IBM Software Group – Rational Software

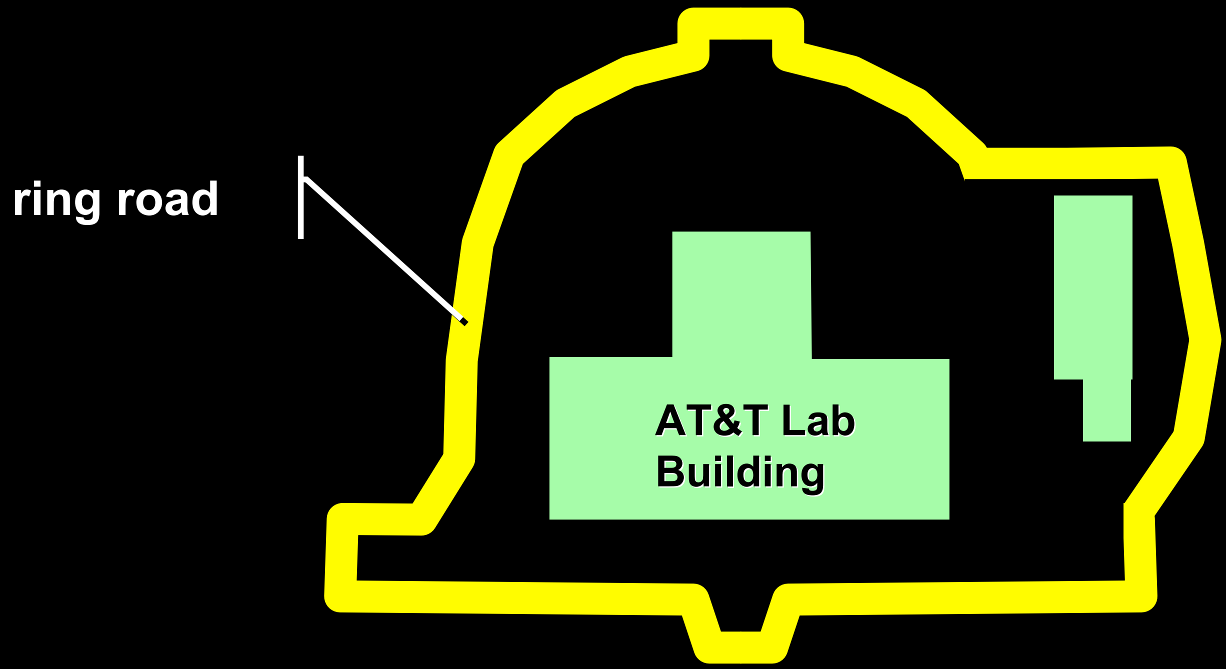
Can the Means Justify the End?

Saving Programs from ~~Programmers~~ Programming

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

A Parable…

In Lisle, Illinois:



- Quiz question: How is this similar to modern-day software?

Another Story…

- The following type of code fragment was included in the program for traffic routing in long distance telephone networks
  -
```
        …
switch (caseIndex) {
case 'A':       route = routeA;
                …
                break;
                …
case 'M':       route = routeM;
                …
case 'N':       route = routeN;
                …
                break;
           …}
```

Missing "break" statement!

- When this code ran, the entire Northeast US lost its long-distance phone service (banks, government institutions, hospitals, businesses…)
- The estimated damage was in the *hundreds of millions of dollars*

…and a More Recent Story

"New FBI Software May Be Unusable"

Los Angeles Times (01/13/05);

A central pillar of the FBI's computer system overhaul, **which has already cost nearly half a billion dollars and missed its original deadline, may be unusable**, according to reports from bureau officials. The prototype … software developed … **at a cost of about $170 million** has been characterized by officials as **unsatisfactory and already out of date**; sources indicate that scrapping the software would entail a roughly **$100 million write-off** while Sen. Judd Gregg … says the software's failure would constitute a tremendous setback. … The computer system overhaul, which has **cost $581 million thus far**, was tagged as a priority by members of Congress ...

Q: Why is Writing Correct Software so Difficult?

A: **COMPLEXITY!**

Modern software is reaching levels of complexity encountered in biological systems; sometimes comprising systems of systems each of which may include millions of lines of code

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., designing a workable network routing system
- *Accidental complexity*
  - introduced by a technology (tools) or technique
  - e.g., building construction without using power tools
- *Modern software development suffers from an excess of accidental complexity*

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 spot the architecture?

...and its Model

Can you see it now?

Breaking the Architecture….

