SAM 2023 Panel Discussion : What happens with MDD after 20 years

Summary. Introducing new methods and tools that support higher abstraction levels, simulation, system analysis, verification and automatic code generation in software development was expected to improve profitability, lower development cost, improve quality, shorten time to market and increased controllability. What happens with MDD after 20 years, is inspired by an assessment performed in 2016, 20 years after the SISU project was terminated in 1996 and is the background for the panel discussion at SAM 2023.

To explore this issue we put together a panel at SAM 23 representing different views on MDD, from programming vs modeling, tool support, graphical modeling and agility, a survey and the SISU project. The following participants presented their views:

Ole Lehrman Madsen, from Aarhus University, presented his highlights from his KeyNote Speech about early and future days of modeling and programming. Emmanuel Gaudin, CEO of PragmaDev, and the chair of SDL Forum Board, presented his views on tool support. Juergen Dingel, from Queens University Ontario presented his views on MDD graphical modeling, agility and open systems. Hessa Alfraihi, from KingstonCollege London, presented highlights from the paper, "A survey on Trends and Insights into the Use of Model-Driven Engineering". Myself, Stein Erik Ellevseth, former ABB AS, Ericsson AS, Alcatel STK, now Retired, represented the SDL view and inspiration from SISU project (89 – 96).

An important question we wanted explore was to what degree MDD contributed to improved profitability in the development and maintenance of embedded software in companies, and the specific goals of 1) correct system quality, 2) shorter time to market, 3) lower development cost and 4) improved project control. Here the goal 1, 3 and 4 are very clear results from MDD notified through the panel. Shorter time to market is still questionable, because time spent in early phases to ensure quality, not always contributed to shorter time to market. However, since products appeared to endure longer in the market, as shown in the products from the remaining 6 SISU participants, this could indicate that with the correct quality, goal (1), the ROI, return on investment, can provide improved product product life cycle profitability. In summary, longer product life cycle may not decrease development cost, goal (3), but for sure significantly contributed to decreased life cycle cost.

If there were any conclusion from the discussion, was that the benefits of MDD still stands, and the future appears bright. But there are threats and obstacles that still needs to be addressed:

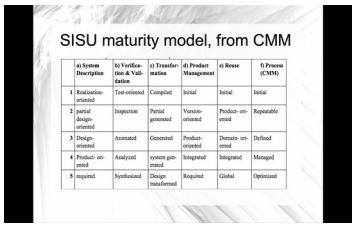
- Methodology enthusiast and improvement agents in the companies went missing, moved or lost interest, maybe due to low management support during the time after the termination in 1996. "We spent a lot of money on SISU, now we expect payback…". The national industrial network that were created during the SISU cooperation was not taken well care of neither on national level nor inside the companies.
- Initially OO languages Simula, Beta and Smalltalk was intended to support modeling as well as programming. However, modeling was early separated from coding, like when SA/SD diagrams started to influence software development. Program execution is a model, but most modeling tools today have limited ways of debugging the execution of a system. Graphics are important initially, later source code rules (check recording).
- Significant benefits of modeling has been broken down by Agile by pushing non verified decisions in early iterations and to do error corrections in the following iterations. Cost of modeling

tools has decreased and tool vendors economy declined, which made support and functionality suffer. Today licenses for training, prototyping and proof-of-concept are more or less free.
- Model interchange is lacking and comprehensive, standardized and universally supported formats is needed. Open model repositories/libraries is necessary, on which the tools vendors should agree on, through common efforts and utilized in standardization bodies. More emphasis on user experience and human factors should be better shared when building techniques and tools.
- Adoption of MDE should be facilitated by improving the appropriateness and flexibility of MDE tools. Training and education to practitioners is needed to increase MDE knowledge and skills. Practitioners expect long term vendor support is needed. However, there is a question on how that can be possible with open source community when software is expected to be free. Can R&D be better utilized to mitigate threats to MDD.

