What I Wish They Taught in Engineering School: Reflections on 30+ Years as a Software Developer

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

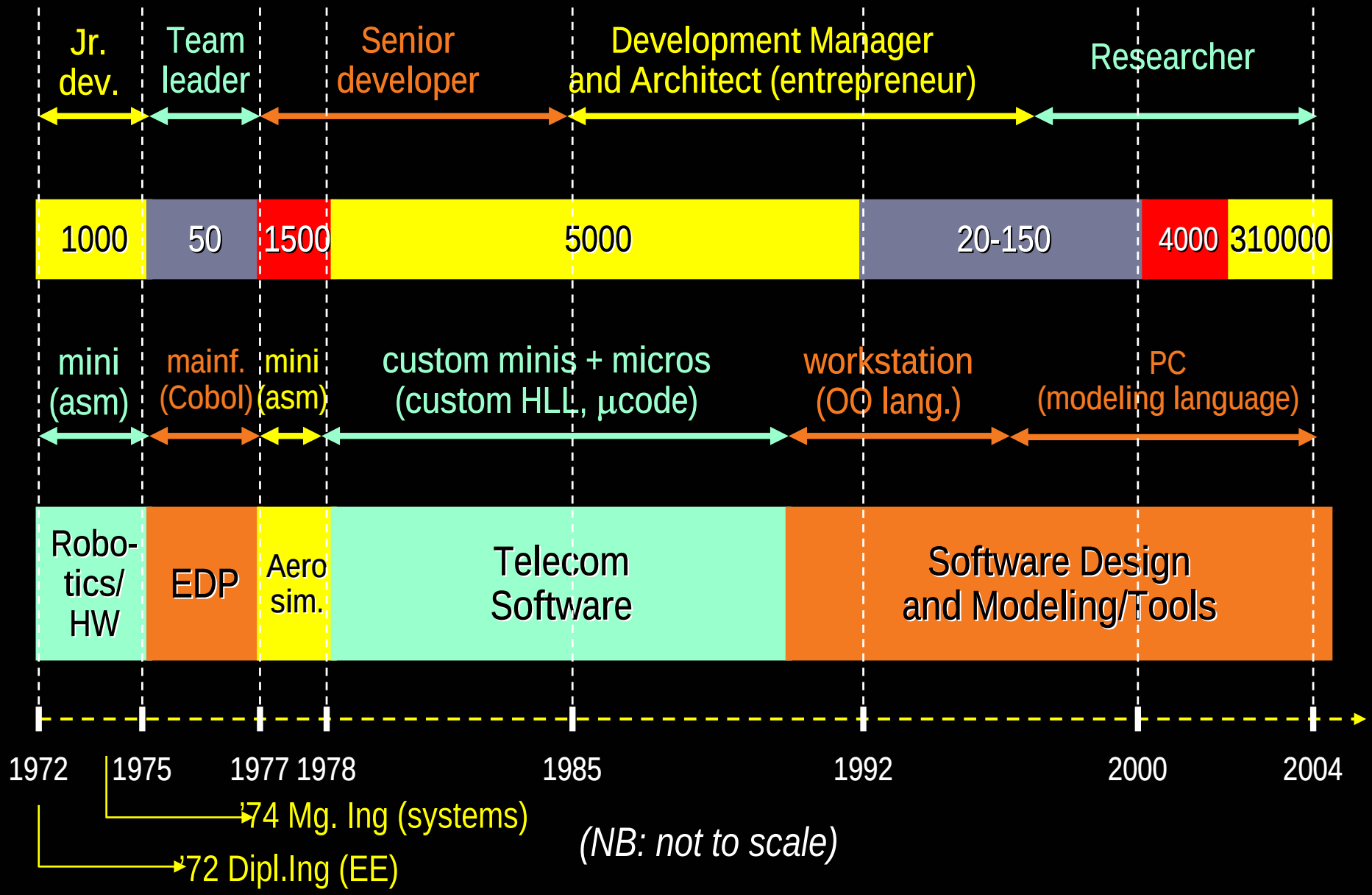
A Symbolic (and True) Anecdote

Speaker speaking to an audience of experienced software developers:

Speaker: *"Software developers don't like to do documentation, because it provides no value to them…"*

Audience: *<nods knowingly in approval>*

My Background (Colorful)



What has Changed from a Developer's Perspective…

- Changes with the greatest impact:
  - Hardware: speed, reliability, cost, capacity
  - Connectivity between computers
  - Team programming environments
  - Human-computer interfaces
  - Sophistication of development tools
  - Complexity of applications
- Most significant non-change:
  - Level of abstraction of programming

A Discipline in Trouble

"New FBI Software May Be Unusable"

