiently nourished at the national level nor inside the companies. There were always different opinions about the benefits and drawbacks of using a higher abstraction level in the companies. Likely reasons were a general resistance to changes as well as the mindset of some programmers, that only source code is worthwhile. Nevertheless, as pointed out in the discussion about Tab. II, the increased maturity achieved during SISU prevailed at least partially.

The discussion panel at SAM 2023 covered another important question about the SISU project, i.e., to what degree MDD contributed to improved profitability in the development and maintenance of embedded software in companies. In particular, the following specific goals were of interest:

1) Correct system quality,
2) Shorter time to market,
3) Lower development costs,
4) Improved project control.

The panelists agreed that MDD, if used properly, can help to achieve the goals 1, 3 and 4. Goal 2, however, was seen as questionable since using MDD usually affords an additional amount of time to be spent during the early phases of a project in order to ensure the necessary product quality. The compliance with the goals will be discussed in greater detail in Sect. VIII.

## III. Modeling, Programming, and Agile Development

While modeling a software system and coding the programs were originally performed in close relationship, developers started quite early to separate these two aspects into completely independent tasks, e.g., by starting to use Structure Analysis (SA) and Structured Design (SD) diagrams to influence software development. Over the years, coding has become the predominant activity in a software project, and modeling was, if at all, limited to using initial diagrams to achieve a good understanding of the structure of a system. On the other hand, applying abstractions early in product development is recognized to be very important for the creation of high-quality software. This evolution towards "code over model" suggests that there might be a general lack of understanding about formal modeling in the software community. This could again be a reason for the relatively low quality of many software products.

Object-Orientation (OO) has been a driving force in software development over the last 40 years. While its terms can be dated back to the Greek philosophers (Platon: the idea and the individual), Object-Oriented modeling as a skill started to emerge in the mid-60'es. Inspired by Hoare´s paper on Record Handling [14], SIMULA started as a language for programming and modeling with emphasis on modeling. The follow-up language Beta [15] was also intended to support modeling as well as programming. However, SIMULA and Beta never became mainstream languages. Smalltalk was the first popular Object-Oriented language and may be a central reason that the reuse of code instead of modeling became the main focus in software development.

OO programming was separated from OO modeling in the late 80s, when the OO mechanism introduced in SIMULA was implemented as an extension of C in C++ and later in Java, C#, Python, JavaScript, and others. The OO programming community focuses primarily on implementing programs without modeling them explicitly. For instance, OO-programmers rarely mirror the environment of a system into types and objects. Instead, they appreciate the benefits of reusing existing publically available software utilities to access the environment of a system. Thus, it seems that programming won the battle of influence in software development against modeling. But then programming effectively **is** modeling, though at a very detailed level. The above-mentioned quality issues with a lot of today's software indicate, however, that this level may be too elaborate, and a modeling approach for programming seems to be missing.

Modeling in the programs can profit from an explicit conceptual framework for OO modeling, see [5]. A good example is the language Beta, for which such a conceptual framework was developed. It is described in Chapter 18 (Conceptual Framework) of the Betabook [15] and focuses on the program execution and visualized interaction and behavior of the objects in the execution of the program. In contrast to that, most state-of-the-art modeling tools have limited ways to debug the execution of a system. In our opinion, a better capability to debug software can be a good step towards software products of higher quality.

One of the major benefits of modeling is to be able to think ahead, be prepared, and make sure that you know what to do before actually doing it. This is achieved by being able to verify both static and dynamic functional properties of the system model by model-checking or simulation. Unfortunately, Agile development ruined this benefit, since it pushes for short-term solutions, i.e., "waterfall in sprints", by bringing the software

TABLE III
THE SIX COMPANIES THAT, IN 2023, STILL USE TECHNOLOGY DEVELOPED IN SISU.

| Company | Modeling Tool | Code Generator | Programming Language | Runtime System | Product |
|---|---|---|---|---|---|
| Zenitel (Stentofon) | SDT 3.x / Tau 4.1 | ProgGen '90+ | C | TST v3.26 | ICX-AlphaCom |
| Indra Navia (Garex) | ~~DaSOM, Ooram~~ | DaSOM/PG → Ooram/PG | C | TST v. | Garex 220 switch |
| | ~~DaSOM, Ooram~~ | DaSOM/PG → Ooram/PG | C | TST v. | Garex 220 terminal |
| | SDT / TAU | SDT/PG / ProgGen | C | TST v. ? | Garex 220 MPC position |
| Thales (Alcatel) | Geode ? | ProgGen ? | CHILL | CRS: CHILL RS | NSS |
| | TAU SDL Suite 4.2 | SDT/PG ? | C ? | SDT RS/CRS | VCS/ACEcom |
| Autronica | | | | Inhouse | Fire Alarm |
| Norsonic | ~~SDT~~ | ~~ProgGen~~ | C | TST | Nor 121 |
| KDA (NFTE) | ~~Geode~~ → SDT 4.6 | ProgGen | CHILL → C | CRS → Inhouse | MRR |
| | Geode → SDT 4.6 | ProgGen | C | Inhouse | CPX |
| | Geode → SDT 4.6 | ProgGen | C | Inhouse | MLT220 |

engineer to program something immediately and correcting the code in the following iterations, if that is done at all. This makes it very likely, that unintentional defects survive the development and testing phases. Therefore they will end up as severe bugs in later phases or even the final product, that have to be fixed under high costs, see, e.g., [16]. A defect created in one phase can cost ten times as much to solve if it is detected and fixed in the next one. Moreover, Agile often results in software products that have not the previously planned structure. One can provocatively add that it does not seem to matter really if we know, where we are going or not. On the other side, it seems that it is initially much more fun to work in Agile, at least, until the usually tedious bug fixing begins.

