CSC2125: Modeling Methods, Tools and Techniques Winter 2018

Marsha Chechik

Department of Computer Science
University of Toronto

Intro and Organizational Meeting

http://www.cs.toronto.edu/~chechik/courses18/csc2125

About Me

- Ph.D. from Maryland
- At UofT since 1996
- General interests: verification (of programs), model-checking, analysis, modeling, product lines, safety

- Office: BA3246, x3820, chechik@cs.toronto.edu
- Office Hours: after class (Monday at 4) and by appointment

Acknowledgements

- (many) slides/ideas from
  - Jourgen Dingel (Queens)
  - Jordi Cabot/J. Bezivin (U. Nantes)
  - KSU CIS 842 (J. Hatcliff and M. Dwyer)
  - T. Ruys (U Twente)
  - J. Atlee (U Waterloo)
  - E. Posse (Queen's)
  - I. Krueger (UCSD)

- (other sources are cited)

Several Similar Terms

Model Driven Development (MDD)

- the general notion that we can construct a model of a software system and transform it into software

Model Driven Architecture (MDA)

- the developer creates a software model that abstracts away the program's execution platform (e.g., the Web, CORBA, ,NET)
- tools can generate an implementation for a specific platform automatically

Model Driven Engineering (MDE)

- the developer creates a model in terms of the user's domain, abstracting away software-technology concepts (e.g., algorithms, execution platform, programming language)
- tools generate an implementation automatically

Model Based Software Engineering (MBSE)

- An approach to software development in which the focus and primary artifacts of development are models (vs programs)

This lecture

- **Motivation**
  - Software development is hard
  - It won't get any easier
  - Need more powerful techniques and tools
- **Course overview / Admin stuff**

What is Software?

"The programs, routines, and symbolic languages that control the functioning of the hardware and direct its operation."

American Heritage Dictionary

Application Domain

Software

Hardware

?

!

What is Software: Crucial

- **Crucial to functioning of modern society**
  - critical infrastructure
    - transportation
    - energy
    - water
    - communication
  - business and finance
  - health care
  - military
  - entertainment
  - education
  - …

What is Software: Complex (Cont'd)

- Windows 95
  - 11 million lines of code
  - > 200 programmers & testers
- Windows NT
  - > 16 million lines of code
- Windows XP
  - 35 million lines of code
- Windows Vista
  - > 50 million lines of code

- Cellphone (2005)
  - 2 million lines of code
- Car
  - 1981: 50,000 lines of code
  - 2005: 10 million lines of code
  - 2010: 100 million lines of code
- Pacemaker
  - 100,000 lines of code

Software is one of the most complex man-made artifacts!

But perhaps "Lines of code" is a poor measure of complexity?!

[Source: "Why Software Fails". R.N. Charette. IEEE Spectrum, Sept 2005]

What is Software: Complex (Cont'd)

- State of a program P
  - snapshot of execution of P mapping variables to values
  - e.g., <x=1, y=0, z=42, flag=true, A=[0,0,0]>
- State space of P
  - set of reachable states of P
- State spaces can be very large
  - in Java, a single integer: 4.2 billion (10^9) possible values
  - a program with 2 integers: >16,000 quadrillion (10^{15}) possible states

What is the size of the state space of Windows XP?

Software is one of the most complex man-made artifacts!


```
Search completed with depth 20

Number of states: 49149
Number of transitions: 65532

VeriSoft Version 2.0.6

real      3:46.5
user         2.6
sys       2:26.4
dingel@innovate:
```

What is Software: Failing

Consequences of this complexity (Cont'd)

- **Failing software**
  - money
    - Examples: ESA Ariane 5, Mars Climate Orbiter, Skype bug in '07, blackout in '04, MS Zune bug in '09, US telephone system, …
    - Cost of errors in software in US in 2001:

US$ 60B

[**Source**: US National Institute of Standards and Technology]

  - lives
    - Therac 25, …

- **More details**
  - Peter Neumann's **`www.risks.org`**
  - Ivars Peterson. Fatal Defect: Chasing Killer Computer Bugs. Vintage Books, New York, 1996.

Example: Therac-25 (1985-87)

- Radiotherapy machine with SW controller
- Several deaths due to burning
- **Problems:**
  - "poor SWE practices",
  - error messages cryptic/undocumented,
  - false error messages,
  - user interface w/o safety checks
- **References:**
  - N.G. Leveson and C.S. Turner. An Investigation of the Therac-25 accidents. Computer, 26(7):18-41, July 1993.

