School of Electrical and Information Engineering

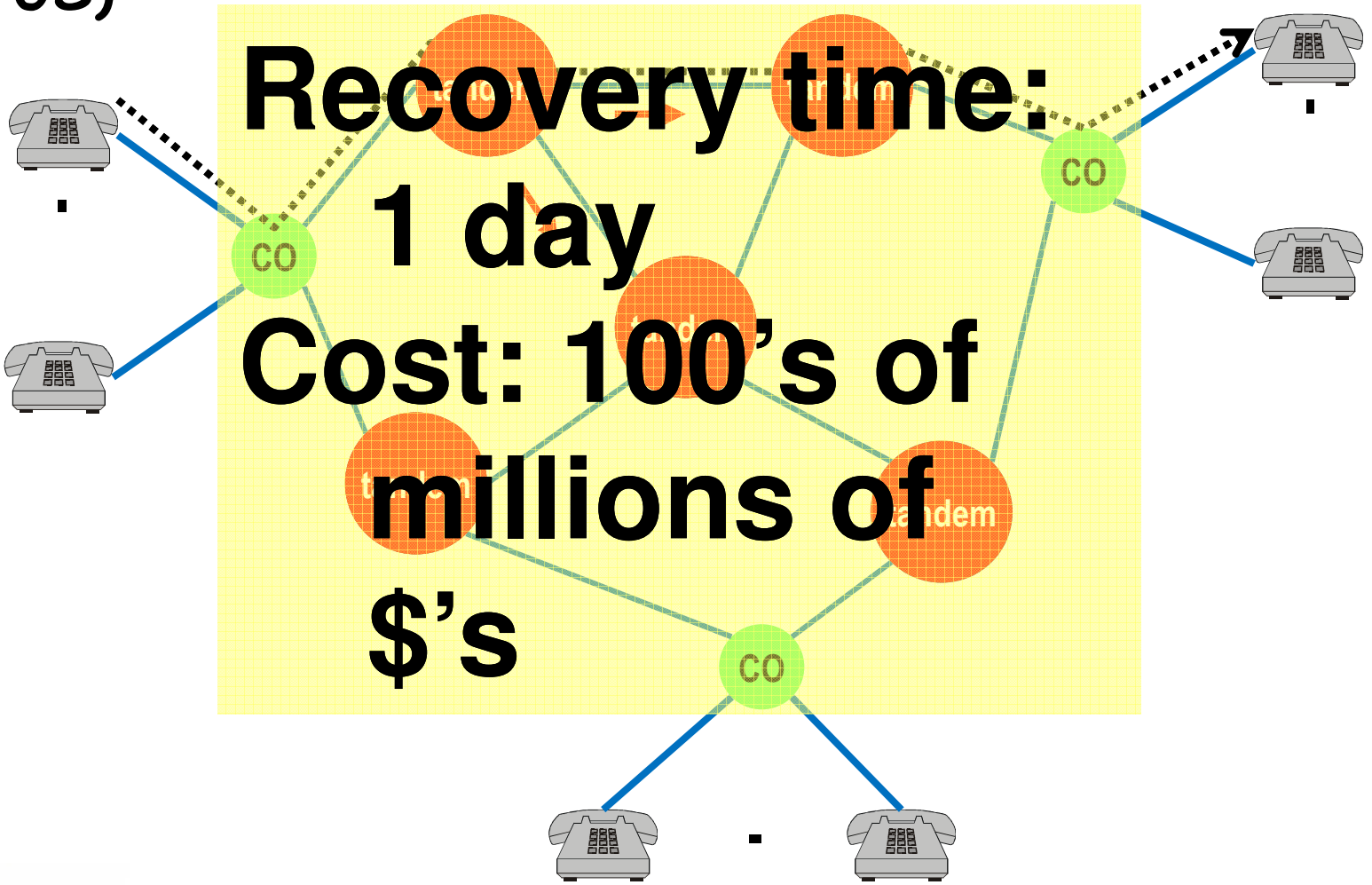
Computer Engineering Laboratory

An Introduction to Model-Based Software Engineering

ELEC 5620

The Tandem Tango…

- 1990: AT&T Long Distance Network (Northeastern US)



The Hidden Culprit

- The (missing) "break" that broke it

```
. . .;
switch (...) {
      case a : ...;
            break;
      case b :...;
            break;
            . . .
      case m : ...;
      case n : ...;
            . . .
};
```

…and, it's all HIS fault!

Ooops! Forgot the "break"…

Overview

- The Problem
- The Premise
- The Results

A Fragment 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 program is about?

The Enemy: Complexity

- Modern software systems are reaching levels of complexity comparable to those of biological systems
  - Systems of systems
- Furthermore, this trend will continue
  - …as will the demand for greater software reliability
- Given the current track record of software projects (>50%), what chances have we got?

WELCOME to HELL!

Fred Brooks on Complexity

- [From: F. Brooks, "The Mythical Man-Month", Addison Wesley, 1995]
- Essential complexity
  - inherent to the problem
  - cannot be eliminated by technology or technique
  - e.g., solving the "traveling salesman" problem
- Accidental complexity
  - due to technology or methods used to solve the problem
  - e.g., building a skyscraper using only hand tools

Mainstream Programming Languages

- Most mainstream programming languages abound in accidental complexity
- These languages are:
  - Difficult to understand
  - Defect intolerant, with a chaotic quality
- Modern variants (e.g., Java, C#) are not significantly more productive compared to the original languages (e.g., FORTRAN)
- (The embarrassing bit) Yet, we have persistently held on to these outdated technologies, investing enormous financial and intellectual resources in improving them
  - ..at the cost of overlooking many new and better approaches

The Impact

- **Abstraction (modeling) of programs is extremely difficult and risky**
  - Any detail can be critical!
  - Eliminates our most effective means for managing complexity
- **Our ability to exploit formal mathematical methods is severely impeded**
  - Mathematics is at the core of all successful modern engineering
- **We <u>must</u> do something different**
  - *"Problems cannot be solved by the same level of thinking that created them" – A. Einstein*

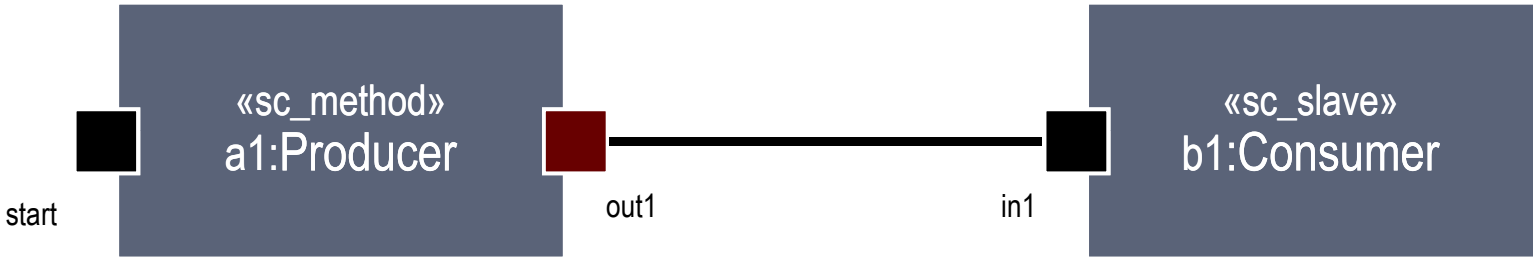
Back to "Modern" Software

```
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 program is about?

…Corresponding UML Model

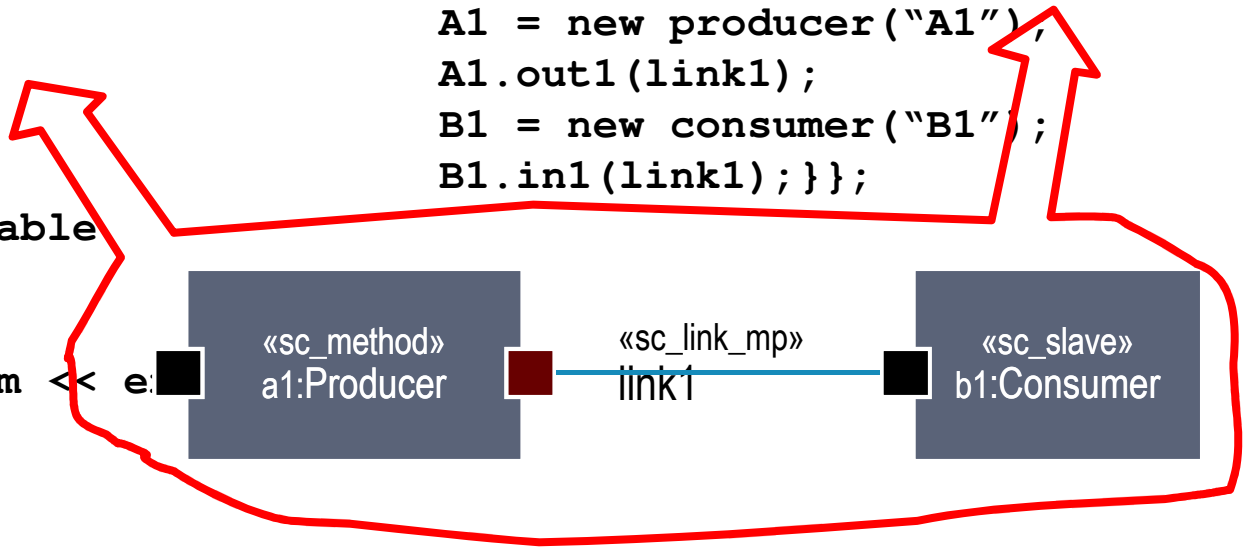


Can you see it now?

The Program and Its Model