In the majority of cases, programming appears to be the main task in software development and, in the end, the code is the product. Models, in contrast, are mainly created for documentation purposes in early development phases. Further, there seems to be a lack of interest in verifying models. Instead, software architects appear to be satisfied with models that are only correct to a certain degree. In our opinion, this attitude is a pity since MDD can effectively reduce development efforts through the graphical representations of models, which significantly facilitates the understanding of the product to be developed. Moreover, MDD helps to improve the communication between teams. This contributes to better quality, fewer defects, and longer product lifetime.

This standpoint is also supported by the majority of the 119 participants surveyed in [4]. They conclude that automating software development tasks, model checking, and source code generation improve the overall efficiency of system engineering. Further, they agree that the improved software quality provides reliable products with less error-prone software. This helps to enhance the life-cycles of the products. A typical example is the well-known memory leakage problem, that can be usually avoided by applying MDD, since defects are normally detected early in the development cycle. Another example is callback hell [17], that can be easily mitigated thanks to the asynchronous event handling used in techniques like SDL. Moreover, the collaboration between development teams is facilitated thanks to a better understanding of the models easing the communication among the team members.

Another key aspect of a model is, that it can be also used to specify the architecture of a system. If the architecture is not correct in the first place, its later fixing will raise significant additional costs. This makes it difficult to compare the time to market of a product developed using Agile with a system built based on MDD including a proper and complete definition of its architecture. It is quite likely that the Agile one will be available sooner but likely carries a lot of bugs or misses some functionality, that will increase its lifetime costs significantly.

Overall, the lifecycle costs have to be taken into account and not only the initial development costs. In particular, the costs of fixing bugs in products, that are already in service, should be considered as well as those needed to maintain the system over the years. However, when using Agile development, these costs are rarely taken into account. Since a properly modeled product has usually fewer bugs and can be easier maintained than a system created using Agile development, its overall costs are often significantly lower. This holds in spite of higher initial costs due to carrying out the MDD.

## IV. TOOLS PERSPECTIVE

Since the tool vendors participating in the SISU project felt that they did not get sufficient support for the newly standardized Object-Oriented constructs of SDL, i.e., SDL-92, they hesitated to build this extension into their tools. The tool vendors also faced technical issues beyond methodology, hardware problems, and software issues (e.g., embedded challenges with real-time performance). Moreover, the general market situation changed since the drop in the oil price to 15 USD per barrel in 1994 created difficulties for many Norwegian companies.

Originally, big companies such as Ericsson, Alcatel / Thales, and Siemens developed modeling tools for their own internal use. Gradually, they started both to use commercially available products and to outsource their own tools. That led to modeling tools and language providers like Capella (Thales), Titan (Ericsson), and Erlang (Ericsson). Competition also came from UML tools, in particular, drawing tools like EA and IntelliJ. The added competition led to a significant drop of the money that could be earned for the modeling tools. For example, the cost for a license started at around 100 kEUR in the beginning of the 1990s, but over time the license costs de-

creased to 10 kEUR. Further, licenses for training, prototyping, and proof-of-concept are more or less free. In consequence, the economic situation of the tool vendors declined, which influenced their ability to support their products and provide new functionality.

Few tool vendors did actually implement SDL-92 into their SDL tool set, whereas Object-Oriented programming languages like C++ and Java expanded vastly. Telelogic implemented SDL-92 into SDT 3.0, that was available in 1995. This amended SDL version was well received, evaluated and piloted by several SISU companies. Thereby, mainly the procedure, process and block types of SDT 3.0 were used, but not so much inheritance-related constructs like *virtual*, *redefine* and *finalized*. That fits in with our knowledge, that the object-oriented SDL-constructs have not been used in any major product development up to now. Nevertheless, these constructs are available in some development tools like PragmaDev and openGeode. In contrast, open source variants exist for most Object-Oriented programming languages (e.g., gcc, java/jvm), and also Integrated Development Environments (IDE) like Eclipse are available for free. These tools are often in use for creating the software of products requiring high degrees of security and safety. It is a question, however, if and to which degree these tools are able to contribute to truly safe and secure products.

During the SISU project and also later, the question came up, whether round trip engineering is really needed, as many developers claim. Round trip engineering is being able to synchronize software artifacts with the model they are derived from. From an MDD point of view, however, it is not needed since all development is centered around the model, from which source code can be automatically generated.

