IBM Rational

From Model-Driven Development to Model-Driven Engineering

Bran Selic
IBM Distinguished Engineer
bselic@ca.ibm.com

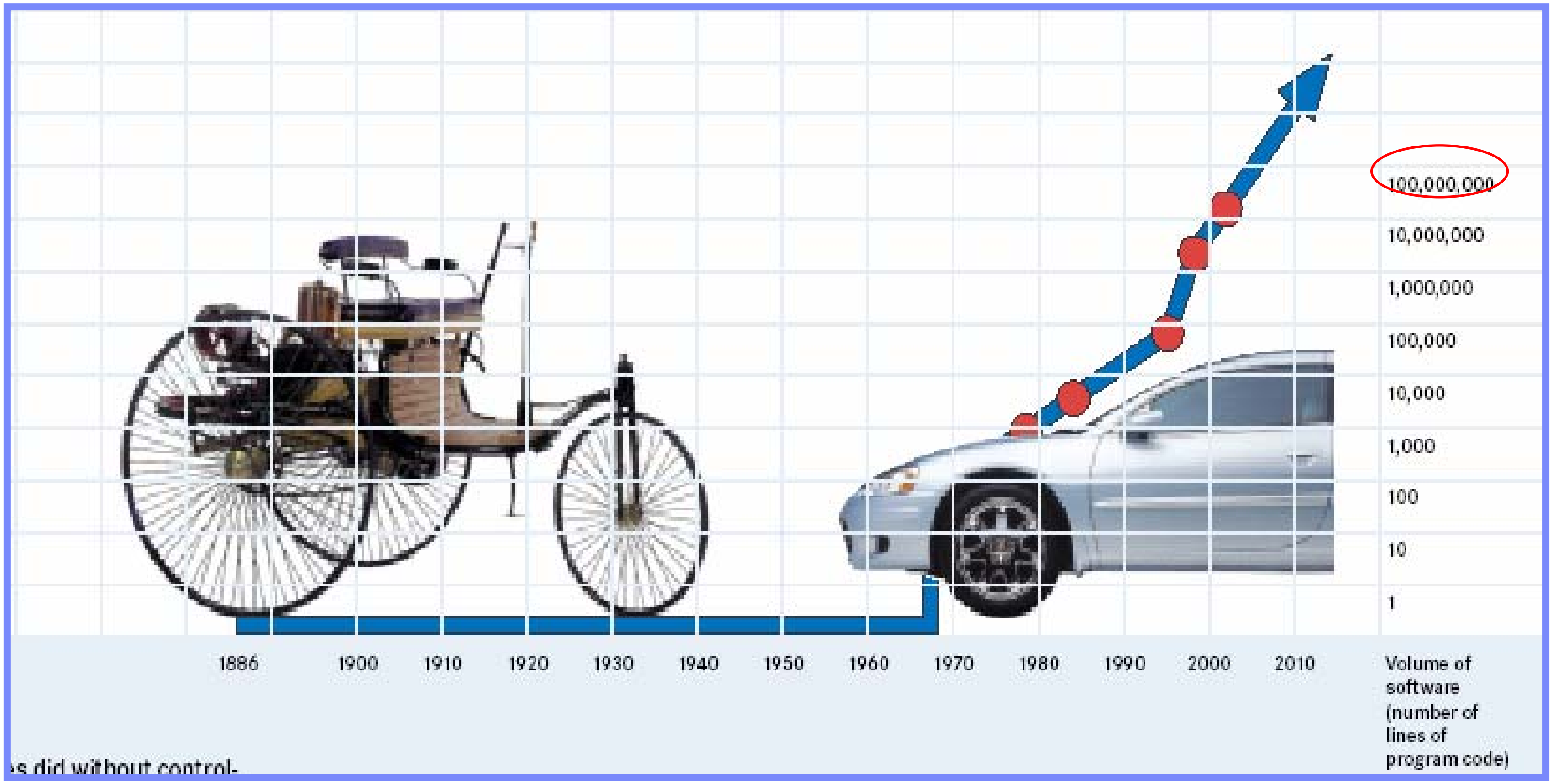
ON DEMAND BUSINESS

Outline

- The impact of software on engineering design
- Introducing model-driven development (MDD)
- Adding the engineering aspect (MARTE)
- Adding the systems aspect (SysML)
- The challenges before us

The Encroachment of Software…

- **Intended as a replacement for basic relay circuitry**

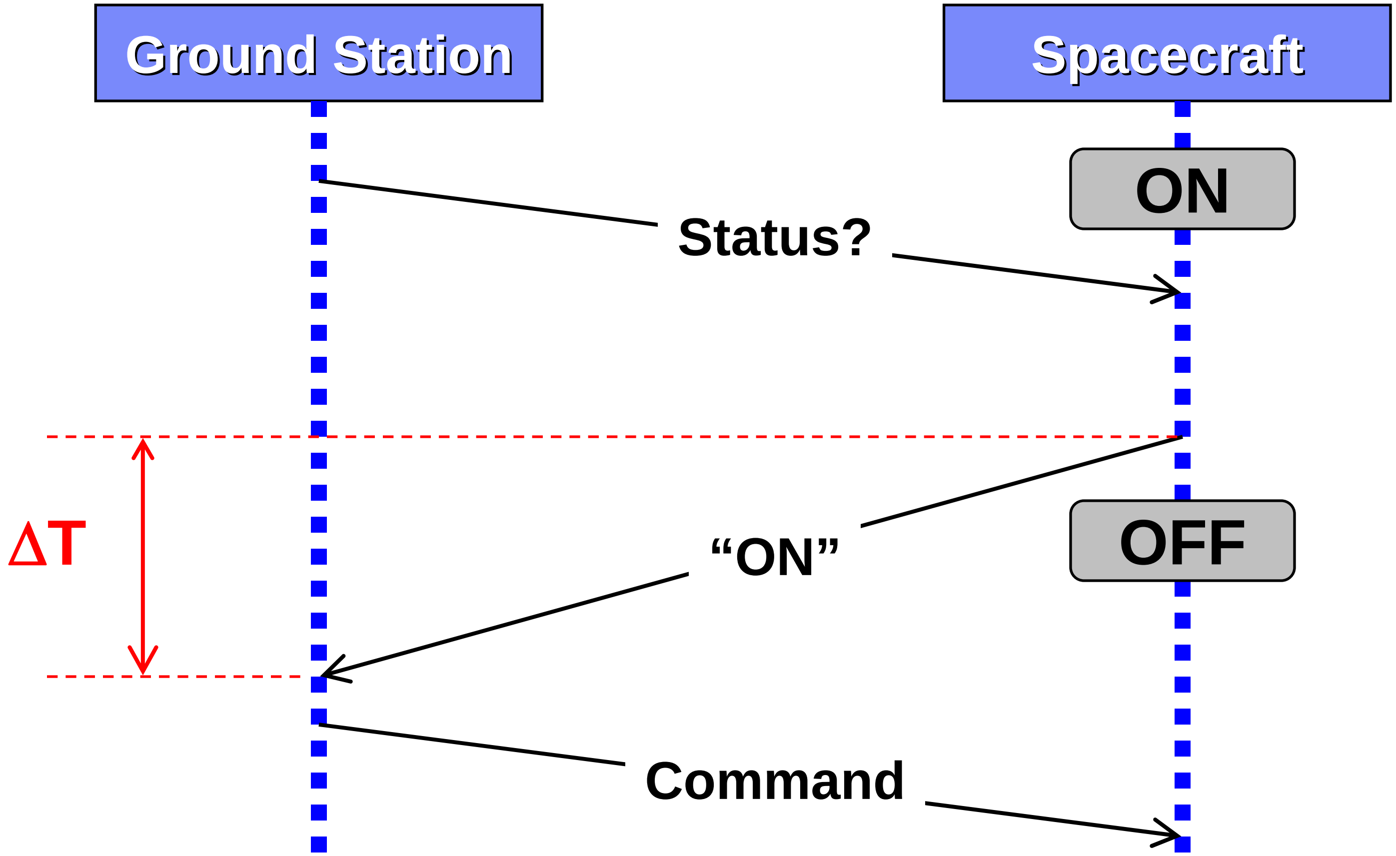


The Essential Complexities of Embedded Software Design

- **Contending with the physical world**
  - An unpredictable and often unfriendly context (Murphy's Law):
    - The need for timely responses
    - Concurrency and distribution
    - Resource limitations (memory, CPU speed, bandwidth, etc.)
    - The likelihood of faults and the need to deal with them
- **The pressure for more sophisticated functionality**
  - Motivated by the apparent flexibility of software
  - Competitive pressures
  - Engineering hubris

Physical World Effects: Example

- The effect of transmission delays



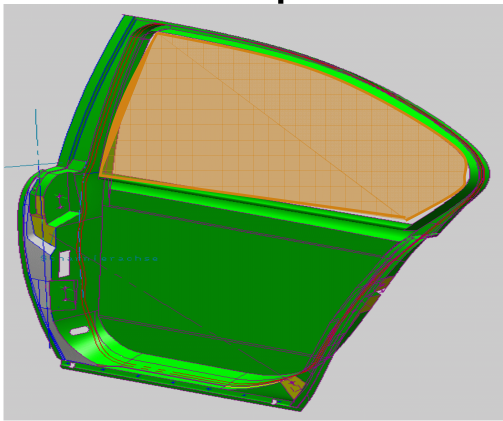
Software Physics: The Great Impossibility Result

It is not possible to guarantee that agreement can be reached in finite time over an asynchronous communication medium, if the medium is lossy or one of the distributed sites can fail

– Fischer, M., N. Lynch, and M. Paterson, "Impossibility of Distributed Consensus with One Faulty Process" *Journal of the ACM*, (32, 2) April 1985.