```
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 << e
```

```
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);}};
```



Overview

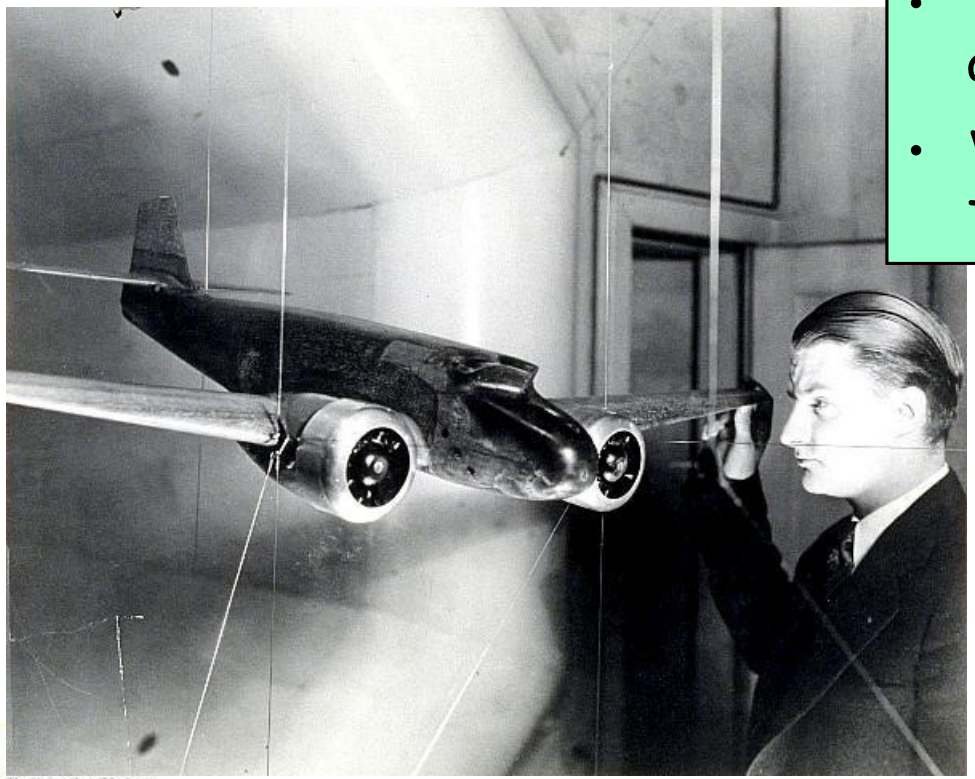
- The Problem
- The Premise
- The Results

Why Do Engineers Build Models?

- To understand
  - …the interesting characteristics of an existing or desired (complex) system and its environment
- To predict
  - …the interesting characteristics of the system by analysing its model(s)
- To communicate
  - …their understanding and design intent (to others and to oneself!)
- To specify
  - ...the implementation of the system (models as blueprints)

Engineering Models

- **Engineering model:**
  - *A selective representation of some system that captures accurately and concisely all of its essential properties of interest for a given set of concerns*



- We don't see everything at once
- What we do see is adjusted to human understanding

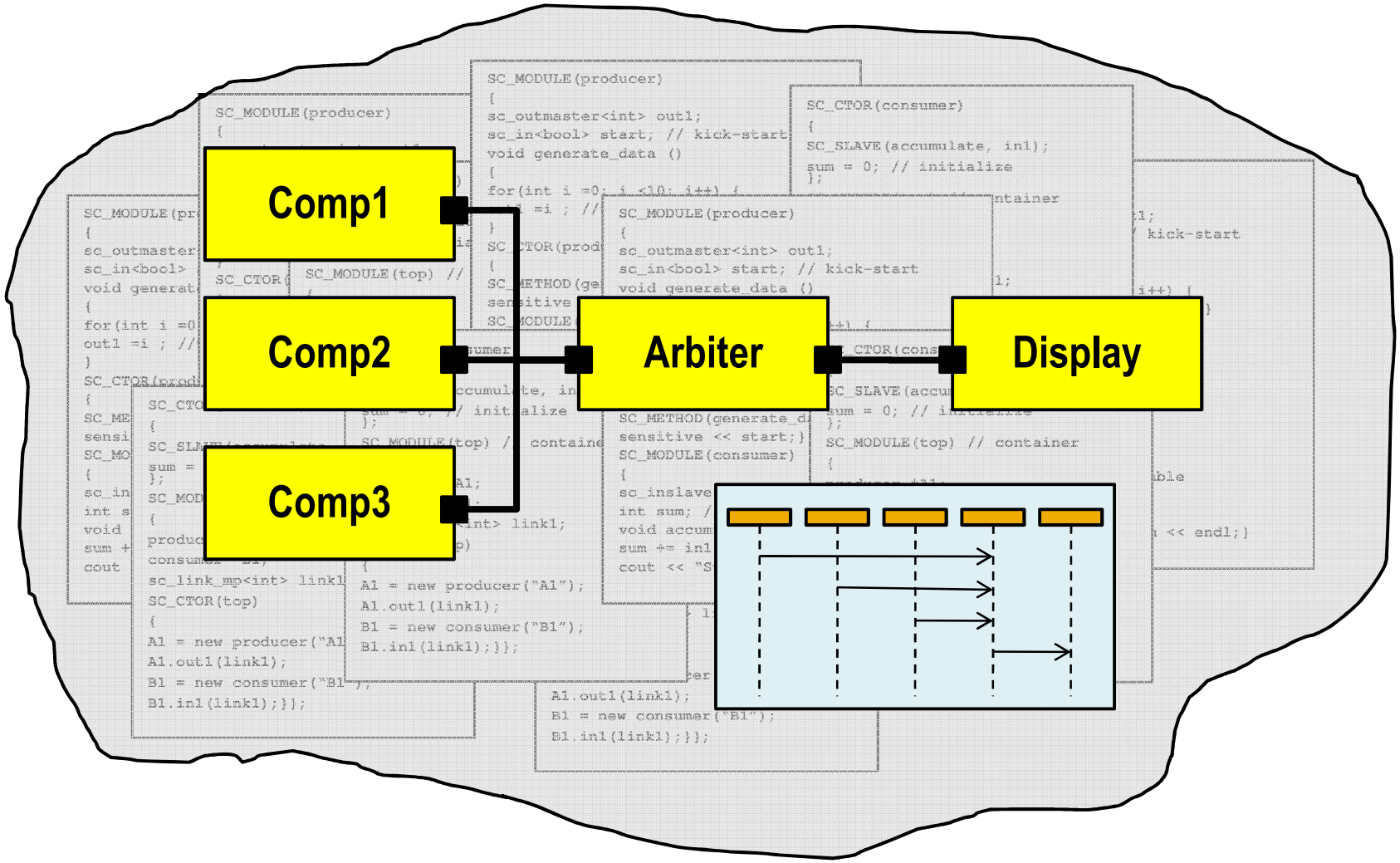
Characteristics of Useful Engineering Models

- **Purposeful:**
  - Constructed to address a specific set of concerns/audience
- **Abstract**
  - Emphasize important aspects while removing irrelevant ones
- **Understandable**
  - Expressed in a form that is readily understood by observers
- **Accurate**
  - Faithfully represents the modeled system
- **Predictive**
  - Can be used to answer questions about the modeled system
- **Cost effective**
  - Should be much cheaper and faster to construct than actual system

To be useful, engineering models must satisfy at least these characteristics!

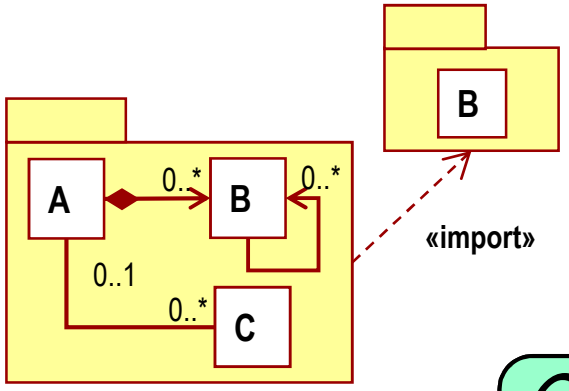
What About Software?

- An abstract representation of a software system

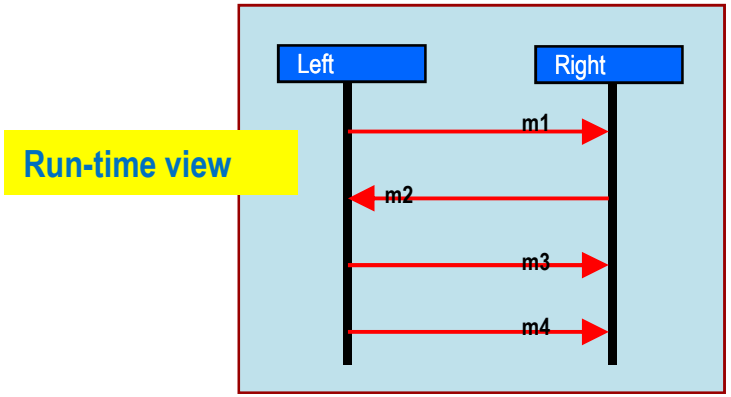


Models of Software

- **Software model**: An engineering model (specified using a modeling language) of some software that represents:
  1. Run-time views of the software: the structure and behavior of the *software in execution* and/or
  2. Design-time views of the software: The structure and content of the *software specification*



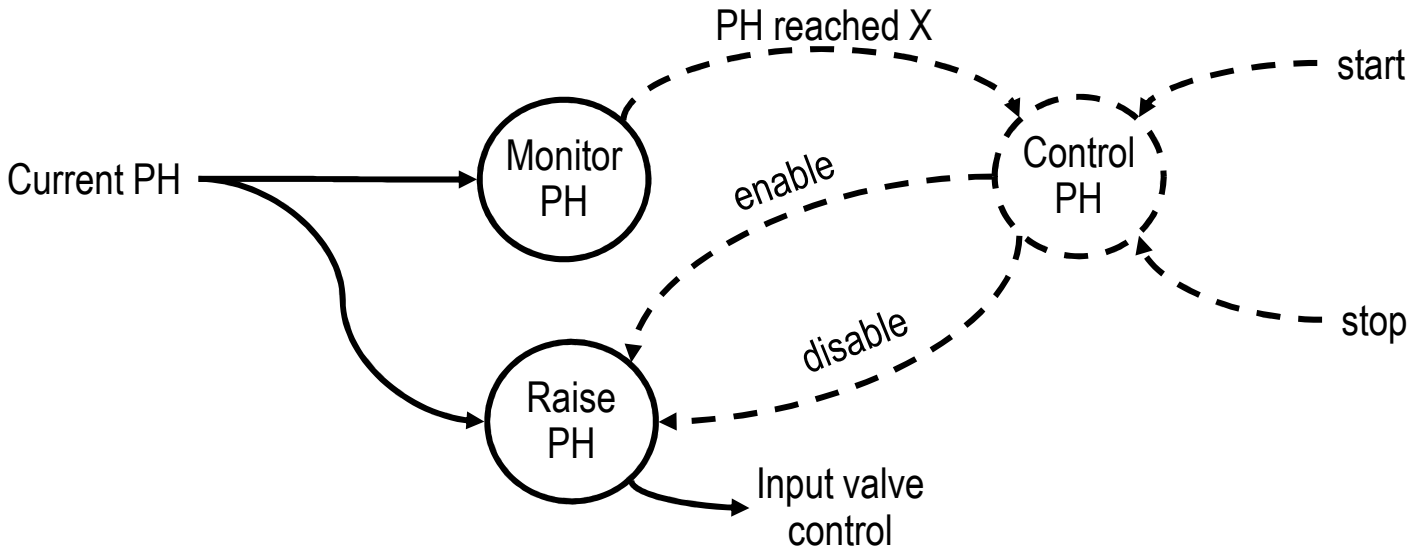
Design-time view



Run-time view

One of the primary motives for many modeling languages is the need to more clearly represent software in execution