Up to now, comprehensive, standardized, universally supported formats, that will allow effective model interchange, are missing. This often leads to so-called tool lock-in of users, i.e., the impossibility to replace a tool, when that is reasonable. That is the case for many UML- and Domain Specific Language (DSL)-based tools as well as Matlab Simulink from Mathworks. To make model interchange possible, the tool vendors should agree on common formats, that can be fixed by standardization bodies like ITU or ETSI. ITU-T has already provided a recommendation for model interchange, namely the Common Interchange recommendation Z.106 [18]. Similar efforts could be made in other bodies, e.g., for XMI UML from OMG [19] and DSLs. Nevertheless, most available UML tools do not support XMI seamlessly, which can easily lead to compatibility issues [20]. Furthermore, XMI has a tendency to run verbosely, which can hamper the development performance [21].

Model interchange will also allow for the creation of open model repositories. Similar to class libraries, that support code creation in many programming languages, recurrent system functionality can be specified and thereafter provided to the tool-users for an easy integration into their models. This would be of great help to achieve both, shorter time to market and lower development costs.

Universally supported formats like Open Services for Lifecycle Collaboration (OSLC) [22] might also help to form open modeling community ecosystems. For example, one can identify how MDD can contribute to system development in emerging areas such as digital twins [23], system engineering [24] and AI by creating suitable repositories dedicated to these fields. These repositories could be highly popular for engineers working in these areas.

## V. User Perspective

The reasons why MDD has not succeeded as expected in the nineties, were discussed in several insightful studies, see, e.g., [25]–[27]. One factor seems to be the significant effort needed to make modeling work, see [26]. Moreover, user experience showed, that graphical representations of models can be quite hard to create, navigate, process, and exchange, in particular, when the models are large, see [25]. In particular, many existing MDD-tools are considered to be too time-consuming and complex in practice, while the benefits of using them are not fully understood. The tool vendors should therefore try to comprehend how their tools are effectively used. That will help them to address the challenges faced by the users. For instance, the provision of model repositories containing meaningful and easily reusable models of often needed system functionality, that was already mentioned in the context of model interchange, might be very attractive for many users.

While methodologies are seen as less problematic, the main difficulties with MDD seem to be caused by deficiencies of the tools and notations like poor debugging support, as we discussed in Sect. IV. Altogether, many practitioners report a lack of knowledge and skills as a barrier preventing a more intensive use of MDD [4], which calls for a better assistance by development experts in the own organization and the tool vendors.

## VI. Other Approaches

In the following, we introduce some other approaches, that bear certain resemblance to SDL-based MDD as used in SISU.

### A. Low Code

Rymer and Appian define Low Code as "products and/or cloud services for application development that employ visual, declarative techniques instead of programming and are available to customers at low- or no-cost in money and training time to begin, with costs rising in proportion of the business value of the platforms" [28]. Some of the goals of Low Code and the means to achieve them, overlap with those of MDD, e.g., the facilitation of system development through code generation [29]. However, Low Code is more specific than MDD, since it primarily focuses on facilitating the creation of certain applications. In contrast, MDD concepts and techniques can be applied during the **entire** software lifecycle, and a broad range of software engineering tasks, e.g., requirements, design, coding, testing, deployment, maintenance, evolution, migration, design recovery in legacy code, are supported.

Since Low Code offers fewer features than MDD, one could also argue that it is a subset. For instance, Low Code is only centered on the creation of software, while MDD also supports its manipulation, translation, and the analysis of models. Thus, MDD can support many software- or system-related tasks through a range of different higher level artifacts, that can be expressed in different, possibly domain-specific and custom-made, languages. Further, these artifacts can be used by different tools for different purposes. MDD has even been applied to writing grant proposals which, in the end, has similarities with creating and maintaining documentation, compliance reports, manuals, etc., see [30]. The relations between Low Code-based development and MDD are discussed in greater detail in [31].

### B. Shift Left

According to [32], Shift Left testing means to start with testing as early as practicably possible in the software lifecycle. This can be easily done using tools for simulation, verification, and product generation, see [33]. The method was developed by Smith in 2001 [34]. It is popular in the DevOps area, which focuses on the automation and continuous measurement and improvement of processes. Further, it is used in hardware development. Combining Shift Left with both, MDD and Agile development is also discussed, see [32].

### C. MDD in Control Automation

In Control Automation, MDD was introduced within the standard IEC 61131-3 in 1993 as an approach to create and implement programmable controllers (PCs) quickly, effectively and at minimum cost using textual and graphical languages [35]. The approach was later extended with the publication of the standard IEC 61499 for distributed industrial automation systems. It allows engineers to create control software architecture from Functional Blocks (FB), which offer a high potential of reuse, see, e.g., [36]. The interfaces of the FBs are connected using edges, and the modeling technique has similarities to UML activity diagrams. In contrast to these, however, control and data flows are described separately.

Matlab from Mathworks has been available since 1979. It received a lot of attention in the Control Automation domain and acted as a simple matrix calculator. Later the Matlab developers introduced the graphical programming environment Simulink, that offers simulation capabilities, which increased the overall success of the tool. Thereafter, automatic code generation from embedded systems was also added, which has shown to be reliable and effective.