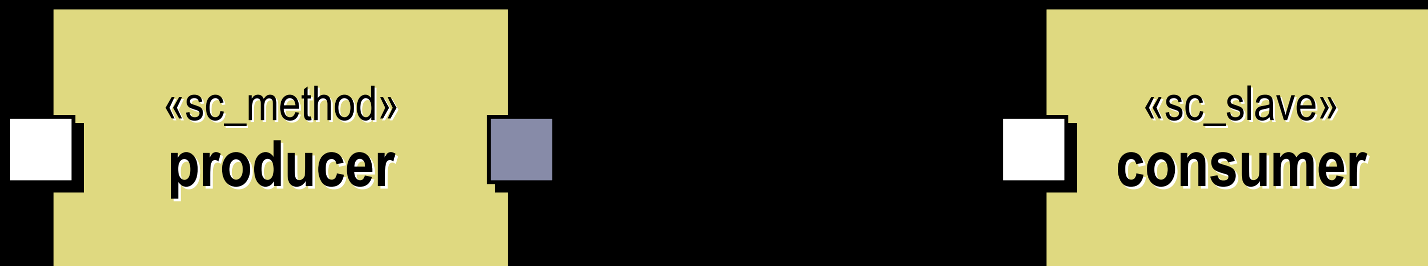
```
SC_MODULE(producer)
{
sc_outmaster<int> out1;
sc_in<bool> start; // kick-start
void generate_data ()
{
for(int i =0; i <10; i++) {
out1 =i ; //to invoke slave;}
}
SC_CTOR(producer)
{
SC_METHOD(generate_data);
sensitive << start;}};
SC_MODULE(consumer)
{
sc_inslave<int> in1;
int sum; // state variable
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 where?

Breaking the Architecture….

⇒ Clearly, models can be useful in software development

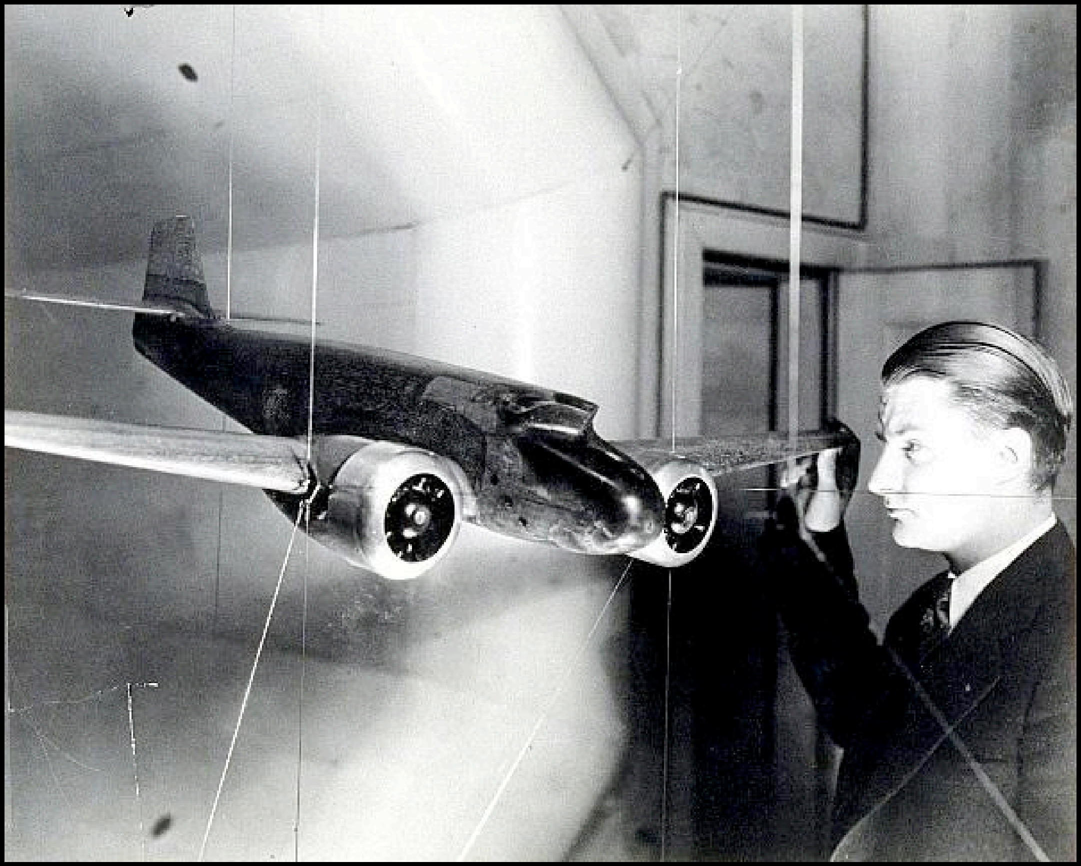
How useful can they be?



Can you see it now?

Use of Models in Engineering

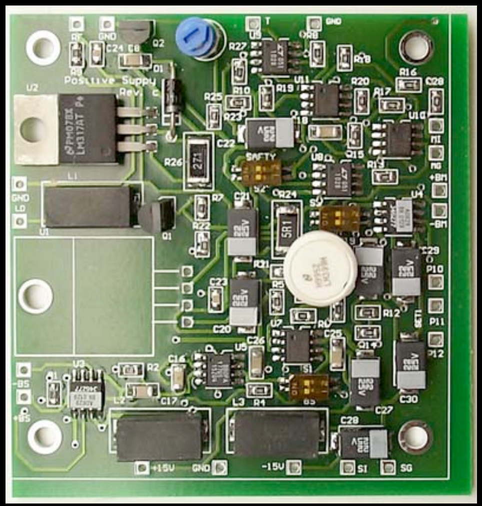
- Probably as old as engineering (c.f., Vitruvius)



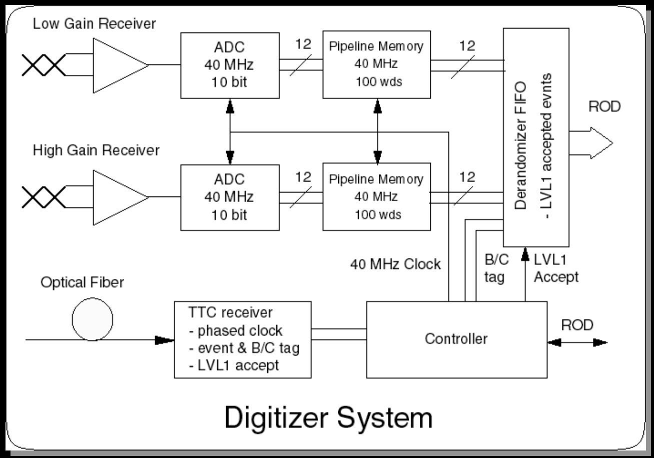
Engineering Models

- Engineering model:

  A reduced representation of some system that highlights the properties of interest from a given viewpoint



Modeled system



Functional Model

- We don't see everything at once
- We use a representation (notation) that is easily understood for the purpose on hand

How Models are Used in Engineering

- To help us understand complex systems
  - Useful for both requirements and designs
  - Minimize risk by detecting errors and omissions early in the design cycle (at low cost)
    - Through analysis and experimentation
    - Investigate and compare alternative solutions
  - To communicate understanding
    - Stakeholders: Clients, users, implementers, testers, documenters, etc.
- To drive implementation
  - The model as a blueprint for construction

Characteristics of *Useful* Engineering Models

- 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
- Inexpensive
  - Much cheaper to construct and study than the modeled system

Useful engineering models must satisfy all of these characteristics!

Back to Our Software 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 << endl;}
```

