Where will be MDE in 2030?

Jean Bézivin

JBezivin@gmail.com

Lecture 4: Where will be MDE in 2030?

- Introduction
- On the evolution of technology
- What to throw away?
- What to keep?
- A research agenda
- Conclusions

The present situation is not ideal

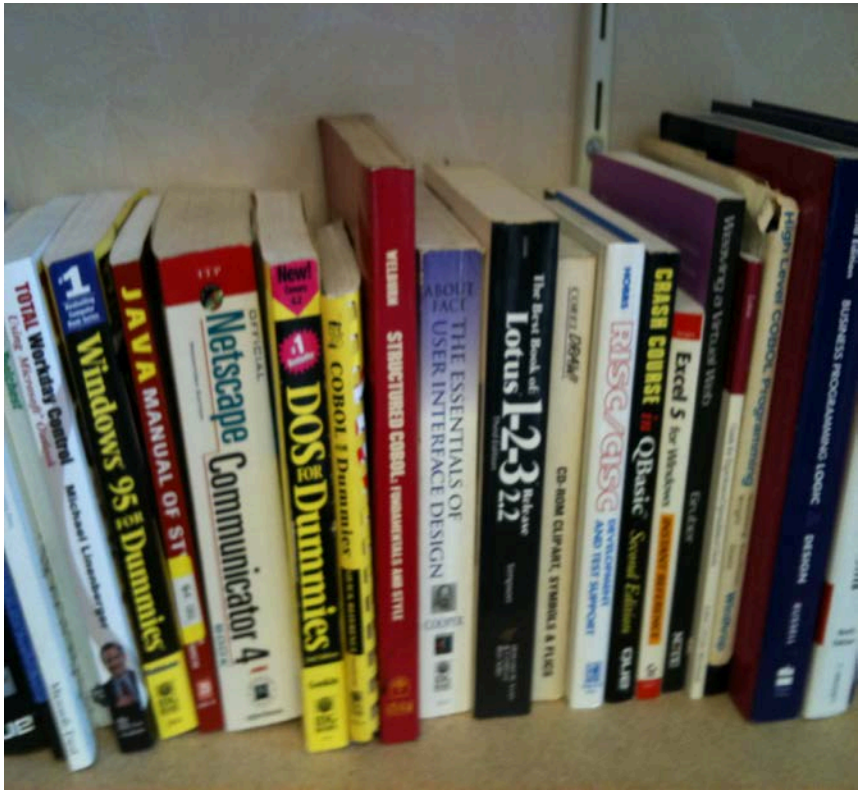
- Ideas spreading slowly
- High expectations not followed by solution delivery
  - Promises made by MDA in 2000 not delivered
- Several big companies not supporting the idea
- MDA not widely adopted by industry
- Tools lacking maturity
- Software landscape not significantly transformed (impacted?) by MDE (not yet?)
- The current iteration may not be the last one.
  - This piece of technology may need an additional cycle to mature and produce maximum impact.

Good time to reflect and react

- Interesting to understand what did work and what did not work
- Identify what are the main criticisms to MDA
- Also identify the good points
- Positive attitude
  - Reflect on the past
  - Critically analyze the present
  - Prepare for the future
- Need to separate three parts
  - Negative balance (throw away)
  - Positive balance (keep)
  - What need yet to be added (research roadmap)
- First a global look at the evolution of software technology in general

Engineering as a succession of hypes

- Is MDE to be recorded as another transient hype in the software technology history?
- Many developers' career paths follow a continuous zigzag from hype to hype.
- We need to focus more on continuity and progresses than on ruptures and failures.
- A solution that does not meet immediate success may just be premature, not always wrong.



Quotations

- ***Those who cannot remember the past are condemned to repeat it***

  George Santayana

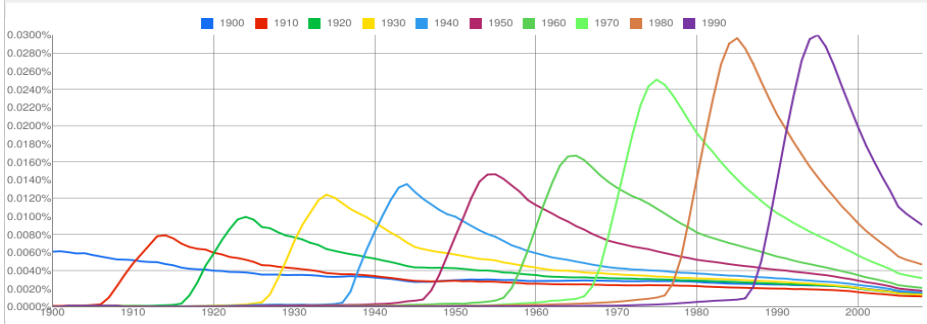
  - Some past experience in software modeling are worth repeating and extending while some are not.

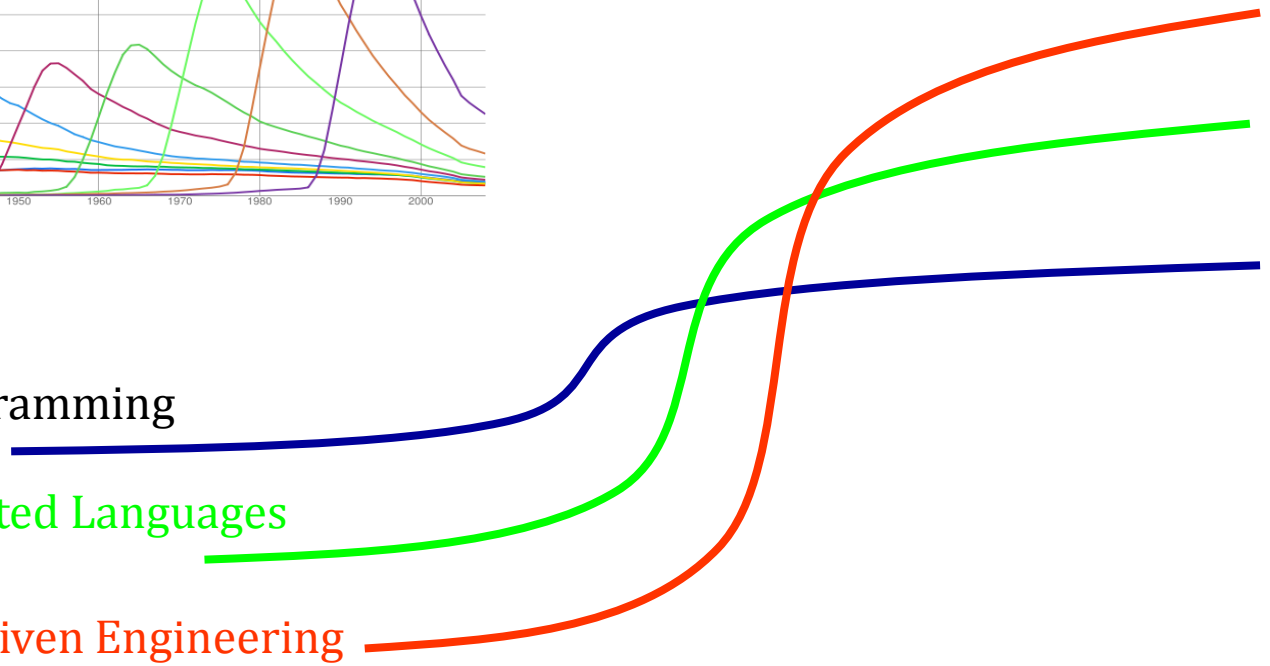
- ***As we advance in the present, history keeps changing***

  Michel Serres

  - There is a record on the birth registration of Ajaccio on the 15 August 1769 at the name of N. di Buonaparte. This came rather unnoticed in the following years until Napoleon become famous. Interpretation of the past is always made with the latest point of view of the present.
  - Interpretation of the past history of the modeling formalisms should be made in the light of the latest knowledge and developments in DSLs.

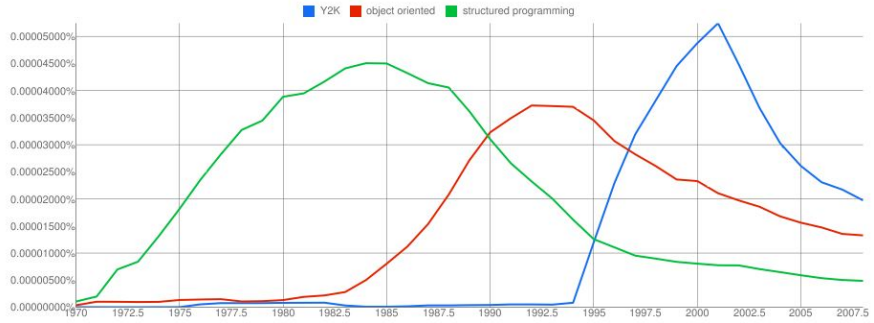
Technology waves



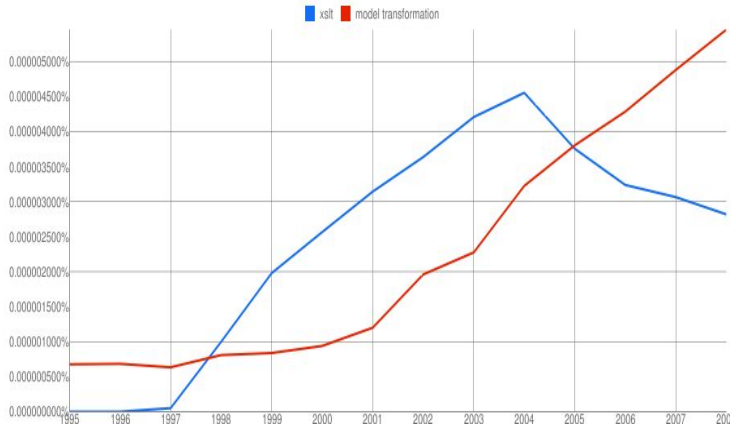


Each wave does not replace the previous one, but complements it.