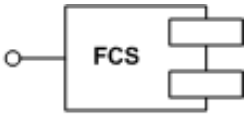
Example: ESA Ariane 5 (June 1996)

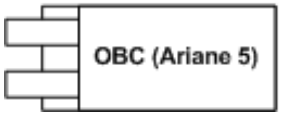
- On June 4, 1996, unmanned Ariane 5 launched by ESA explodes 40 seconds after lift-off
- One decade of development costing $7 billion lost
- Rocket and cargo valued at $500 million destroyed

- What went wrong?
  - Bad reuse of code from Ariane 4
  - Bad fault-tolerance mechanism
  - Bad coding practices

Example: ESA Ariane 5 (June 1996) (Cont'd)

- Example of how not to do reuse:
  - Parts of Flight Control System (FCS) taken from Ariane 4
  - Horizontal velocity much greater for Ariane 5
  - Unprotected conversion operation in FCS causes error
  - On-board computer (OBC) interprets error code as flight data
  - …
  - Launcher self-destructs





- Example of how not to achieve fault-tolerance:
  - FCS and backup FCS identical, thus backup also failed
- Example of how not to code:
  - When code caused exception, it wasn't even needed anymore
- References:
  - [Gle96] and **www.ima.umn.edu/~arnold/disasters/ariane.html**

Example: NASA Mars Climate Orbiter (1999)

- Some programs worked in English units, some metric units
- Conversion from English to metric forgotten
- Instead of 65 miles probe attempted to orbit 65 km

  (40 miles) above Mars
- $327M lost
- References:
  - **`http://mars.jpl.nasa.gov/msp98/orbiter/`**

Example: NASA Mars PathFinder

- Launched December 4, 1996
- A few days after landing on Mars, the Sojourner rover tasks began missing their deadlines causing **total system resets**
- Problem: **priority inversion** is the scenario where a low priority task holds a shared resource that is required by a high priority task
- Reference:

```
http://research.microsoft.com/en-us/um/
                people/mbj/mars_pathfinder/
                 Authoritative_Account.html
```

Mars Pathfinder launched on 4 December 1996

Example: Skype

The New York Times
nytimes.com

August 17, 2007

Error in Skype's Software Shuts Down Phone Service

By BRAD STONE

SAN FRANCISCO, Aug. 16 — The online telephone service Skype was not working for much of the day on Thursday, leaving its 220 million users, some of them small businesses that had given up their landlines, without a way to call colleagues, customers and friends.

Executives at Skype, a division of eBay that is based in Luxembourg, said its engineers worked throughout the day to bring the service back online. But they said that while they had pinpointed the source of the problem, they still did not know why it had resulted in a network failure, and they could not ensure that the service would be running smoothly again by Friday.

"There is a chance this could go on beyond tomorrow, but it's our hope that it's going to be resolved," Kurt Sauer, Skype's chief security officer, said. "What happened today was caused by a unique set of events, the genesis of which is not entirely understood."

Example: The Blackout Bug

- 50 Million people w/o electricity
- Worst black out in North American history
- Cause: Race condition in alarm system (10^6Loc of C)

Tracking the blackout bug

Kevin Poulsen, SecurityFocus 2004-04-07

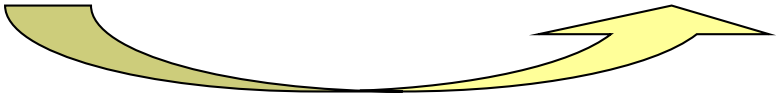
<snip>

languages. Eventually they were able to reproduce the Ohio alarm crash in GE Energy's Florida laboratory, says Unum. "It took us a considerable amount of time to go in and reconstruct the events." In the end, they had to slow down the system, injecting deliberate delays in the code while feeding alarm inputs to the program. About eight weeks after the blackout, the bug was unmasked as a particularly subtle incarnation of a common programming error called a "race condition," triggered on August 14th by a perfect storm of events and alarm conditions on the equipment being monitored. The bug had a window of opportunity measured in milliseconds. "There was a couple of processes that were in contention for a common data structure, and through a software coding error in one of the application processes, they were both able to get write access to a data structure at the same time," says Unum. "And that corruption led to the alarm event application getting into an infinite loop and spinning." **Testing**