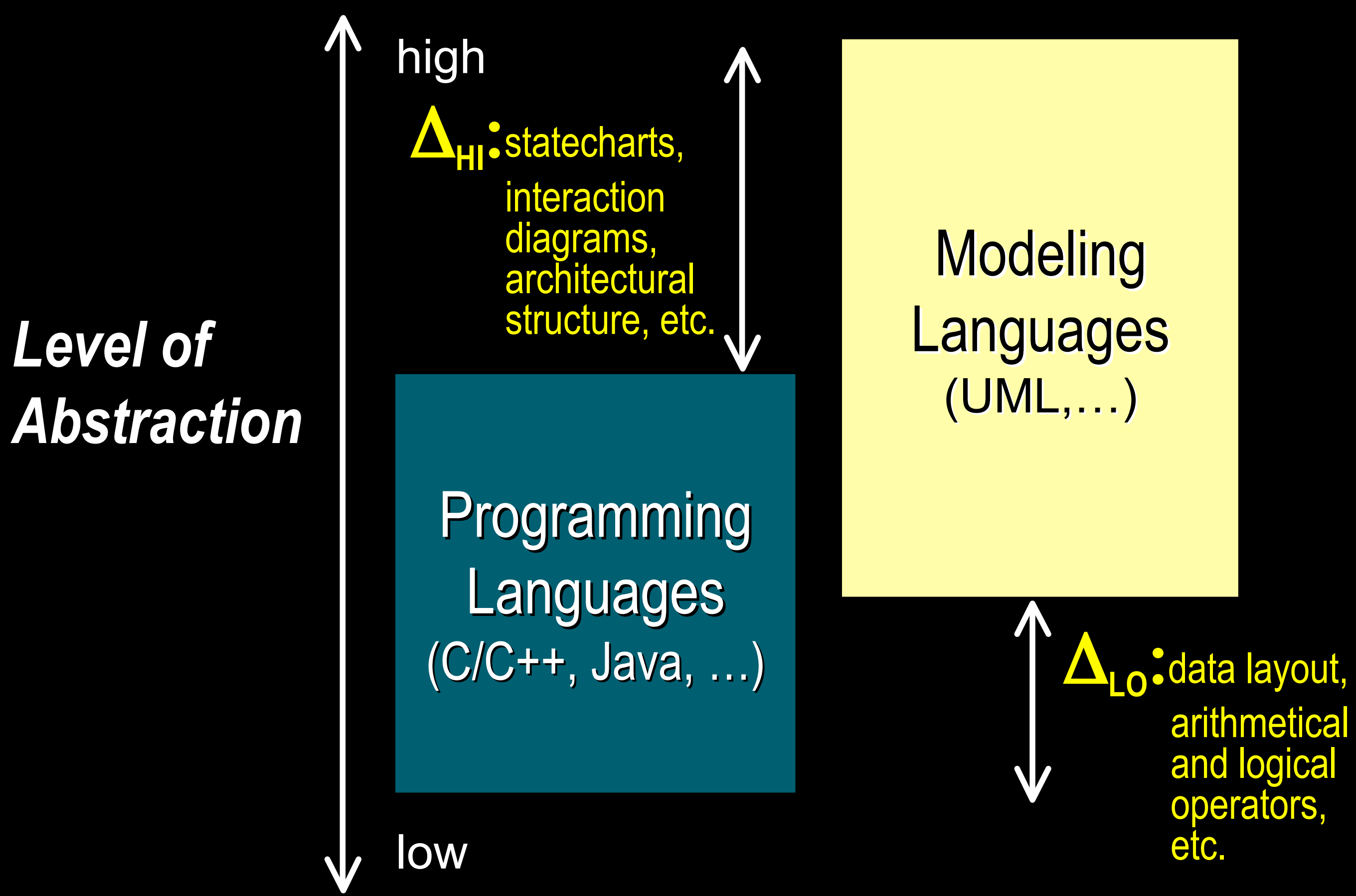
```
SC_CTOR(consumer)
{
SC_SLAVE(accumulate, in1);
sum = 0; // initialize
};
SC_MODULE(top) // container
{
producer *A1;
consumer *B1;
sc_link_mp<int> link1;
SC_CTOR(top)
{
A1 = new producer("A1");
A1.out1(link1);
B1 = new consumer("B1");
B1.in1(link1);}};
```