Complex Functionality

- **A real-world example: the window closing problem**
  - Electronically-operated windows could not be closed when car was traveling past a certain speed



- **A classical case of "feature interaction"**
  - Conflict between safety constraint and desire for automation

The Consequences…

- Software has become the dominant problem in many engineering systems
- Over 50% of embedded projects are months behind schedule[1]
- 25% of embedded projects are abandoned[2]
- Only 44% of designs are within 20% of expectation[1]
- Over 50% of the total development effort spent on testing (75% for safety critical systems)

[1]Electronics Market Forecasters, April 2001

[2]Embedded Developer Systems Survey, Summer 2001

Outline

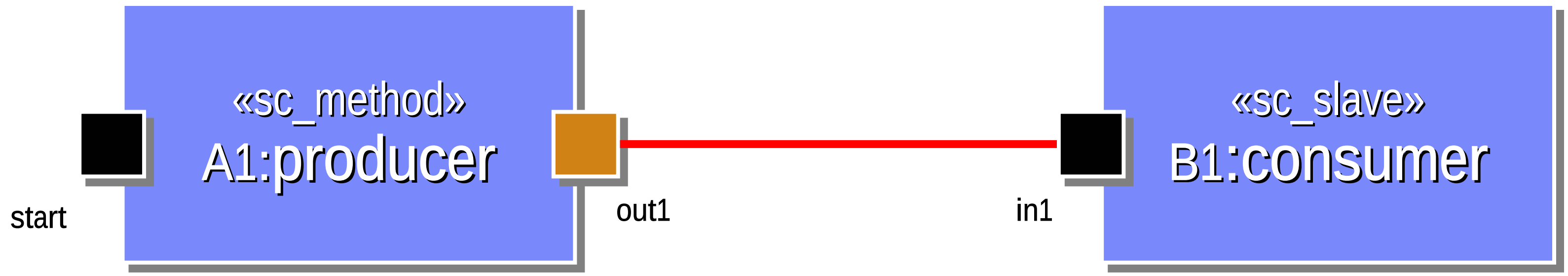
- The impact of software on engineering design
- Introducing model-driven development (MDD)
- Adding the engineering aspect (MARTE)
- Adding the systems aspect (SysML)
- The challenges before us

A Bit of Modern Software…

```
SC_MODULE(producer)
{
sc_outmaster<int> out1;
sc_in<bool> start; // kick-start
void generate_data ()
{
for(int i =0; i <10; i++) {
out1 =i ; //to invoke slave;}
}
SC_CTOR(producer)
{
SC_METHOD(generate_data);
sensitive << start;}};
SC_MODULE(consumer)
{
sc_inslave<int> in1;
int sum; // state variable
void accumulate (){
sum += in1;
cout << "Sum = " << sum << endl;}
SC_CTOR(consumer)
{
SC_SLAVE(accumulate, in1);
sum = 0; // initialize
};
SC_MODULE(top) // container
{
producer *A1;
consumer *B1;
sc_link_mp<int> link1;
SC_CTOR(top)
{
A1 = new producer("A1");
A1.out1(link1);
B1 = new consumer("B1");
B1.in1(link1);}};
```

Can you see what this software does?

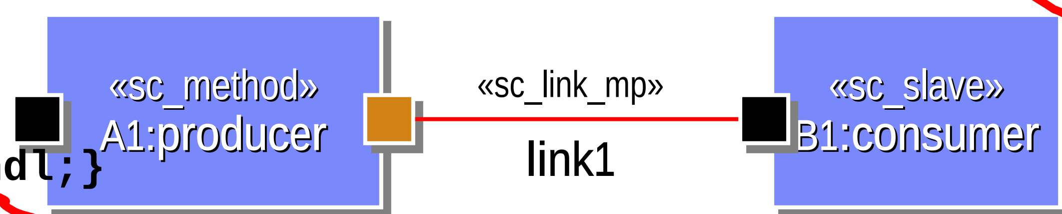
…and its Model



Can you see it now?

The Model and the Code

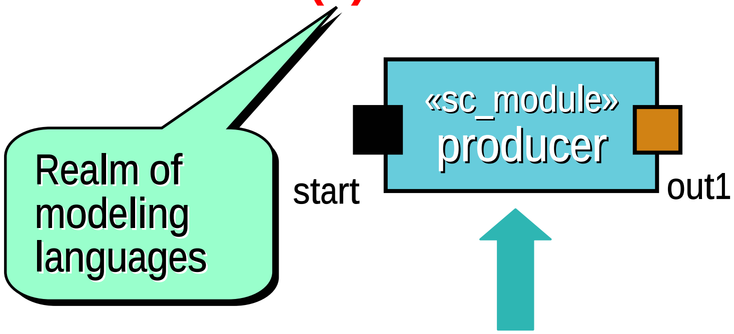
```
SC_MODULE(producer)
{
sc_outmaster<int> out1;
sc_in<bool> start; // kick-start
void generate_data ()
{
for(int i =0; i <10; i++) {
out1 =i ; //to invoke slave;}
}
SC_CTOR(producer)
{
SC_METHOD(generate_data);
sensitive << start;}};
SC_MODULE(consumer)
{
sc_inslave<int> in1;
int sum; // state variable
void accumulate (){
sum += in1;
cout << “Sum = “ << sum << endl;}
SC_CTOR(consumer)
{
SC_SLAVE(accumulate, in1);
sum = 0; // initialize
};
SC_MODULE(top) // container
{
producer *A1;
consumer *B1;
sc_link_mp<int> link1;
SC_CTOR(top)
{
A1 = new producer(“A1”);
A1.out1(link1);
B1 = new consumer(“B1”);
B1.in1(link1);}};
```



Model-Driven Development (MDD)

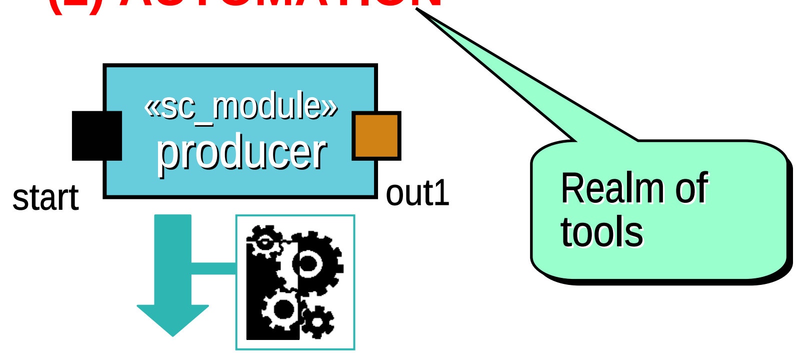
- **An approach to software development in which the *focus* and primary artifacts of development are *models* (vs programs)**
- **Based on two time-proven methods:**

(1) ABSTRACTION



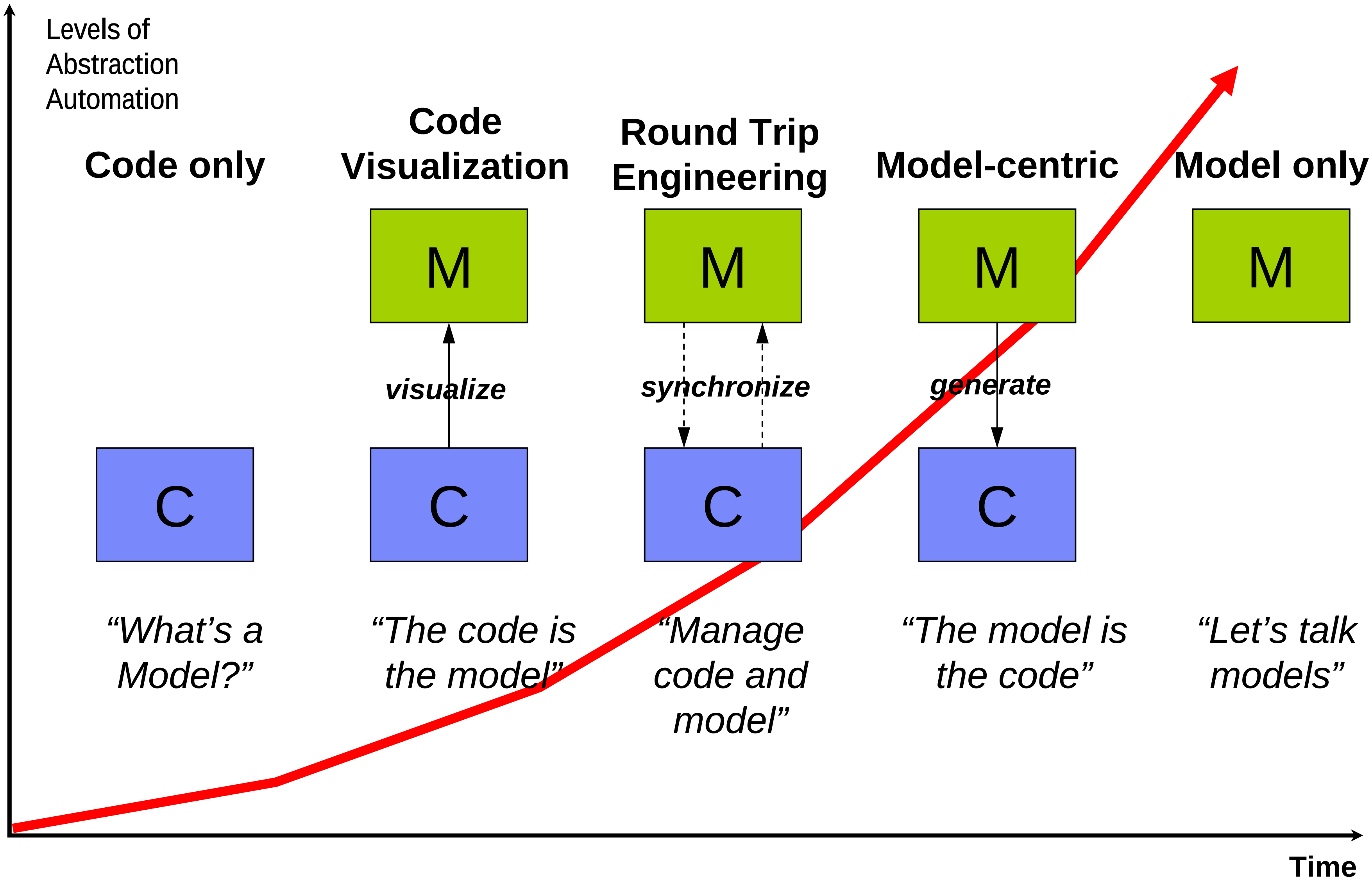

```
SC_MODULE(producer)
{sc_inslave<int> in1;
int sum; //
void accumulate (){
sum += in1;
cout << "Sum = " <<
sum << endl;}
```

(2) AUTOMATION




```
SC_MODULE(producer)
{sc_inslave<int> in1;
int sum; //
void accumulate (){
sum += in1;
cout << "Sum = " <<
sum << endl;}
```

Styles of MDD: The MDD Maturity Model



State of the Art in MDD

- **Example: Major Telecom Equipment Vendor**
  - Adopted MDD Tooling
  - Rose RealTime, Test RealTime, RUP
- **Product 1: Radio Base Station**
  - 2 Million lines of C++ code
  - 100 developers
- **Product 2: Gateway**
  - 300,000 lines of C++ code
  - 30 developers
- **Product 3: Network Controller**
  - 4.5 Million lines of C++ code
  - 400 developers
- **Performance:**
  - Within ± 15% of hand coding

Sampling of Embedded Software Developed Using MDD

Automated doors, Base Station, Billing (In Telephone Switches), Broadband Access, Gateway, Camera, Car Audio, Convertible roof controller, Control Systems, DSL, Elevators, Embedded Control, GPS, Engine Monitoring, Entertainment, Fault Management, Military Data/Voice Communications, Missile Systems, Executable Architecture (Simulation), DNA Sequencing, Industrial Laser Control, Karaoke, Media Gateway, Modeling Of Software Architectures, Medical Devices, Military And Aerospace, Mobile Phone (GSM/3G), Modem, Automated Concrete Mixing Factory, Private Branch Exchange (PBX), Operations And Maintenance, Optical Switching, Industrial Robot, Phone, Radio Network Controller, Routing, Operational Logic, Security and fire monitoring systems, Surgical Robot, Surveillance Systems, Testing And Instrumentation Equipment, Train Control, Train to Signal box Communications, Voice Over IP, Wafer Processing, Wireless Phone

MDD Helps, but…

- **By itself it does not provide answers to the following types of questions that are key in engineering systems design:**
  - Will the proposed software architecture satisfy its required deadlines?
  - How much buffer space do I need to provide for the anticipated traffic load?
  - Will the system meet its availability and reliability requirements?
  - Etc.