Impact of some technological changes

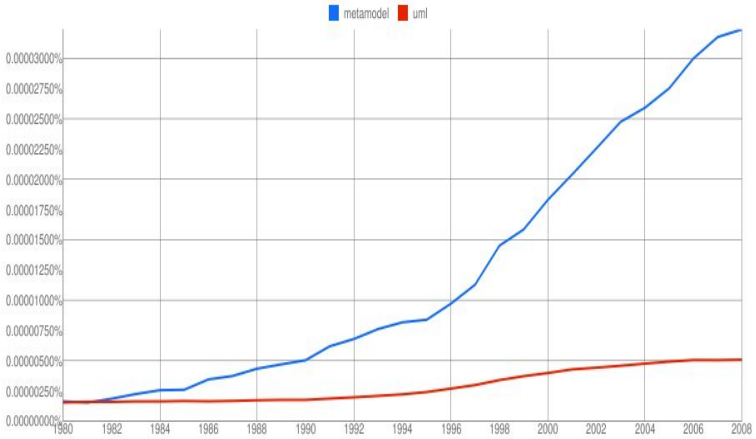


Structured Programming, **Object Oriented**, **Y2K**



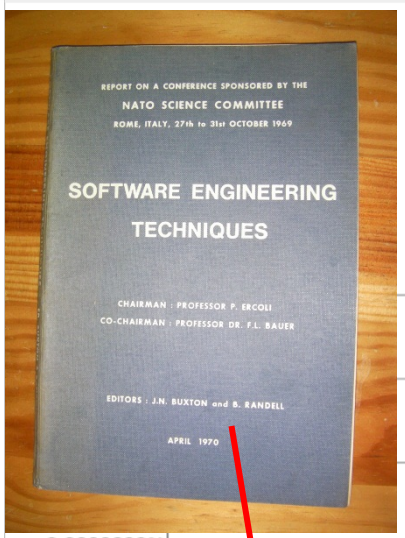
XSLT, **Model Transformation**

Different trends
(different scale)
(different timeframes)



Metamodel, **UML**

Measure of the perceived importance



Google books Google Ngram Viewer

(Raw Ngram buzzword observations)

Possible multiple successive peaks

18 ± 3 Software Technology Maturation (W. Riddle)

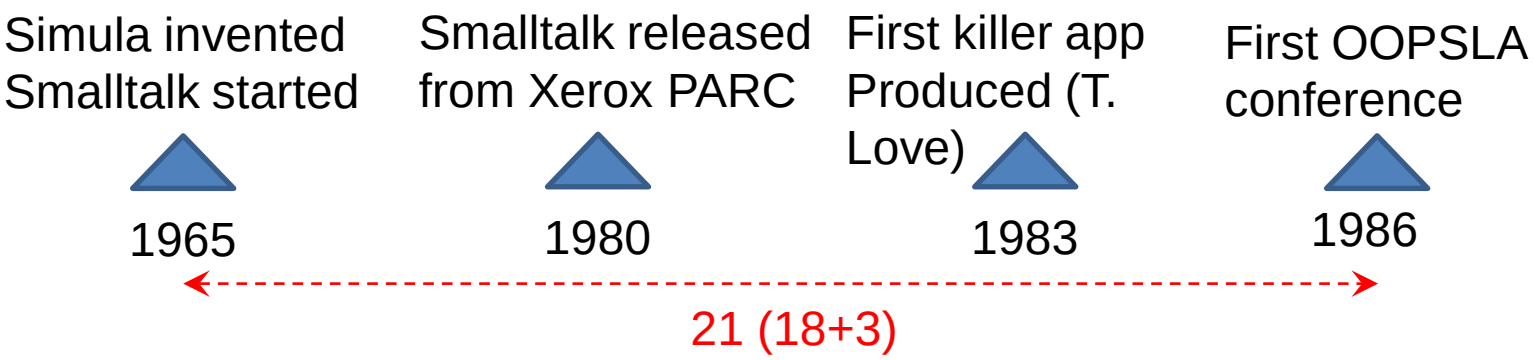
- 18 ± 3 Software Technology Maturation
- 15 to 21 years to mature a technology to the point that it can be popularized and disseminated to the technical community at large
- If we take the MDA announcement (2000) as the starting point for MDE, the jury is still out, but not for a long time.
- Without a killer application, OOP or OT would have never become mainstream

Cost Models
0-1966- appearance of first collection of cost-related data
1-1976- appearance of a usable system (Price S)
2-1978- alternative systems are available (for example, COCOMO)
3-1981- publication of Boehm's text

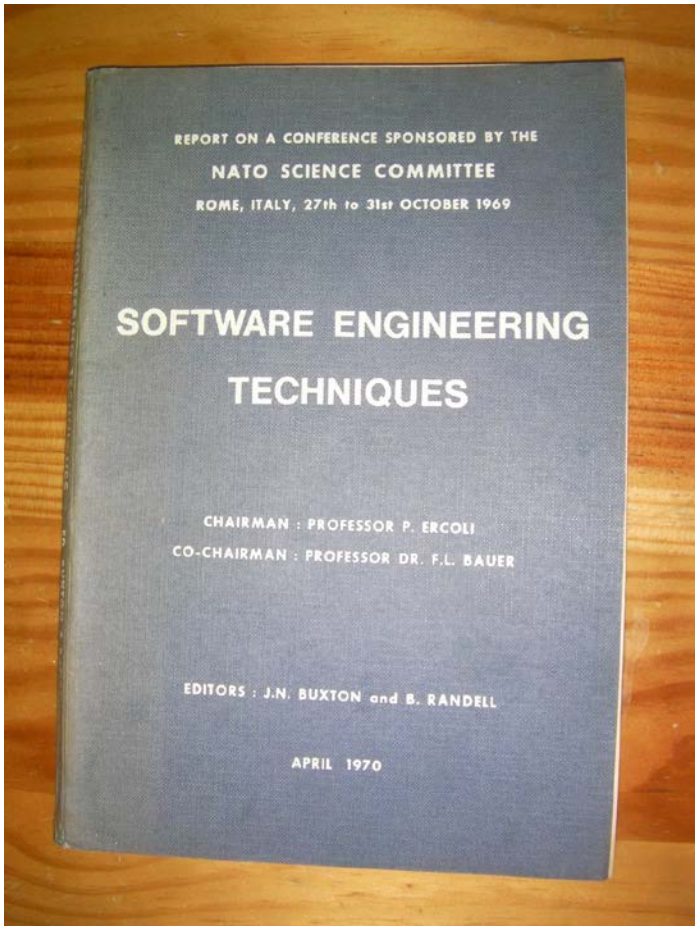
Smalltalk-80
0-1965- Kay's thesis defines concept of a personal computerized notebook
1-1972- preliminary version of Smalltalk is available
2-1976- major new version of Smalltalk appears
3-1981- other companies start porting the Smalltalk-80 system to their computers
4-1983- Smalltalk-80 available as a commercial product

SREM
0-1968- ISDOS system demonstrates applicability of attribute-value-relation approach to pre-implementation activities
1-1973-74- first concrete definition of the SREM system appears
2-1977- first release of the SREM system
3-1981- Vax version available

Unix
0-1967- appearance of the Multics system
1-1971- initial versions of Unix available
2-1973- Unix system debuts at Sigops conference
3-1976- collection of papers appears and system begins to be widely used in academic community
4-1981- announcement of Unix System III



Only 3 technology maturation cycles



1960 + 3 x 18 = 2014

Software Engineering

0-1960- inadequacy of existing techniques for large-scale software development noted in several projects (for example SAGE)

1-1968- concept of software engineering is articulated at Workshop on Software Engineering at Garmisch Partenkirchen

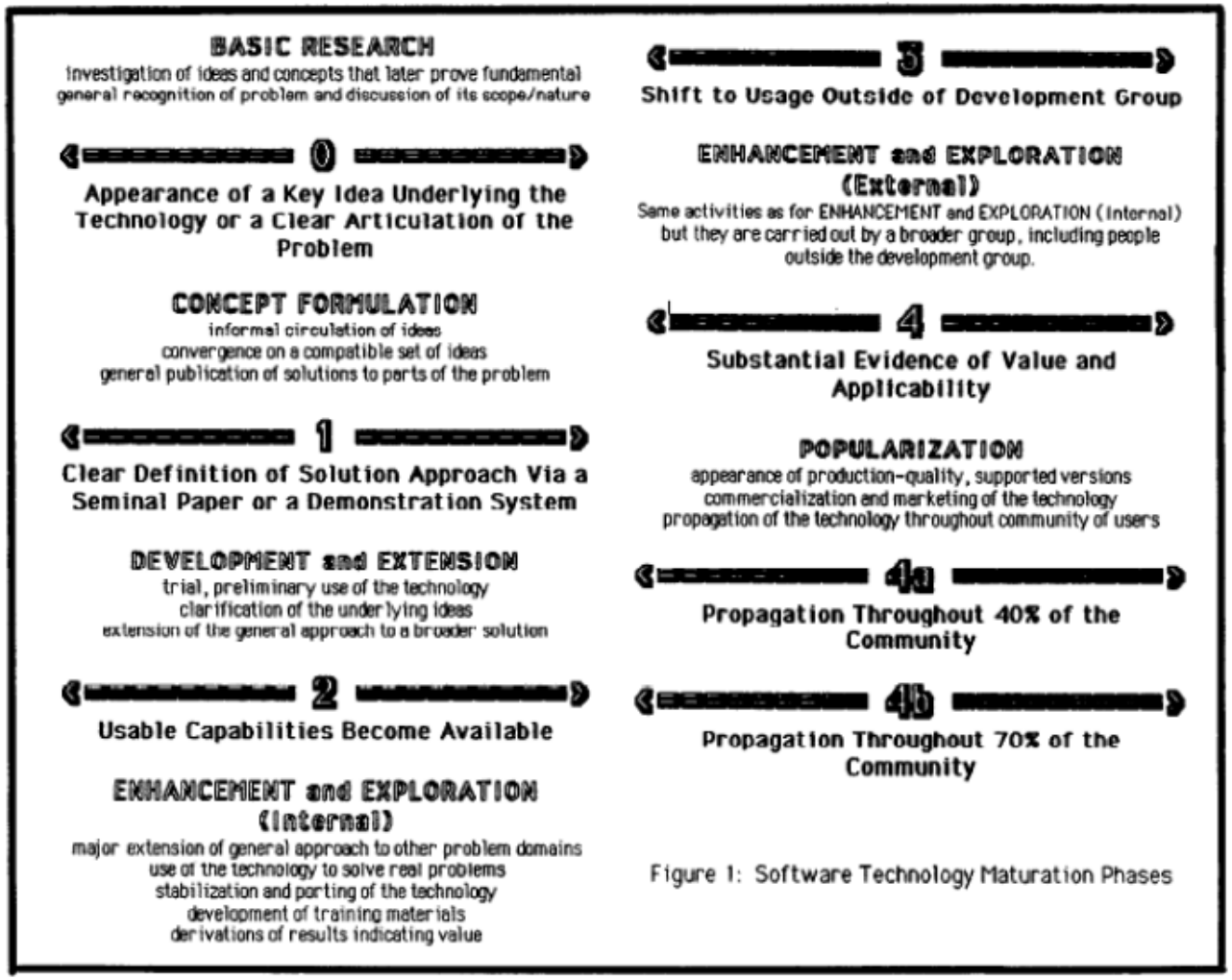
2-1973-74- general collections of papers appear and policy guidelines are established in various communities

3-1978-79- texts and generally usable systems supporting software engineering appear (for example, the SREM system)

4-1983- use of software engineering shifts to a larger community through actions such as the Del auer directive and the definition of a Software Engineering Institute

Riddle Software Technology Maturation

MDE?