The Model-Driven Development Approach

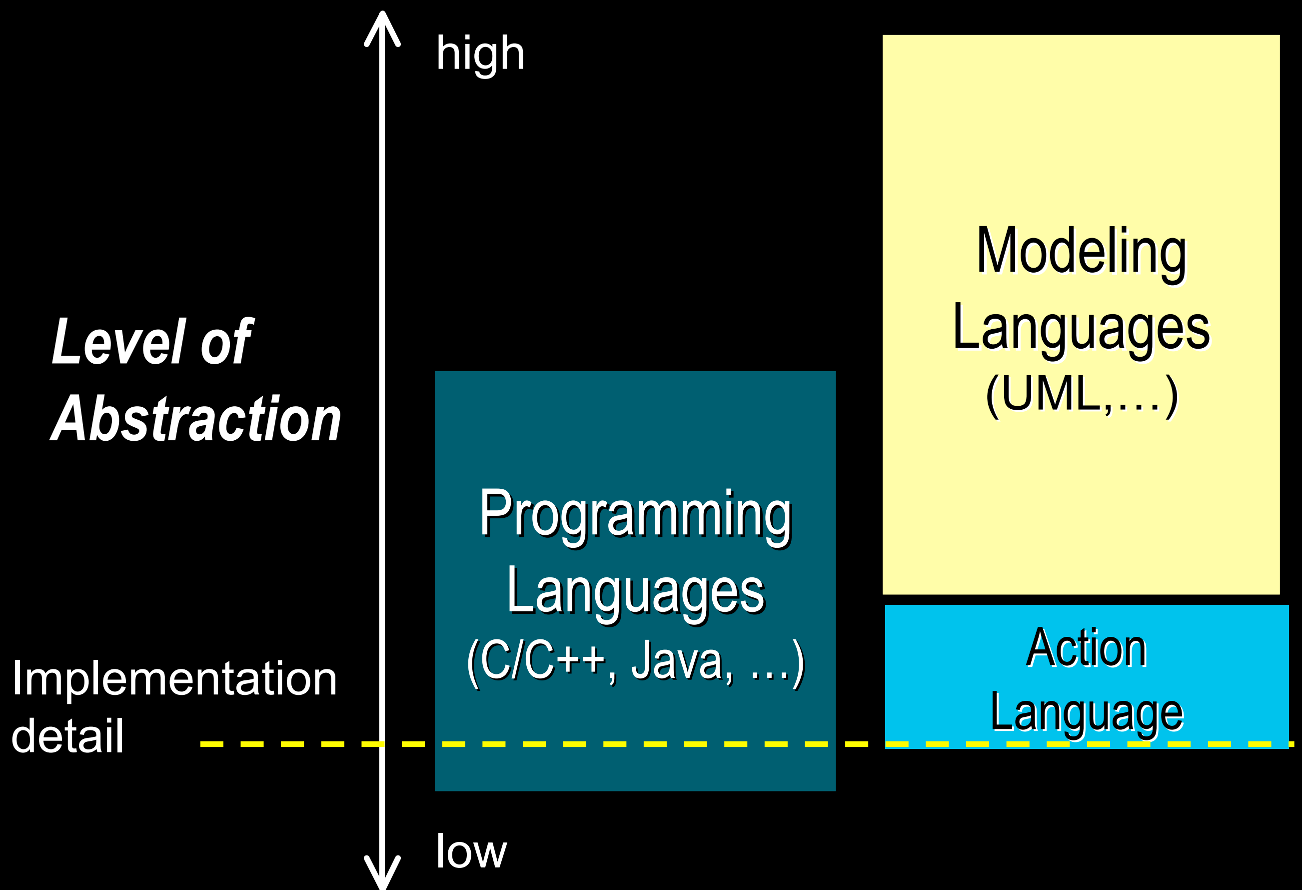
```
SC_MODULE(producer)
{
sc_outmaster<int> out1;
sc_in<bool> start; // kick-start
void generate_data ()
{
for(int i =0; i <10; i++) {
out1 =i ; //to invoke slave;}
}
SC_CTOR(producer)
{
SC_METHOD(generate_data);
sensitive << start;}};
SC_MODULE(consumer)
{
sc_inslave<int> in1;
int sum; // state variable
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);}};
```