### D. MDD in Automotive

AUTOSAR (AUTomotive Open System ARchitecture) is an automotive and software industry standard framework as well as an open system architecture for intelligent mobility. Matlab Simulink has an AUTOSAR blockset that enables designers to create AUTOSAR components, which can be verified by simulation and automatically create target code. Another example of a popular tool-set for vehicular software engineering is the AUTOSAR extension to the tool Rhapsody by IBM Rational [37]. It utilizes UML class diagrams, that allow developers to exemplify inheritance, aggregation, and composition of other models, which makes them attractive to describe aspects of vehicles. Both extensions have proven to be good means to enable, accelerate, and simplify the development process of automotive software, since they allow developers to focus on creating robust and efficient applications, that meet today's demanding industry requirements.

### E. Reactive Blocks

This is a UML-based MDD technique for distributed reactive systems [38]. It was developed at NTNU within a Ph.D. project [39] under the name SPACE and later commercialized. Reactive Blocks allows for the separate specification of often reusable sub-functionality in special models. These so-called building blocks can be easily composed to more comprehensive blocks or system models, such that a complex system description can be specified as a hierarchy of relatively small blocks. Moreover, this concept makes it possible to create repositories for relevant application domains, that consist of reusable functionality, which can be seen as a first step towards model interchange, as discussed in Sect. IV.

In the building blocks, behavior is modeled by UML activity and state machine diagrams. Since these diagrams were not provided with a formal semantics by the OMG, an independent semantics was defined for them [40]. It was influenced by the semantical foundation, also used in SDL. With the exception of system descriptions, each block is further accompanied with an External State Machine (ESM), a behavioral interface [41]. This is an abstract state machine, that allows a user to combine a building block with its environment without having to understand the details of its functionality.

The definition of a formal reactive semantics for the diagrams made it also possible to build a model checker into the tool-set [38]. It allows the tool-users to verify automatically, that the models fulfill certain correctness properties like the compliance of building blocks with their ESMs. Moreover, the tool contains a code generator from the system descriptions to executable Java code [42].

Unfortunately, Reactive Blocks could not attract a sufficient number of users to make it commercially viable, such that the company marketing it had to be terminated over time due to the lack of investments. This is another example of the missing attractiveness that MDD faces nowadays.

## VII. Formal Methods

A clear benefit of Model-Driven Development (MDD) is the fact that the models can be based on formal methods. That allows us to verify formally, that they fulfill certain properties, which are important to guarantee a high system quality. Applying formal methods makes it also possible to carry out proofs, that a more detailed model is a correct refinement of a more abstract one. Thus, MDD can incorporate techniques such as model checking and simplification of design through using abstractions. Further, a formal semantics is necessary for the automatic generation of compilable and executable code

from the models. In consequence, the use of formal methods leads, in general, to a better quality of the created software. Hence, it is recognized as good software practice to utilize formal methods, when the software to be developed needs to meet high quality expectations.

Nevertheless, many engineers are reluctant to use formal methods for verification purposes, since they are perceived as highly complex mathematics, and the users fear that working with formal techniques is difficult and time-consuming. In Sect. IX, we discuss possible reasons for this disinclination.

A possibility to mitigate this reluctance, is to find ways to make the use of formal methods easier for the engineers. For instance, one could employ so-called lightweight formal methods, that are relatively easy to handle and therefore also suited for non-experts, see, e.g., [43]. The best way, however, is to integrate formal methods into the MDD-tools in a manner that they are invisible for the tool-users, see [44]. The integral model checker in Reactive Blocks [38] is a good example.

## VIII. SISU Goals Evaluated

In the following, we look again at the four specific goals of interest for the SISU project introduced in Sect. II, and discuss if and to what degree these goals were effectively reached:

1) **Correct system quality:** As mentioned above, a major benefit of modeling is to be able to think ahead, be prepared, and make sure that one knows, what to do before actually doing it. This is achieved by having the opportunity to develop systems at different levels of abstraction. Moreover, formal verification and automatic generation technology can be used to refine from one level to the next one. These properties allow engineers to keep a better overview and therefore to find design errors early, which, in general, leads to better system quality than without applying MDD. Evidence is the significant number of tools developed in the SISU project, that are still in use, see Tab. III.
2) **Shorter time to market:** If MDD is supported by a prototype process, it has indeed the potential to shorten time to market. However, the development time is not always shorter, since the efforts for verification and validation cost a fair amount of time. Nevertheless, the better quality of the developed systems will vastly shorten the time for finding and mitigating development errors, which can have a positive impact on the time needed to bring a product to the market. In consequence, it is not clear if the average time to market is shorter or longer by using MDD.
3) **Lower development costs:** The significant number of products built within the SISU project that are still in operation, indicates that systems engineered based on MDD usually endure longer in the market. This indicates that, with the correct quality, the Return On Investment (ROI) can provide an improved product life cycle and, as a result, a higher profitability. In summary, MDD may not reduce the overall development costs, but contributes to significantly reduced life cycle costs.
4) **Improved project control:** Table II depicts that, in 1992, the project management of all companies participating in SISU either depended highly on individuals (level 1) or just used basic procedures to monitor cost, progress, and functionality (level 2). Until 1996, all 15 companies managed to improve their management, mostly by one level. While some SISU participants were satisfied by adding some basic procedures (level 2), most of them defined and standardized more in-depth procedures and integrated them in their defined product development processes, that covered both, management and technical development throughout the particular organization (level 3). Interestingly, with a single exception, all companies either kept their standard from 1996 until 2016 or even improved it. Thus, evidence suggests that the use of MDD stimulates the definition of procedures, which contributes to a more organized management of the product engineering process.