<snip>

In the Future: Two Main Forces

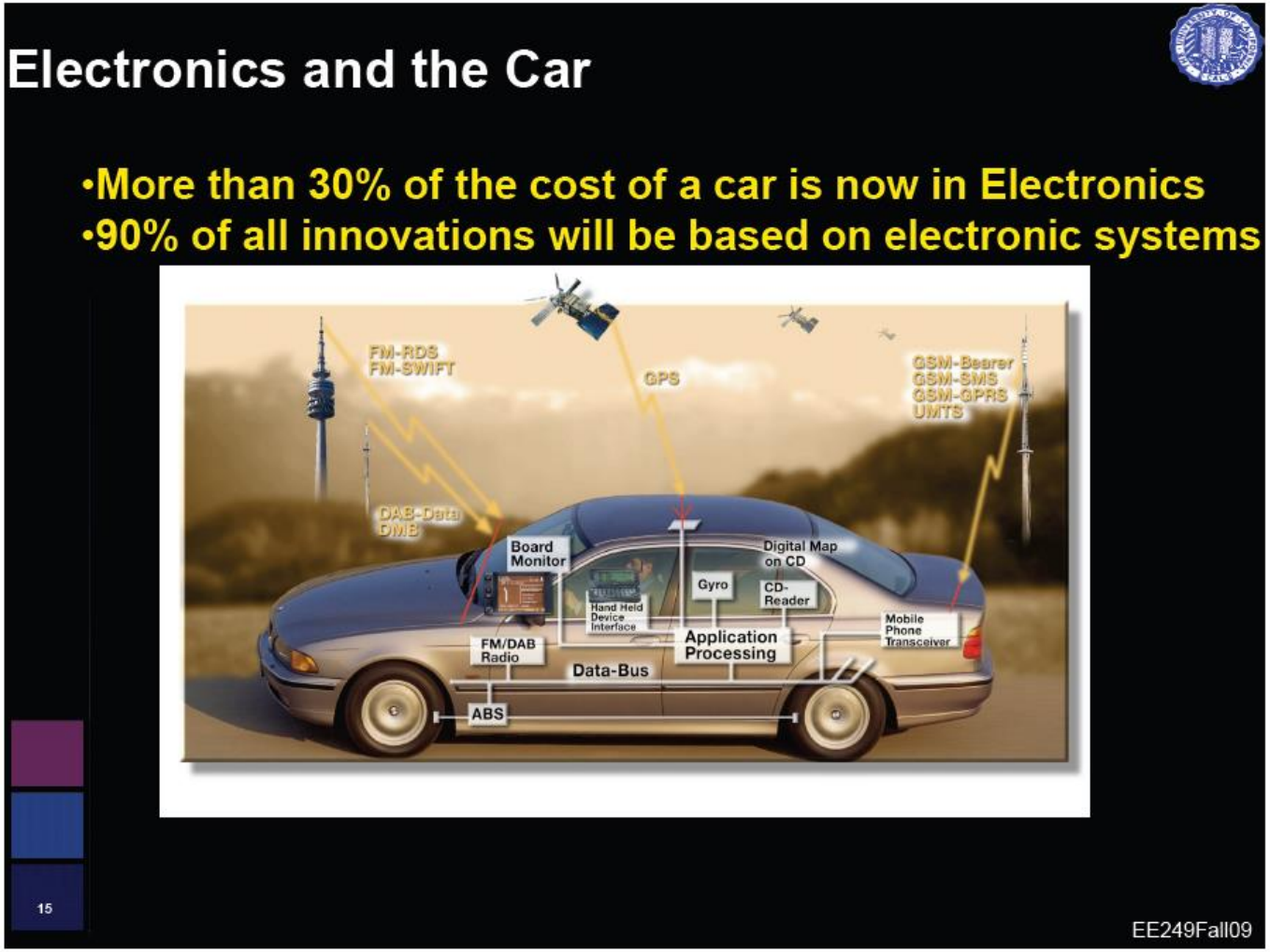
	Today	Tomorrow
1. Computerization:	mechanic & manual	electronic & automatic
2. Integration:	stand-alone & incompatible	networked & interoperable