Modeling vs Programming Languages

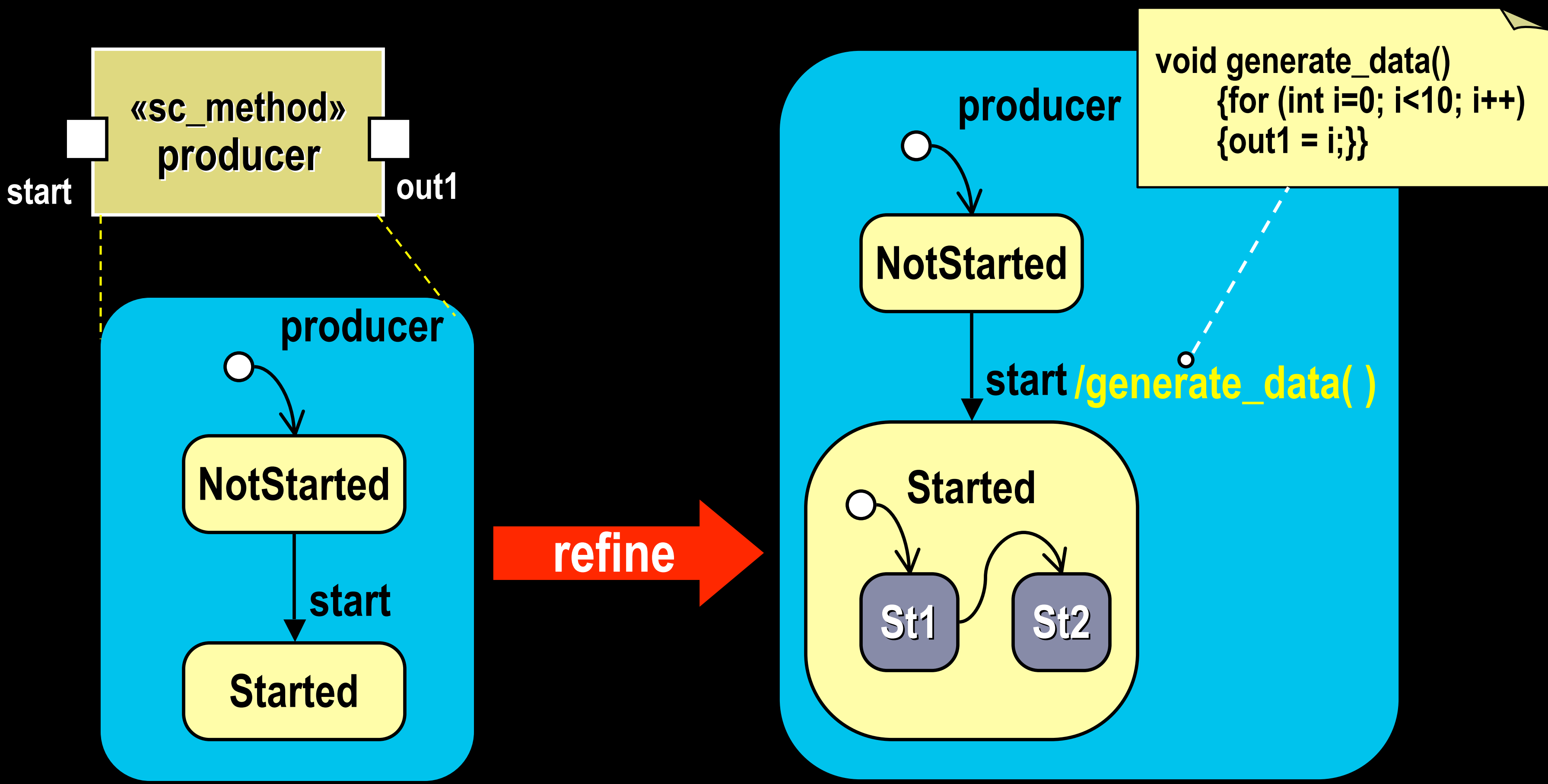
- Cover different ranges of abstraction



Models: Filling in the Detail



Model Evolution: Refinement



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

The Remarkable Aspects of Software

IBM

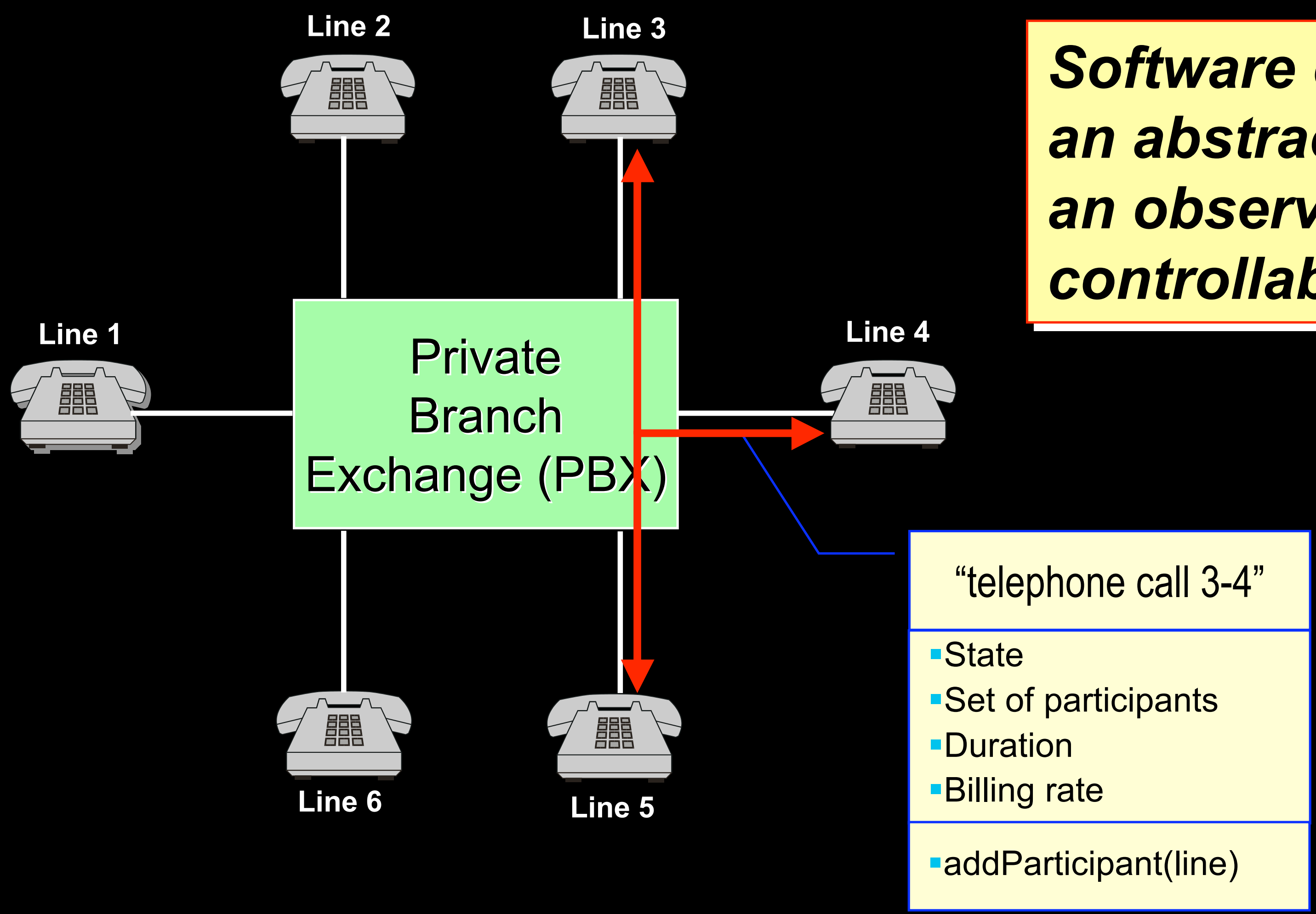
- *Software has the unique property that it allows us to evolve abstract models into full-fledged implementations without changing the engineering medium, tools, or methods!*
- *It also allows us to generate abstract views directly and automatically from the implementations*