Modeling Software: SA/SD

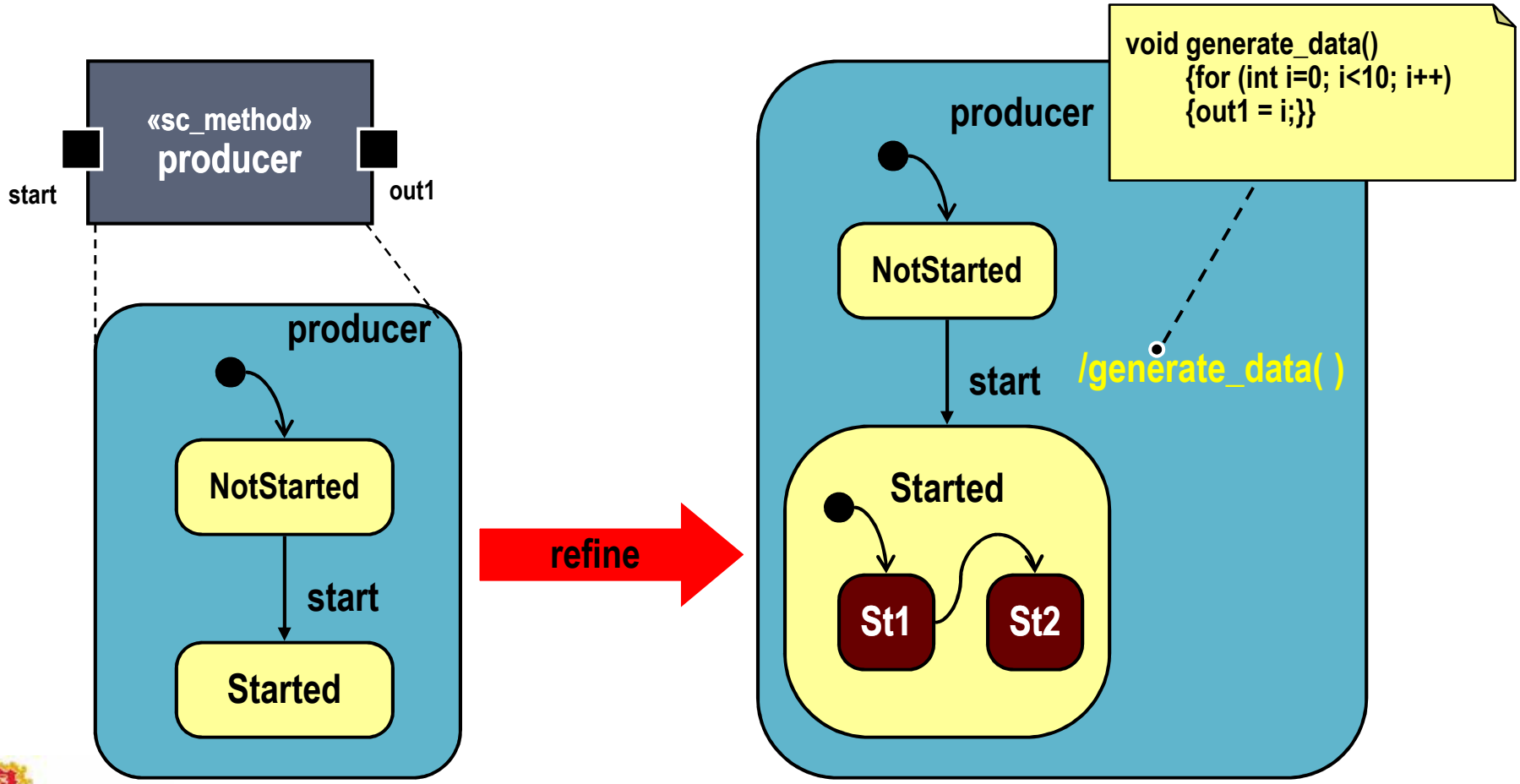


"…bubbles and arrows, as opposed to programs, …never crash"

-- B. Meyer
"UML: The Positive Spin"
American Programmer, 1997

Modern MBSE Development Style

- Models can be refined continuously until the application is fully specified ⇒ ***the model becomes the system that it was modeling!***



A Unique Feature of Software

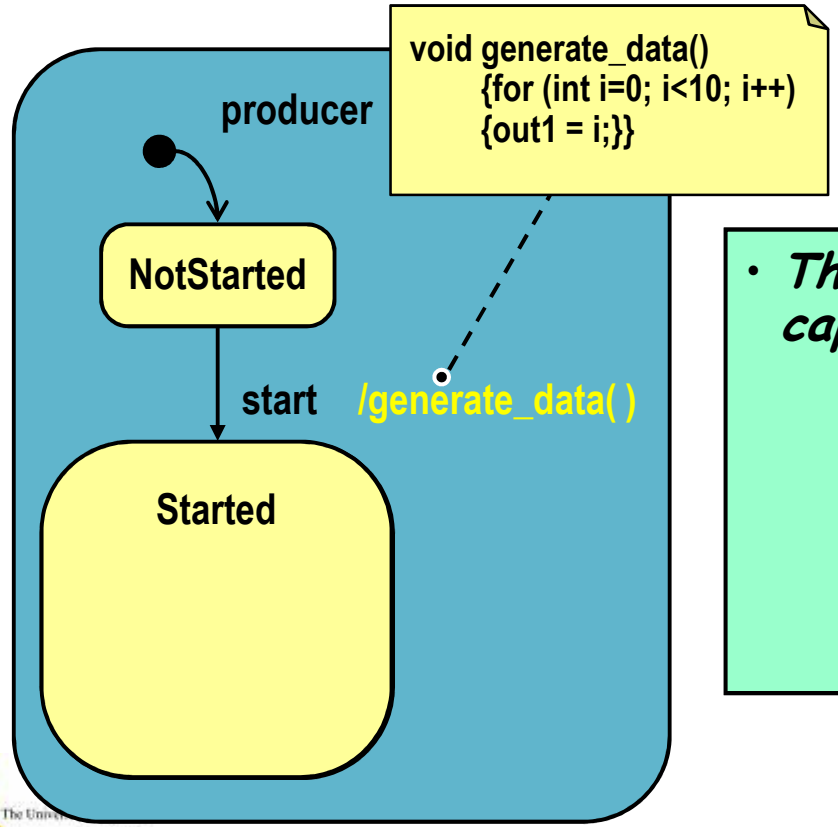
- **A software model and the software being modeled share the same medium—the computer**
  - Which also happens to be our most advanced and most versatile automation technology

Software has the unique property that it allows us to directly evolve models into implementations without fundamental discontinuities in the expertise, tools, or methods!

⇒ High probability that key design decisions will be preserved in the implementation and that the results of prior analyses will be valid

But, if the Model is the System…

- …do we not lose the abstraction value of models?



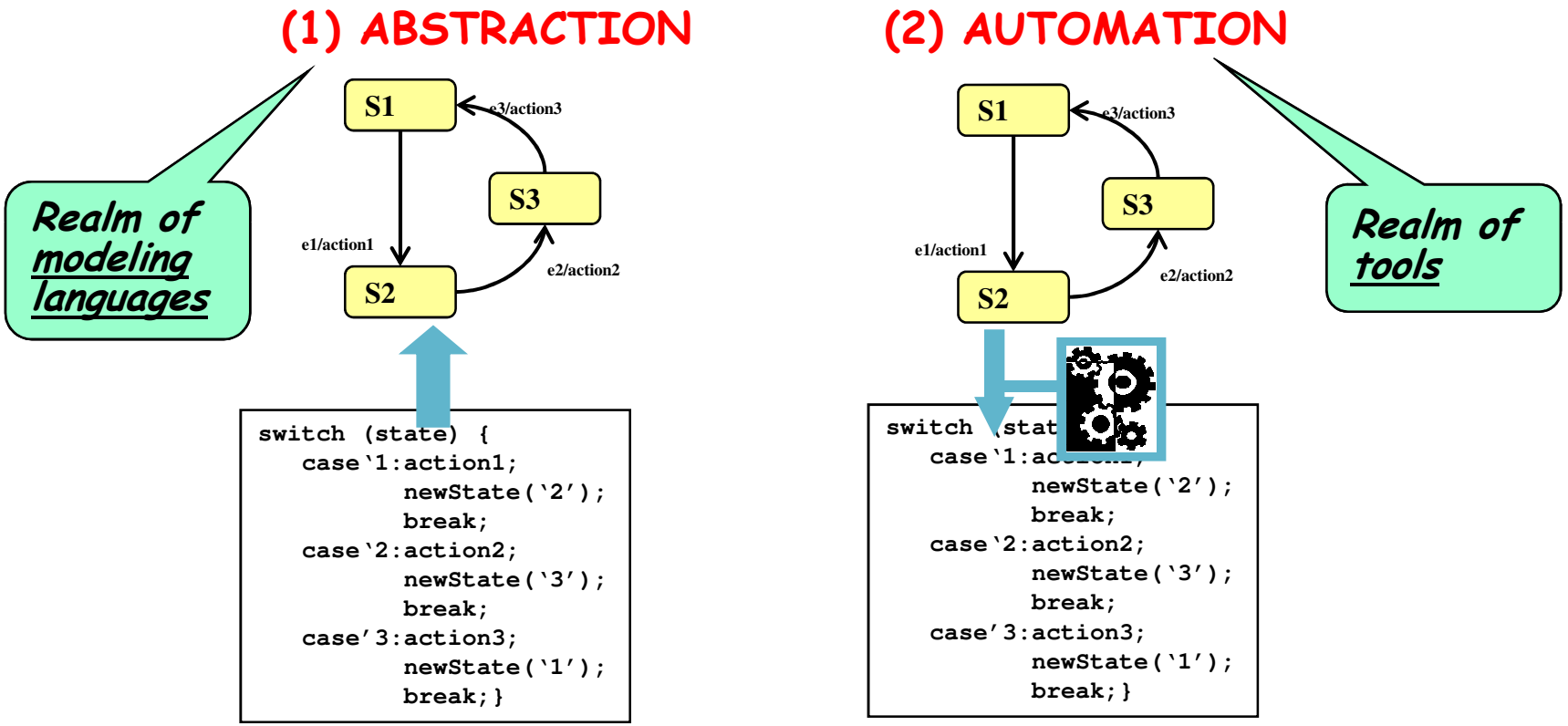
- *The computer offers a uniquely capable abstraction device:*

  Software can be represented from any desired viewpoint at any desired level of abstraction

  The abstraction is inside the system and can be extracted automatically

The Model-Based Engineering (MBE) Approach

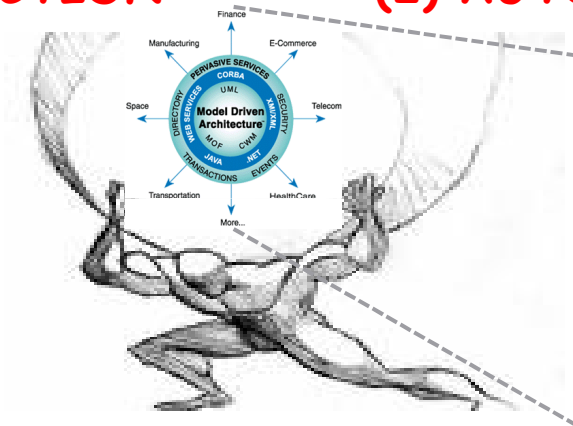
- An approach to system and software development in which software models play an indispensable role
- Based on two time-proven ideas:

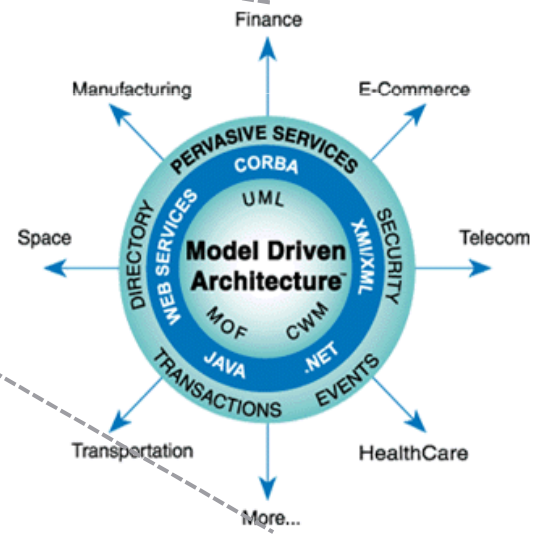


Model-Driven Architecture (MDA)™

- In recognition of the increasing importance of MBE, the Object Management Group (OMG) is developing a set of supporting industrial standards

(1) ABSTRACTION (2) AUTOMATION





(3) INDUSTRY STANDARDS

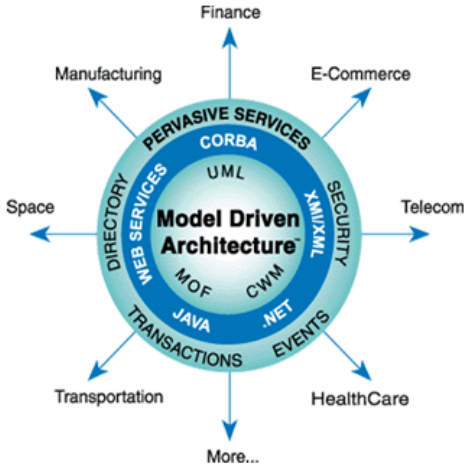
- UML 2
- OCL
- MOF
- SysML
- SPEM
- …etc.

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

The Value of Standards

- **Standards are good because;**
  - They support specialization
    - Standards define interfaces between different specialization domains
    - Specialization is beneficial since it allows complex topics to receive due attention (no need to worry too much about other specialties)