Los Angeles Times (01/13/05);

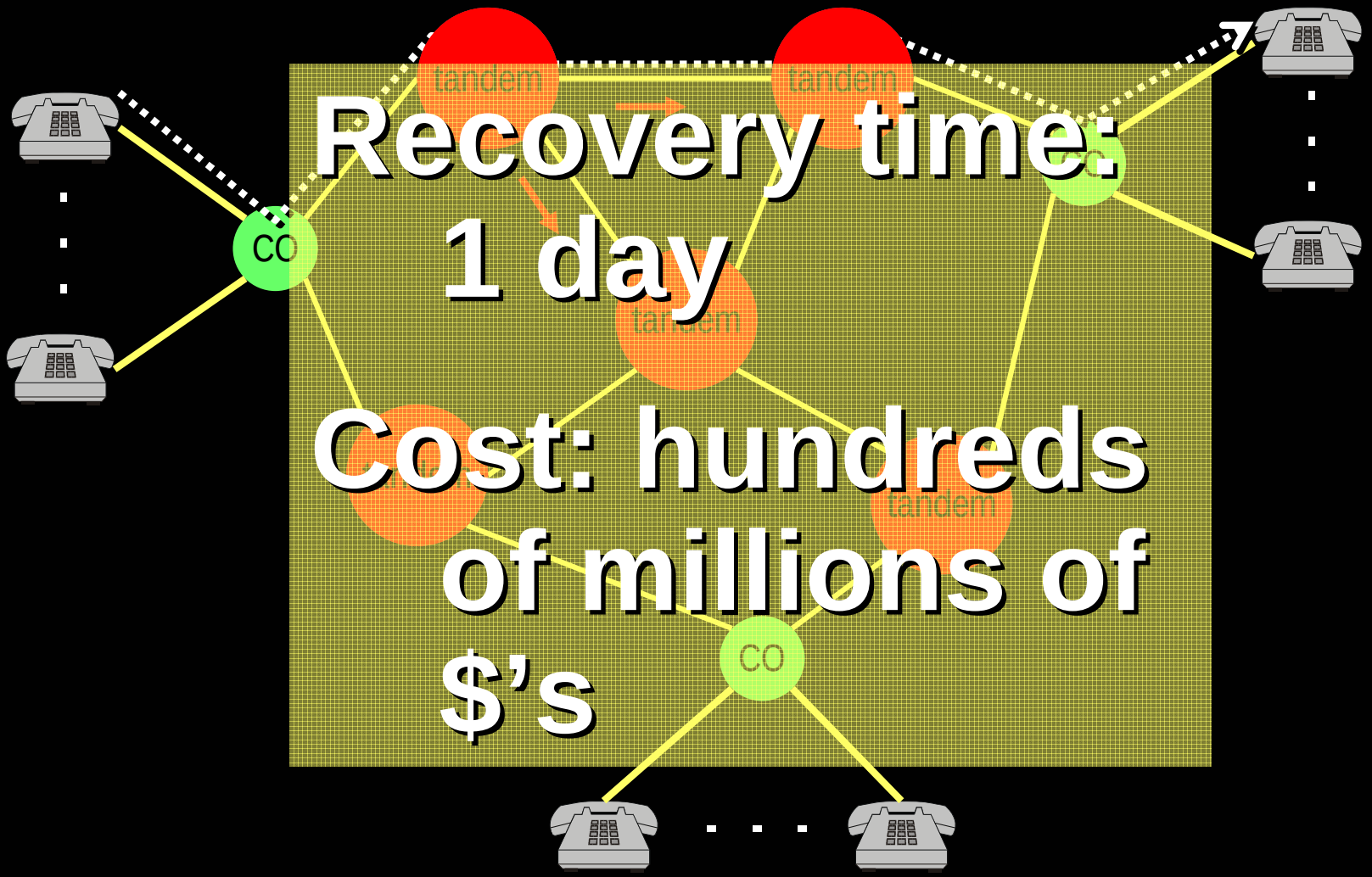
A central pillar of the FBI's computer system overhaul, which has already cost nearly half a billion dollars and missed its original deadline, may be unusable, according to reports from bureau officials. The prototype … software developed … at a cost of about $170 million has been characterized by officials as unsatisfactory and already out of date; sources indicate that scrapping the software would entail a roughly $100 million write-off while Sen.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

On the Peculiar Nature of Software, Software Technologies, and the Psychology of Programming*

*NB: from the unscientific perspective of an amateur psychologist

The Anatomy of an Engineering Disaster

- 1987: AT&T Long Distance Network (Northeastern US)



The Root Cause

- Missing “break” statement in a software module
  - one (missing) line among millions