Stein Erik Ellevseth, SISU and SDL perspective. Retired

The SISU project, Support to Integrated System Utvikling (Development) was a Norwegian national project in two phases from 1989 – 1992 (initially 28 mill nok) which focused on the use of tools to be integrated based on SDL, SA/SD, ER, OSDL, and to support code generation and followed-up by SISU II 1993 – 1996 (51 mill nok) which focused on business improvements using tools from SISU I. 19 companies including 2 tools vendors and 4 research organizations were involved. In addition there were contributions through the EU program ESPRIT. The project was based on notations that ruled through 70s and 80s, including SA/SD, ER, SDL, Simula and OMT. OMT was one of the forerunners of UML which was introduced in the mid 90s. SISU contributed significant to SDL support, simulation, code generation and runtime support for SDL systems for the 19 companies. The 4 research institutes acted as technology providers, trainers and mentors for the companies through excellent management by Geir Melby, former Telox.

To answer this was by measuring the maturity level in different disciplines during the project and assessment later. A model inspired by CMM, the Capability Maturity Model, was created for the disciplines System Description, Verification & Validation, Transformations, Product Management, Reuse and Process. Each company was given a score in each of the discipline at the start of the project and at the end of the project. A plan was created to achieve this goal.



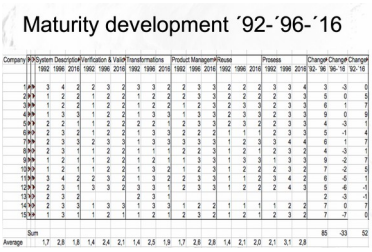
<Figure SISU Maturity model>

The technology had its root in companies and the Technological University of Trondheim (NTH/NTNU) and the Simula environment at Norwegian Computing Center NCC (Norsk Regnesentral, NR) in Oslo. SISU contributed to SDL ´92 Object orientation in SDL which also made the foundation of UML 2.0. A SISU Textbook, Engineering Real Time Systems, Haugen/ Braek ISBN 0-13-0344448-6 was published in 1993 and is still being used. In SISU II we created an electronic book, ELB, available at SDL Forum Society, https://sdl-forum.org/Tools/TIMe/HTML/elb/mainpage/mainpage_t01.htm

SISU project proved the above expectations to be achievable in many cases. Assessing the SISU project initiative 20 years later formed the bases for this claim. The SISU initiative was regarded as a great success among the participants and in academia thirty years ago. It gave the MDD community object-orientation in SDL (Specification and Description Language, Z.100), MSC (Message Sequence Charts, Z.120) and later UML 2.0 (Unified Modeling Language from OMG, Object Management Group). The methodologies and tools introduced are still in use, proving that the technology harvested from the SISU project contributed to sustainable software development and economic benefits for the participating companies. Further, people who worked in these companies matured the industry, spread the methodologies, tool and standards by influencing the companies they later moved to.

After the termination in 1996 methodology enthusiast and improvement agents in the companies went missing, moved or lost interest. The national industrial network that were created during the SISU cooperation was not taken well care of neither on national level and inside the companies. There were different views of the benefits of a higher abstraction level in the companies a) Resistance to changes and b) Real programmers: source code rules, was and still is an “obstacle”. Tool vendors hesitated in applying new constructs with object-orientation from the standardization of SDL. Missing target support in the language and the tools, to faster adapt to new hardware. Technical issues beyond methodology, hardware issues, software issues (embedded challenges with real time performance, etc). Market situation changed, and when the oil price dropped to 15 USD per barrel in 1994 may have created difficult situation for many companies.

In preparation of the panel discussion, several of the SISU industrial participants that were assessed in 2016 regarding their use of MDD and tools and if the products developed during SISU still were in operation. Of the participating 19 SISU companies, 6 companies today still use results today in products still in service after 30 years.