more features, capabilities,
productivity, efficiency, …

Computerization: Example

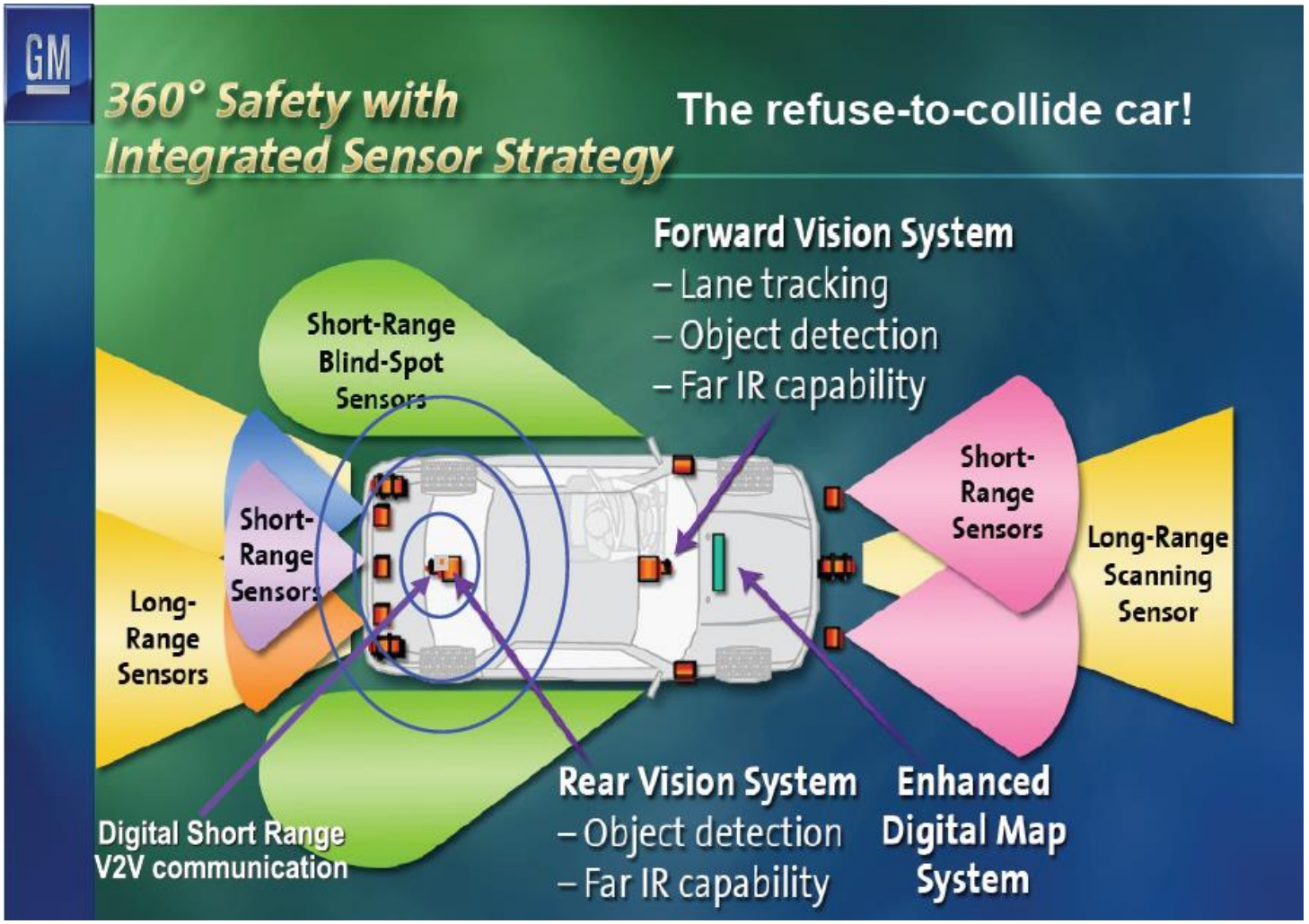
- for innovation



[from A. Sangiovanni-Vincenticelli]

Computerization: Example (Cont'd)