In summary, MDD can have a positive impact on quality, development cost, and project control, while the goal of reaching a shorter time to market is ambiguous. However, these goals may be also achieved without MDD, which makes the effective contribution of MDD unclear. While, in general, better quality is recognized to be a result of MDD, shorter time to market and lower developments costs are usually not.

## IX. Recommendations

As already elaborated, Model-Driven Development (MDD) supports the use of higher abstraction levels in program development. This makes the application of suitable tools for simulation, system analysis, verification, and automatic code generation possible. Unfortunately, however, many programmers do not seem to use models during the program development, while modelers, also called architects, appear to be satisfied with models that are only correct to a certain degree. In consequence, both user groups are not willing to fully utilize the advantages provided by MDD. In spite of this quite unsatisfactory situation, the panel members at SAM2023 see a bright future for MDD, if and when certain ambitious goals are met. We will introduce these goals in the following.

### A. Tool Support

As discussed in Sects. IV and V, the tool-developers should align their tools more with the expectations of the engineers. In particular, the wide gap between the MDD languages and the tools supporting them needs to be bridged better. This will help the users to adjust modeling and programming more easily by using modeling frameworks in programming languages. To achieve this, we need tools allowing their users to create collections of executable models such as frameworks of finite state machines, that make the specification and verification of the architecture of a system possible. Then, the source code effectively is the model, which helps us to avoid round trip engineering, see Sect. IV.

Most important for the adoption of MDD, however, is to train and educate the tool users in order to increase their

abilities to apply the tools effectively. Training should target designers and architects as well. Also, the goal of the training should not just be the more effective use of tools, but also an appreciation of and ability to: 1) think abstractly, 2) take a "whole system view" that encompasses end users, and 3) reliably "transfer design intent" [25]. Further, long term vendor support and also advice management are also desirable. Nevertheless, it is not clear, how that can be aligned with today's open-source community approach, according to which software should be (nearly) free.

### B. User Experience and Human Factors

Users shall be encouraged to use MDD efficiently, sell the generated high quality products, and feel, that applying models can be real fun. In consequence, they need to gain confidence, that correct and robust product code can be engineered using MDD. As sketched in Sect. VII, this can be achieved by providing the users with techniques and tools, that allow them to carry out formal-based analysis and code generation without being bothered with complex formalisms.

Besides the availability of powerful and highly automatic tools, user-friendly and scalable support for hybrid textual and graphical modeling is essential, too. Therefore, creators of MDD building techniques and tools should put particular emphasis on user experience and human factors. For instance, one could learn from Apple, which centered many of its products around user friendliness. Moreover, fundamental interdisciplinary research should be conducted in order to find out, how effective (graphical) modeling languages should look like, since we do not have a good enough understanding, what kinds of concrete syntax developers find intuitive (see [45]).

### C. Education

Another important goal is that practitioners should be better trained to gain the knowledge and skills necessary to use MDD effectively and to apply formal methods. That would likely be another relevant keystone to ease their reluctance against applying formal methods. An important question in this regard is if today's university education fails to present formal methods in a way, that attracts engineers instead of scaring them off. As discussed above, there are possibilities to make the use of formal methods in software development easier. Introducing such technologies should be part of the curriculum. Moreover, it should be better communicated to students that it is very important to focus on the end user needs in a software project instead of just looking at technology issues, see [25].

A possible way to get answers to these questions is to address the education institutions, universities, technical universities and colleges, about how these institutions teach modeling and programming as well as the corresponding formal methods. Moreover, it is interesting, how they adapted their education on system development in the last 20 or 30 years. We may start by interviewing authors with a university background, who have submitted papers to relevant conferences like MODELS, SAM, or the SDL Forum.

### D. Research Funding

In spite of the fact, that there are enough aspects of MDD that still need to be researched, the public support for R&D projects concerning MDD seems to have dropped in the last 10 to 15 years. For example, it is not clear if a valuable project like SISU would have been funded today. Therefore, we should also address the issue with the funding organizations and other public communities and ask about their policies with respect to improving software quality. Moreover, we should discuss, which R&D tasks are most pressing, how they can be financed, and who is best suited to tackle them.

### E. UML Community

The rise of UML 2.0 over the years negatively impacted the SDL-based cooperation, that was established during SISU. A remarkable quote during the assessment after 20 years was: "*UML has basically killed all other similar technologies on its way. 15 years later (2016), UML is still far from what other technologies such as SDL could do*" [3]. UML was expected to become the better modeling language. However, this prediction did not come true yet, since SDL Z.100 is still better suited with respect to verification and code generation of formal models than UML. While the SDL community is fully aware of this fact, apparently UML-centered developers as well as programmers in general either do not know it or simply do not care.