⇒ This ensures perfect accuracy of software models; since the model and the system that it models are the same thing

Software: Beyond Mere Physical Abstraction

IBM

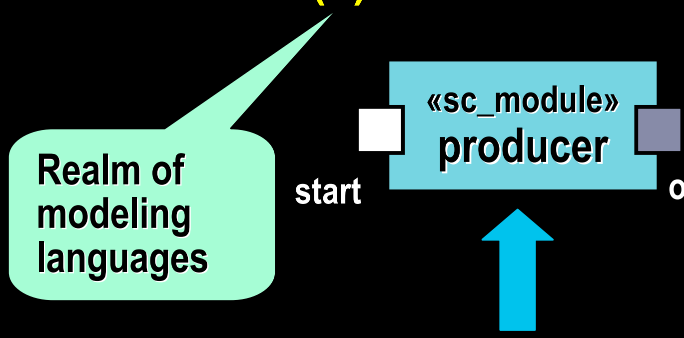
Software can make an abstraction into an observable and controllable reality!



Model-Driven Style of Development (MDD)

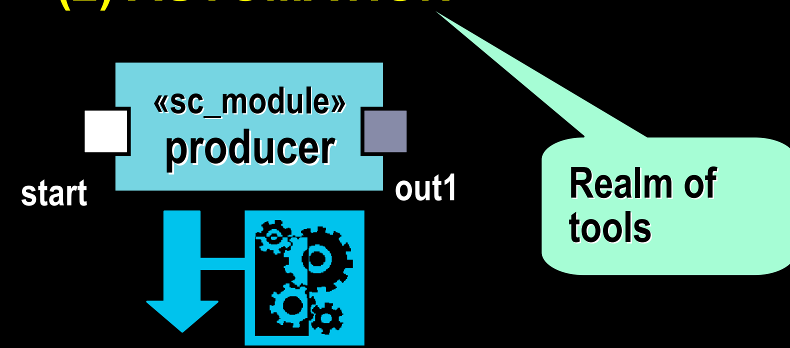
- An approach to software development in which the focus and primary artifacts of development are models (as opposed to programs)
- Based on two time-proven methods

(1) ABSTRACTION




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

(2) AUTOMATION

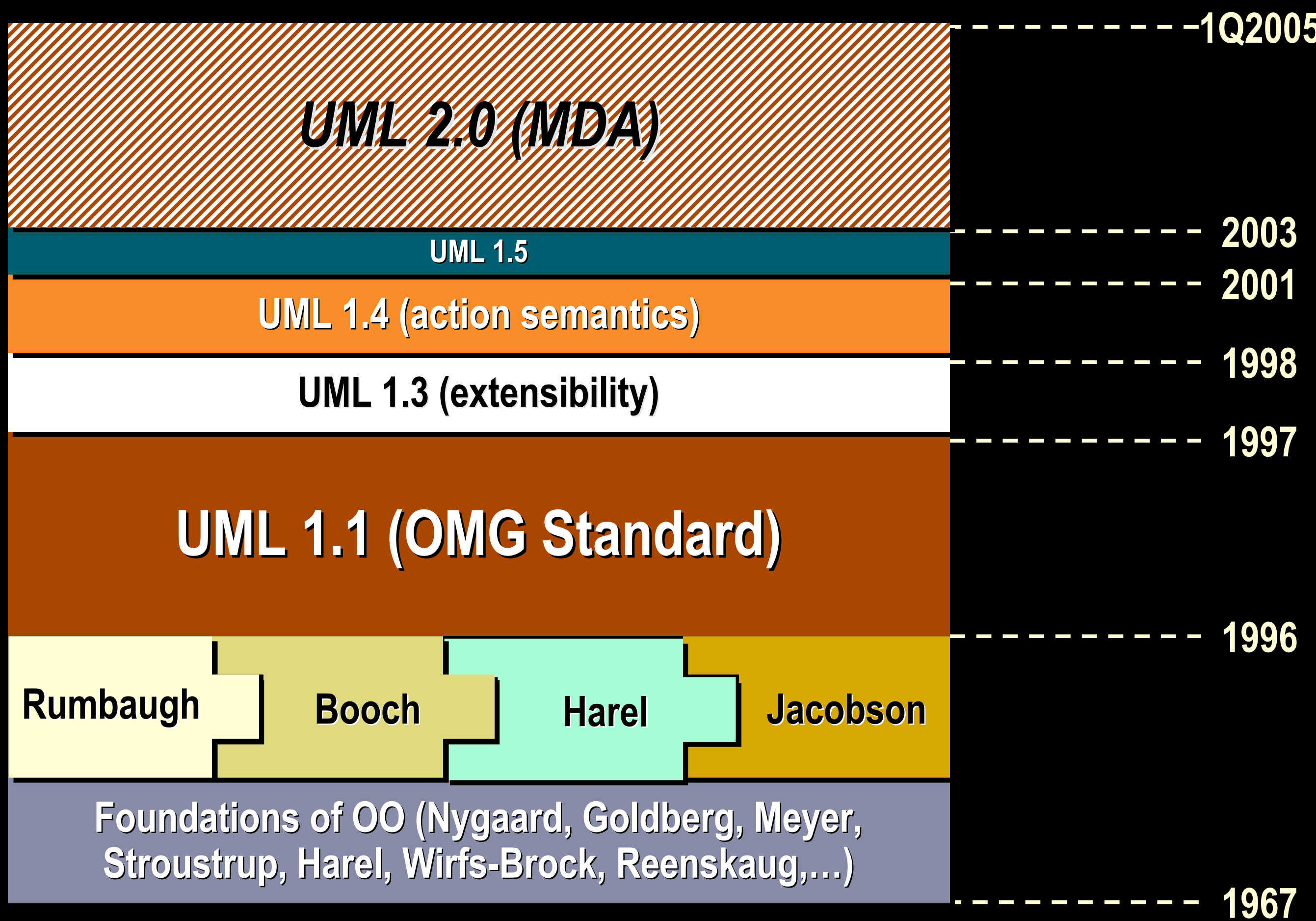



```
SC_MODULE(producer)
{sc_inslave<int> in1;
int sum; //
void accumulate (){
sum += in1;
cout << "Sum = " << sum
<< endl;}
```