BASIC RESEARCH
investigation of ideas and concepts that later prove fundamental
general recognition of problem and discussion of its scope/nature

0

Appearance of a Key Idea Underlying the Technology or a Clear Articulation of the Problem

CONCEPT FORMULATION
informal circulation of ideas
convergence on a compatible set of ideas
general publication of solutions to parts of the problem

1

Clear Definition of Solution Approach Via a Seminal Paper or a Demonstration System

DEVELOPMENT and EXTENSION
trial, preliminary use of the technology
clarification of the underlying ideas
extension of the general approach to a broader solution

2

Usable Capabilities Become Available

ENHANCEMENT and EXPLORATION (Internal)
major extension of general approach to other problem domains
use of the technology to solve real problems
stabilization and porting of the technology
development of training materials
derivations of results indicating value

3

Shift to Usage Outside of Development Group

ENHANCEMENT and EXPLORATION (External)
Same activities as for ENHANCEMENT and EXPLORATION (Internal) but they are carried out by a broader group, including people outside the development group.

4

Substantial Evidence of Value and Applicability

POPULARIZATION
appearance of production-quality, supported versions
commercialization and marketing of the technology
propagation of the technology throughout community of users

4a

Propagation Throughout 40% of the Community

4b

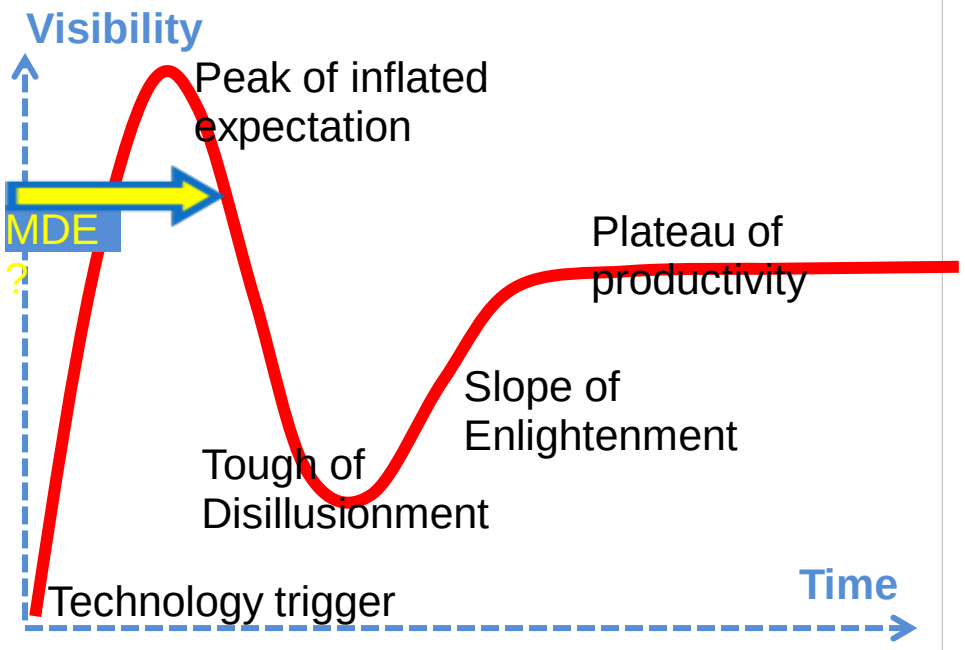
Propagation Throughout 70% of the Community

Figure 1: Software Technology Maturation Phases

OT?

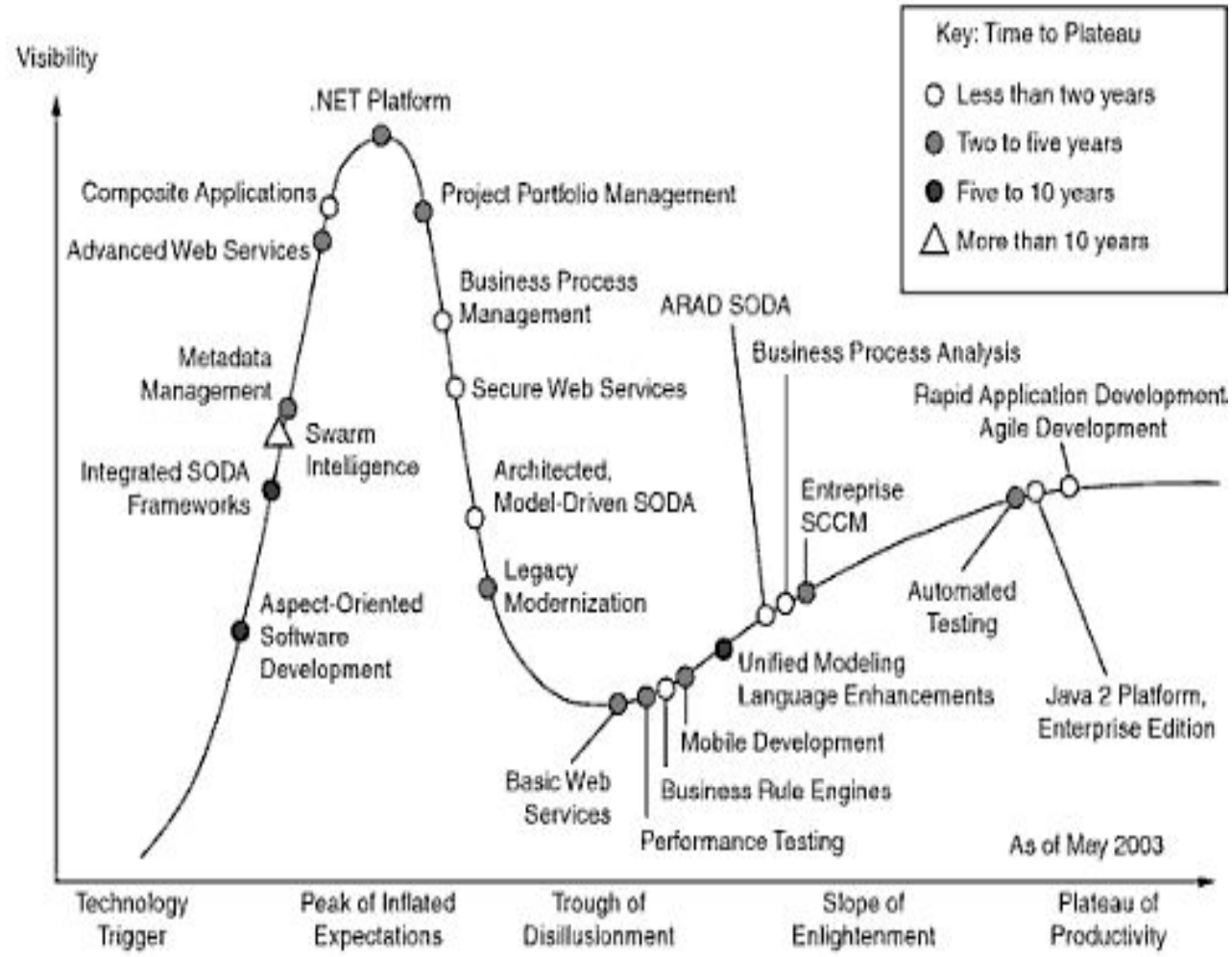
Gartner ’s Hype Cycle Key Phases

- A **hype cycle** is a graphic representation of the maturity, adoption and social application of specific technologies:
  - **Technology Trigger:** A potential technology breakthrough kicks things off. Early proof-of-concept stories and media interest trigger significant publicity. Often no usable products exist and commercial viability is unproven.
  - **Peak of Inflated Expectations:** Early publicity produces a number of success stories—often accompanied by scores of failures. Some companies take action; many do not.
  - **Trough of Disillusionment:** Interest wanes as experiments and implementations fail to deliver. Producers of the technology shake out or fail. Investments continue only if the surviving providers improve their products to the satisfaction of early adopters.
  - **Slope of Enlightenment:** More instances of how the technology can benefit the enterprise start to crystallize and become more widely understood. Second- and third-generation products appear from technology providers. More enterprises fund pilots; conservative companies remain cautious.
  - **Plateau of Productivity:** Mainstream adoption starts to take off. Criteria for assessing provider viability are more clearly defined. The technology’s broad market applicability and relevance are clearly paying off.

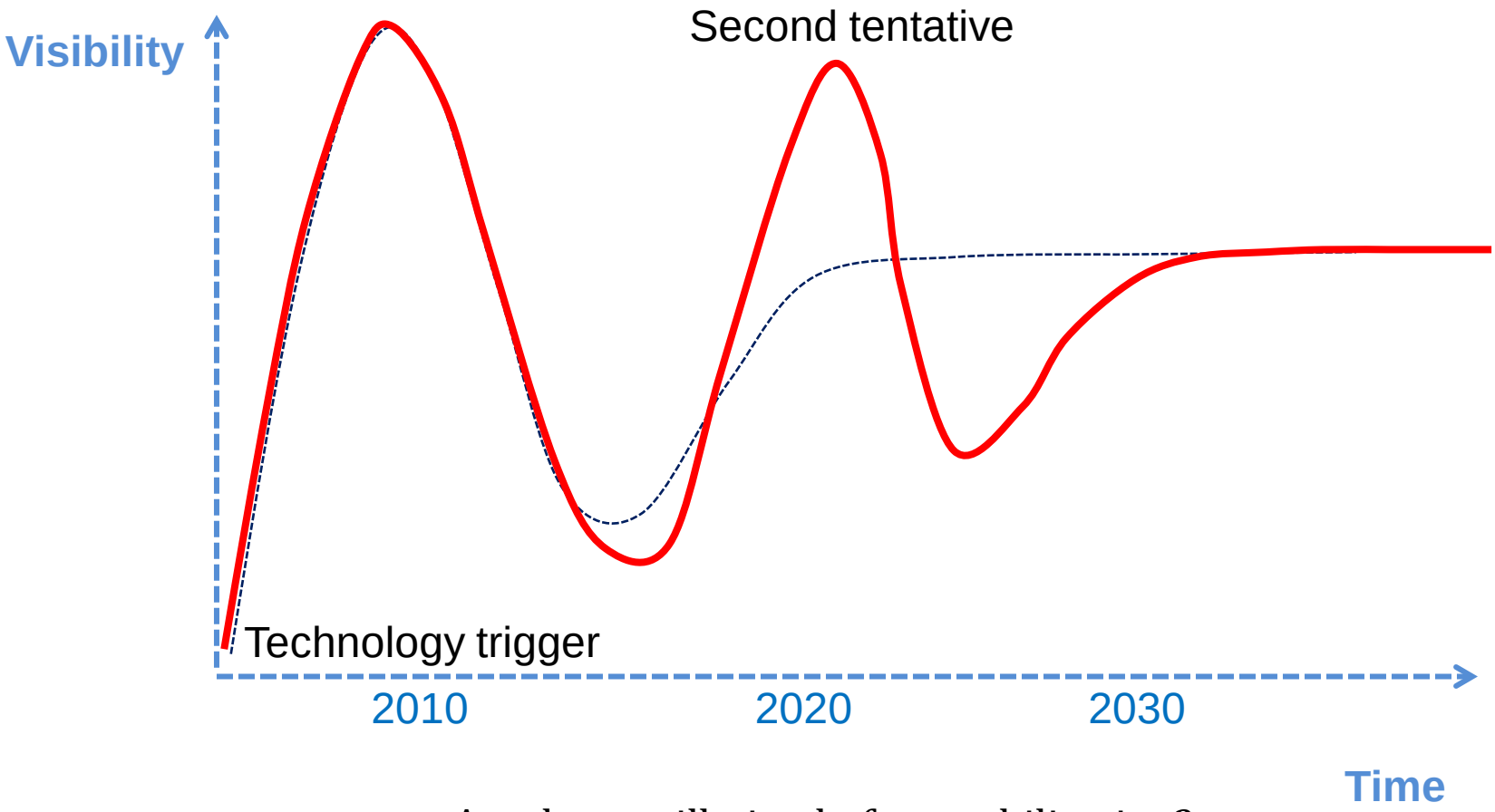


http://www.gartner.com/technology/research/methodologies/hype-cycle.jsp

Gartner's MDA adoption (2003)