⇒**MDD is not enough**

Outline

- The impact of software on engineering design
- Introducing model-driven development (MDD)
- Adding the engineering aspect (MARTE)
- Adding the systems aspect (SysML)
- The challenges before us

Our Theme

Engineering ***(Merriam-Webster Collegiate Dictionary)*** :

the application of science and mathematics by which the properties of matter and the sources of energy in nature are made useful to people

Software vs Engineering

The Old View of Things:

"All machinery is derived from nature, and is founded on the teaching and instruction of the revolution of the firmament."

- Vitruvius
***On Architecture*, Book X**
1^{st} Century BC

…and the New:

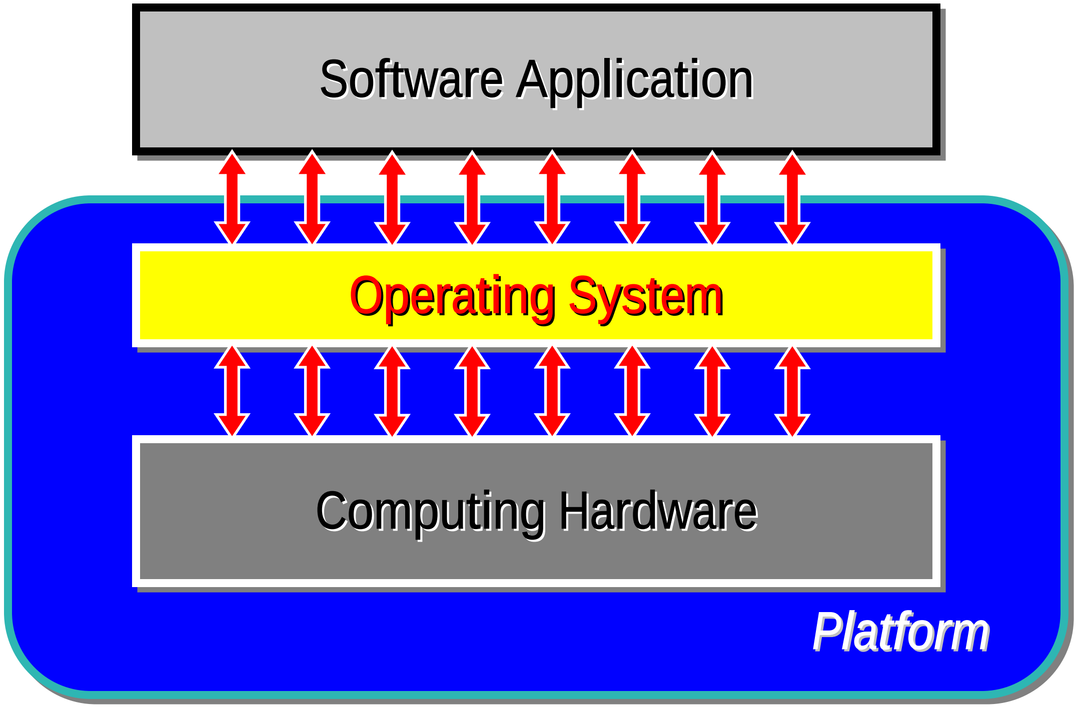
"Because [programs] are put together in the context of a set of information requirements, they observe no natural limits other than those imposed by those requirements. Unlike the world of engineering, there are no immutable laws to violate."

- **Wei-Lung Wang**
Comm. of the ACM (45, 5)

May 2002

What are Programs Made of?

The Raw Material of Programs



- **Platform:**

  the combination of software and hardware required to execute a program

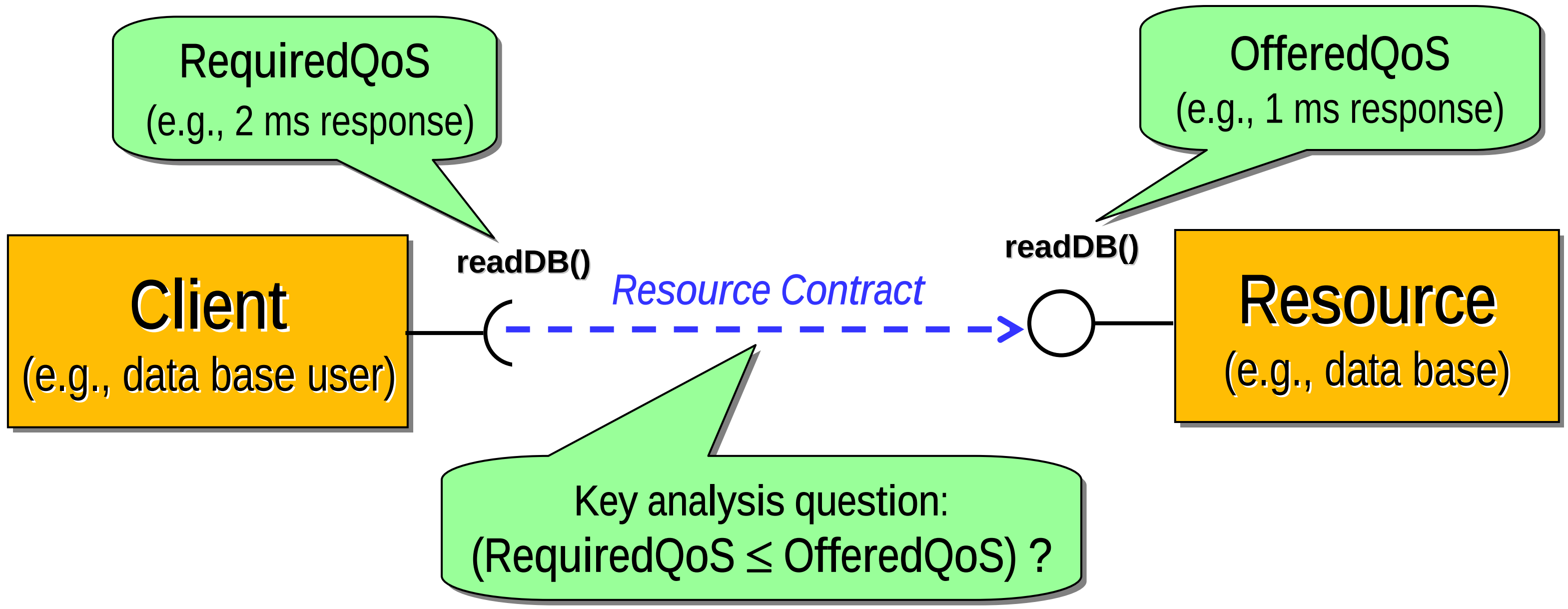
- **A platform constitutes the "raw material" of software whose physical properties can have a major impact on the KPIs of a system and may even affect its design**

Modeling Platforms

- ***Resource*:**

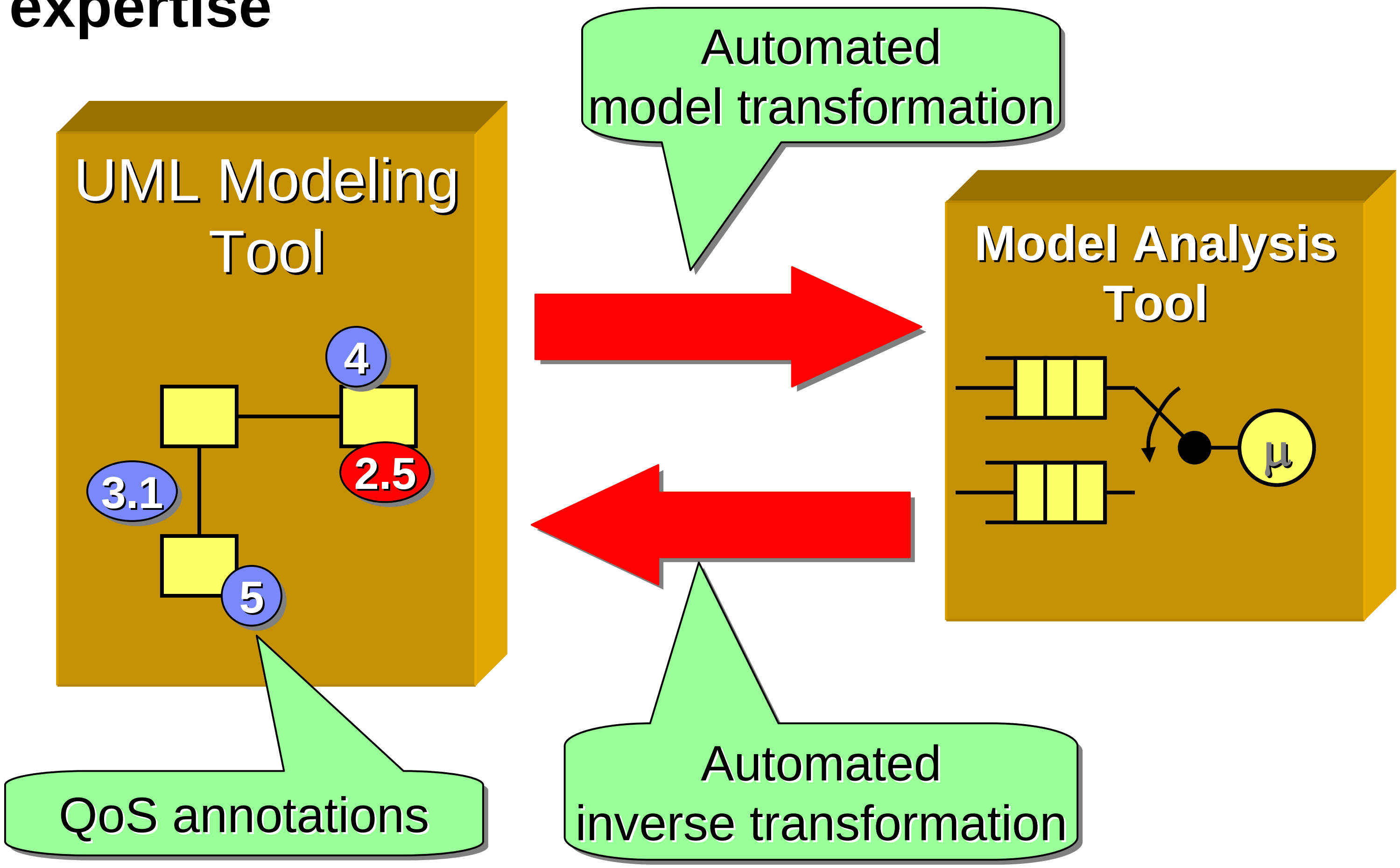
  an element that provides one or more services whose capacities (Qualities of Service (QoS)) are limited due to the properties of the underlying platform