The SDL community and, in particular, the SDL Forum Society may benefit from establishing a cooperation with the OMG, since there is nearly no competition between the two groups anymore. This may contribute to increased knowledge of the benefits of modeling, verification and automatic code generation to software quality and life cycle costs. To make this task simpler, we can rely on the UML profile Z.109 of SDL 2010, since it gives evidence, that SDL and UML complement and overlap each other.

Also a closer cooperation with groups like developers of control systems hardware, automotive, or DevOps systems, who all seem to use model-based technology to a greater extent than software engineers, might be helpful.

### F. Standardization

The more intense application of model-based technology by other communities seems, at least in parts, driven by the fact, that the engineers are simply obliged to use model-driven technology. In control system engineering, for instance, standards like IEC 61131-3 and IEC 61499, see Sect. VI-C, are followed during the creation of control software for technical systems, that often require the use of models in the development process, see, e.g., [36]. These standards are constantly maintained and, when necessary, revised.

In contrast, the ETSI standards, guidelines, and handbooks appear to be out-of-date. The SDL Forum Society could contribute to update them, e.g., by adding the newer developments in Methods for Testing and Specification (MTS) to the standards. It is also interesting to involve the International Telecommunication Union (ITU), which should be a driving

force for the use of MDD in the development of communication protocols and distributed systems, but today does not seem to support the use of MDD in a similar fashion anymore.

As outlined in Sect. IV, an essential aspect to improve the user experience is model interchange, i.e., the ability of software engineers to handle different development steps by varying tools, such that a particular task can be carried out with the best suited tool. This could be achieved through the definition of comprehensive, standardized, and universally supported formats. To achieve that, the tool vendors should make common efforts to normalize such formats in standardization bodies like ITU or ETSI.

### G. SISU Legacy

The assessment of SISU 30 years after finishing the project is coming up. A central goal of this new assessment shall be to learn more about the current state of the SISU developments still used in 2016, see Tab. III, and the future plans of the corresponding companies. We already learned, that some of these products are not maintained anymore and are planned to be phased out in the next years. For instance, one product is scheduled to be taken out in 2030 or 2031. It is therefore highly interesting, by what kind of technology these systems shall be replaced with, how the future developments shall be carried out, and what kind of features MDD-based technologies should have in order to be considered again.

## GLOSSARY

*Modeling Languages:*

| | |
|---|---|
| CMM | – Capability Maturity Model |
| DSL | – Domain Specific Language |
| ER | – Entity Relationship diagram |
| MSC | – Message Sequence Chart |
| SA/SD | – Structured Analysis / Structured Design, 70/80´s modeling, see [6] |
| SDL | – Specification and Description Language |
| SOM | – Strukturert (SDL) Orientert Modeling |
| UML | – Unified Modelling Language |

*Modeling Tools:*

| | |
|---|---|
| DaSOM | – Digital version of SOM |
| EA | – Enterprise Architect, UML tool |
| Eclipse | – Java IDE |
| Geode | – SDL tool from Verilog (later Telelogic) |
| IntelliJ | – Java IDE |
| OOram | – Object Oriented Role Analysis Modelling |
| Reactive Blocks | – UML-based development tool for Java code |
| Tau SDL Suite | – Former SDT |
| SDT | – SDL Design Tool, IBM Rational, former TeleLogic |
| Tau G2 | – A UML tool from IBM Rational, former Telelogic supporting code generation and model verification |

*Run Time Systems (RTOS):*

| | |
|---|---|
| CRS | – CHILL Runtime System |
| RS | – Runtime System |
| TST | – Telox SDL (runtime) Tool |

*Code Generators:*

| | |
|---|---|
| DaSOM/PG | – DaSOM Program Generator |
| OOram/PG | – OOram Program Generator |
| ProgGen | – Program Generator, Sintef Delab |
| SDT/PG | – SDT Program Generator |

*Programming Languages used in SISU:*

| | |
|---|---|
| Ada | – Ada programming language |
| C | – The C programming language |
| CHILL | – CCITT High Level Language, Z.200 |
| Occam | – CSP based transputer lang |

*Products:*

| | |
|---|---|
| ACEcom | – Air traffic Control system (part of SISAM), Alcatel/Thales |
| CPX | – Trunk- and Access-Switch for Military Tactical network KDA (NFTE) |
| Garex | – Gustav A Ring Exchange |
| ICX Alphacom | – Former Stentofon product |
| MLT | – Multi Linje Terminal, KDA (NFTE) |
| MRR | – Multi Role Radio, Alcatel/KDA (NFTE) |
| Nor 121 | – A hand-held sound analyser, Norsonic |
| NSS | – Nodal Switching System, Alcatel/Thales |
| SISAM | – Sikkert Samband (Secure Communication) for Air Traffic Control (luftromkontroll), Alcatel/Thales |
| VCS | – Voice/Vehicle Communication System, Alcatel/Thales |

*Standardization Bodies:*

| | |
|---|---|
| ETSI | – European Telecommunications Standards Institute |
| IEC | – International Electrotechnical Commission |
| ITU | – International Telecommunication Union |
| OMG | – Object Management Group |

*Companies:*

| | |
|---|---|
| Autronica | – Fire alarm products |
| Indra Navia | – former Garex |
| KDA | – Kongsberg Defence and Aerospace, former NFTE |
| NFTE | – Norsk Forsvars Teknologi Ericsson |
| Norsonic | – Sound analyzing products |
| Thales | – former Alcatel Defense Division |
| Xenitel | – former Stentofon |

## REFERENCES


[1] D. C. Schmidt, "Guest Editor's Introduction: Model-Driven Engineering," Computer, vol. 39, no. 2, pp. 25–31, 2006.