A possible scenario for MDE



Another oscillation before stabilization?

Boehm's Pattern

- Discussed in «*A view of 20th and 21th Century Software Engineering*» in ICSE'2006, Keynote, Shanghai, China.
- Hegelian view of software engineering past.
- Most explanations of software engineering evolution from 1950 to present follow a similar pattern
  - Thesis
  - Antithesis
  - Synthesis
- Boehm applies this to:
  - Similarity of software and hardware; COTS
  - Structured programming; Formal methods
  - Waterfall process; Software processes
  - Concurrency control
  - Open Source
  - Software crafting; Agile methods
  - etc.

Negative points: Models Have Failed

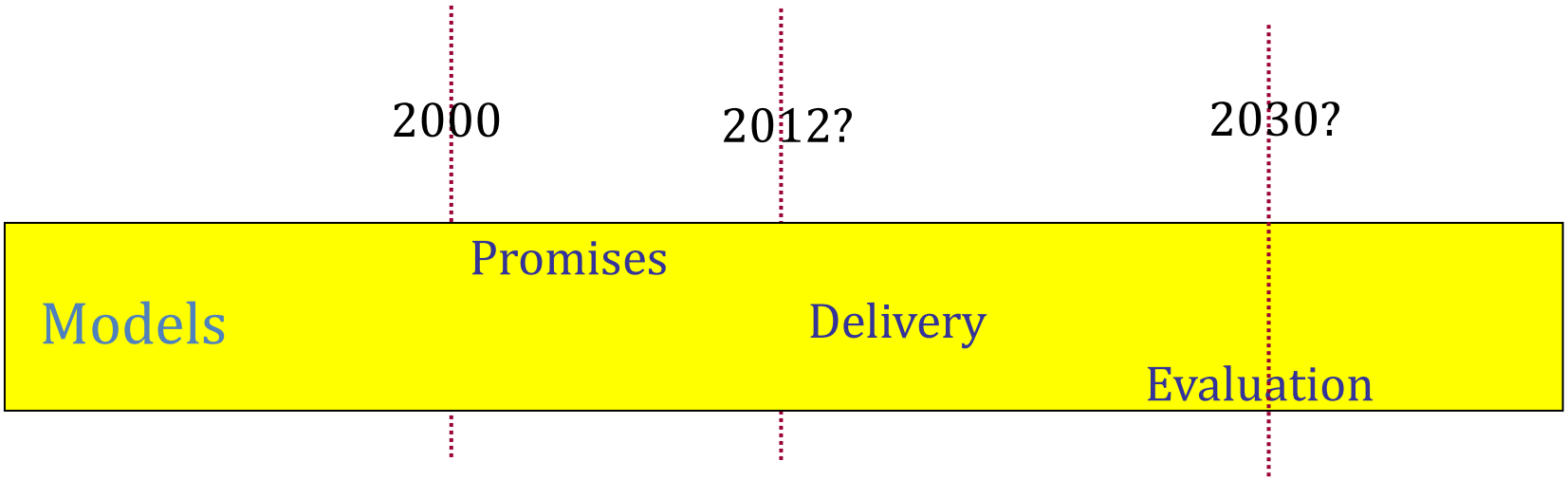
- Models have failed, at least temporarily.
  - Deployment of MDE seems today to have reached a standstill.
  - Immense hopes greeted the MDA™ initial proposal as a possible way to regenerate the entire software engineering practices
    - ✓ Recognition today that its impact is rather limited and its perspectives quite confined.
    - ✓ Observation that the process is not at the same level that in case of technology take-off like OT in the 80's
- What went wrong?
  - No sustainability
    - ✓ lack of model portability in time and space
  - No killer app
  - Decreasing importance of standards (including UML)
  - Many other factors

Many technologies have a limited life

- Remember AD/Cycle by IBM?
- Remember MDR/Netbeans (excellent MDE platform)?
- Remember Microsoft RTIM?
- Remember Microsoft OSLO?
- Remember Microsoft DSL tools?
- How long will EMF last?
- What will follow EMF?
- Will there be a migration path from EMF?

Promises and deliveries

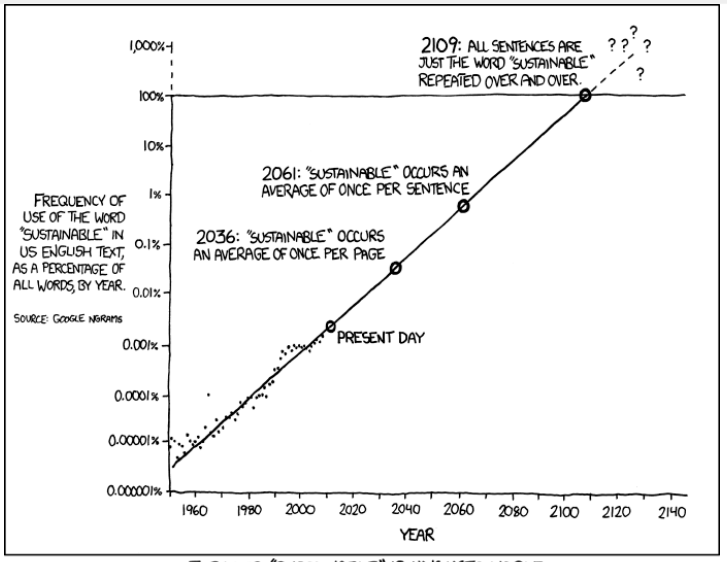
- **What has been promised and what has been achieved**
  - **MDA made many promises in 2000**
    - ✓ **And has problems delivering in 2012**



The strongest commitment of MDA was on protecting
all software investments from deprecation,
over long periods of time.

Sustainable Modeling?

- The first promise/commitment of MDA™ was on **sustainability**:
  - "Developers gain the ultimate in flexibility, the ability to regenerate code from a stable, **platformindependent model** as the underlying infrastructure shifts over time".
  - "ROI flows from the reuse of application and domain models across the software lifespan--especially during long-term support and maintenance, the most expensive phase of an application's life".



PERMANENT LINK TO THIS COMIC: HTTP://XKCD.COM/1007/
IMAGE URL (FOR HOTLINKING/EMBEDDING): HTTP://IMGS.XKCD.COM/COMICS/SUSTAINABLE.PNG

Sustainability is the new hype, but is this hype sustainable?

Many other technologies also are lacking sustainability

From the MDA whitepaper (Nov. 27, 2000)

- As the pace of technology continues to quicken, and the demands of integrating your existing legacy systems, your new intranet, and your ebusiness fall on your shoulders, you need an architecture that makes interoperability central to your infrastructure …
- The bad news is that there will never be a single operating system, single programming language, or single network architecture that replaces all that has passed; the good news is that you can still manage to build systems economically in this environment …
- Over the past decade or more, companies have endured a succession of middleware platforms …
- For years we've assumed that a clear winner will emerge and stabilize this state of flux, but it's time to admit what we've all suspected: The sequence has no end!
- And, in spite of the advantages (sometimes real, sometimes imagined) of the latest middleware platform, migration is expensive and disruptive …

From the MDA whitepaper (Nov. 27, 2000)

- There is a way to manage this situation, and it's based on the core modeling standards from OMG. What we have is the ability to apply modeling technology to pull the whole picture together.

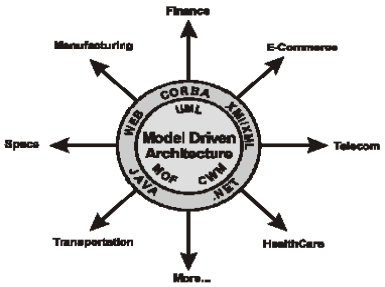


Figure 1: OMG's **Model Driven Architecture**

- Companies that adopt the MDA gain the ultimate in flexibility: the ability to derive code from a stable model as the underlying infrastructure shifts over time. ROI flows from the reuse of application and domain models across the software lifespan …

- **PIM**: Whether your ultimate target is CCM, EJB, MTS, or some other component or transaction based platform, the first step when constructing an MDA based application will be to create a platformindependent application model expressed via UML in terms of the appropriate core model …

- **PSM**: Platform specialists will convert this general application model into one targeted to a specific platform such as CCM, EJB, or MTS. Standard mappings will allow tools to automate some of the conversion. In Figure 1, these target platforms occupy the thin ring surrounding the core

Some MDA success stories but no killer app. yet

"The Architecture of Choice for a Changing World"

Looking Glass Networks	ABB Research Center	Lockheed Martin
U.S. Government Intelligence Agency	ff-eCommerce	Swedish Parliament
The Open System Architecture for Condition Based Monitoring (OSA-CBM) Project	Swisslog Software AG	Deutsche Bank Bauspar AG
CGI	UNext	Carter Ground Fueling Ltd.
BankHOST	Postgirot Bank AB	Gothaer Versicherungen
E-SoftSys	Austrian Railways	Danzas Group
Magnet Communications, Inc.	National Services Industries	How CodeGenie worked for AMS
Credit Suisse	M1 Global Solutions	DaimlerChrysler
Siemens Transportation Systems	ObjectSecurity and Fraunhofer FOKUS: AD4 Virtual Airspace Management System	Cube Model: MDA Meets Open Source
National Cancer Institute		

http://www.omg.org/mda/products_success.htm

What is a Killer App?