    - E.g., a vendor specializing in analyzing UML models need not worry about providing a UML editing tool
  - Standards imply vendor independence
    - Users have a choice of different vendors (no vendor "tie-in")
    - Forces vendors into competing and improving their products
- **Standards are not good because:**
  - They are almost always compromises; i.e., suboptimal solutions
- **Standards are the key to success to much of modern technological innovation**

MDA™ Standards Architecture



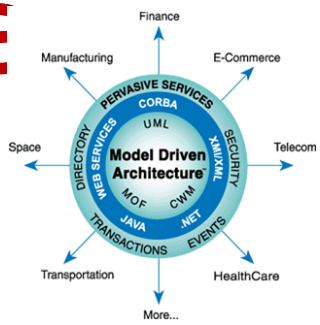
UML Profiles

UML 2 | CWM | SPEM | ODM (Ontology) | RAS

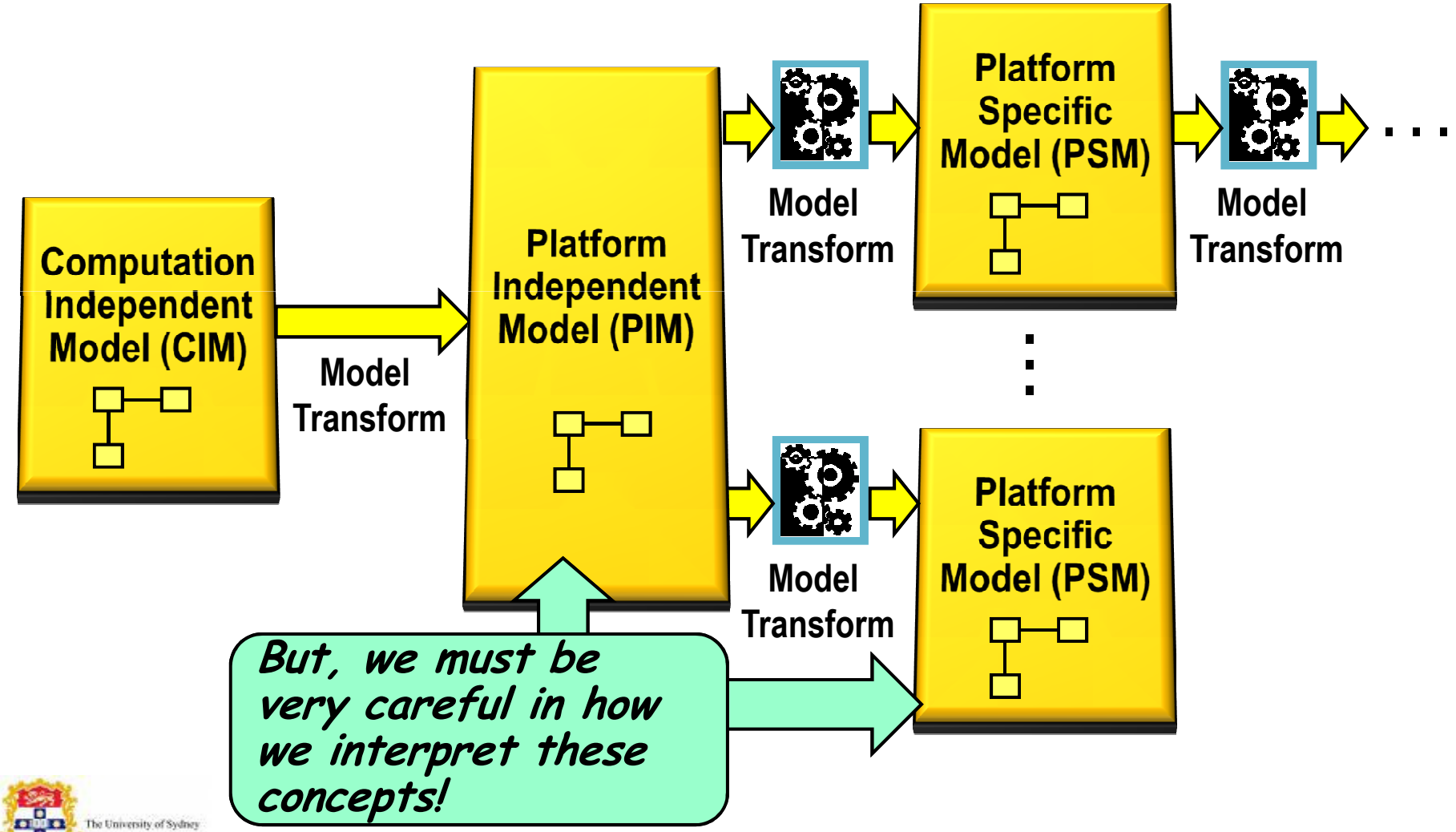
Core Specifications:
MOF 2, XMI 2.1, MOF Versioning, MOF QVT, MOF to Text

UML 2 Infrastructure

The MDA™ Interpretation of MBSE

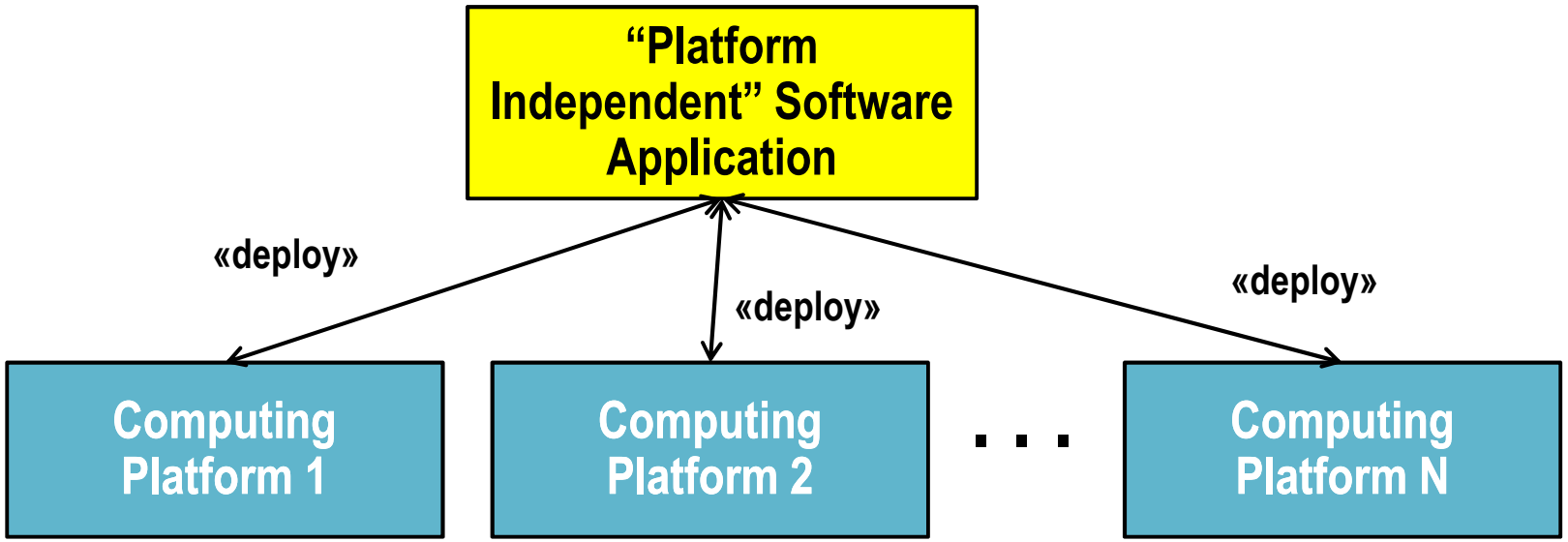


- A cascade of successively refined models leading to one or more implementations



The Concept of "Platform Independence"?

- A highly desirable objective
  - Separation of concerns – reduces apparent problem complexity
  - Enables portability



Does "platform independence" mean that we can ignore platform concerns when designing our application?

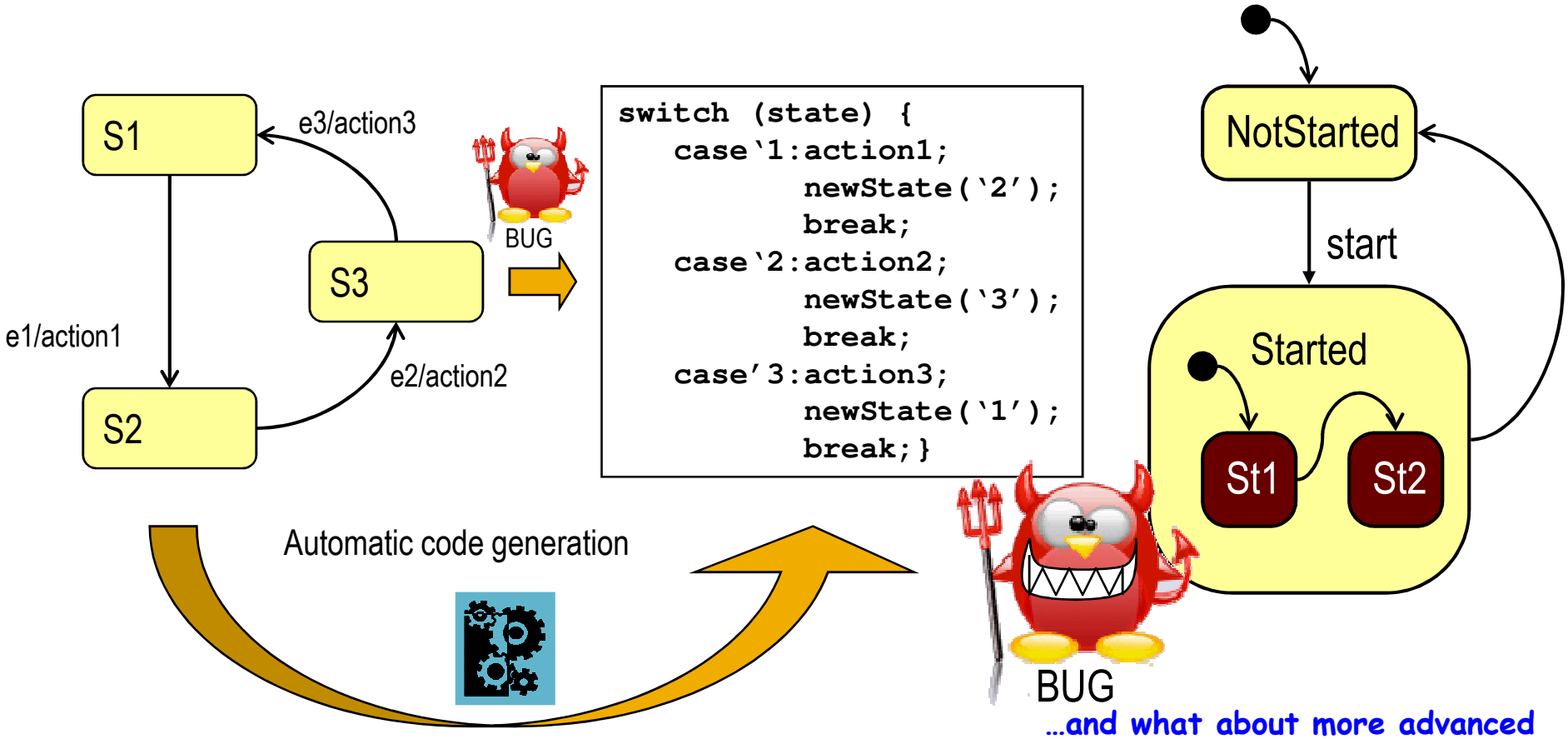
Interpreting the MDA™ View

- *PLATFORM INDEPENDENCE is ...the quality that* the model is independent of the features of a platform of any particular type
  - NB: not independent of the platform as a whole
- A *PLATFORM INDEPENDENT MODEL (PIM)*...exhibits a specified degree of platform independence so as to be suitable for use with a number of different platforms of similar type.

⇒*"platform independence" does NOT imply platform ignorance!*

MBE Opportunity: Code Generation

- The accidental complexity of current programming languages can be greatly reduced by the appropriate use of computer-based automation

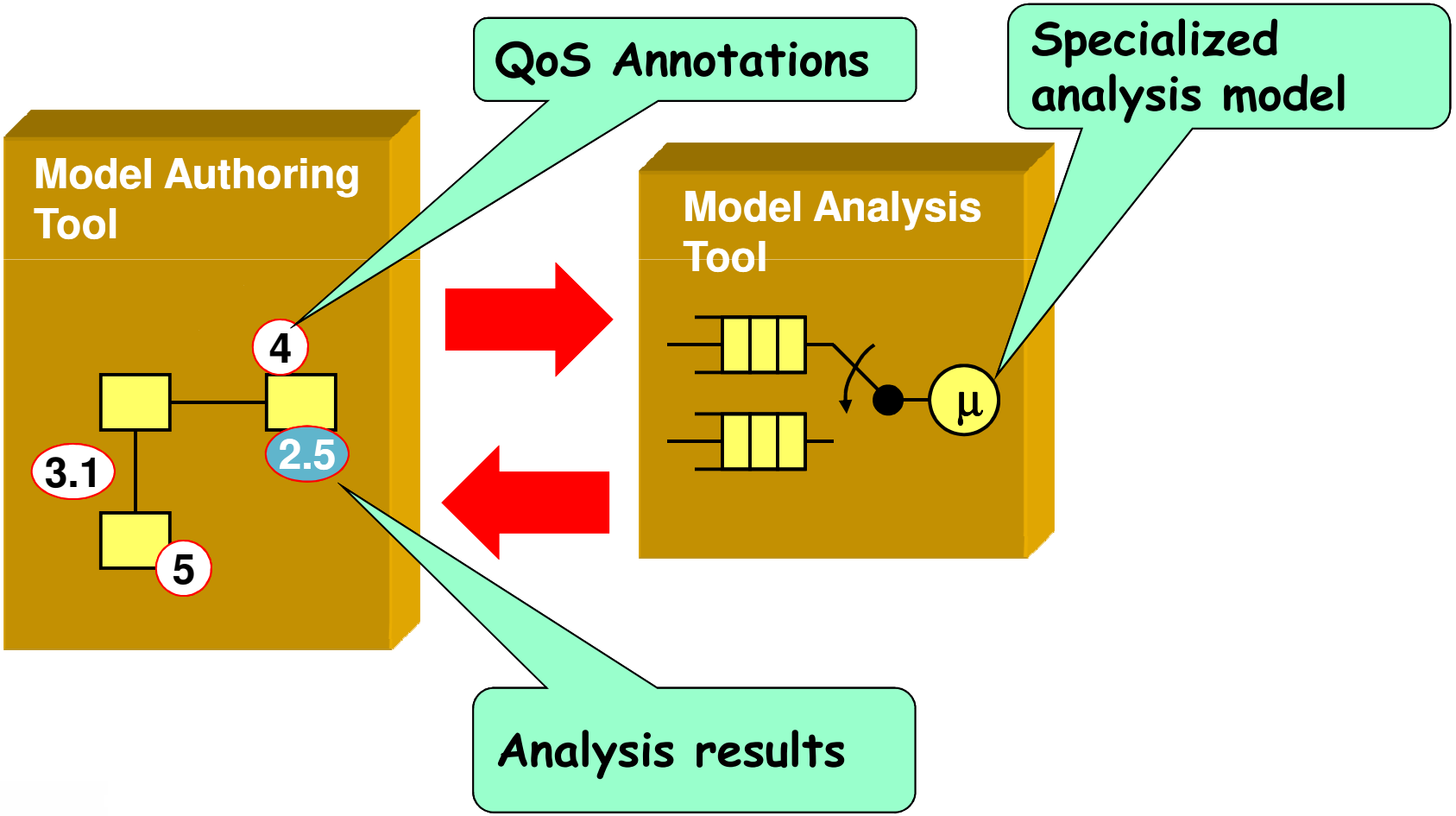


MBE Opportunity: Exploiting Formal Methods

- Given the possibility of making modeling language constructs better behaved than programming language constructs, it is possible to exploit formal methods that could not handle the semantic complexity of programming languages
  - E.g., state machines, Petri nets
  - Model checking, theorem proving