- **These capacities are expressed as QoS characteristics that can be formally analyzed and predictions made**



Automating Complex KPI Analyses

- **Reduces need for rare and expensive analysis expertise**

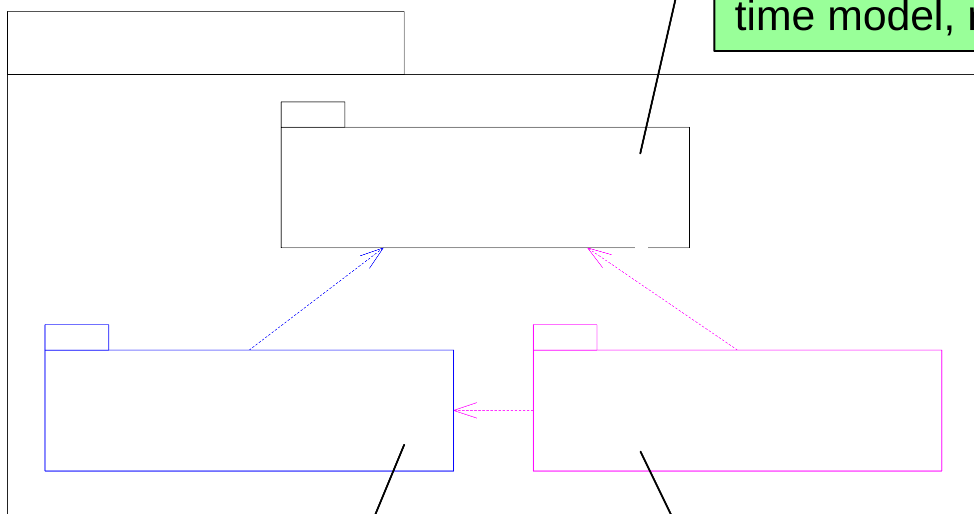


Introducing Physics to MDD: The MARTE Profile

- **UML profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE)**
  - An OMG standard profile, based on UML 2
- **Support precise modeling of key RTE systems phenomena**
  - Qualitative and quantitative modeling of HW and SW and relationships between them
- **Supports automated analyses of KPIs of RTE systems**
  - Schedulability analyses
  - Performance analyses

Architecture of the MARTE specification

Foundations for RTE systems modeling and analysis (e.g., time model, resource model)

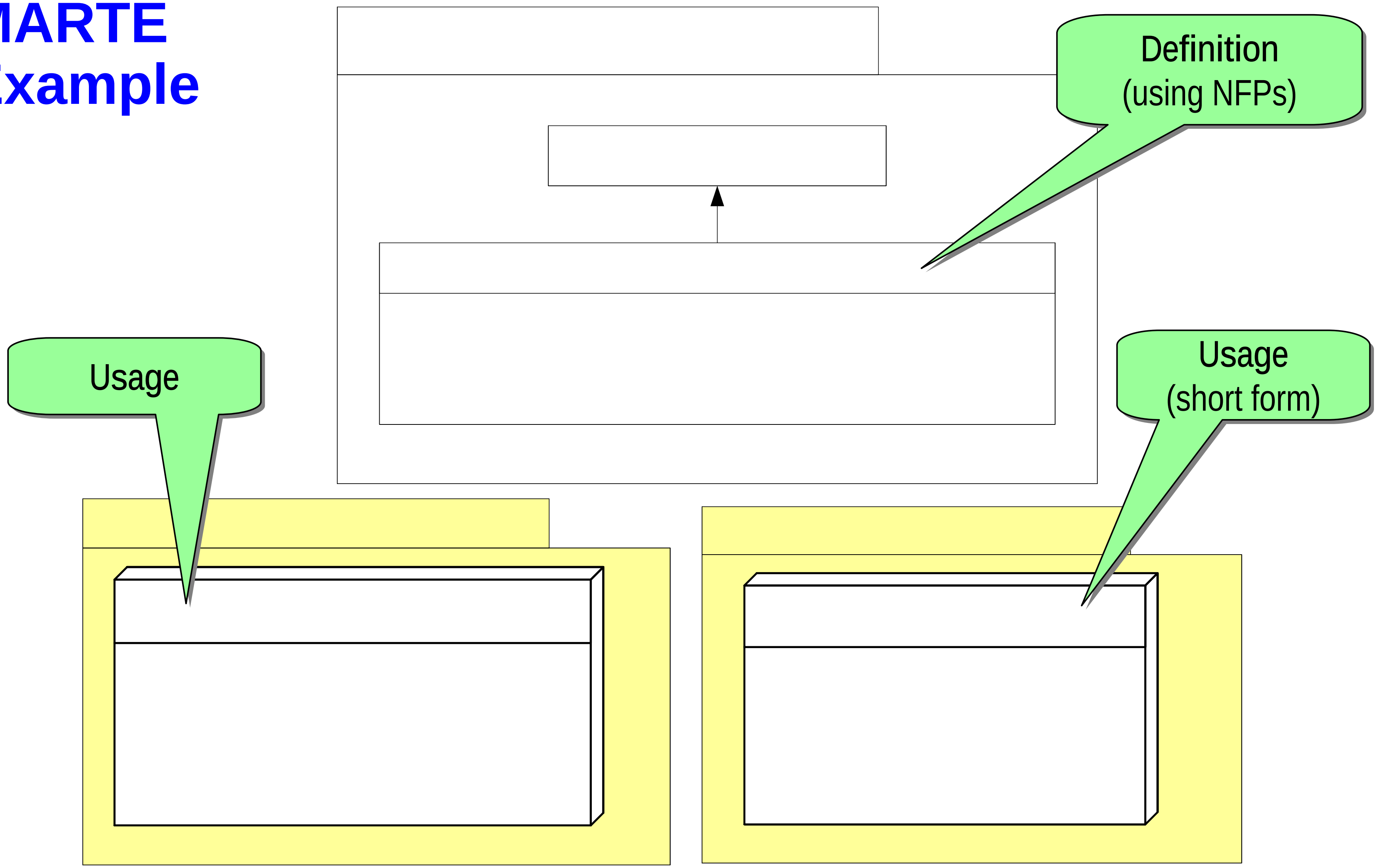


Specialization for precise modeling of RTE systems (e.g., CPUs, concurrency threads)

Specialization formal analyses of RTE systems (schedulability, performance)

(Slide credit of S. Gerard)

MARTE Example



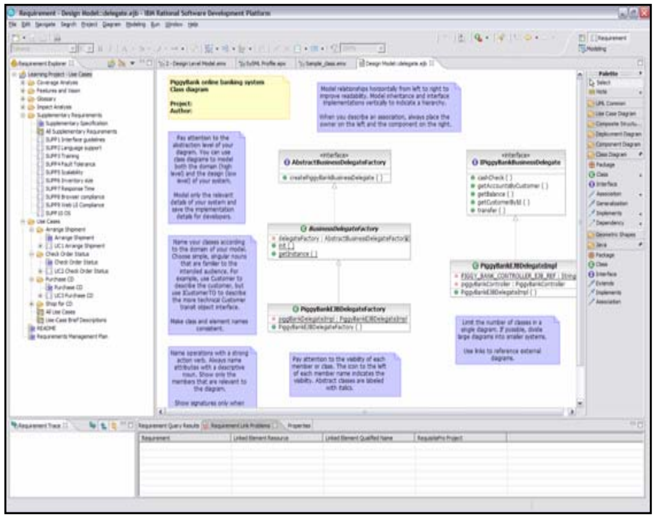
(Slide credit of S. Gerard)

Outline

- The impact of software on engineering design
- Introducing model-driven development (MDD)
- Adding the engineering aspect (MARTE)
- Adding the systems aspect (SysML)
- The challenges before us

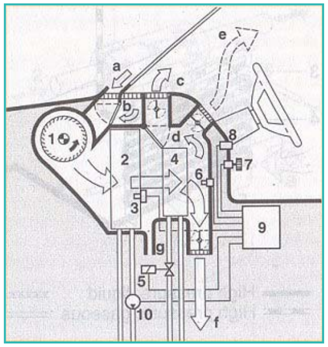
The System of Systems Design Problem

- **Early domain specialization often leads to:**
  - Inadequate requirements coverage
  - Suboptimal designs
  - Integration problems