Schlumberger

In 1983 at Schlumberger, I used Smalltalk-80 to build a tiny prototype of an analyst workstation for viewing a diverse collection of data from oil wells.[6] A working demonstration was built with only 220 lines of new code and lots of reused classes provided by Xerox. It provided an interactive interface to data residing on a VAX computer. This interface furnished iconic menus for accessing maps, measurement data, analysis reports, and photographs. It also did simple plotting of measurement data.

I showed my work to any number of software professionals in the company and asked them to estimate how much effort it would have required to develop comparable facilities in FORTRAN, the most commonly used language at Schlumberger at that time. The smallest estimate I ever received was 10,000 lines of code. While certainly not a commercial product, it was a commercial problem, and the leverage provided by objects seemed most impressive.

This project was also interesting because of some important observations about the use of objects. One day, a company Vice President was in my office looking at the small amount of work I had done. He said, "do you mean I paid you a year's salary to produce 220 lines of code?" At first, I felt terrible. Then, I began to realize how inappropriate it is for us to feel obligated to produce bulk in return for our compensation.

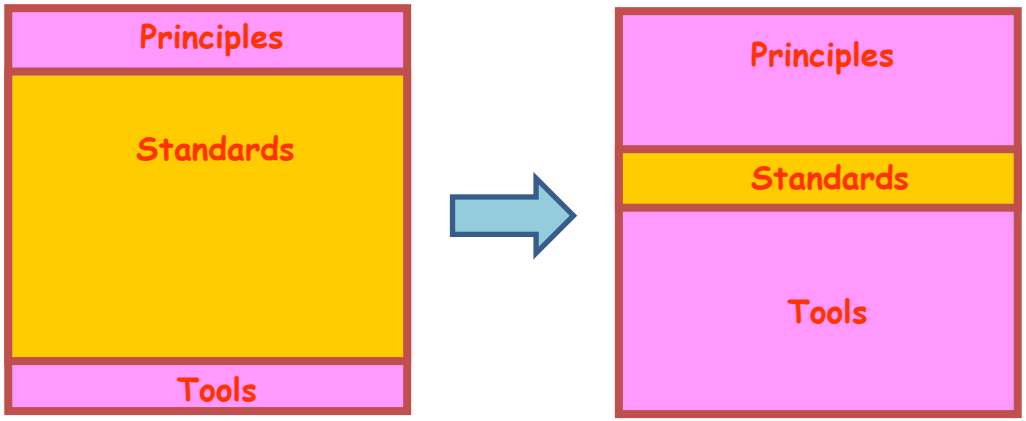
How I had actually spent that year was learning Smalltalk-80 as the first commercial user of that system. Much of the year was spent reading source code I thought I could reuse in the prototype application I was building. Documentation for Smalltalk did not become available until the next year!

Tom Love experiment at Schlumberger (see also the Analyst At Xerox)

220 lines of Smalltalk
vs.
10,000 lines of Fortran

A killer app. should provide measurable and reproducible evidence that the new proposal offers an order of magnitude improvement over previous solutions.

Decreasing importance of standards



- MDE has been characterized by
  - low quality of standards
    - including UML
  - high importance given to them by some companies and researchers
- The focus is now moving more on tools and principles

UML and MDE: friend or foe, devil or angel?

- UML was the conclusion of the OOADTF and the beginning of MDA
  - Everything started with UML and this is probably the main problem of MDE
- UML is a loosely defined language
  - UML is a language with one syntax and an infinity of semantics
- UML is not a badly designed language
  - Because it was never designed at all
  - It is the result of industrial consensus, obtained through a precise process (committee invention)
- Bad modularity principles (profiles)
- UML as a visual syntax for C++
  - UML as a better « programming » language?

UML as the modeling language archetype?

- UML as the typical visual language.
  - Many still wrongly associate MDE with visual modeling.
  - MDE has later shown that textual modeling (Xtext) is often better than visual modeling.
- UML as a general purpose language (GPL).
  - MDE has demonstrated the interest of precise and focused Domain Specific Languages (DSLs).
  - UML considered as the archetype of modeling languages, illustrates a property that is at the extreme opposite of the main MDE basic principles.
- UML is often considered as an OO modeling language
  - MDE has demonstrated the benefits of multiparadigm modeling, considering not only objects, but rules, events, functions, tables, processes, etc.
  - UML has wrongly conveyed the idea that modeling was OO modeling
- UML is known for its complexity, by the size of its metamodel and its rapid evolution
  - MDE promotes the idea of simple languages that could be combined, with controlled execution

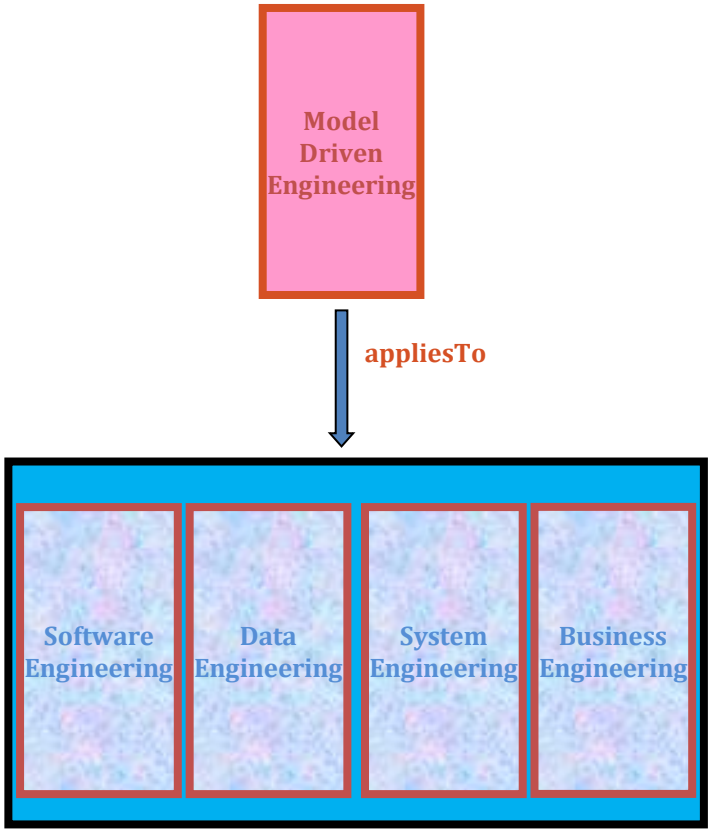
Lack of focus

- If MDE is the solution then what is the problem?
  - What is the precise application scope of MDE?
  - Complete usability taxonomy of models, metamodels and transformations?
    - ✓ To some people MDE is the capacity to efficiently use UML, even for blueprint
    - ✓ To some people MDE is the capacity to transform UML into Java or J2EE
    - ✓ To some people MDE would allow to work with BPMN and UML and bridge the business and IT efficiently
  - Different levels
    - ✓ Core (representation, storage)
    - ✓ Basic operations (transformations)
    - ✓ Specialized model facilities (non agnostic)
- MDD - or MDE for code generation (e.g. UML to Java) - is quite well understood, but other advanced usages of MDE are still much less mastered.
  - Difference between MDE and OT: lack of common agreement

Wrong focus

- Also
  - MDE concentrated too much on models of code and not enough on models of data
  - MDE concentrated too much on models of solutions and not enough on models of problems
  - MDE concentrated too much on Information Systems models and not enough on Business models
  - MDE concentrated too much on modeling in the small and not enough on modeling in the large
  - etc.

Initially MDA was for software engineering



Executability vs. Precision

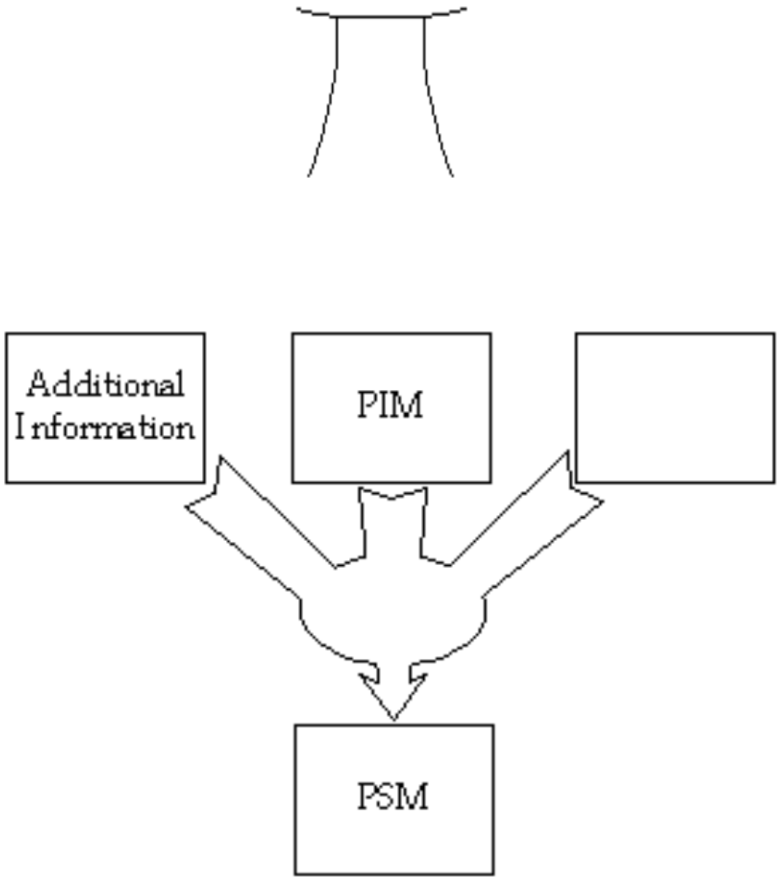
- One important ambiguity has been to let the idea propagate that all MDE-models, and particularly UML, could be made executable.
- MDE promotes the idea that some models may be executable but not all
  - A perverse corollary of this is that non executable models are not precise
  - Many models may be executable and however very precise
    - ✓ Precision is not always obtained through executability
    - ✓ Many models may contribute to the final generation of executable code, while not being themselves executable
    - ✓ Some models may be progressively made executable (e.g. by transformation, progressive enactment)

Too complex

- Metamodels are much too complex
  - Typically a metamodel may contain 5000 elements which is too much
  - A metamodel is intended to remain open (not code or grammar)
  - 500 elements for a metamodel should be a maximum
- Too many metamodels
  - Each metamodel is a silo
  - Allow some communities to hide from other ones
- Relations between metamodels are not understood
  - Overlapping is frequent
  - False similarities (i.e. different concepts with similar names) happen frequently
- MDE as viewed today may be adding accidental complexity instead of helping to master complexity