[2] S. E. Ellevseth, "Threats to Model Driven Development — What happened to Eighteen Companies Twenty Years later," https://sdl-forum.org/Tools/Threats_2017_paper.pdf, 2016, accessed: 2024-11-12.

[3] ——, "Summary of the SISU gathering 27th September 2016," https://sdl-forum.org/Tools/SISUgatheringMinutesEng.pdf, 2016, accessed: 2024-11-07.

[4] H. Alfraihi and K. Lano, "Trends and Insights into the Use of Model-Driven Engineering: A Survey," in 15th System Analysis and Modelling Conference (SAM), published in 2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), Västerås, Sweden, 2023, pp. 286–295.

[5] O. L. Madsen and B. Møller-Pedersen, "What Your Mother Forgot to Tell you About Modeling – and Programming," in ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), Västerås, Sweden, 2023, pp. 200–210.

[6] O. Faergemand and A. Olsen, "Introduction to SDL-92," Computer Networks and ISDN Systems, vol. 26, no. 9, pp. 1143–1167, 1994.

[7] M. C. Paulk, C. V. Weber, B. Curtis, and M. B. Chrissis, The Capability Maturity Model: Guidelines for Improving the Software Process. Addison-Wesley Professional, 1994.

[8] H. P. Dahle, M. Löfgren, O. L. Madsen, and B. Magnusson, "Mjølner — A Highly Efficient Programming Environment for Industrial Use," Lund Institute of Technology, Sweden, Mjølner Report 1, 1986, also in Proceedings of the 15th Simula Users' Conference, Jersey, 1987.

[9] J. L. Knudsen, M. Løfgren, B. Magnussen, and O. L. Madsen, Object-Oriented Software Development Environments: The Mjølner Approach. Prentice-Hall, 1994.

[10] Ø. Haugen, B. Møller Pedersen, and T. Weigert, "Structural Modeling with UML2.0," https://www.uio.no/studier/emner/matnat/ifi/INF5120/v17/resources/papers/haugen-mp-weigert-umlforreal.pdf, University of Oslo, Educational Paper, 2000, accessed: 2024-11-12.

[11] Ø. Haugen, "Comparing UML 2.0 Interactions and MSC-2000," in System Analysis and Modeling (SAM), ser. LNCS, no. 3319. Springer-Verlag, 2005, pp. 65–79.

[12] R. Bræk and Ø. Haugen, Engineering Real Time Systems: An Object-Oriented Methodology Using SDL. Prentice Hall, 1993.

[13] SDL Forum Society, "About TIMe - The Integrated Method," https://sdl-forum.org/Tools/TIMe/HTML/elb/mainpage/mainpage_t01.htm, 1999, accessed: 2024-09-21.

[14] C. Hoare, "Record Handling," Algol Bulletin, vol. 21, pp. 39–69, 1965.

[15] O. L. Madsen, B. Møller-Pedersen, and K. Nygaard, Object-Oriented Programming in the BETA Programming Language. Addison Wesley, 1993.

[16] R. Speed, "Study finds 268% Higher Failure Rates for Agile Software Projects," https://www.theregister.com/2024/06/05/agile_failure_rates/, 2024, accessed: 2024-11-12.

[17] A. Kumar, "What is Callback Hell?" https://medium.com/@avinashkumar151199/what-is-callback-hell-a4594574e3c7, 2023, accessed: 2024-12-02.

[18] ITU, "Recommendation Z.106 (06/21)," https://www.itu.int/rec/T-REC-Z.106-202106-I, 2021, accessed: 2024-09-22.

[19] OMG, "About the XML Metadata Interchange Specification Vesion 2.5.1," https://www.omg.org/spec/XMI, 2015, accessed: 2024-09-22.

[20] U. Zusāne, O. Nikiforova, and K. Gusarovs, "Several Issues on the Model Interchange Between Model-Driven Software Development Tools," in Tenth International Conference on Software Engineering Advances (ICSEA), Nov. 2015.

[21] A. Benelallam, A. Gómez, M. Tisi, and J. Cabot, "Distributed Model-to-model Transformation with ATL on MapReduce," in 2015 ACM SIGPLAN International Conference on Software Language Engineering, ser. SLE 2015. ACM, 2015, pp. 37–48.

[22] OASIS Open Project, "Open Services for Lifecycle Collaboration," https://open-services.net/, 2008, accessed: 2024-12-02.

[23] M. Attaran and B. G. Celik, "Digital Twin: Benefits, Use Cases, Challenges, and Opportunities," Decision Analytics Journal, vol. 6, no. 100165, 2023.

[24] INCOSE, "SE Vision 2035," https://www.incose.org/publications/se-vision-2035, 2024, accessed: 2024-09-22.