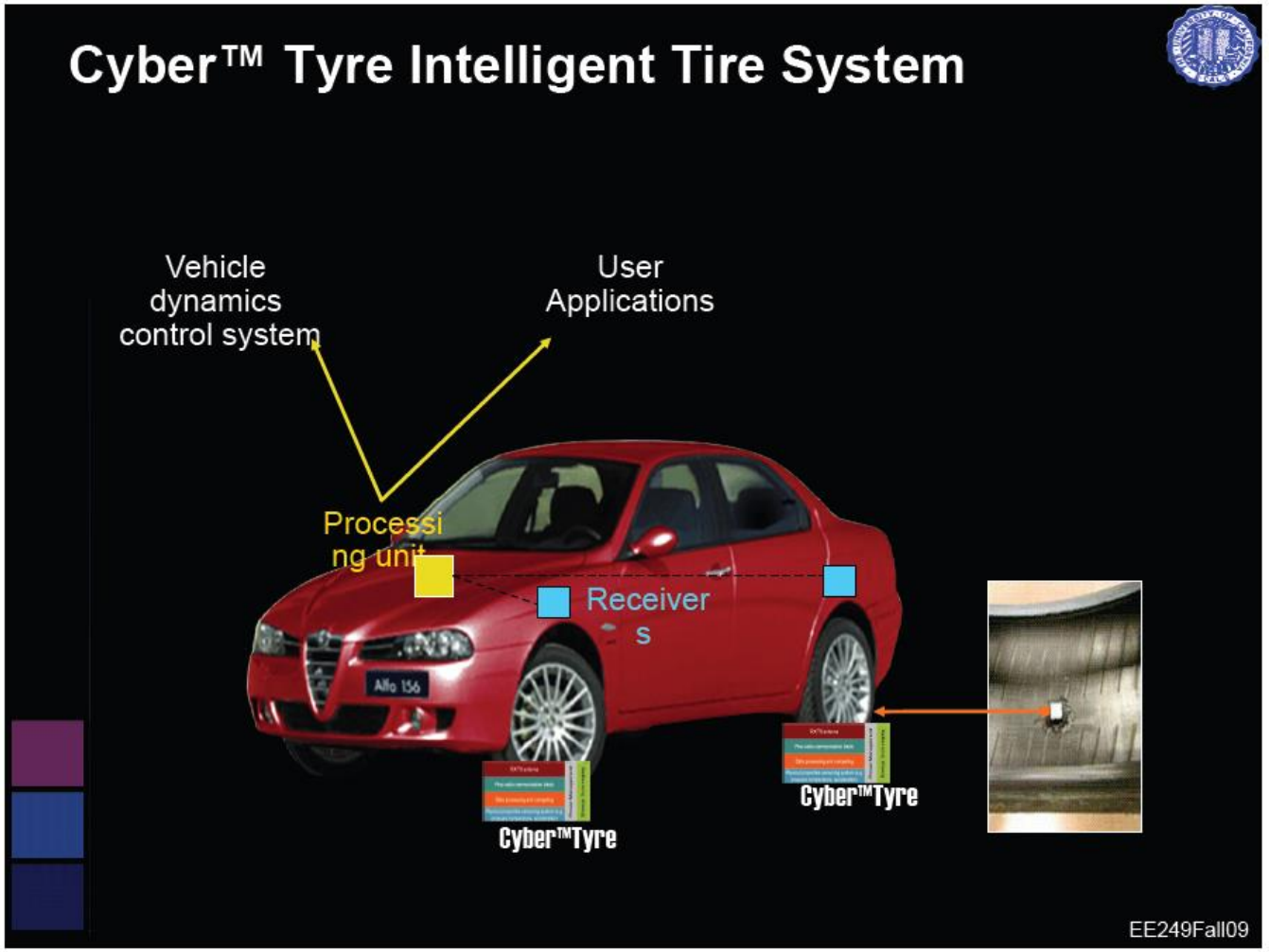
- for safety



[from A. Sangiovanni-Vincenticelli]

Computerization: Example (Cont'd)