The work on platforms was one of the MDE big failures

- The CIM/PIM/PSM was a false good idea.
- The notion of a platform model was never taken seriously.
- No consensus on platform metamodels.
- Join development of:
  - A platform
  - A set of DSLs for abstracting this platform
  - A set of transformations for generating to the platform
- More research needed on practical platforms (e.g. HTML5, Javascript/Json, etc.) used in the context of software modeling



XMI Failure

- From SMIF to XMI, a good start
- XMI as the final interoperability solution, the first mistake
- XMI with UML is part of MDE legacy
- JSON is only a partial replacement
- The proportion of native data expressed in XMI is completely marginal and will not increase
- Modeling tools should be able to natively manage a number of formats, including XML and binary formats
- XMI will eventually disappear, creating an additional problem of maintenance for UML models

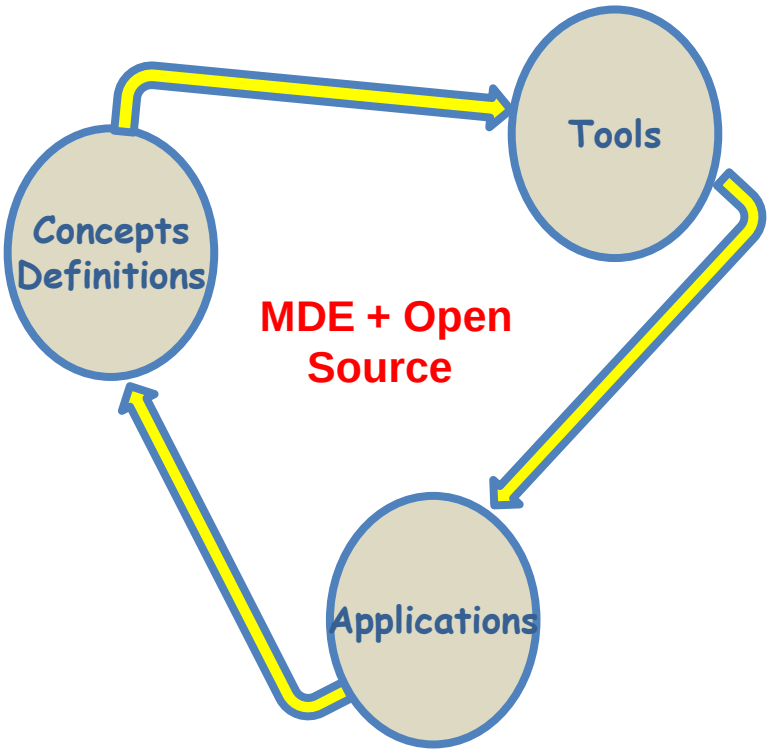
Other problems

- Too easy ways to define metamodels, without sufficient (any) validation.
  - Metamodels loosely linked to reference implementations
  - Metamodels of rather poor quality, evolving through permanent revision tasks forces
- Lack of concern about the possible overlapping of metamodel-based standard agreements
  - Lack of global view on the relations between models and collections of models is a serious difficulty
  - Alternative solution is to use only one unique metamodel (UML?)
- Lack of understanding of the real nature of "model decorations" and its various consequences.
  - These decorations usually pollute the model, cannot be easily separated or cumulated.

Other problems

- Lack of clear distinction between descriptive and prescriptive modeling.
- Lack of a clear classification or taxonomy of models and model operations.
- No general automated way to compare/relate models and metamodels
- No automated way to manage metamodel versions
- No known mature solution to collaborative modeling
- etc.

Reasons to hope



- We learned a lot in 10 years of automated software modeling
- Solid background for the next iteration

Some ideas learned

- MDE is a solution for the separation and integration of various aspects of information systems with the help of typed graphs
- MDE (Modelware) is a specific branch of Software Language Engineering (SLE)
  - Examples of other branches (technical spaces)
    - ✓ Grammarware
    - ✓ XMLware
    - ✓ Ontologyware
    - ✓ etc.
  - Each branch has its own engineering advantages and drawbacks
  - No one will preclude all others even if competition holds
  - More emulation than competition: TSs are complementary
- MDE is not:
  - Visual programming
  - UML
  - PIM to PSM
- All models are not executable, even if some are
- Models may be completely precise and nevertheless computer-understandable

Basic MDE Principles

1. **Representation principle**
   - A model M is a representation of a system S
2. **Multiple view principle**
   - Several models may provide different representations of the same system S
3. **Conformance principle**
   - Any model M conforms to the language of its metamodel MM
4. **3-level principle**
   - Any metamodel MM conforms to the same self-conforming metametamodel MMM
5. **Transformation principle**
   - The most important operation applicable on models is a transformation
6. **Weaving principle**
   - Abstract correspondences between models are represented as models
7. **Megamodel principle**
   - Model elements may be considered as models
8. **Unification principle**
   - All models specialize a common abstract model

We learned a lot

- Assuming
  - We know that MDE is a branch of language engineering
  - We know that automation needs transformations
  - We know that a relation between models may be represented as a model
  - We know that sets of models (conforming to different metamodels) may be represented as models
  - We know that it should be possible to define a strong and regular modeling framework (improved EMF) that will make easy to define and manage an important number of small and precise composable DSLs
- But this structured library of small metamodels does not yet exist
  - Biggest challenge ahead for MDE

Transformation is key

- Without practical transformations, no reasonable automation
  - Remember the «*Bubbles don't crash*» criticism on models by Bertrand Meyer?
  - Transformation may crash ;) and are most useful
  - Transformations draw the boundary between old contemplative modeling and modern MDE
- Different kinds of transformations
  - From model to model (M2M)
  - From model to system (M2S)
  - From system to model (S2M)
  - With a special simplification when the system belongs to another 3-level structured technical space
- Characteristics of transformations
  - Basic properties
  - Time of execution
  - Different purposes

Characteristics of transformations

- Basic properties
  - Metamodel-based
  - Rule-based
  - Transformations as models
- Time of execution
  - Software production
  - Software maintenance
  - Software operation
- Different purposes
  - Measure
  - Verification
  - Synchronization (e.g. round-trip when this applies)
  - Interoperability
  - Reusability (of functionality)
  - Most situations

Ubiquitous DSLs

- The future of IT is with DSLs
- But DSLs also have a bright past
  - Excel (initially Visicalc)
  - SQL
  - and much more
- However the real application scope of DSLs is yet unknown and rapidly growing
  - Generative programming (i.e. generating GPL code from other DSLs)
  - But this is only the emerged part of the iceberg
- The discovery that MDA™ and then MDE was strongly related to DSLs is only a recent idea but a strong result
- The basic layers of any MDE stack should contain DSLs for defining metamodels, defining correspondances between models and defining executable transformations

Research agenda

- Just some ideas
  1. Relations between programming and modeling
  2. Definition & promotion of a better M3
  3. Understanding modularity in modeling
  4. Revisiting model transformations
  5. Relations between Tools and Metamodels (last lesson)
- Most of the efforts should be first spent on the basic MDE layers

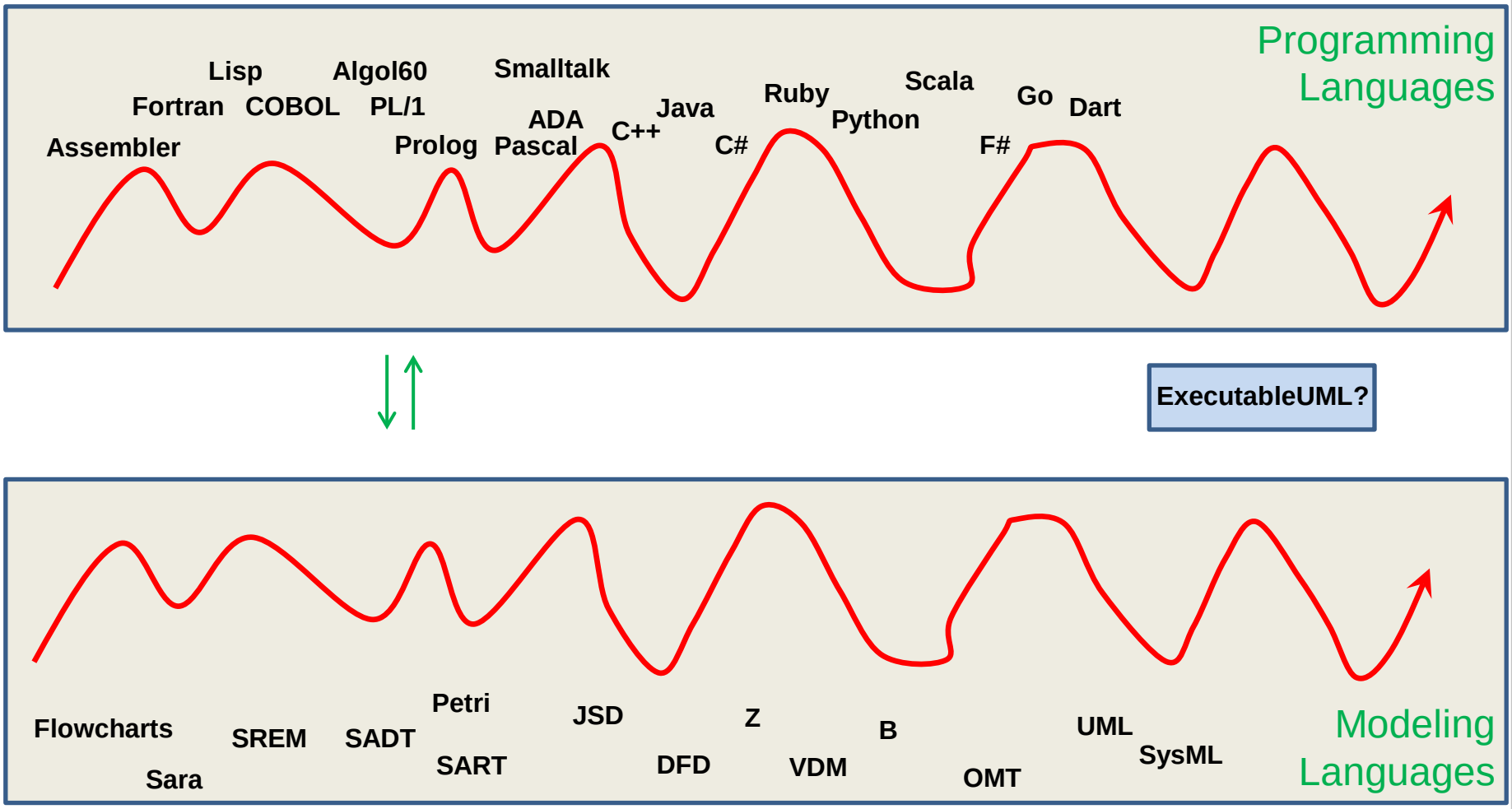
Confusion between programming and modeling