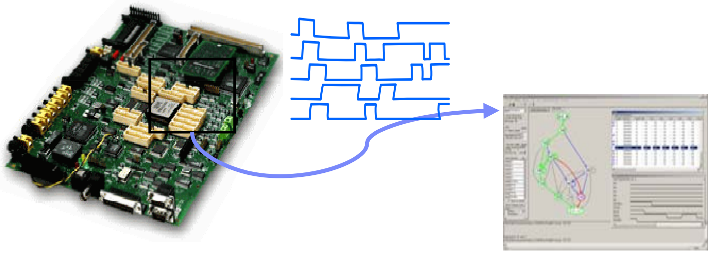


Software system

Mechanical system



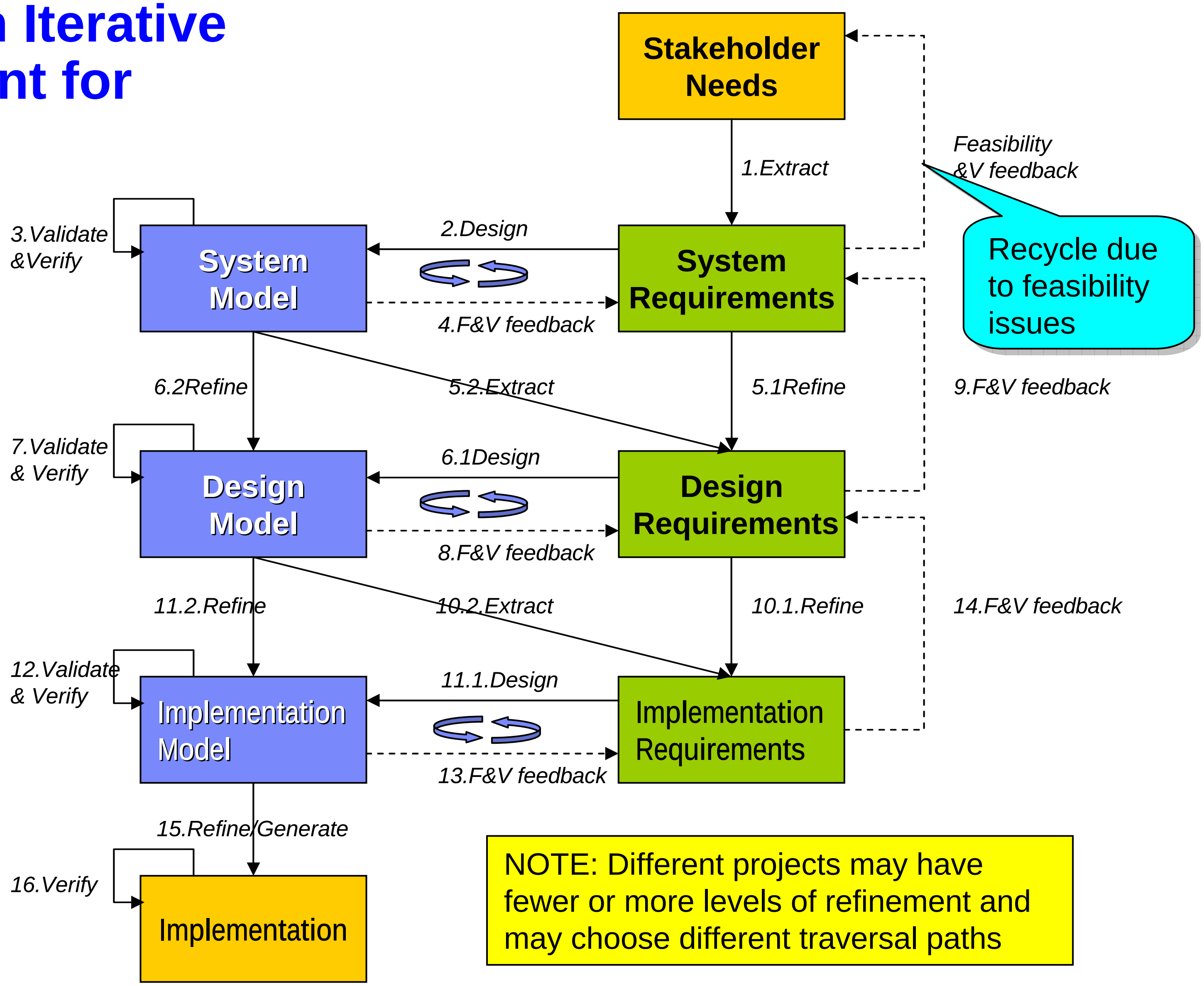
Electronics system



Systems Engineering (SE)

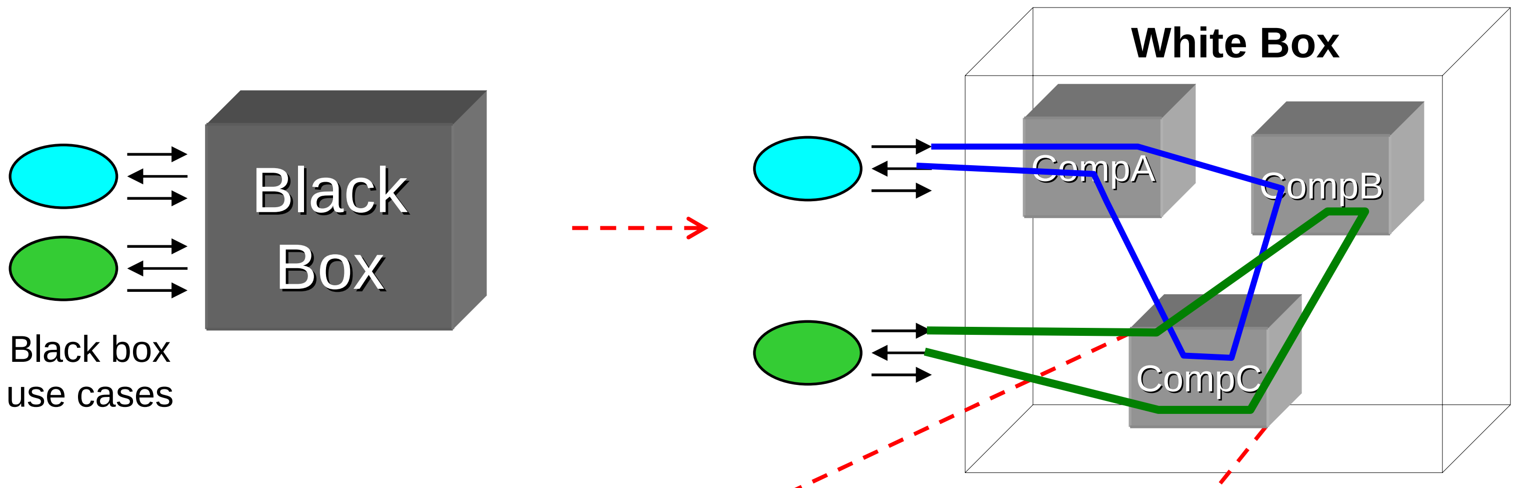
- "***Systems engineering*** ***is a holistic, product oriented engineering discipline whose responsibility is to create and execute an interdisciplinary process to ensure that customer and stakeholder needs are satisfied in a high quality, trustworthy, cost efficient, and schedule compliant manner throughout a system's life cycle.*** **" (International Council On Systems Engineering – INCOSE)**
- **SE is a mature discipline based on principles developed over 50 years ago**
  - Weak support for software modeling
  - Need to adopt it to iterative design model common in MDD

Risk-Driven Iterative Development for Systems



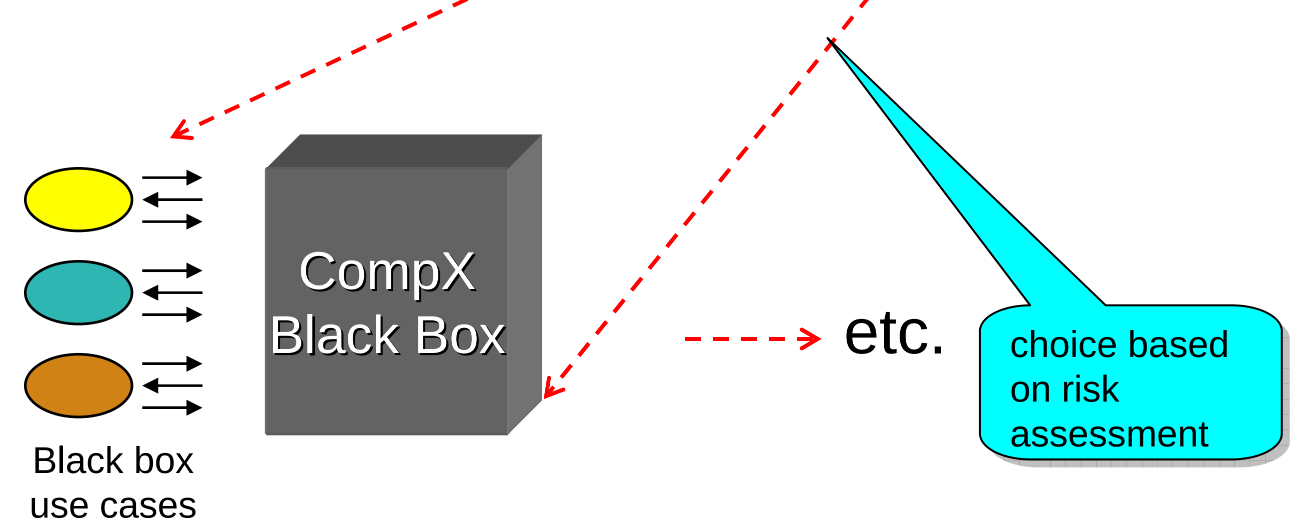
NOTE: Different projects may have fewer or more levels of refinement and may choose different traversal paths

Risk-Driven IterativeDevelopment: RUP-SE Process



Use-case flowdown:

- Focus on one-level of requirements at a time
- Decompose systems, not requirements