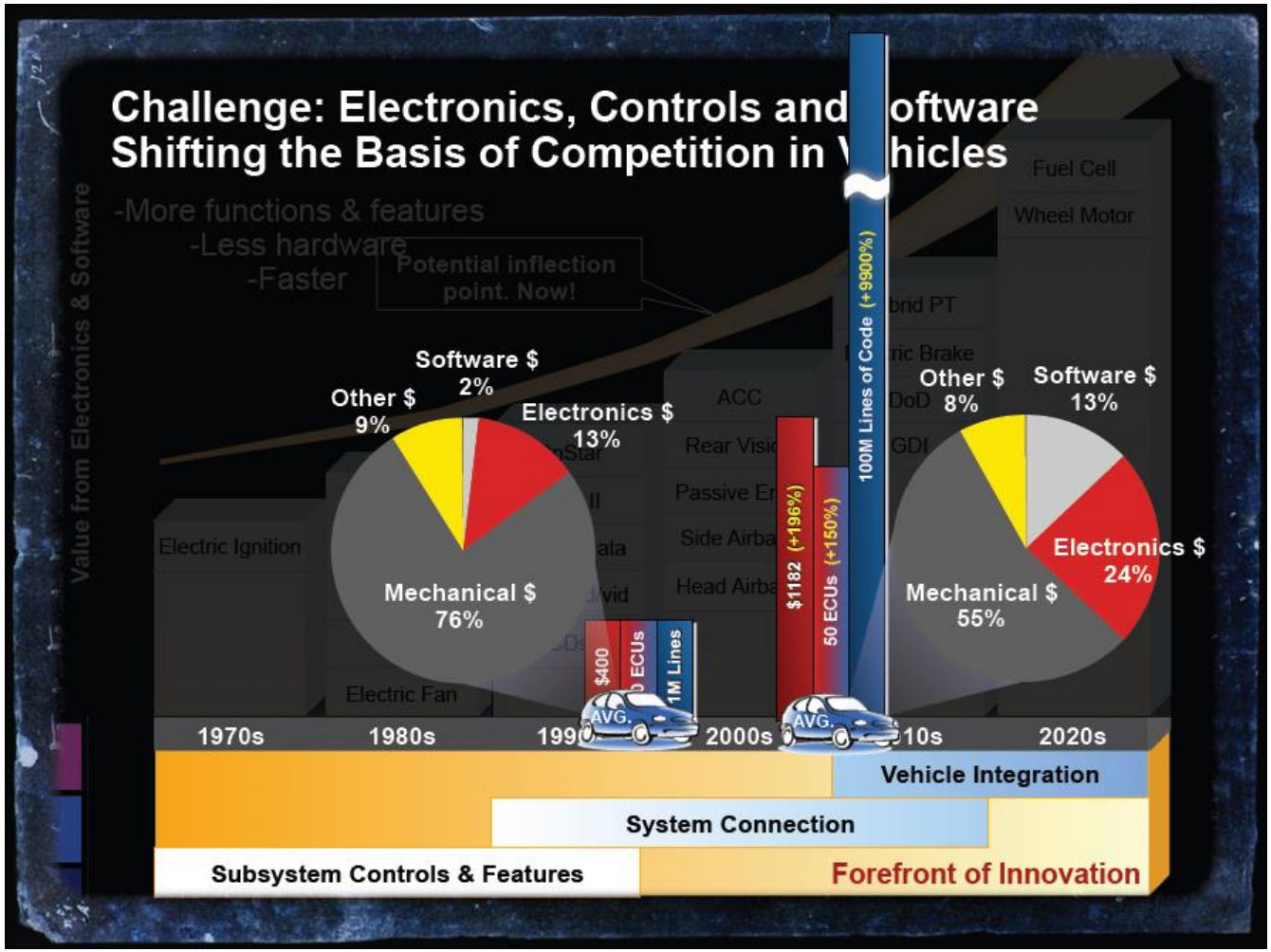
- The tire as intelligent sensor



[from A. Sangiovanni-Vincenticelli]

Computerization: Example (Cont'd)

- Software content could increase 100x in next 5-6 years!



[from A. Sangiovanni-Vincenticelli]