Recovery time: 1 day

Cost: hundreds of millions of $’s

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

Execution “fell through” unintentionally into the next case

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 tens of millions of lines of code

…any one of which may bring down the entire system at great expense

Fred Brooks on Complexity

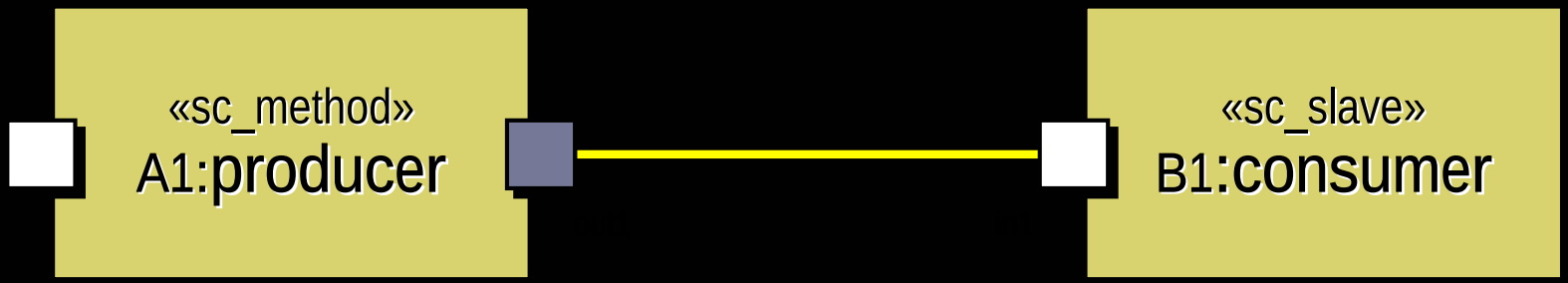
- [From: F. Brooks, "*The Mythical Man-Month*", Addison Wesley, 1995]
- *Essential complexity*
  - inherent to the problem
  - cannot be eliminated by technology or technique
  - e.g., solving the traveling salesman problem
- *Accidental complexity*
  - due to technology or methods used to solve the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

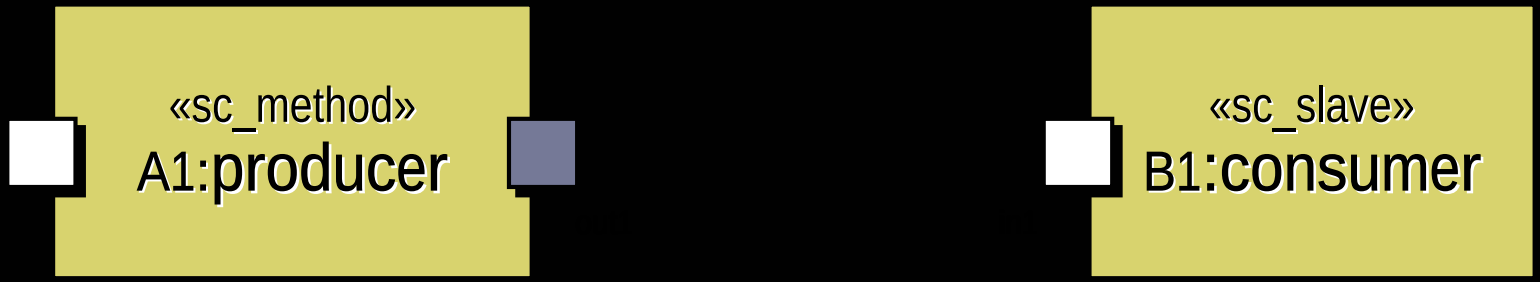
```
SC_CTOR(consumer)
{
SC_SLAVE(accumulate, in1);
sum = 0; // initialize
};
SC_MODULE(top) // container
{
producer *A1;
consumer *B1;
sc_link_mp<int> link1;
SC_CTOR(top)
{
A1 = new producer("A1");
//A1.out1(link1);
B1 = new consumer("B1");
//B1.in1(link1);}};
```

Can you see where?

Breaking the Architecture….

⇒ Clearly, models can be useful in software development

How useful can they be?