- We need to work on formal semantics of modeling languages

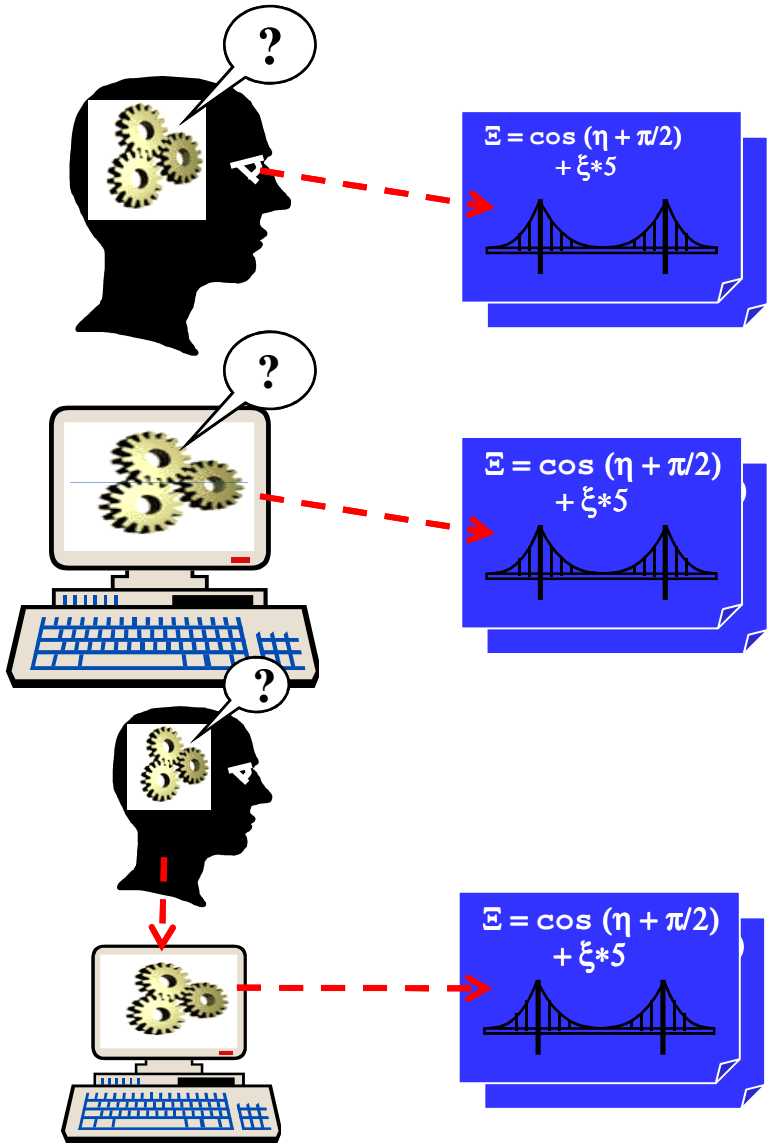
MBE Opportunity: Automated Model Analysis

- Complementary inter-working of specialized tools based on shared standards



Learning From Models

- By inspection
  - mental execution
  - unreliable
- By formal analysis
  - reliable (provided the models are accurate)
- By execution
  - more reliable than inspection
  - direct experience/insight



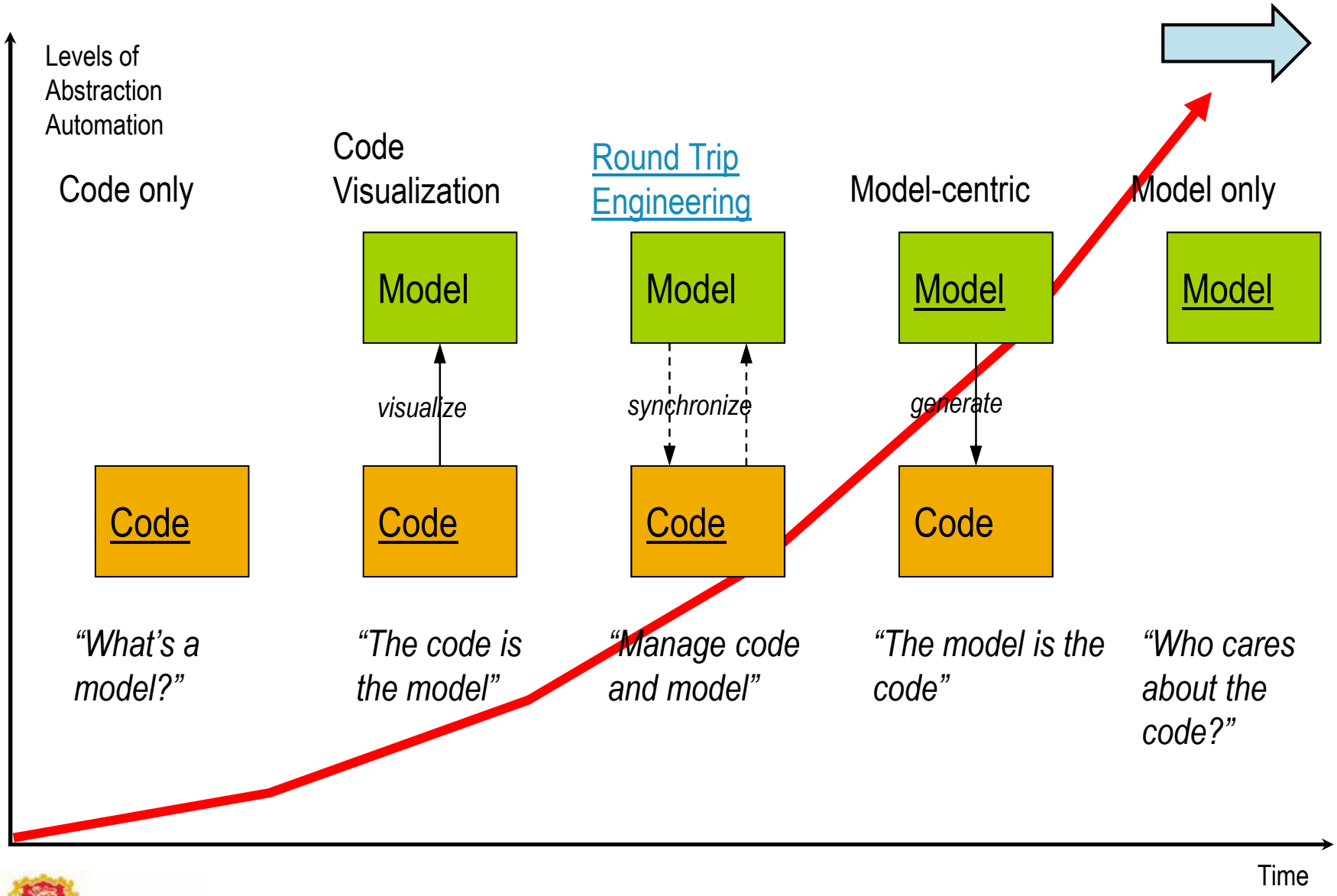
MBE Opportunity: Model Execution

- D. Harel: "Models that are not executable are like cars without engines"
- Ability to execute a model on a computer and observe its behavior
- Key capabilities
  - Controllability: ability to start/stop/slow down/speed up/drive execution
  - Observability: ability to view execution and state in model (source) form
  - Partial model execution: ability to execute abstract and incomplete models
- Executable specifications: overcoming the limitation of paper-based specifications

MBE Opportunity: Model Transformations

- **Multiple purposes:**
  - Model viewing: abstraction and refinement
  - Domain-to-domain model transformations: e.g., UML to queueing network model
  - Model-to-code transformations (code generation)
- **Multiple styles**
  - Declarative
  - Operational
- **Standard: OMG's upcoming MOF Queries, Views, and Transformations (QVT)**
- **We need a comprehensive theory of model transformations comparable to compiler theory**

Styles of MBSE



Roundtrip Engineering

Implementation transformation

Reverse engineering

NB: Slide idea borrowed from an **itemis AG** presentation

Automatic Code Generation

- **A form of model transformation (model to text)**
  - To a lower level of abstraction
- **State of the art:**
  - All development done via the model (i.e., no modifications of generated code)
  - Size: Systems equivalent to ~ 10 MLoC
  - Scalability: teams involving hundreds of developers
  - Performance: within ±5-15% of equivalent manually coded system

Overview

- The Problem
- The Premise
- The Results

Major Telecom Equipment Manufacturer

- **MBE technologies used**
  - UML, Rational Technical Developer, RUP
- **Example 1: Radio Base Station**