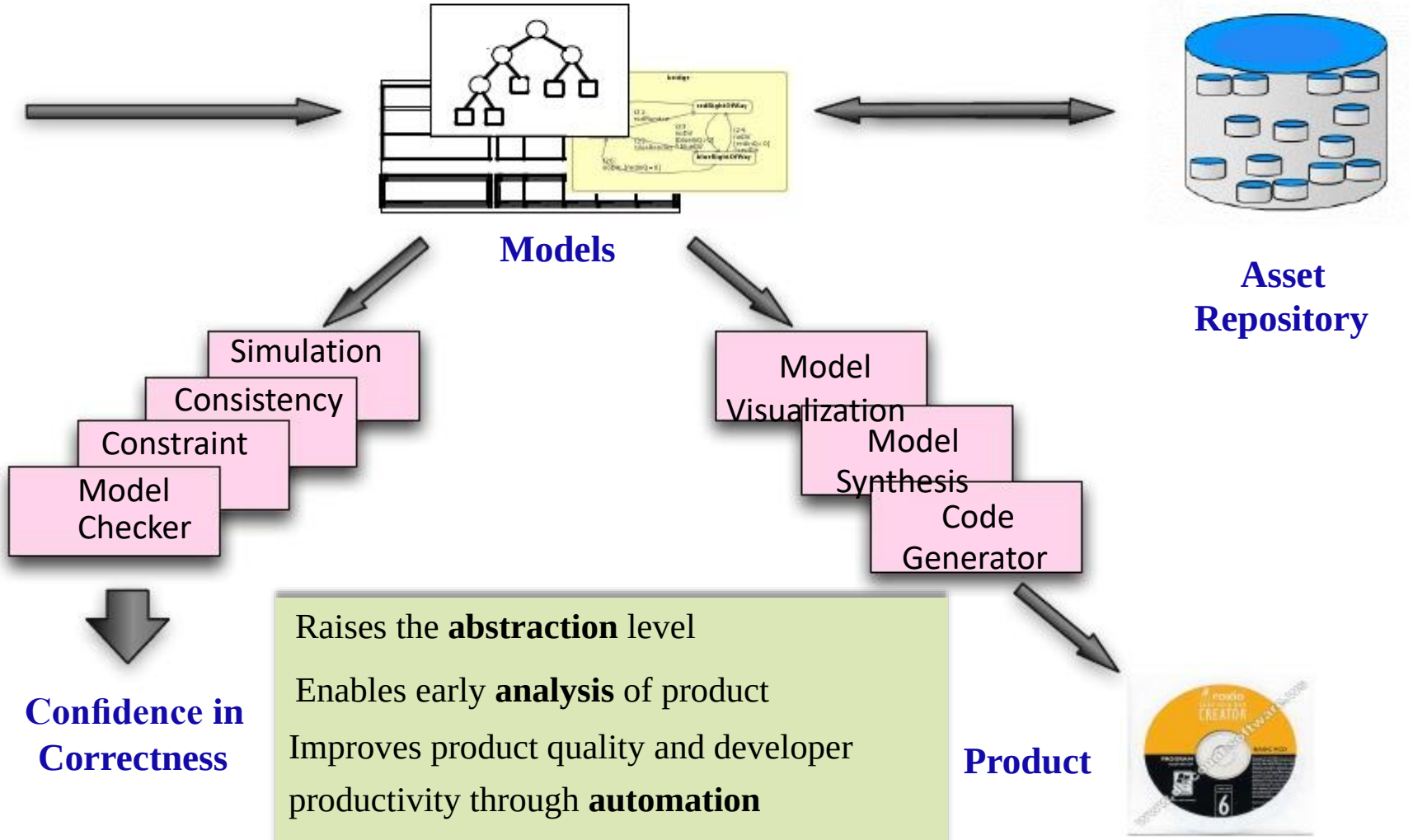
Integration: Examples

- Government
  - IRS tax system: 100 million lines of code
- Health care
  - HL7 standards (**`www.hl7.org`**)
    - for exchange, management and integration of electronic healthcare information
- Energy
  - "smart-grid" projects in US
- Transportation
- Business and finance
- Military
- Communications

"Systems of Systems"
"Ultra-large-scale Systems"

A possible solution?
Model-Based Software Engineering

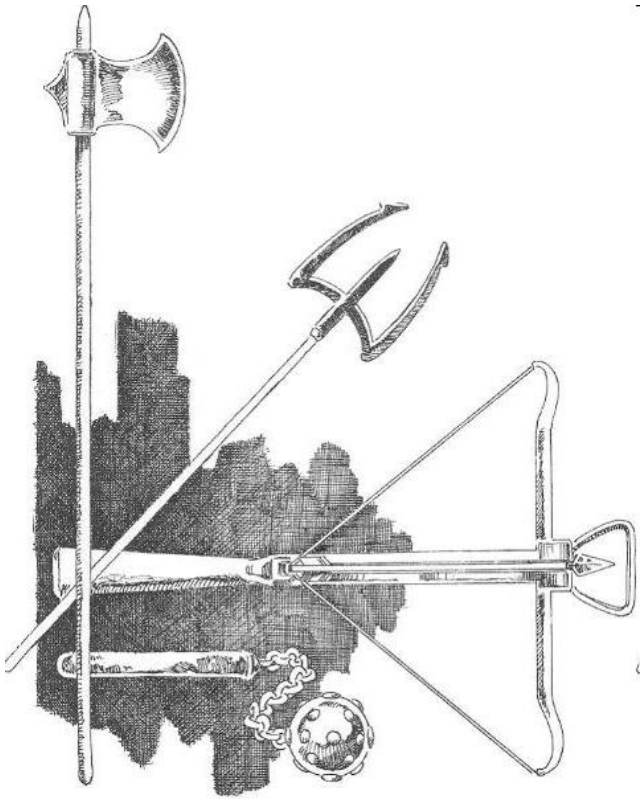
An approach to software development in which the focus and primary artifacts of development are models (vs programs)



Modeling - a weapon to tame complexity?