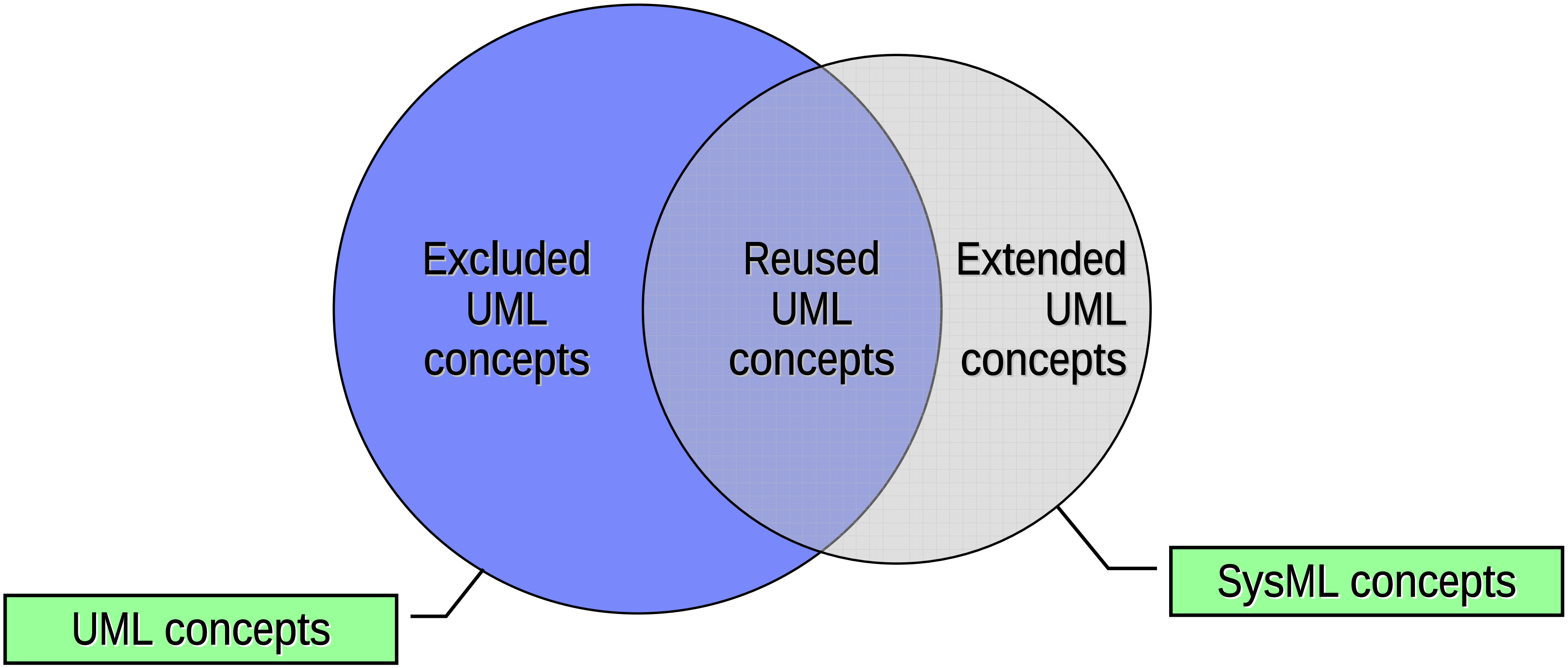


Enter SysML…

- **A graphical <u>modeling language</u> adopted by the OMG, in collaboration with INCOSE and AP233**
  - a UML profile that represents a subset of UML 2 with extensions for heterogeneous (SW/HW) modelling
  - Takes advantage of significant UML tooling support and experience
- **Supports the specification, analysis, design, verification, and validation of systems that include hardware, software, data, personnel, procedures, and facilities**
- **Supported by multiple vendors**

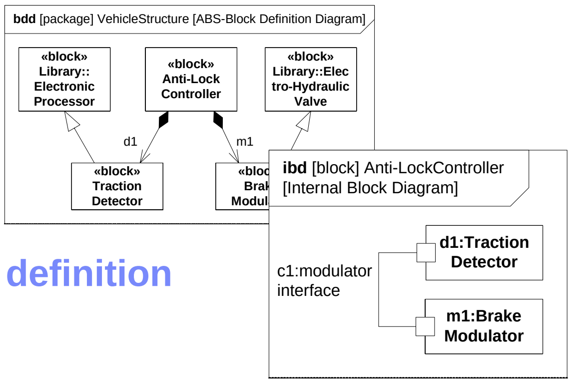
UML 2 and SysML

- **Uses a subset of UML concepts**
  - Simplified language
  - Provides SE-specific customization of certain UML concepts
  - However, it is possible to combine the excluded concepts if desired