  - 2 Million lines of C++ code (87% generated by tools)
  - 150 developers
- **Example 2: Network Controller**
  - 4.5 Million lines of C++ code (80% generated by tools)
  - 200 developers

Benefits

80% fewer bugs

30% productivity increase

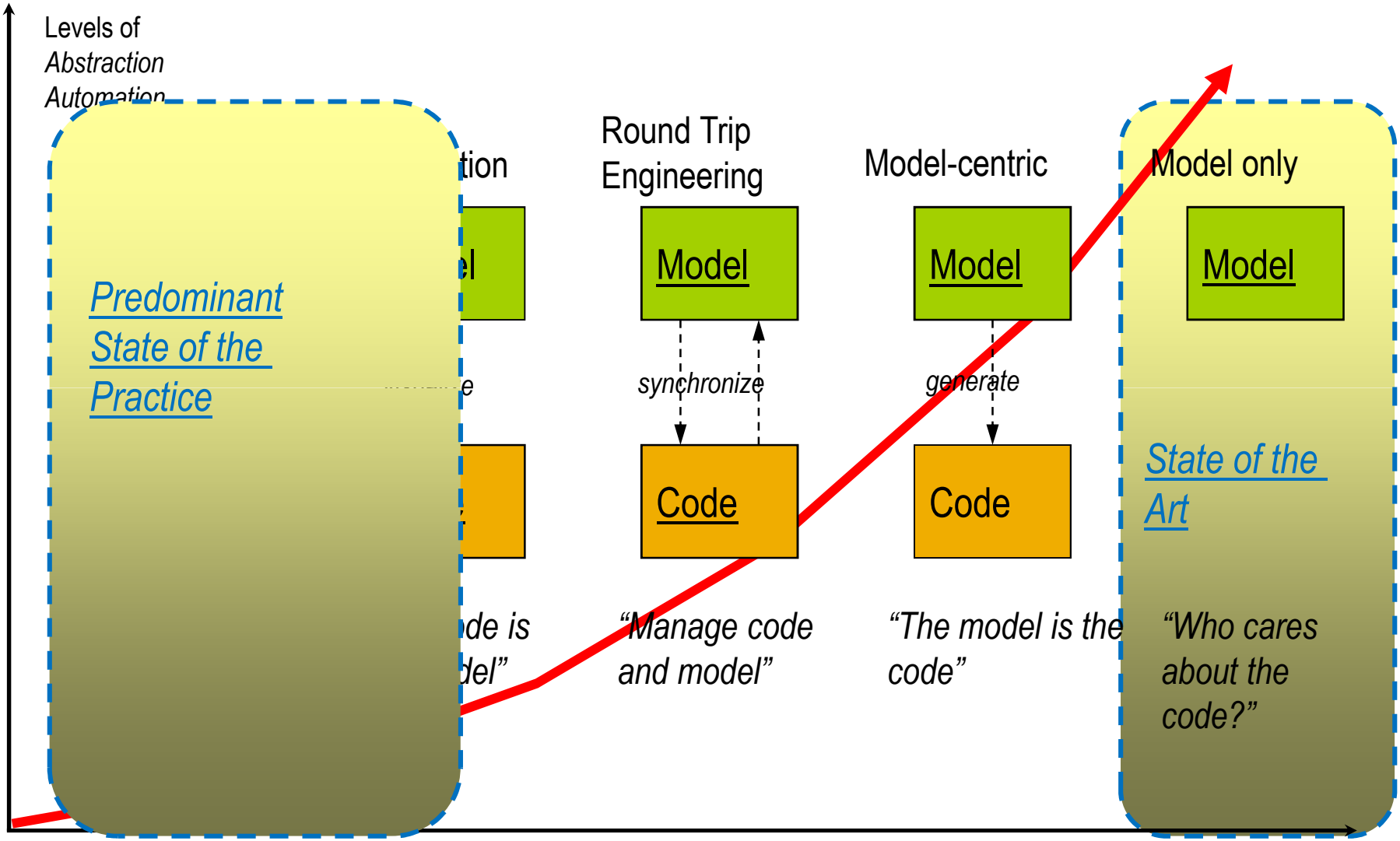
..and a Few Extreme Cases

- **Major Equipment Manufacturer 1:**
  - Code production rate went from 40 LoC/day to 250 Loc/day (>600% improvement)
- **Major Equipment Manufacturer 2:**
  - Code production rate went from 200 LoC/week to 950 Loc/week (~500% improvement)
  - 6-person team developed 120 KLoC system in 21.5 weeks compared to planned 40 weeks (~100% improvement)
  - Fault density (per line of code) reduced 17-fold (1700%)

Sampling of Successful MBE Products

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

MBSE in Practice



If this stuff is so good, why isn't everybody doing it?

Root Causes of Low Adoption Rate

- Categories of impediments
  - Technical problems
  - Social/Cultural issues
  - Economic factors
- Key Point: It is not sufficient to address only the technical issues

Summary: MBSE

- MBSE is an approach to software development based on raising the level of abstraction and level of automation beyond current standard practice
- There have been numerous successful applications of MBSE in industrial practice
- The uptake of MBSE in practice is still slow due to
  - Immaturity of the discipline – weak theoretical underpinnings
  - Required investment in training and tooling
  - Discontinuities involved in switching to new methods and tools
  - Cultural factors

School of Electrical and Information Engineering

Computer Engineering Laboratory

Supplement: The Impediments to Greater MBSE Adoption

Bran Selic

Technical: Immaturity

- **Most MBE technologies are immature**
  - Inadequate understanding of underlying theoretical foundations
  - Most innovation developed ad hoc by commercial enterprises to solve specific and immediate market needs
- **Example: Modeling language design**
  - Insufficient experience and understanding of the problem and characteristics of potential solutions

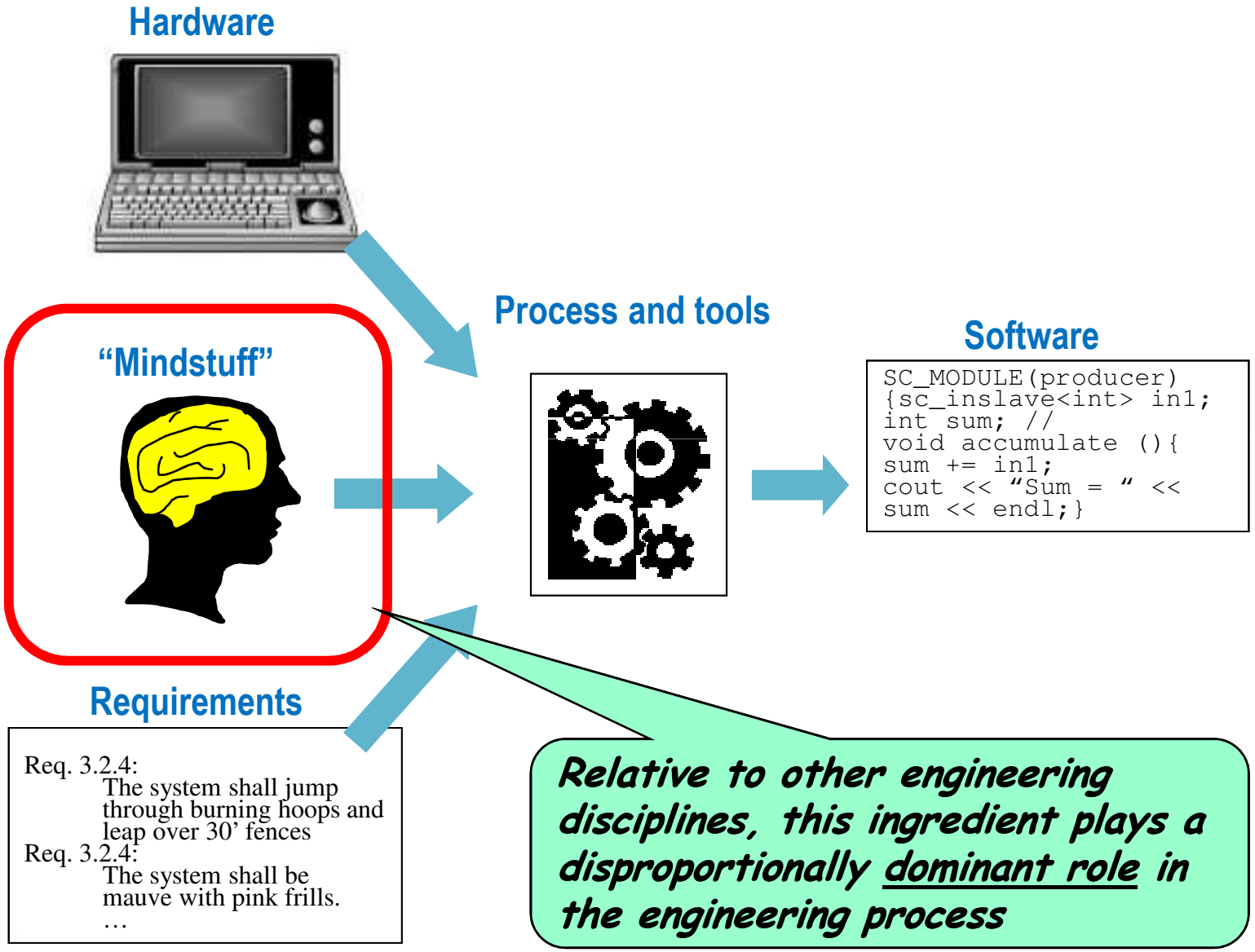
Technical: Usability and Interoperability