[25] B. Selic, "What will it Take? A View on Adoption of Model-based Methods in Practice," Software & Systems Modeling (SoSym), vol. 11, no. 4, pp. 513–526, 2012.

[26] J. Whittle, J. Hutchinson, M. Rouncefield, H. Burden, and R. Heldal, "Industrial Adoption of Model-driven Engineering: Are the Tools Really the Problem?" in 16th International Conference on Model-Driven Engineering Languages and Systems (MODELS). Miami, FL, USA: Springer-Verlag, 2013, pp. 1–17.

[27] G. Mussbacher, D. Amyot, R. Breu, J.-M. Bruel, B. H. C. Cheng, P. Collet, B. Combemale, R. B. France, R. Heldal, J. Hill, J. Kienzle, M. Schöttle, F. Steimann, D. Stikkolorum, and J. Whittle, "The Relevance of Model-Driven Engineering Thirty Years from Now," in Model-Driven Engineering Languages and Systems (MODELS). Springer International Publishing, 2014, pp. 183–200.

[28] J. Rymer and K. Appian, "The Forrester Wave™: Low-code Development Platforms for AD&D Pros, Q4 2017," Forrester Research, Cambridge, MA, USA, report 120, 2017.

[29] M. Hirzel, "Low-Code Programming Models," Communications of the ACM, vol. 66, no. 10, pp. 76–85, 2023.

[30] D. S. Kolovos, N. Matragkas, J. R. Williams, and R. F. Paige, "Model Driven Grant Proposal Engineering," in 17th International Conference on Model-Driven Engineering Languages and Systems (MODELS). Valencia, Spain: Springer-Verlag, 2014, pp. 420–432.

[31] D. Di Ruscio, D. Kolovos, J. de Lara, A. Pierantonio, M. Tisi, and M. Wimmer, "Low-code Development and Model-driven Engineering: Two Sides of the Same Coin?" Software and Systems Modeling (SoSyM), vol. 21, no. 2, pp. 437–446, 2022.

[32] D. Firesmith, "Four Types of Shift Left Testing," https://web.archive.org/web/20150905082941/https://insights.sei.cmu.edu/sei_blog/ 2015/03/four-types-of-shift-left-testing.html, 2015, accessed: 2024-11-29.

[33] B. Hughes, "What's In A Name: Shift-Left and the Quest for Quality-Driven Development," https://www.mabl.com/blog/whats-in-a-name-shift-left-and-the-quest-for-quality-driven-development, 2022, accessed: 2024-11-29.

[34] L. Smith, "Shift-Left Testing," Dr. Dobb's Journal, vol. 26, no. 9, 2001. [Online]. Available: https://www.drdobbs.com/shift-left-testing/184404768

[35] K. Thramboulidis and G. Frey, "An MDD Process for IEC 61131-based Industrial Automation Systems," in International Conference on Emerging Technologies and Factory Automation (ETFA). Toulouse, France: IEEE, 2011, pp. 1–8.

[36] A. Zoitl and R. Lewis, Modelling Control Systems using IEC 61499. IET, 2014, no. 11021.

[37] "AUTOSAR Modeling," https://www.ibm.com/docs/en/engineering-lifecycle-management-suite/design-rhapsody/10.0.1?topic=modeling-autosar, 2024, accessed: 2024-11-29.

[38] F. A. Kraemer, V. Slåtten, and P. Herrmann, "Tool Support for the Rapid Composition, Analysis and Implementation of Reactive Services," Journal of Systems and Software (JSS), vol. 82, no. 12, pp. 2068–2080, 2009.

[39] Frank Alexander Kraemer, "Engineering Reactive Systems: A Compositional and Model-driven Method Based on Collaborative Building Blocks," Ph.D. dissertation, Norwegian University of Science and Technology, Trondheim, 2008.

[40] F. A. Kraemer and P. Herrmann, "Reactive Semantics for Distributed UML Activities," in Formal Techniques for Distributed Systems (FMOODS/FORTE), ser. LNCS, no. 6117. Springer-Verlag, 2010, pp. 17–31.

[41] ——, "Automated Encapsulation of UML Activities for Incremental Development and Verification," in Model Driven Engineering Languages and Systems (MODELS), ser. LNCS, no. 6117. Springer-Verlag, 2009, pp. 571–585.

[42] F. A. Kraemer, P. Herrmann, and R. Bræk, "Aligning UML 2.0 State Machines and Temporal Logic for the Efficient Execution of Services," in On the Move to Meaningful Internet Systems (CoopIS, DOA, GADA, and ODBASE), ser. LNCS, no. 4276. Springer-Verlag, 2006, pp. 1613–1632.

[43] A. Zamansky, M. Spichkova, G. Rodriguez-Navas, P. Herrmann, and J. O. Blech, "Towards Classification of Lightweight Formal Methods," in 13th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE). Funchal, Portugal: SciTePress, 2018, pp. 305–313.

[44] J. Rushby, "Disappearing Formal Methods," in 5th IEEE International Symposium on High Assurance Systems Engineering (HASE), Albuquerque, NM, USA, 2000, pp. 95–96.

[45] D. Moody, "The "Physics"' of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering," IEEE Transactions on Software Engineering, vol. 35, no. 6, pp. 756–779, 2009.