- Search for making UML into a visual executable language led to the belief that modeling languages and programming languages are similar
- This is still one of the strongest debate in the community but also one of the major problems of MDE
- Another view is that they are different but complementary
- Using modeling languages may result in generating executable code through model transformation
- Some programming languages (JavaScript for example) are not languages to program but language to generate (see GWT) and are excellent candidates to be jointly used with modeling languages

Modeling and programming languages

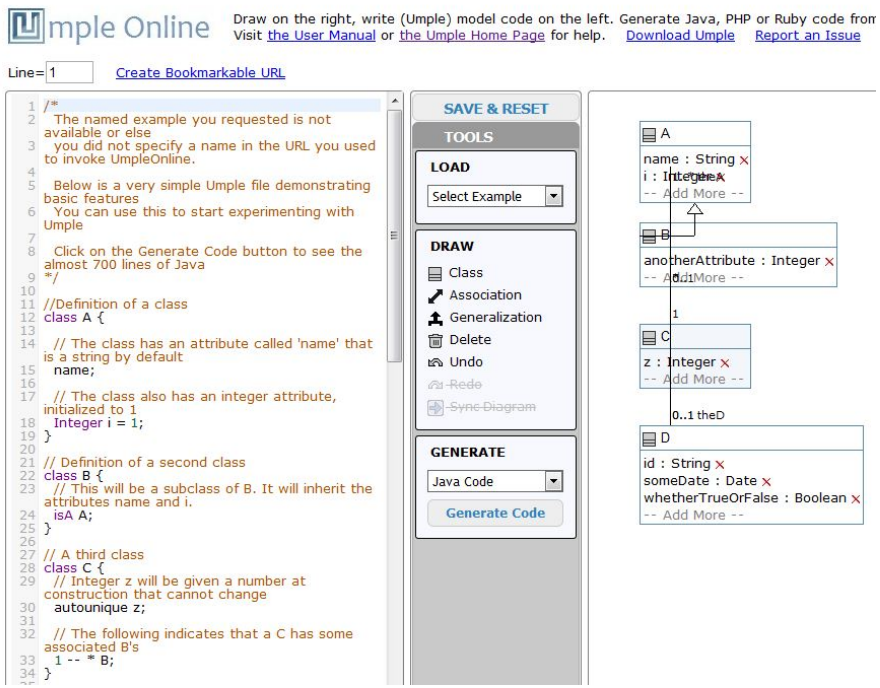
- Is there a difference between models and programs?
  - Some have coined the term ***mogram*** to describe the common entity
- Main difference (?)
  - All programs are executable
  - Some models are not executable
- Programs are models (a special case of)
  - A programming language has a precise and fixed execution semantics
  - Few models are natively executable
  - Some models may be transformed into an executable program
- But there are some other differences and non-differences between a model and program

The two parallel tracks



The impossible dream

- A language with all the nicest properties of programming languages and modeling languages.
  - Either by making a modeling language executable (executable UML)
  - Or by including modeling features in a programming language (UMPLE)
- Is this feasible?
  - Many goals of both kinds of languages are in complete opposition



http://cruise.site.uottawa.ca/umple/

Models and Programs

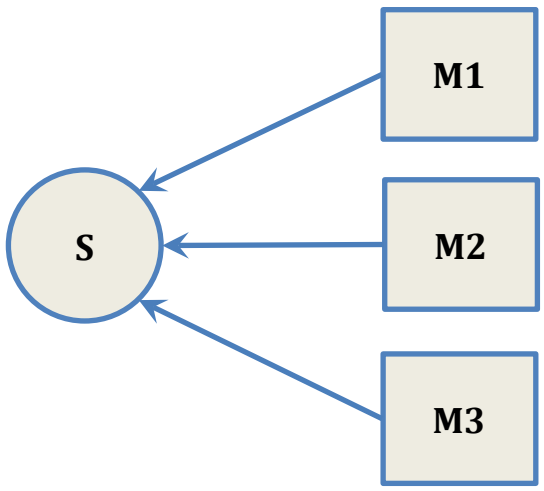
- Technical space notation
  - Distinction between a *MDE-model* and a *Programming-model*
  - Both are *Software-models*
  - In the following we name *models* MDE-models and *programs* Programming-models
- Issue of representation
  - Usually a model is a graph and a program is a tree
- Issue of appearance
  - Usually programs are expressed by a concrete textual syntax while many models are expressed in a visual syntax
  - This is being less significant now that many models are also expressed textually
- Issue of conformance
  - A program conforms to a grammar
  - A model conforms to a metamodel
  - Quite important (see next slide)

Grammars vs. Metamodels

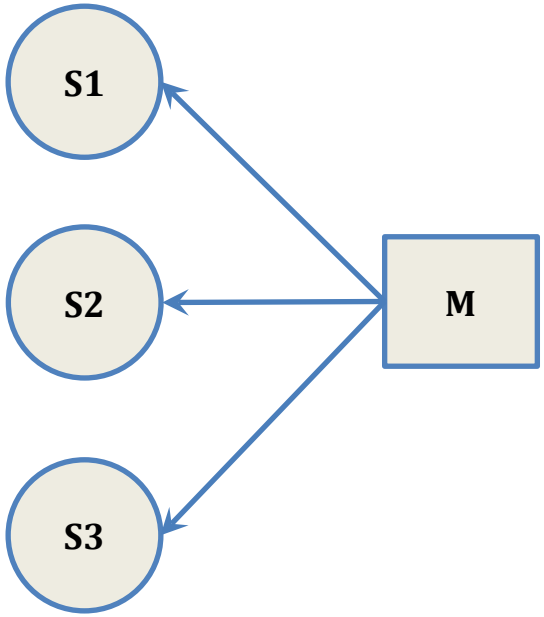
- Issue of representation
  - Grammars are trees
  - Metamodels are graphs
- Issue of visibility
  - Metamodels are open
  - Grammars are rather closed
- Issue of evolutivity
  - Grammars are (usually) fixed
  - Metamodels are (usually) variable
- Issue of maturity
  - Grammarware has 50+ years
  - Modelware has much less maturity

Not to be confused

Multimodeling

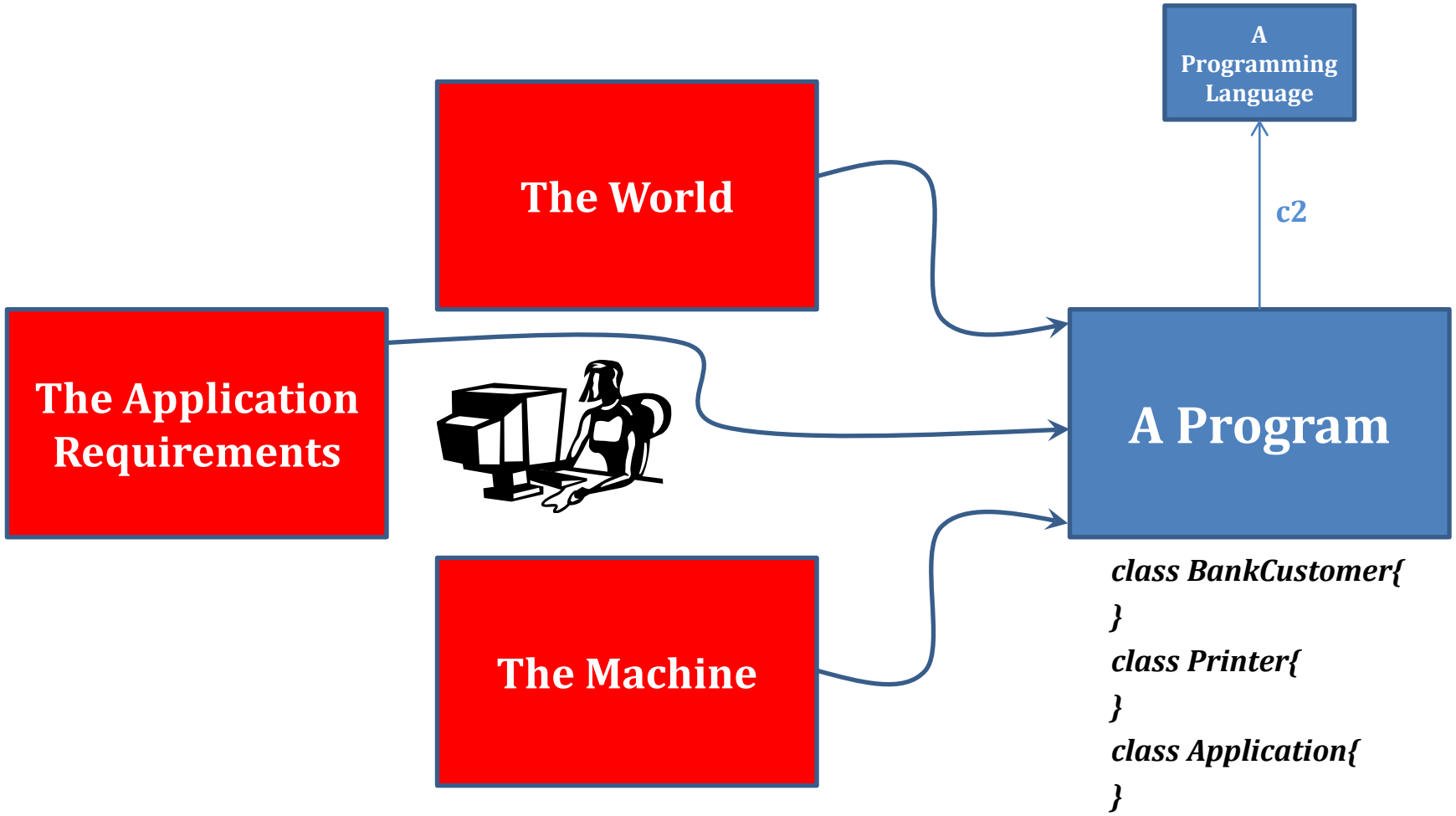


Composite Modeling



A usual program is an example of a composite model, representing at the same time the domain, the execution platform, and the application user requirements.