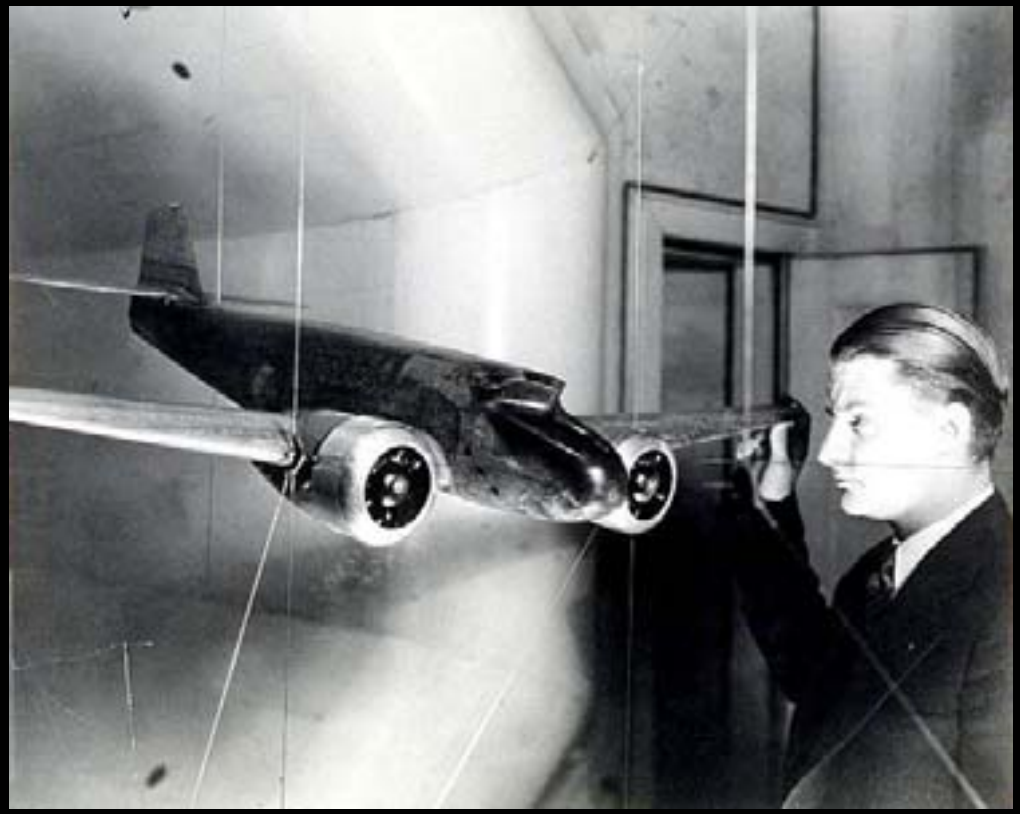


Can you see it now?

Use of Models in Engineering

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

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

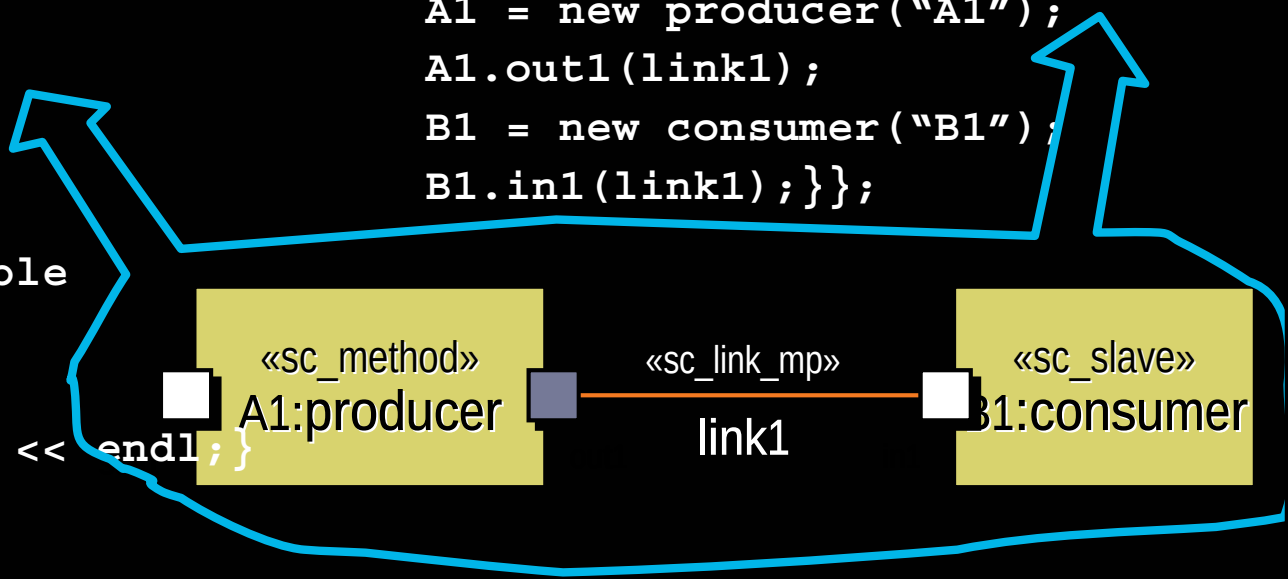


- We don't see everything at once
- What we do see is adjusted to human understanding