SISU SDL Support 2016 2023

	SDL tool	Codegenerator	Language	RTOS SDL sch	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
				TST v.	Garex 220 terminal
	SDT / Tau	SDT/PG / ProgGen	C	TST v ?	Garex 220 MPC posi
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
KDA (NFTE)	Geode -> SDT 4.6	ProgGen	C	Inhouse	CTX

<Figure SISU Maturity 1996→2016> <Figure SISU 6 remaining companies 2023, green is confirmed>

More details can be found in the summary from the SISU assessment meeting is available :

https://sdl-forum.org/Tools/SISUgatheringMinutesEng.pdf

Quota from the society: “ *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.”*

Ole Lehrman Madsen, Programming & Modelling view from Simula and Beta; Aarhus University, Denmark

Modeling was early separated from coding, like when SA/SD diagrams started to influence software development. Initial diagrams is good to provide a good understanding of structure and abstraction early in product development is the road/key to success. OO modeling as a skill started to emerge in the mid-80'es and later OO modeling was separated from OO programming with Java and C++ in the 80s. OO terms can date back to the greek philosophers (Platon: the idea and the individual). Is there a lack of understanding of modeling in the software community, that will cause the products to suffer good quality models?

SIMULA started as a language for programming and modeling; inspired by Hoare´s paper on Record Handling, with emphasis on modeling. The follow-up language, Beta, was also intended to support modeling as well as programming. However Beta, like SIMULA, never became a mainstream language. Smalltalk was the first language to make OO popular and when the OO mechanism of SIMULA were implemented as an extension of C in C++ and later Java, C#, Python, JavaScript and others became mainstream used by most programmers in the world.
The OO programming community focus on making programs and appreciates the benefits of reuse. However, reuse is only a side-effect that became an important aspect to OO programmers. Programmers seem to model, not explicitly, and there is little attention to modeling in their programs, like mirroring the environment into their programs, in domain, system and environment objects and types.
Since program execution is the model, programming/programmers won the battle of influence of software development. However, since programming "is" modeling, a modeling approach for programming seems to be missing. Further, since program execution is the model, there is a challenge on how to describe and observe the behavior during runtime execution. MSCs are only snapshots which do not give the complete picture.
Future is still bright, and it is possible to set ambitious goals for future of MDE. First, the gap between tools and languages is still too wide and needs to be bridged as the same for modeling and programming. In the 90´s we experienced that the SDL tools were syntactically driven, instead of compiling to identify errors, you were syntactically guided through development and to some degree semantically too. An explicit conceptual framework for OO modeling is needed to support modeling in the programs. This is also the case for OO programming as for Beta, where such a conceptual framework was developed. Beta focused on the program execution and visualize interaction and behavior of the objects in the execution of the program. Most modeling tools today have limited ways of debugging the execution of a system. When we are able to improve debugging, we may have a winner!

BetaBook: Ole Lehrmann Madsen, Birger Møller-Pedersen, and Kristen Nygaard. 1993. Object-Oriented Programming in the BETA Programming Language. Addison Wesley, Wokingham, England.

Emmanuel Gaudin, Tools perspective; CEO PragmaDev

One of the major benefits of modeling is to be able to think ahead, be prepared and making sure you know what to do before doing it. This is done by being able to verify the functional properties of the system model, statically and dynamically by simulation. It is software quality assurance at its best.

However, Agile broke this benefit down. Because it pushes to just go for it and do corrections in the following iterations. Agile pushes for short term solutions due to the short iteration intervals and later rely on correcting the bugs in the following iteration - if we are lucky. This can cause unintentional mistakes to "survive" development and testing, and to end up as severe bug in the product. This creates a lot of bug fixes which needs to be handled. It is as if the development goes uncontrolled ... by the flow. It appears as if this is a much more fun way to work … being the programmer/tester/developer who found and solved the defect (i.e. bug). It is like it does not matter if we do not know where we are going.

Experiences show several issues that modeling has not succeeded as expected. Models are mainly made for documentation, but the process appears to be code driven. Coders sometimes tell (middle) management "do you want paper or working code?". There is lack of interest to verify models. Architects (Modelers) appear to be satisfied with a model that is more or less correct. Even though sometimes developers are modelers, not all modelers are developers. Difficult decision are delayed, into the next phase without considering the costs of late bug fixing. "We fix" the bugs in the coding and test phases. It is known that a defect created in one phase can cost 10 times more to solve if it is detected and fixed in the next phase. We (They?) do not think ahead anymore, but push important decisions in front of us. We have talked about document driven vs model driven development, or should we say code-driven vs model driven. Programming/Coding today appear to be the main aspect in software development. It is true, and in the end the code is the product. While emacs appear to be the main favorite tool for many developers today, maybe the usage was due to lack of knowledge from the complexity of SDL/MDD tools. It is possible to include / add sdlang-mode in emacs which I do not know how well known it is?