SysML Basics

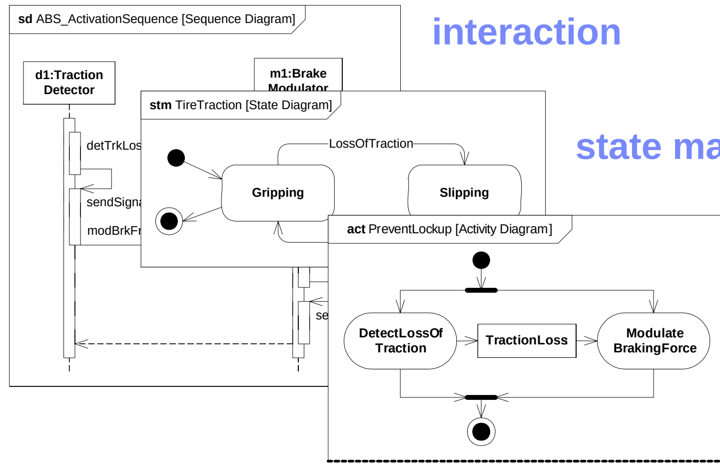
1. Structure



definition

use

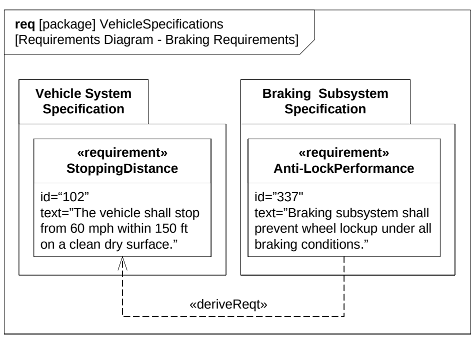
2. Behavior



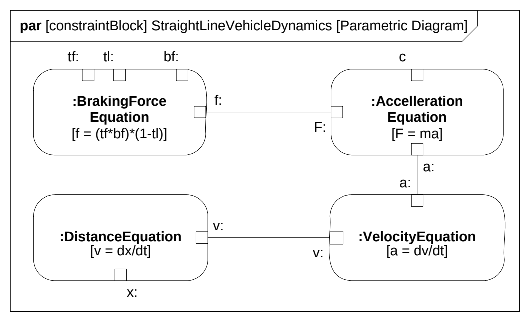
interaction

state machine

activity/
function

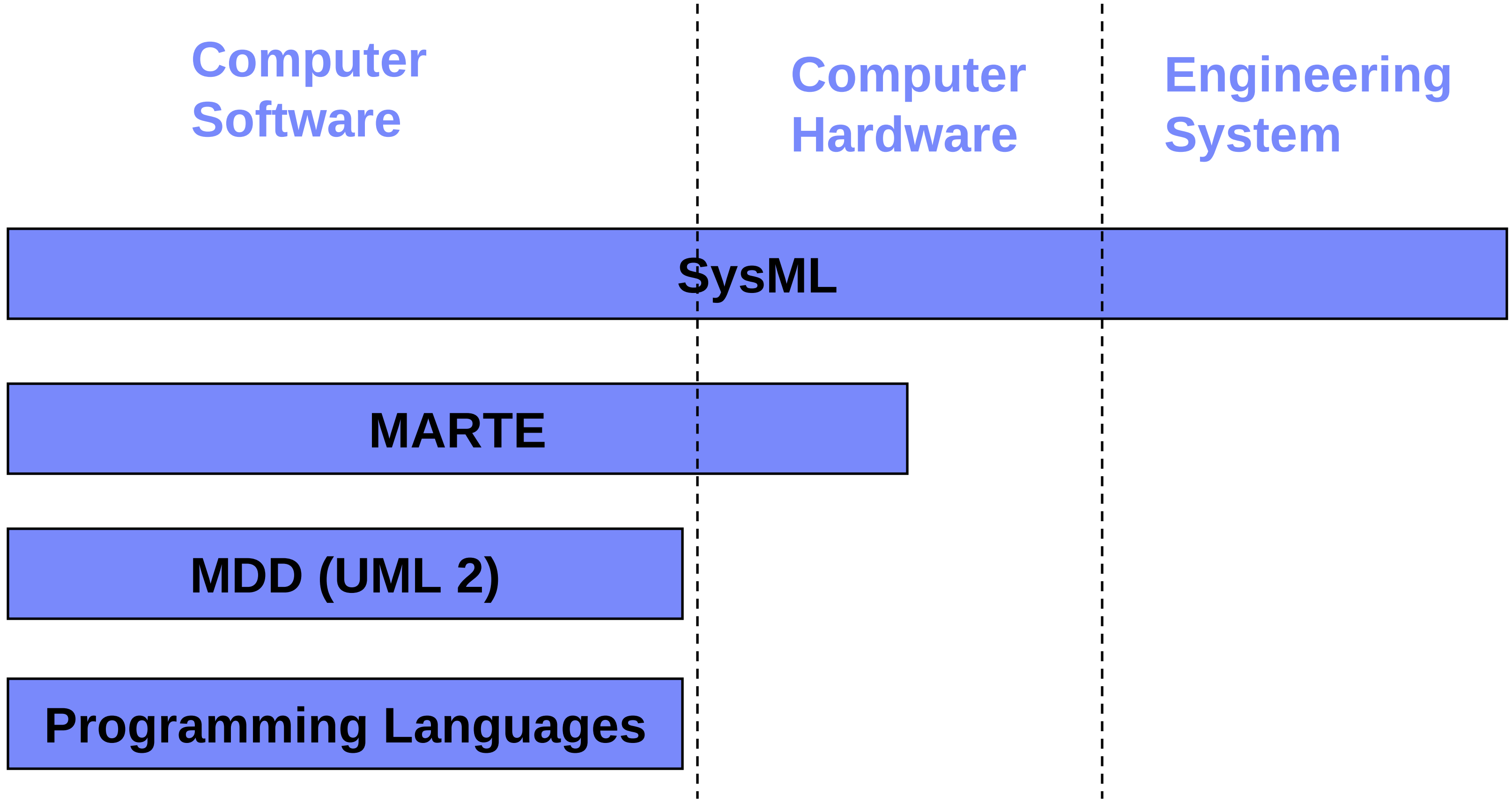


3. Requirements



4. Parametrics

The Solution Stack So Far…



- **Where do we go from here?**

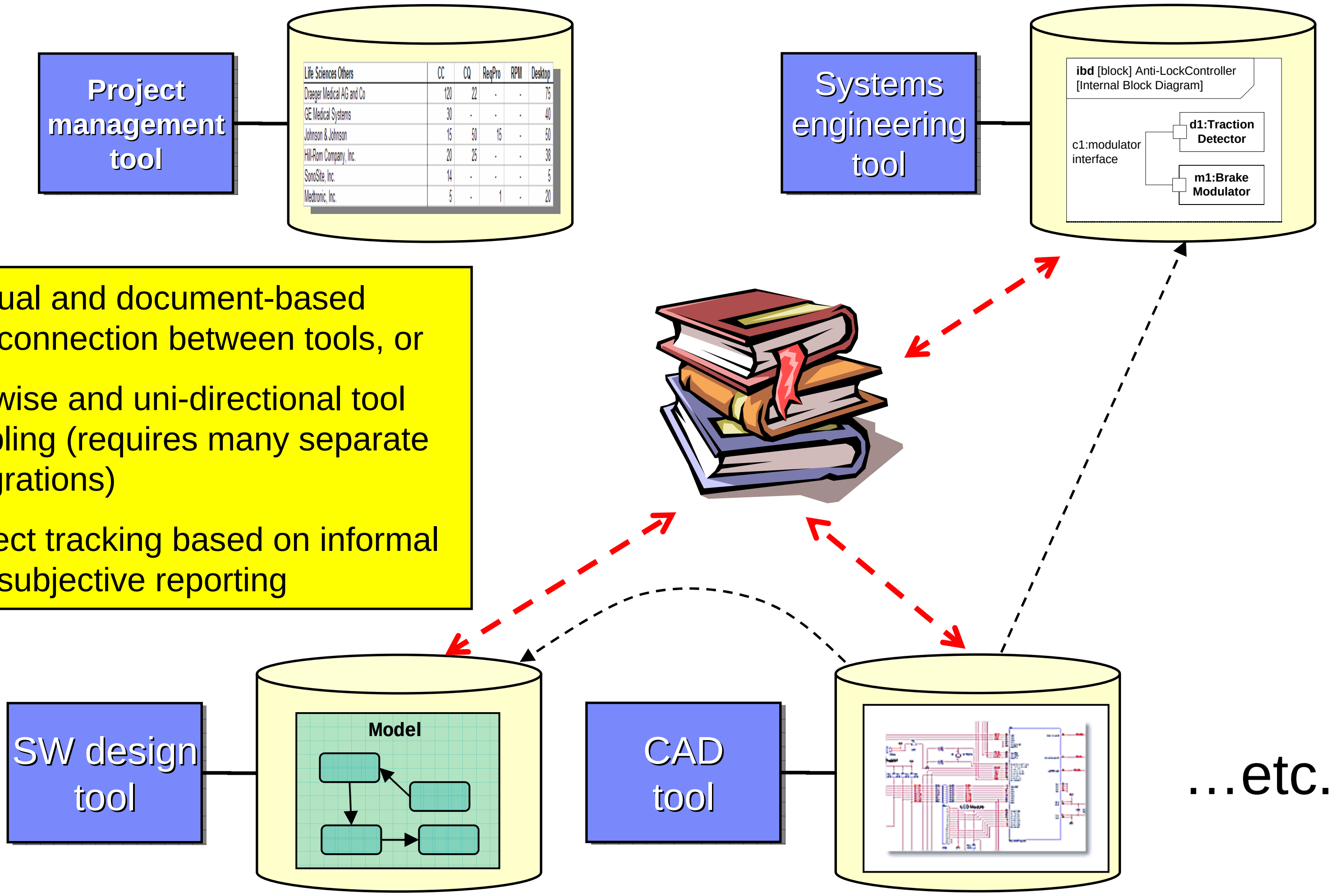
Outline

- The impact of software on engineering design
- Introducing model-driven development (MDD)
- Adding the engineering aspect (MARTE)
- Adding the systems aspect (SysML)
- The challenges before us

Major Systems Design Issues

- **Complex, flawed, and changing requirements**
- ***Governance* issues:**
  – Tracking real (vs subjective) progress to ensure timely delivery
  – Ensuring that the right product is delivered
- **Designs have to cope with full complexity of real-world phenomena**
  – E.g. concurrency, partial failures, effects of distribution, response-time deadlines
  – High levels of risk persist late into the development cycle
- **Heterogeneous and disconnected design processes, tools, and data**

Major Pain Point: Designs Disconnect



- Manual and document-based interconnection between tools, or
- Pairwise and uni-directional tool coupling (requires many separate integrations)
- Project tracking based on informal and subjective reporting

A Conceptual Framework for Systems Development

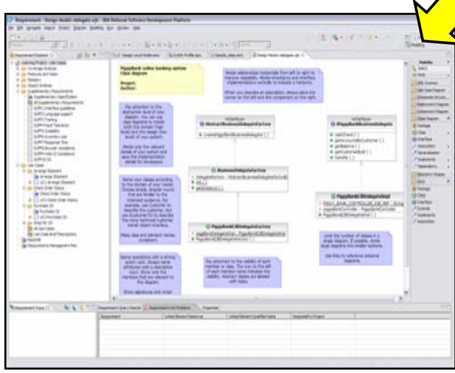
Systems Development

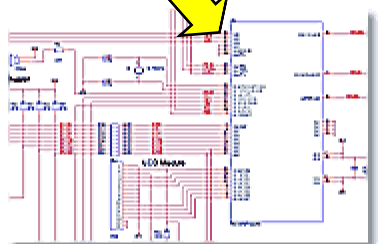
Systems of systems architecture, hardware/software co-design

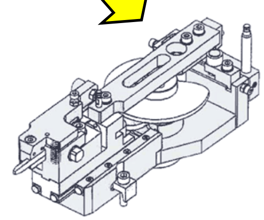
Domain (Subsystem) Development

System <u>Development</u> Governance

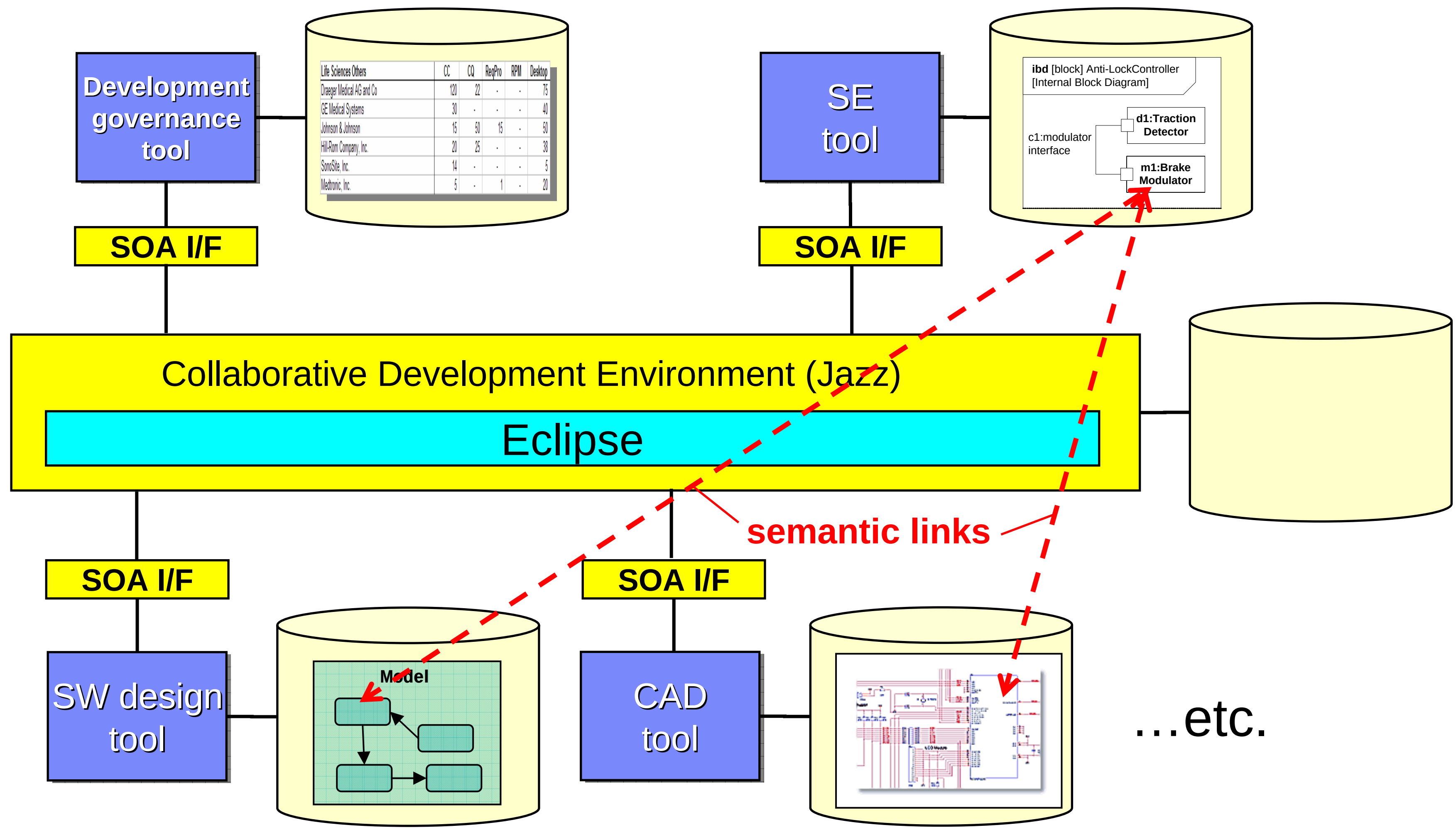
Collaborative engineering, development process management, information management





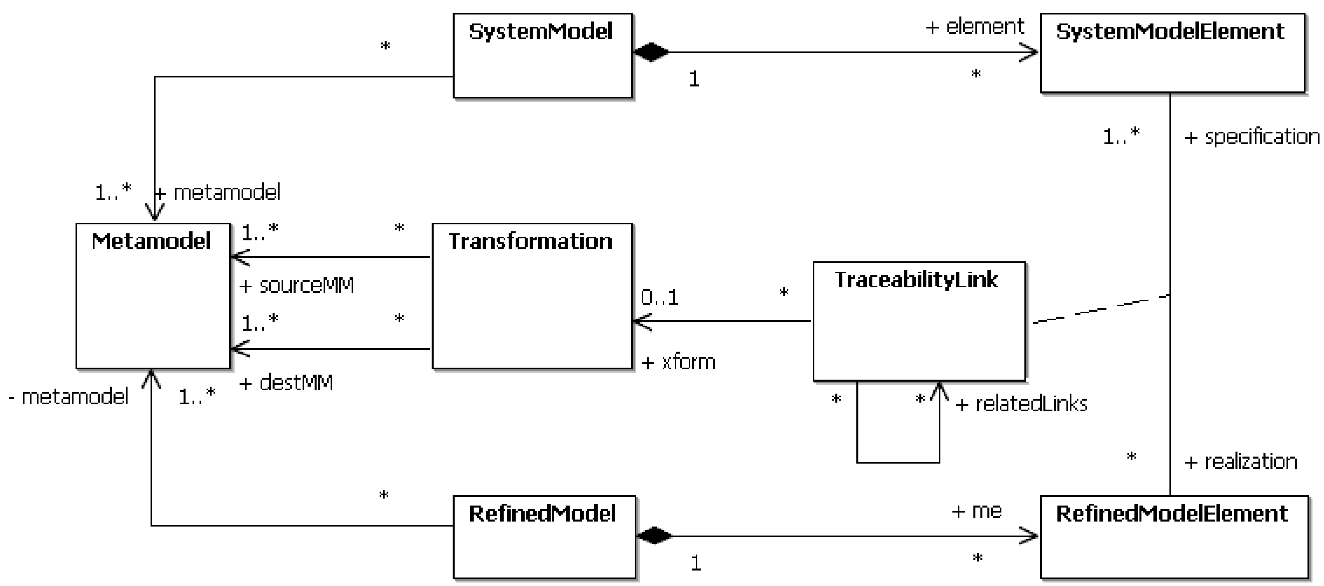


A Tooling Architecture for Systems Design



Semantic Links

- **For**
  - Requirements traceability
  - Links between system model and domain-specific models
- **Different levels of sophistication**
  - From simple hyperlinks to…
  - Sophisticated “intelligent” links (caching, transforms, etc.)