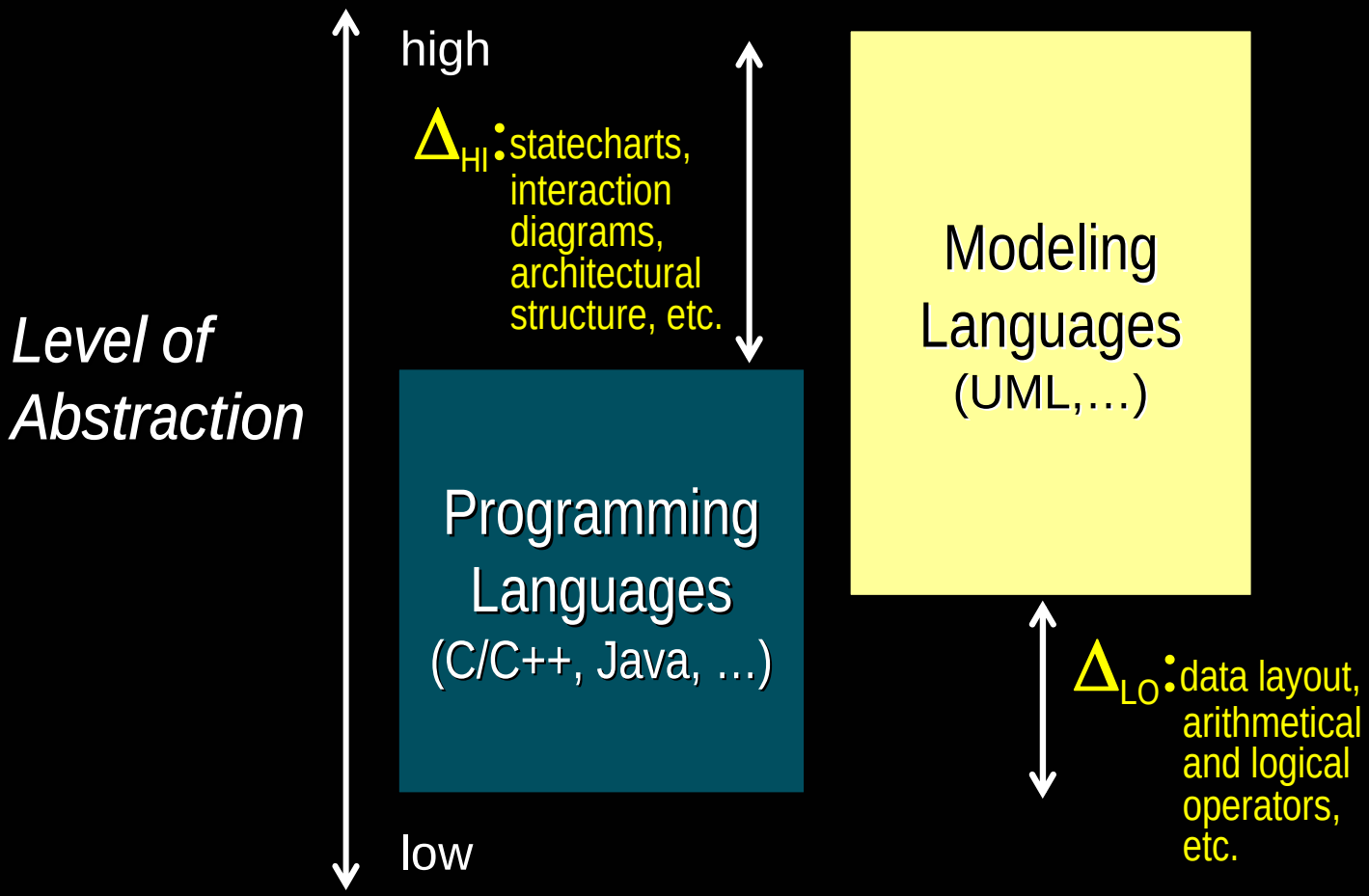
The Model and the Code

```
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;}
```