The big companies, Ericsson, Alcatel/Thales, Siemens, etc. developed (modeling) tools originally for their own internal use. Gradually they changed position about sorting out what is their core business ... and started to use commercially available tools and outsourced their modeling tools; Capella (Thales), Titan (Ericsson), Erlang (Ericsson). Competition also came from UML tools (drawing tools like EA, IntelliJ,..). Is round trip engineering really needed, as many developer claim?

Cost of modeling tools started out at 100 kEUR per license in the beginning (of 90´s), which at that time was good business. Later, license costs decreased to 10kEUR, per license. Tool vendors economy declined during the 90´s, which may have influenced support and functionality. Few (none?) tool vendors did actually implement SDL92, object oriented SDL into their SDL tool set, while object oriented programming languages like C++ and Java expanded. Programming tools is also suffering, by open source products, and GCC is heavily used. It is a question whether gcc, java/jvm providing safe and secure products.

At present licenses for training, prototyping and proof-of-concept are more or less free. These tools are also in use software in products requiring security and safety.

Future is still bright, and it is possible to set ambitious goals for the future of MDE.

One of the key aspect of a model is to specify the architecture of the system. If the architecture is not correct in the first place it is costly to fix later on. That has a major impact on quality.

It is difficult to evaluate the time to market of, let's says, an agile developed product that is not fully defined, and a model based product that is fully defined. It is quite likely that the agile one will be available sooner but as you said full of bugs or with missing functionalities.

Again the life cycle costs has to be taken into consideration, not only initial development costs, but that is rarely the case. For example the cost of fixing bugs in products in service should be taken into account. The cost of maintaining the system over the years as well. I believe a properly defined system through modeling will have less bugs and will be easier to maintain.

In summary, MDE has great impact on quality, development cost and project control, while shorter time to market is questionable.

Juergen Dingel, graphical modeling, agility and open systems; Queen's University Ontario, Canada

Starting with the successes. MDE can be very effective and reducing development efforts through graphical representations of models which can greatly facilitate understanding of the product to be developed and improve communication between teams. This contributes to better quality, less defects, and longer product lifecycle.

However, there is a range of reasons why modeling has not succeeded as expected. The adoption of MDE (or lack thereof) has been the subject of insightful several studies, including [1,2], and there are great successes and failures of MDE. Why has modeling not succeeded as expected in the nineties. First of all, MDE takes effort to make work, and according to [1]. Also, Experiences show (according to [2]) that graphical representations of models also can be hard to create, navigate and process and exchange. Key issues to this appear to be poor user experience especially in the context of large models and the lack of comprehensive, standardized, universally supported formats that allow model interchange and prevent tool lock-in. Lock-in is typical for UML tools, DSL tools and Matlab Simulink. There is no common interchange formats available. Furthermore, the lack of generally accessible, public, comprehensive model repositories is holding back the community and the complicates the formation of 'open modeling community ecosystems'.

Future is still bright, and it is possible to set ambitious goals for future MDE. First of all, we need to encourage present users to be excited end enthusiastic to use it, sell it and feel that its fun and efficient to use. The user need to have confidence that the verified model generates readable and robust code to the end user products. Therefore user-friendly and scalable support for hybrid textual/graphical modeling is essential. More emphasis on user experience and human factors should be better shared when building techniques and tools. Can we learn something from Apple who invented user friendliness in its products? Model interchange through comprehensive, standardized and universally supported formats. Like in C/C++, python, etc. open model repositories/libraries is necessary, on which the tools vendors should agree on, through common efforts and utilized in standardization bodies like ITU, ETSI, and similar. ITU-T has already a recommendation for model interchange, namely the Common Interchange Format, recommendation Z.106. Are there similar in other bodies for UML and DSLs?

Conducting MDE research and identify how modeling and MDE can contribute to up-and-coming areas such as digital twins, systems engineering, and AI.