- The "naked" complexity of current software technologies is a major deterrent to their use
  - Menu-mania and lack of interoperability
  - Require significant intellectual investment to master
  - Deflects effort and resources from core problem
- Tool design requires deep understanding of
  - The technical aspects of the application domain
  - The habits and psychology of users
  - The nature and flow of the development process
  - The business context in which the tools are used
- Usability must be a fundamental part of the system architecture
  - Difficult if not impossible to retrofit

Cultural: Lack of Awareness and Vision

- **Many practitioners remain unaware of the potential and achievements of MBE**
  - Lack of verifiable evidence
  - Enterprises are often secretive about their successes due to competitive reasons
- **Technological ruts (ratholes?)**
  - Many (most?) fractitioners tend to focus on technologies instead of solutions $\Rightarrow$ strong resistance to change
  - Incremental thinking
    - e.g. OOPSLA '07 panel on future of programming languages
  - "Problems cannot be solved by the same level of thinking that created them" – A. Einstein

The Idiosyncrasies of Software - 1



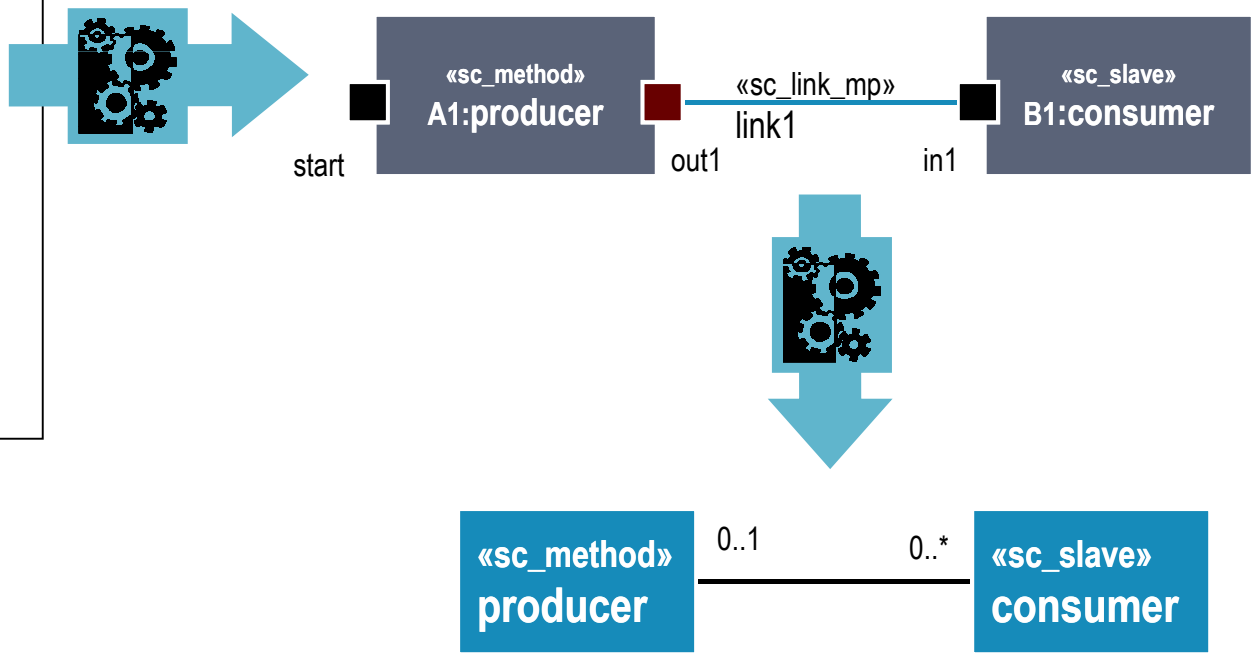
Some Consequences

- Products are much less hampered by physical reality
  - …but, not completely free
- The effects of aptitude differences between individuals are strongly accentuated
  - Productivity of individuals can differ by an order of magnitude
  - Not necessarily a measure of quality
  - …or intelligence
- The path from conception to realization is exceptionally fast (edit-compile-run cycle)
  - Often leads to an impatient state of mind
  - …which leads to unsystematic and hastily conceived solutions (hacking)
  - Also yields a highly seductive and engrossing experience
  - …so that, often, the ultimate product becomes secondary

The Idiosyncrasies of Software – 2

- In all other engineering disciplines abstractions (and models) are artifacts that are necessarily distinct from the systems that they abstract
  - Results in divergence and inaccuracy of abstractions/models
- *Uniquely, in software, the abstraction can be integrated with its system and can be extracted automatically when required*