Types of Automation with MDD

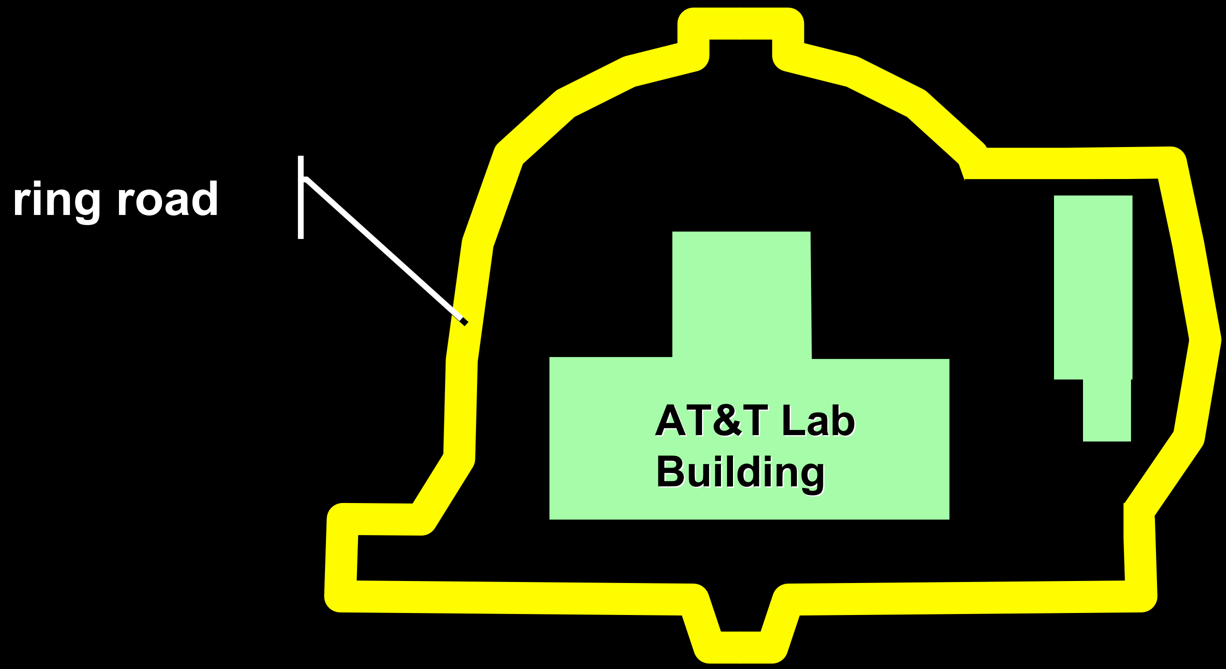
- Computer-based model transformations
  - Code generation, pattern application, abstraction,…
- Computer-based validation
  - Formal methods (qualitative and quantitative)
- Computer-based testing
  - Automated test generation, setup, and execution
- Computer-based model execution (simulation)
  - Particularly execution of abstract and incomplete models
    -- when most of the important decisions are made
- Computer-supported reuse
  - Using computers to store, find, and retrieve re-usable components

UML 2.0: an MDD Language



Back to the Beginning…

Lisle, Illinois:



- The new design team was unaware of the high-level view

Architectural Decay in Software

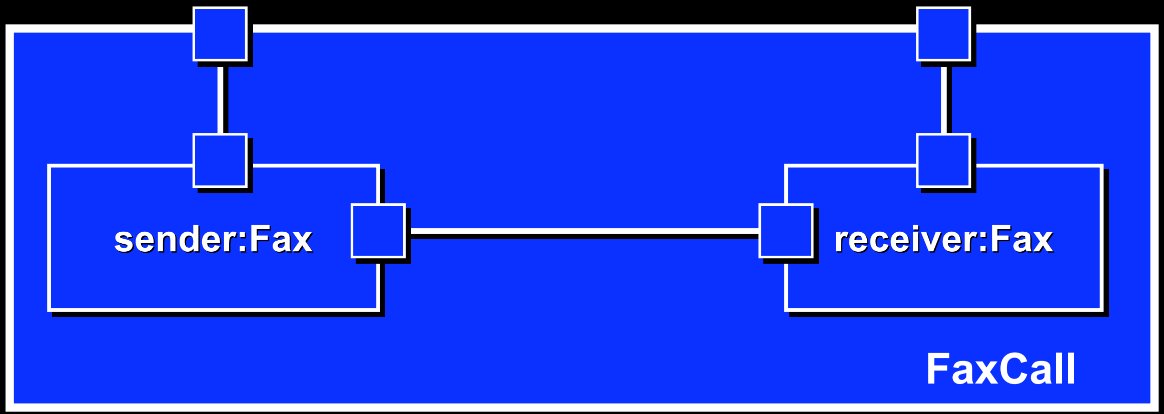
IBM

- The gradual divergence of a program from its intended architecture caused by successions of seemingly minor code modifications
- Ultimate causes
  - Inability to identify architectural intent
  - Inability to enforce architectural intent
- Typically occurs during low-level maintenance work

UML 2.0 Architectural Specification

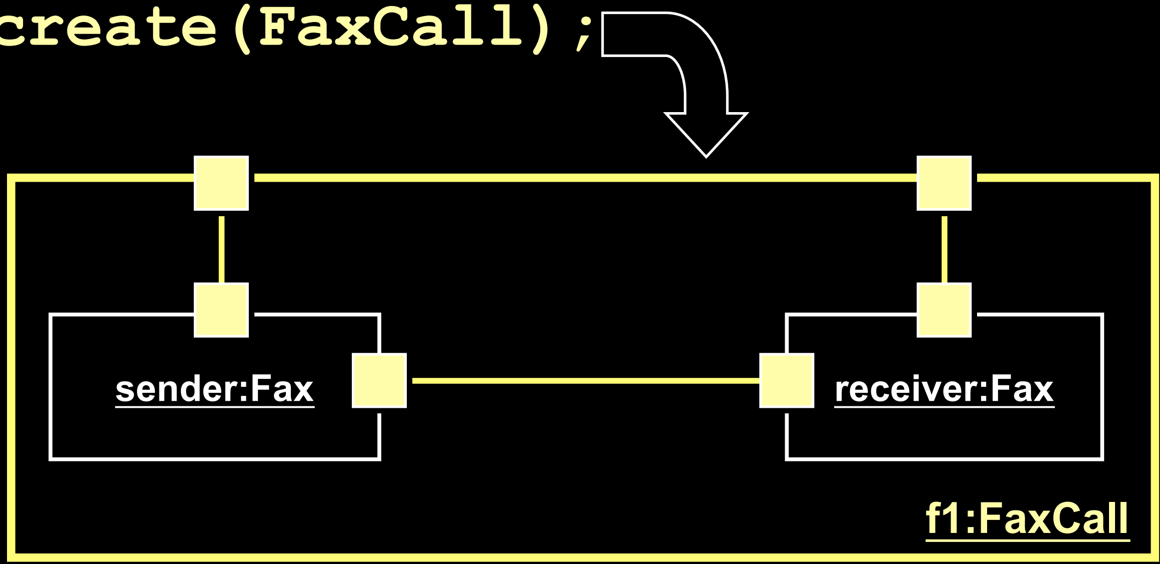
IBM

The Design:



The Implementation (with automatic code generation):

```
f1 := create(FaxCall);
```



Full Automatic Code Gen: State of the Art

- Complete code generation available in specific domains
- Efficiency
  - performance and memory utilization:
    within ±5-15% of equivalent manually coded system
- Scalability
  - compilation time (system and incremental change):
    within 5-20% of manual process
    eliminates need to manually change generated code
  - system size:
    - Complete systems in the order of 4MLOC have been constructed using full code generation
    - Teams of over 400 developers working on a common model

MDD in Practice

- The following large-scale industrial products were all developed using <u>complete automatic code generation</u>:

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, Operations And Maintenance, Optical Switching, Industrial Robot, Phone, Private Branch Exchange (PBX), 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

The Application of MDD

- If MDD can help us construct more reliable software faster, why isn't everyone doing it?
- The most obstinate resistance to MDD comes from software practitioners – one of its main intended beneficiaries
- Reasons:
  - Immature or missing tools
  - Inadequate results (not fast enough, too big,…)
  - Lack of control over the implementation
  - Paradigm shift
  - Culture: is the medium the message?

The Human Side of Software

IBM

- The ultimate objective of any technology is to be useful to humans
- Yet, technologists often expect humans to adapt to technologies
  - E.g., Bhopal tragedy (1984) – training vs design
  - E.g., the $1B missing “break” statement incident
- The unparalleled flexibility and adaptability of software makes it an ideal medium for constructing much more human-friendly technologies
- …starting with the technology used to construct software itself

Conclusions

- We cannot keep trying to develop 21st century software using technological frameworks devised for solving 1950s' problems
- New technologies, such as MDD, based on time-proven trusted methods (abstraction, automation), provide a clear way forward
- But, their success depends on an awareness of and a dedication to the human users for whom all software is ultimately constructed
  - The medium is not the message, the means are not the end
- The Fortran box has been finally breached and it is our responsibility to reach outside