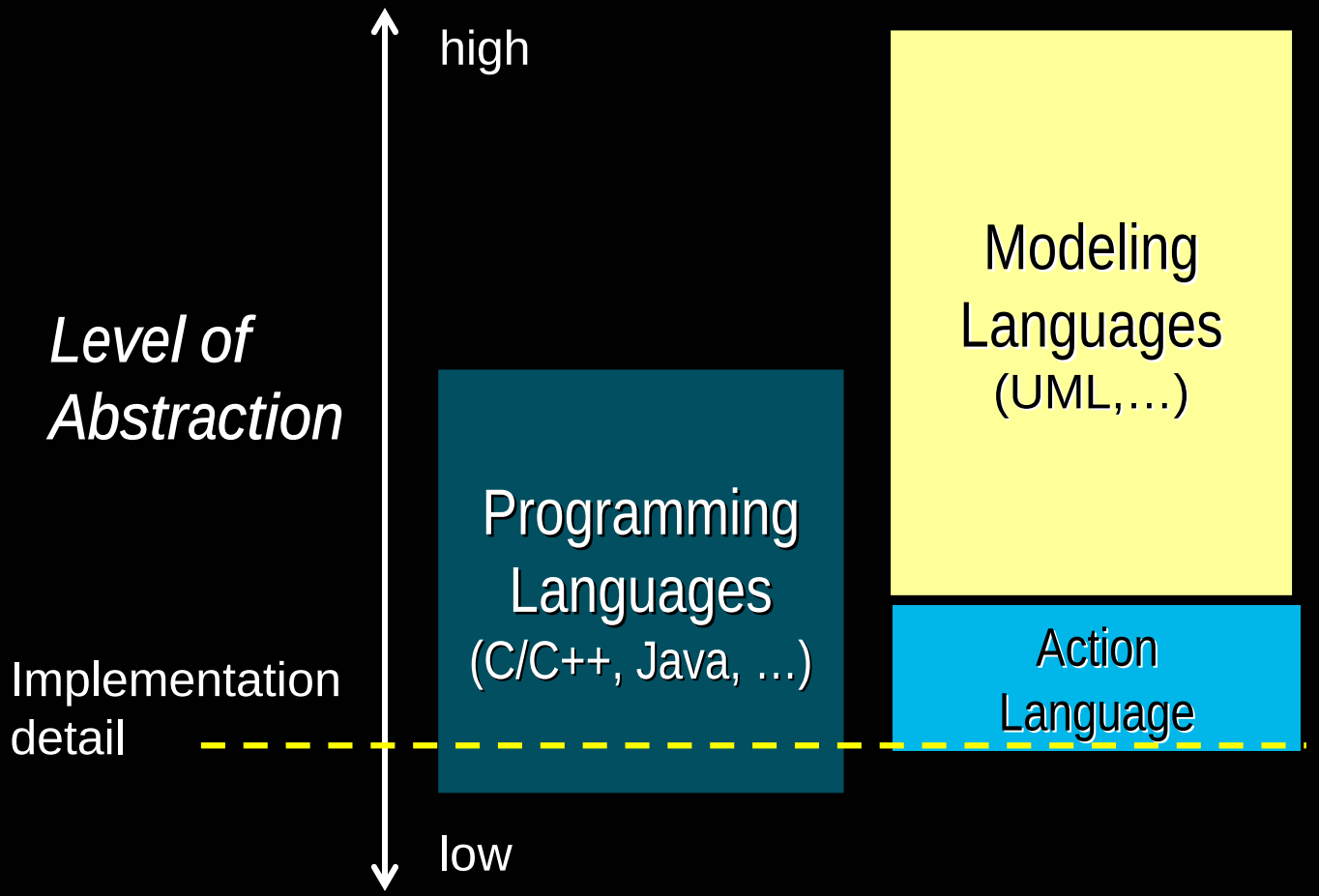
```
SC_CTOR(consumer)
{
SC_SLAVE(accumulate, in1);
sum = 0; // initialize
};
SC_MODULE(top) // container
{
producer *A1;
consumer *B1;
sc_link_mp<int> link1;
SC_CTOR(top)
{
A1 = new producer("A1");
A1.out1(link1);
B1 = new consumer("B1");
B1.in1(link1);}};
```