```
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)
```



So, Is Programming = Mathematics?

Edsgar Wybe Dijkstra (1930 – 2002)

- *"I see no meaningful difference between programming methodology and mathematical methodology"* (EWD 1209)
- *"[The interrupt] was a great invention, but also a Pandora's Box…essentially, for the sake of efficiency, concurrency [became] visible…and then, all hell broke loose"* (EWD 1303)

Two Opposing Views

"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

"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
1st Century BC

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.

- *In many practical systems, the physical platform is a primary design constraint that cannot be overcome by layers of software*

  Computer system = software + hardware

- *Yet, students are still being taught that "platform concerns" are second order issues*

The University of Sydney

More on Software Physics

"Time has been systematically removed from theories of computation, since it has been viewed as representing the annoying property that computations take time."

E. Lee, UC Berkeley

- Failure to distinguish between computational theory and software engineering practice
- Lack of fundamental quantitative skills
  - Basic "back of the envelope" calculation
- Consequences
  - 7-second dialtone delay
  - 8 GB PC application

Social: The Great Inertial Mass Problem

- Numerous generations of software practitioners were raised with this culture
  - ~12-25 million programmers in the world
  - ...most of them holding on to what they know and unwilling to move outside their technological rut comfort zones
- How to overcome this enormous inertial mass?

Economic: Present Day Business Culture

- **Predominantly based on short-term return on investment (ROI)**
  - Markets today force focus on quarterly results
  - Business and technology development plan horizons are rarely meaningful beyond 12 months
  - Reward structure based on short-term results
- **Foundational research and introduction of new technologies requires more distant horizons and long-term investments**
  - Today's model of research funding is strongly tied to short-term market relevance
    - Not conducive to research into fundamentals
    - Hampers ground-breaking outside-the-box innovation

What Can We Do?

What is Engineering?

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

Why "Software Engineering"?

- **Misleading term**
  - The objective is not to develop software but useful systems
  - Software should be just one of the tools used by engineers for solving engineering problems
- **Consequences:**
  - Software engineers often identify themselves not by their domain expertise (e.g., telecom, financial systems, aerospace) but by their technology expertise (e.g., C++, EJB, Linux)

    "When the only tool you have is a hammer, all problems start looking like nails"
  - Technology obsolescence and suboptimal solutions
  - High degree of resistance to technological innovation and change

Getting Closer to the End User

- **There is an unfortunate lack of awareness of and respect for end users**
  - Personal gratification should not come solely from having designed and constructed the system, but from seeing it in use
  - The medium is not the message
- **Implies achieving a deep level of understanding of the value of the system to the customer**
  - Implies a scope of skills and knowledge that extends far beyond the technical domain
  - Required at every level (not just system architects)

"The [engineer] should be equipped with knowledge of many branches of study and varied kinds of learning, for it is by his judgment that all work done by the other arts is put to test. This knowledge is the child of practice and theory."

- Vitruvius

***On Architecture,* Book I (1st Century BC)**

An Unexpected Source of Inspiration

"The glow retreats, done in the day of toil;
It yonder hastes, new fields of life exploring;
Ah, that no wing can lift me from the soil,
Upon its track to follow, follow soaring! ."
-- Goethe, Faust

- **"In an instant, I saw it all."**

  Nikola Tesla describing the moment of insight that led to the invention of the rotating magnetic field and the alternating current electric motor – considered one of the 10 most important modern inventions

The Value of a Broader Education

- **More than just finding inspiration for technical solutions in non-technical sources**
  - Although, higher levels of general literacy are sorely needed (particularly writing skills)
- **Understanding and respect for the greater social, cultural, economic context in which technical inventions function**
  - Understand when and how to apply technological solutions
  - Avoid often futile attempts to solve non-technical issues with yet more technology
  - Reduce current glut of confusing and problematic technologies that cause more problems than they solve

Understanding the Business Case

- There is often a justifiable reason why the "best" technical solution is not the best solution for a given situation
  - E.g., cost of retraining
  - Perhaps the most frequent (and most futile) complaint of software developers worldwide
  - Based on the assumption that technical concerns (e.g., technical elegance) are always paramount
  - Often reflects a lack of awareness of overriding non-technical issues
- Engineers must be trained to understand and appreciate the greater business context

Speaking of Business…

- **Prepare software experts for work in a business-oriented environment**
  - They work for organizations that solve business problems
  - They may become entrepreneurs
- **"Must know" topics**
  - Economics fundamentals: how markets work
  - Basics of business management and administration
  - Basics of accounting and key legal aspects (e.g., IP law)
  - Professional ethics
  - Basics of psychology and sociology
  - Project management/work organization
  - The essentials of marketing
  - Presentation skills

On the Technical Side

- **Abstraction plays a central role in software**
  - More so than any other engineering discipline
- **Mathematics is an excellent foundation for developing and honing abstraction skills**
  - …and may even be directly applicable to the technical problems ☺
  - Mathematical logic
  - Probability theory
  - Discrete mathematics
  - Optimization theory
  - History of technology and mathematics
- **An understanding of the physics underlying software**

Theory and Practice

- *"The difference between theory and practice is much greater in practice than it is in theory"*
- The divide is growing
- Most practitioners disdain theory
  - Unfortunate, since some theory could help them substantially
- Most theoreticians don't understand practice
  - Unfortunate, since they could work on more useful lines of research
- Educational requirements:
  - Instill an appreciation for the value of theory
  - Instill an understanding of the pragmatics of industrial software development

Teaching the Pragmatics

- **Educational examples tend to be naïve and small**
  - Little or no team programming
  - "Greenfields" (vs maintenance) type of development
  - Small scale gives an incorrect basic impression about the nature of software development
- **Proposal: develop a multi-year "product" project in SE courses**
  - Requires work in teams (learning the dynamics of teams)
  - Requires understanding of others' designs (and an appreciation of the value of documentation)

Conclusions

- **Software development *is* engineering**
  - …which does not preclude the notion of software science
- **Software offers a truly unique engineering medium**
  - Dominated by ideas rather than physical reality

    …but not completely
  - The ability to define and realize our own realities
- **This requires a unique combination of new and old engineering principles**
  - We have yet to discover the right balance
- **An understanding of users and their circumstances is crucial**
  - Developing an engineer's sense of responsibility and perspective
  - Inevitably, this requires a broader outlook and interests that extend beyond specific technologies

"Concern for man himself and his fate must always constitute the chief objective of all technological endeavors...in order that the creations or our minds shall be a blessing and not a curse to mankind. Never forget this in the midst of your diagrams and equations."

-- A. Einstein, 1931