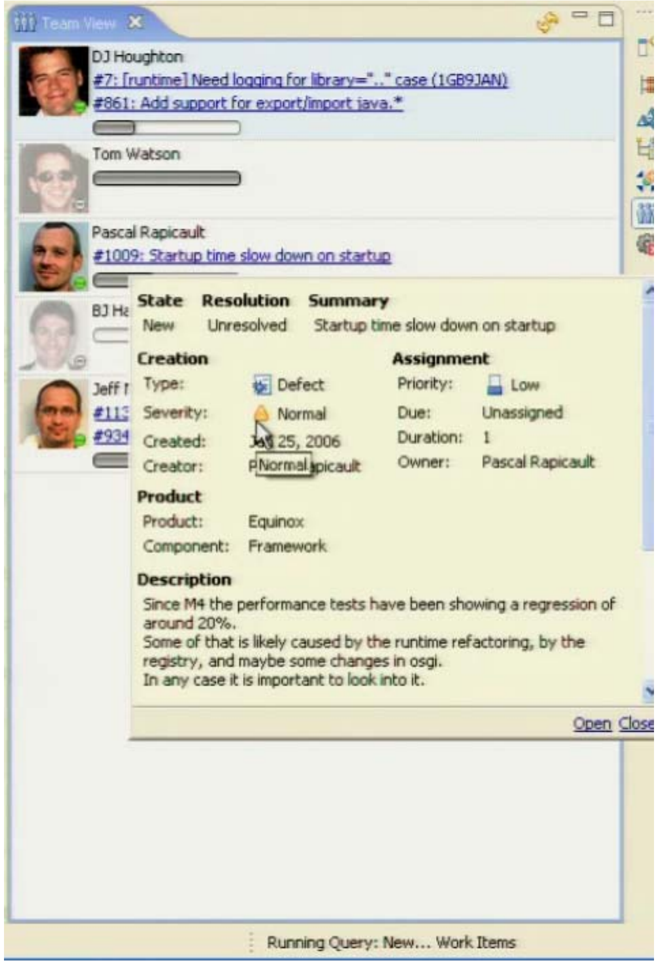


The Jazz Platform

- **An open source platform for collaborative distributed development with direct support for certain common team capabilities**
  - **Process Enactment**: Process rules, approval flows, based on RUP and agile practices drawn from Eclipse experience
  - **In-context Collaboration**: Collaborate around artifacts, work items, with full context awareness
  - **Change Management**: Versioning/Baselining of all project assets
  - **Defect Tracking**: Basic but extensible defect tracking
  - **Reporting and Project Health**: Basic but extensible reporting capabilities
  - **Team Build**: Extensible team build integration (e.g. CruiseControl, BuildForge)
  - **Cross-Lifecycle Traceability and Auditability**: Provide end-to-end lifecycle traceability
- **Analogous to Eclipse for the product development teams – custom products built as Jazz plug-ins**
- **Web site: jazz.net**

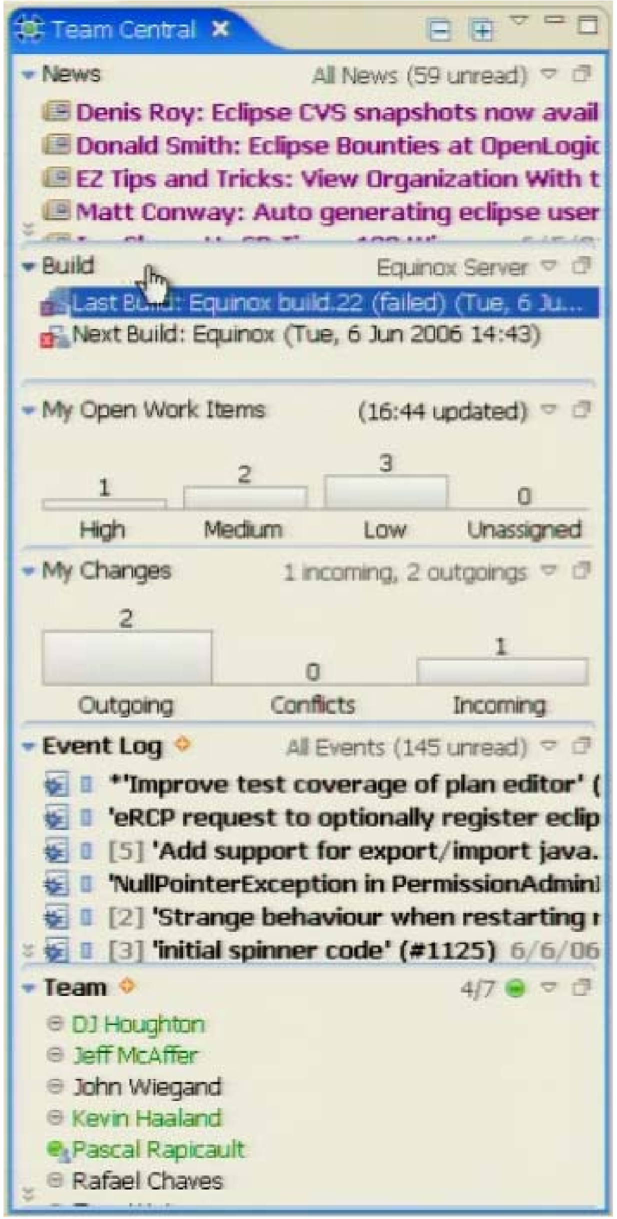
Jazz Capability: Team Awareness

- **Shows team members and their online status**
- **Shows what the team is working on**



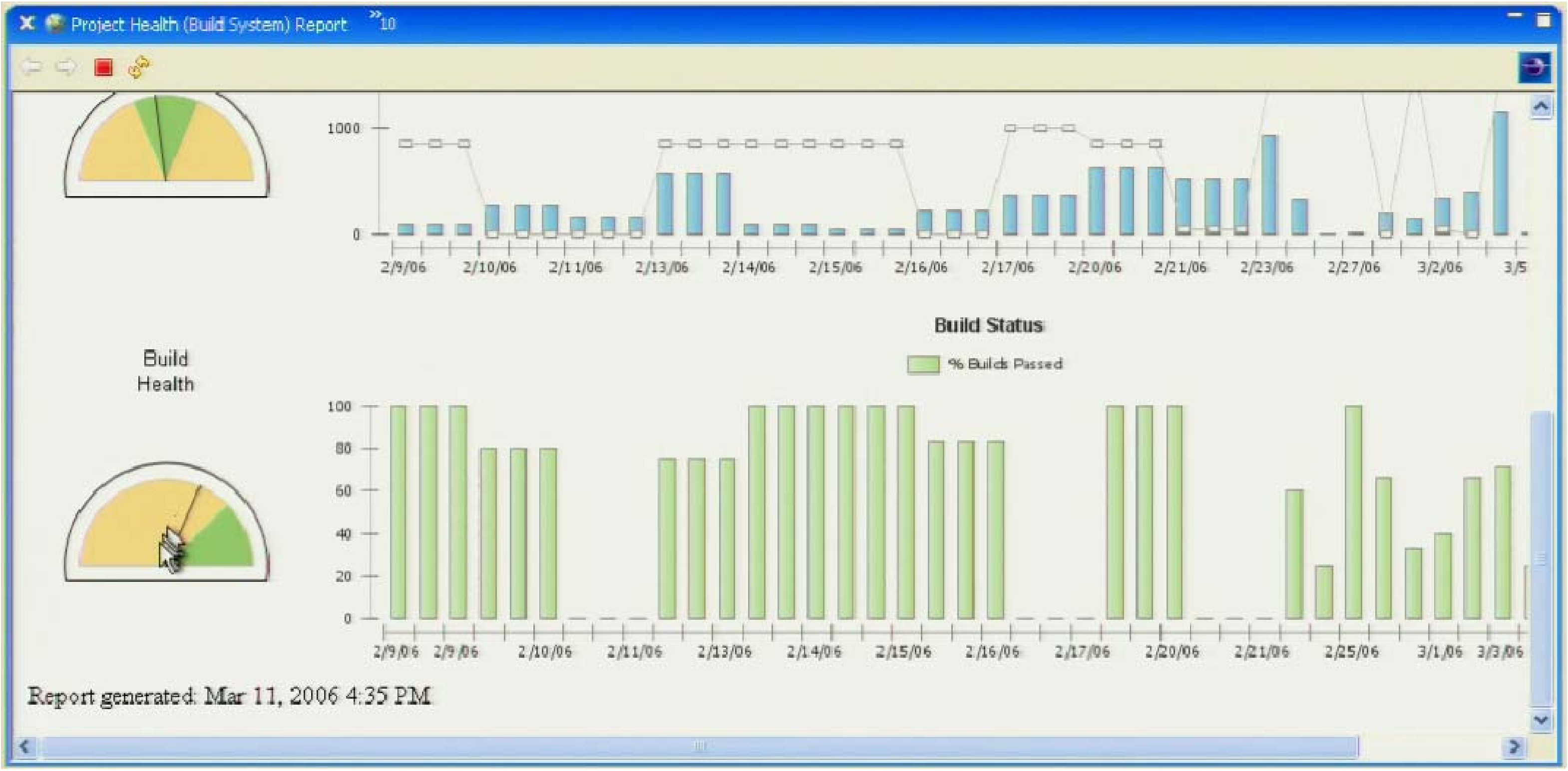
Jazz Capability: *Team Central*

- **Shows what is happening on project**
  - News & events
  - Build status
  - What's being worked on
  - Changes
- **Configurable (RSS feeds)**
  - New kinds of information easily added
- **Personalizable**
  - Each team member can tailor to their needs



Project Health Reporting

- **Based on data collected in real-time from actual development work**
  - Always accurate
  - No extra effort required to gather data



jazz

Conclusion

- **Software has proven to be both a blessing and a curse in the engineering of systems**
  – Offers unparalleled flexibility
  – Historically has been separated from mainstream engineering that has resulted in some spectacular failures and much distress
- **With MDD and MARTE we are now able to inject the "engineering" ingredient into software design**
- **Through SysML and in combination with new MDD-based methods and tools, we have the opportunity to effectively design and implement complex systems that combine software and hardware**
- **However, much research is required to develop appropriate languages, tools, and methods that would maximally exploit this potential**


Thank You