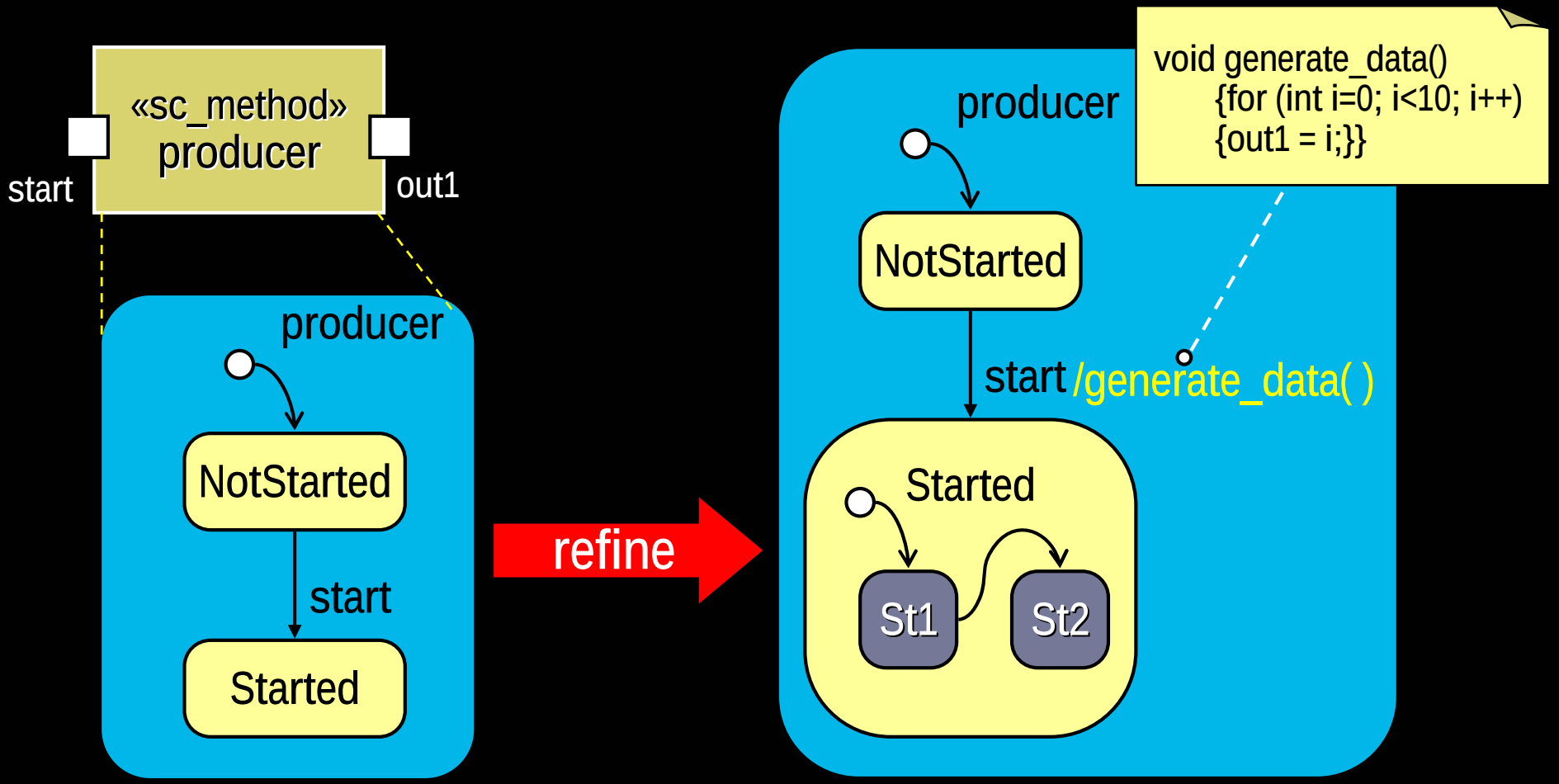
Modeling Languages vs Programming Languages



Models: Filling in the Detail



Model Evolution: Refinement

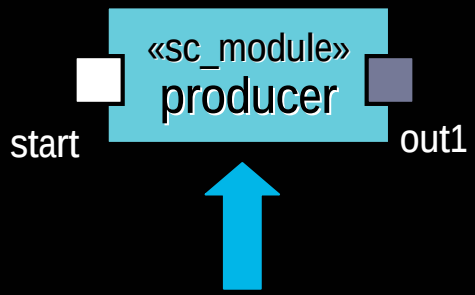


- Models can be refined continuously until the application is fully specified ⇒ *the model becomes the system that it was modeling!*

Model-Driven Development

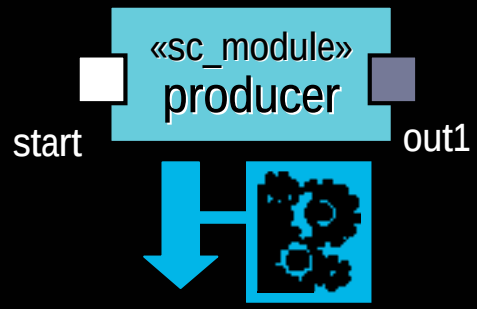
- Model-Driven Development (MDD): An approach to software development in which the focus and primary artifacts of development are models (as opposed to programs)
- Based on 2 time-proven approaches:

(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;}
```

MDD: State of the Art and Adoption

- Systems using fully automated code generation from models written using a modeling language:
  - Size: Complete systems equivalent to ~ 5 MLoC and involving several hundred developers
  - Performance: within ±5-15% of equivalent manually coded system
  - There are many similar examples of successful MDD projects
- Yet, MDD is practiced by only a small percentage of software developers
  - In fact, the vast majority dislike it and many actively oppose it!
- Why?

The Idiosyncrasies of Software – 1

Hardware

"Mindstuff"

Process and tools

Software

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

Requirements

Req. 3.2.4:

The system shall jump through burning hoops and leap over 30' fences

Req. 3.2.4:

The system shall be mauve with pink frills.

…

Relative to other engineering disciplines, this ingredient plays a disproportionally *dominant role* in the engineering process

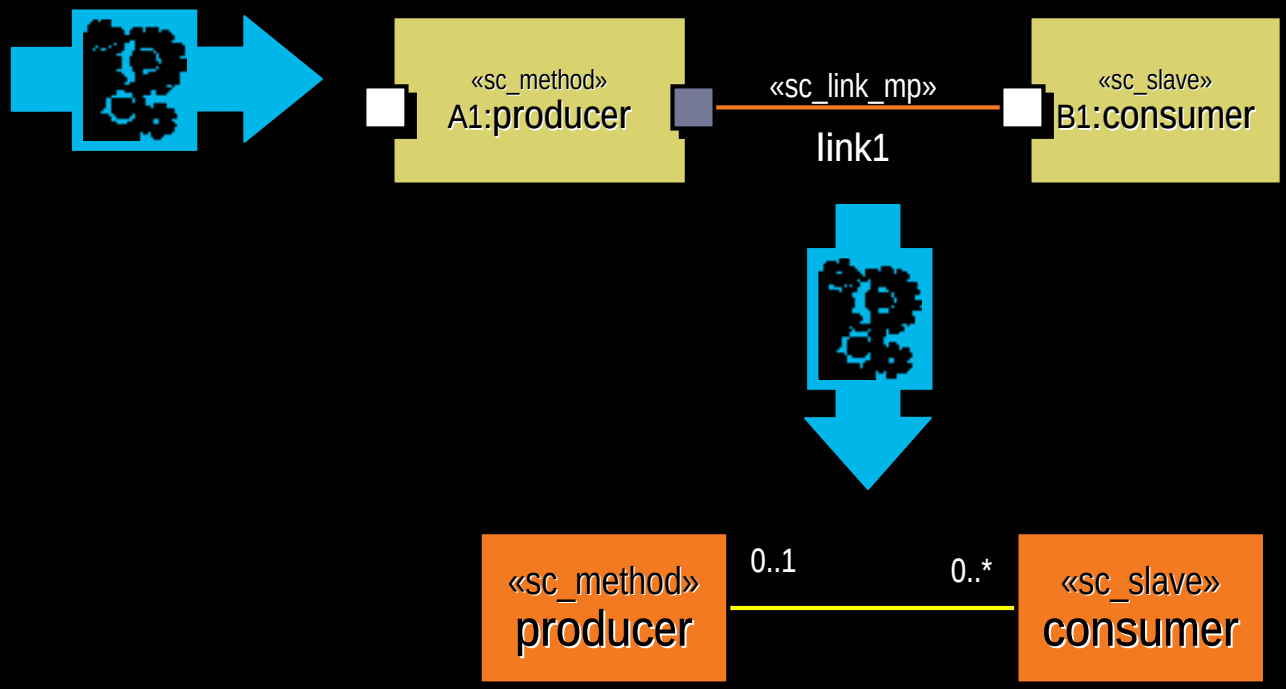
Some Consequences

- 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 medium becomes the message

The Idiosyncrasies of Software – 2

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

```
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 setting is a primary design constraint that cannot be overcome by layers of software*
- *Yet, students are still being taught that "platform concerns" are second order issues*

What I (and Others) Did Not Learn in School

– But Should Have

or, How can Education Help?

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"?

- Dubious premise
  - 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 solution-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

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 direly 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 reason why the “best” technical solution is not the best solution
  - 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., 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 context

Speaking of Business…

- Prepare software experts for work in a business-oriented environment
  - They may become entrepreneurs or they may work in an entrepreneurial environment
- "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

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 of Industrial Software Development

- 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)

Conclusion

- Software is a truly unique engineering medium
  - Dominated by the human mind rather than physical reality

    …but not completely
  - The ability to define our own realities
- This requires a unique combination of new and old engineering principles
  - We have yet to discover the right balance
- The role of education is crucial
  - Developing an engineer's sense of responsibility and perspective
  - Inevitably, this requires a broader education that extends beyond specific technologies