Programming is difficult



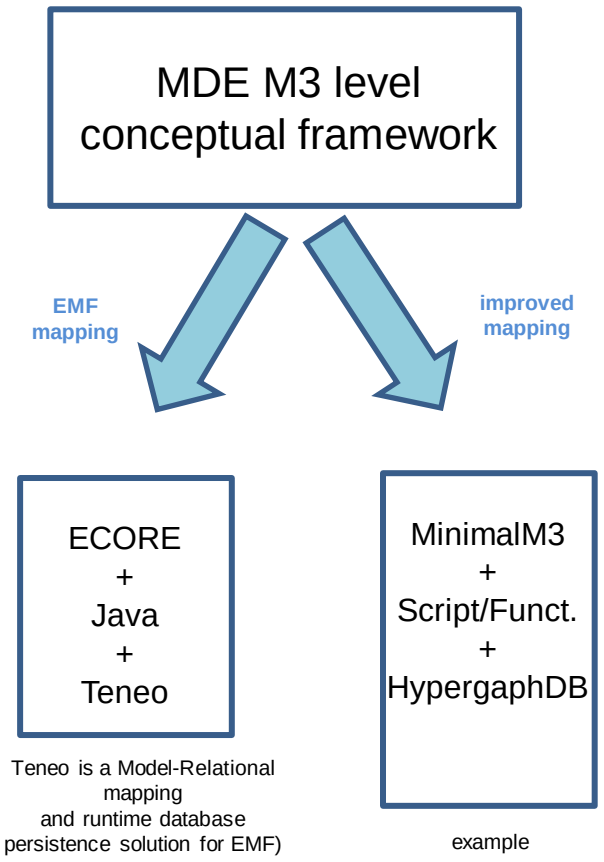
Towards a new M3

- The current M3 (metametamodels) have been defined by opportunity
  - First inspired by UML like MOF (for tooling reasons)
  - Then aligned with UML (for political reasons)
  - Thus object-oriented and class-based by opportunity, not by objective
  - Then simplified by necessity (EMOF and ECORE)
  - UML was class-based by objective (OOAD and targeting Java and similar languages)
- A M3 should be paradigm-agnostic
  - Allowing to build any kind of metamodel (OO, functional, rule-based, event-based, process-based, etc.)
  - There is no need to be visual for a M3
- A M3 defines the representation system of the corresponding TS
  - It should be defined with maximum care

Towards a new M3

- Characteristics of an ideal M3
  - Minimalist (and not universalist)
  - Based on typed hypergraphs
  - Implementable on advanced Big Data systems (cloud, NoSQL, etc.)
  - Allowing libraries of metamodels (lattices with true incremental deltas)
  - Allowing domain evolution (class-based is not the obvious solution)
  - Implemented on an open-source basis for rapid shared adoption
  - With robust serialization (Json or XML-based)

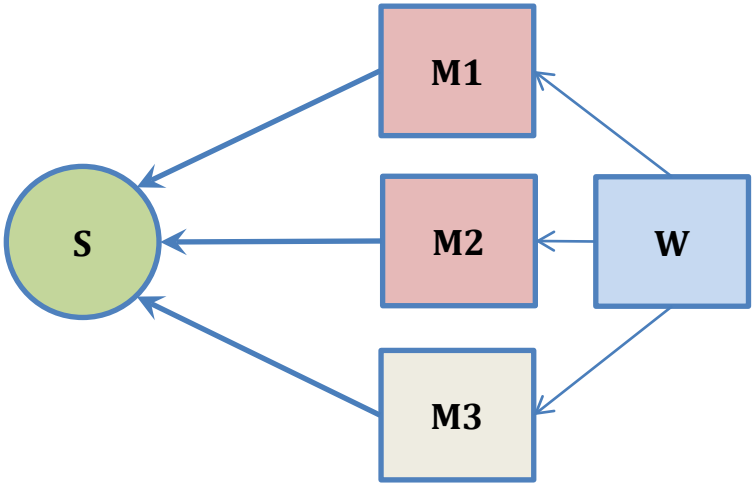
Towards a new M3



- Defining a new M3 is defining a new TS
  - It should be conceptually sound
    - Typed hypergraphs
  - It should be efficient and scalable
    - Use some graph database (bigdata, cloud, NoSQL, etc.)
      - Ex. HypergraphDB, Neo4J, OrientDB, CouchDB, etc.

Modularity for Models

Multimodeling



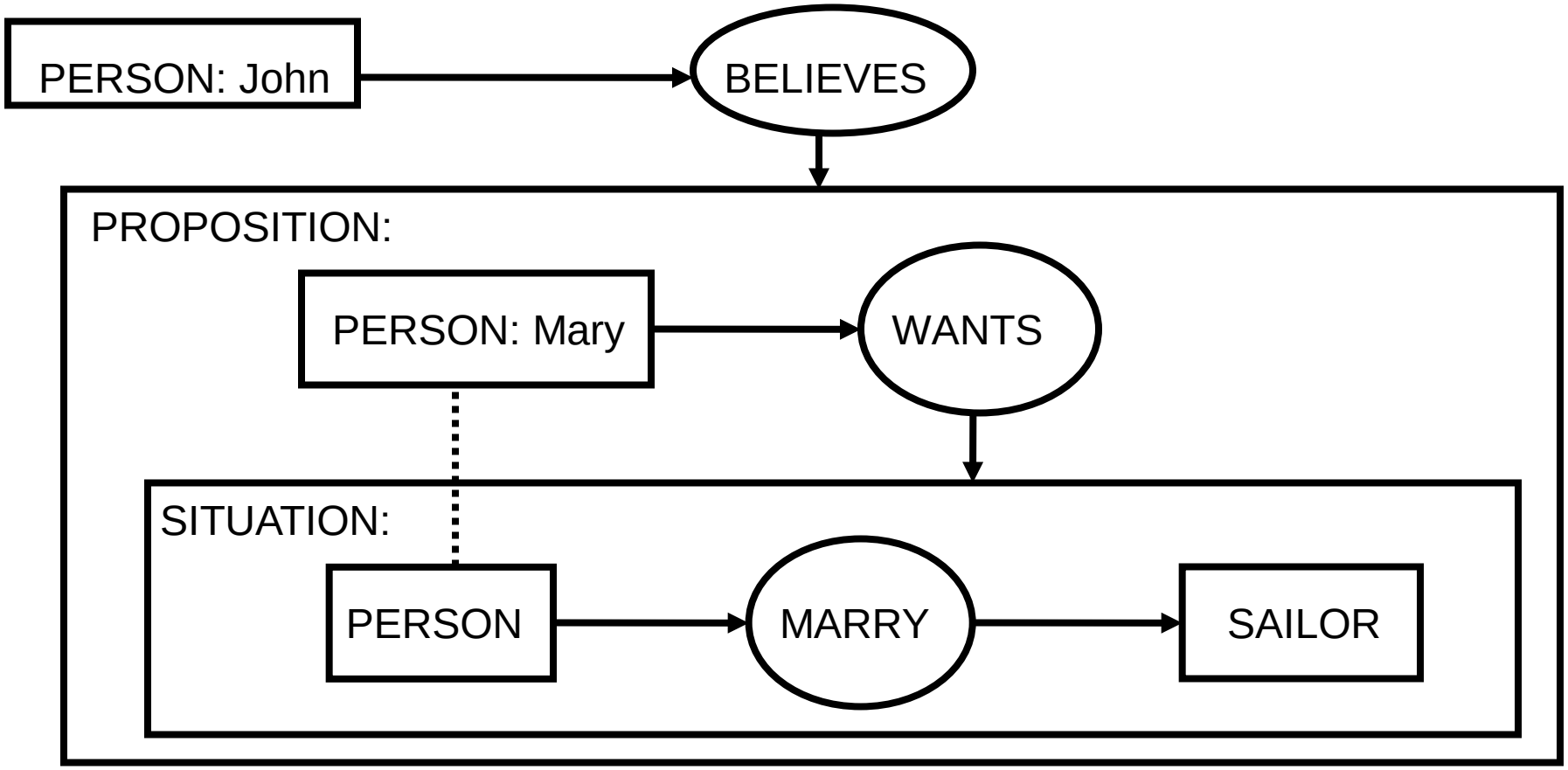
- By nature MDE is modular
- It promotes full separation of aspects

Lack of good model modularity devices

- Initially the definition of UML by Rational (formerly an ADA company) was to borrow the **package** mechanism to the programming language community.
- IBM (S. Cook) proposed a better solution named **schemes** that was rejected for political reason
- By lack of a better idea, people considered models composed of packages in MDA/MDE
- To solve problems, OMG introduced the concept of **profiles** that make the situation much worse
- When the need to exchange models by serialization arose (SMIF), MDE/MDA borrowed solutions to the XML community (XMI) and therefore introduced the **name space** mechanism in the representation of models
- Today there is still no specific model modularity and people are struggling with packages, profiles and namespaces

Contexts and coreferences (Sowa graphs)

"John believes that Mary wants to marry a sailor"



Revisiting model transformations

- Three objectives
  - Use
    - ✓ Focus on usability and acceptance
  - Reuse
    - ✓ Focus on sharing libraries of transformation components
  - Re-architecture the IS
    - ✓ Radically different
    - ✓ Propose a clear alternative to OO-architecture
    - ✓ Closer to data-flow architectures of the 70's
- The killer app will be here
  - Need to find a significant application written in OO style (e.g. Java)
  - Rewrite it with metamodels and transformations
  - Compare the building and maintenance costs

Conclusion

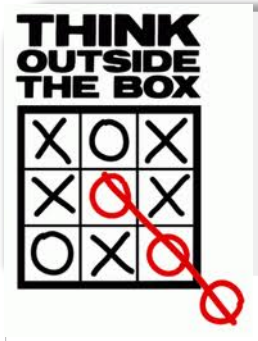
- Even if we know some limited success stories on MDE
  - We still miss the great MDE killer app
  - MDE is not following the hype curve of OT
  - UML has made very difficult the independent expression of innovative software modeling ideas
- Did MDE miss the boat?
  - Industrially probably yes on this iteration
  - Except for some small niches
  - On the research side we learned a lot, and we have now a vision for the future (this century)
  - There is still a chance that the next iteration of MDE will provide the basis to get the software industry out of the bad situation where it is installed now

Robin Milner's Grand Challenge for the century

Language is the raw material of software engineering, rather as water is the raw material for hydraulic engineering…

A more thorough science-based approach to informatics and ubiquitous computing is both necessary and possible. We often think in terms of models, whether formal or not. These models, each involving a subset of the **immense range of concepts** needed for ubiquitous computer systems, should form the **structure of our science**…

Even more importantly, the **relationships** (either formal or informal) among them are the cement that will hold our towers of models together. For example, how do we derive a model for senior executives from one used by engineers in designing a platform for business processes, or by theoreticians in analyzing it?



MDE is not «Linear Research»

The original B-tree paper was bounced; the data cube paper was bounced. The original transaction paper was bounced. Any paper that is non-linear is going to get bounced.

Jim Gray, SIGMOD Record,
Vol. 32, No. 1, March 2003

- It is time the researchers realize that MDE is not a small delta in software engineering or programming languages
- The problem is not to invent a marginally better programming language (executable UML?)
- The question is to invent radically new ways of producing software as a real answer to the real needs of people
- Providing the research community with a new advanced modeling framework may allow the non-linear experiments that will explore completely these new ways to produce useful and reliable software applications.

Thanks

- Questions?
- Comments?

JBezivin@gmail.com