MDE has impact on quality, development cost, project control and shorter time to market. However, the use of MDE is neither necessary nor sufficient to deliver these goals. The MDD contributions above other development technologies is not clear.

References:
[1] J. Whittle, J. Hutchinson, M. Rouncefield, H. Burden, R. Heldal. Industrial Adoption of Model-Driven Engineering: Are the Tools Really the Problem? MODELS'13. 2013
[2] B Selic. What will it take? A view on adoption of model-based methods in practice. SoSyM 11 (4), 513-526. 2012
[3] F.A. Lopes, M. Santos, R. Fidalgo and S. Fernandes. Model-driven networking: A novel approach for SDN applications development. 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM). Ottawa, ON, Canada, 2015, pp. 770-773, doi: 10.1109/INM.2015.7140372.
[4] IETF. Intent-Based Networking - Concepts and Definitions, 2021. https://www.ietf.org/archive/id/draft-irtf-nmrg-ibn-concepts-definitions-05.html.
[5] International Council on Systems Engineering (INCOSE). Systems Engineering Vision 2035. 2022.https://www.incose.org/publications/se-vision-2035
[6] Martin Hirzel. Low-Code Programming Models. Communications of the ACM, Volume 66, Issue 10, pp 76–85
[7] Mohsen Attaran, Bilge Gokhan Celik. Digital Twin: Benefits, use cases, challenges, and opportunities. Decision Analytics Journal. Volume 6, March 2023.
[8] ITU. Z.106: Specification and Description Language - Common interchange format for SDL-2010. 2021.https://www.itu.int/rec/T-REC-Z.106-202106-I
[9] OMG. XML Metadata Interchange (XMI). 2015. https://www.omg.org/spec/XM

Hessa Alfraihi, MDD survey perspective; King's College London, UK

Model-Driven Engineering (MDE) has gained popularity as a software engineering approach that promises to improve productivity, profitability, software quality and reduce development as well as life cycle costs. MDE/MDD is a software development methodology that uses formal models that can be executed and verified as well as automatically generate compileable and executable code from. This provides clear benefits with MDE, i.e. better quality, etc. Focus is on model checking, automatic code generation and simplification of design through abstractions which is recognized as good software practices.

From a total of 119 participants, representing a diversed group of organizations and roles, we have received their experiences through a survey where they conclude improved efficiency by automating software development tasks, model checking, source code generation. The improved software quality further provides reliable products with less error-prone software which is proved to enhanced products lifecycles. As an example will memory-leakage usually becomes history and defects are detected early in the development cycle. Further they experience collaboration between teams MDE/MDD facilitate better understanding and communication among development teams.

However, its adoption in practice still remains limited, due to challenges with MDE tools, notations, and to some degree methods. There are indications that while MDE is used in various domains, its

adoption is still not widespread, with many practitioners are reporting lack of knowledge and skills as a barrier. While better quality is recognized, shorter time to market and lower developments cost is not clear, While methodologies are less problematic, difficulties with MDE concern mainly is caused by tools deficiencies and notations, like poor debugging support and ….?

Further is level of assistance is not well supported neither from the development organization nor the tool vendors.

Despite its benefits, formal modeling has not succeeded as expected and MDE adoption in the industry still remains limited. The challenges with tools, notations, and motivation have hindered its widespread implementation. The need to understand the current usage and challenges is more crucial than ever to facilitate the effective adoption of MDE in practice.

Future is still bright, and it is possible to set ambitious goals for future of MDE. First, our results suggest that the adoption of MDE should be facilitated by improving the appropriateness and flexibility of MDE tools and may most important to provide training and education to practitioners to increase MDE knowledge and skills. Long term vendor support is needed, as well as advice management. However, there is a question on how that can be possible with todays open source community approach that claim software should be free, which influence the tool vendors economy. However, there are no free lunch. There is always a cost to it, depending on how.

Further, the goals to future surveys should mitigate threats to validity. There is a question on how to use R&D, that can research questions that needs to be asked.

References :

SAM 23 paper no 4 & slides : Trends and Insights into the Use of Model-Driven Engineering: A Survey