- abstraction
- automation
- analysis
- decomposition
- reuse

} key ingredients to MDD (and engineering in general)



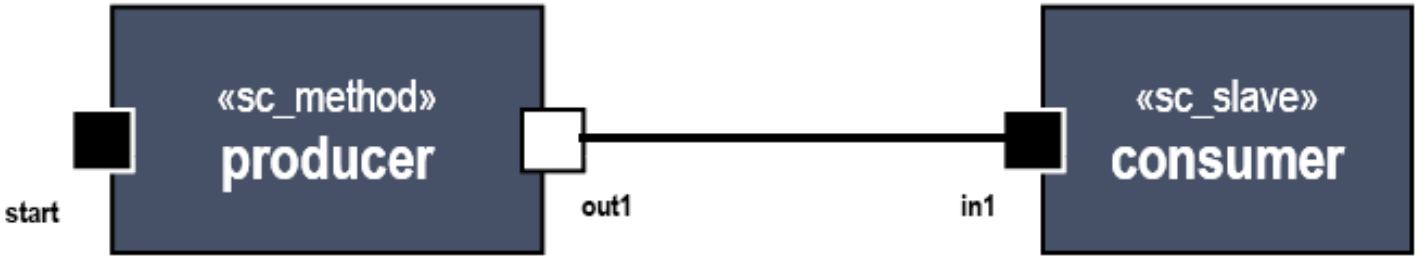
A Small Fragment of Modern Software…

```
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;
}
```

```
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 (UML 2)



Can you see it now?

Bill Gates on the Topic

"Modeling is the future ...

And the promise here is that you write a lot less code, that you have a model of the business process ...

So, modeling is pretty magic stuff, whether it's management problems or business customization problems or work-flow problems, visual modeling ...

It's probably the biggest thing going on ..."

Bill Gates. "What is Bill Gate Thinking? Interview", eWeek.com, 3/30/2004

A look over the fence

Software Engineering currently isn't like engineering at all!

Engineering

1. build (mathematical) models
2. analyze models rigorously
3. refine models
4. build artifact
5. little testing

Characteristics

- Very rigorous
- "front-loaded"
- **Main QA technique:**
  Modeling & analysis

Software Engineering

1. some (informal) modeling
2. build artifact
3. some (informal) reuse
4. lots of testing

Characteristics

- Mostly informal
- "back-loaded"
- **Main QA technique:**
  Testing (often >50% of total development effort)

A look over the fence (Cont'd)

engineering:

"The application of scientific and mathematical principles to practical ends such as the design, manufacture, and operation of efficient and economical structures, machines, processes, and systems"

American Heritage Dictionary

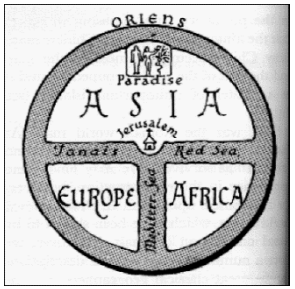
software engineering:

"The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, that is, the application of engineering to software" IEEE Standard 610.12

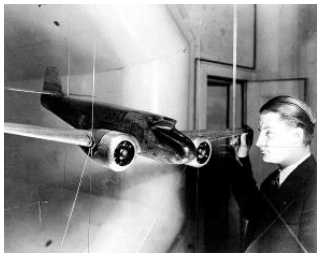
Yeah, right!

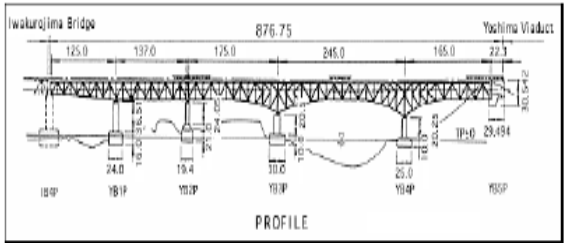
Modeling to the rescue

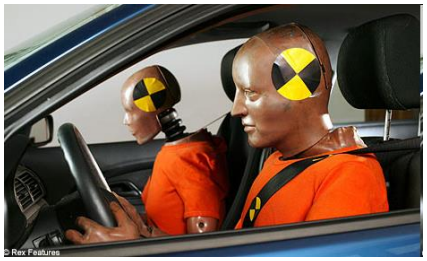
- **Modeling is key to almost all human decision making**



- **Modeling is key to other engineering disciplines**







- **However, the role of models in software development is still relatively small**
  - documentation, communication
  - when was the last time you've used a model of software for analysis?

What is Software Development?

"The system shall do this, that, and the other thing"

manual,
costly,
error-prone

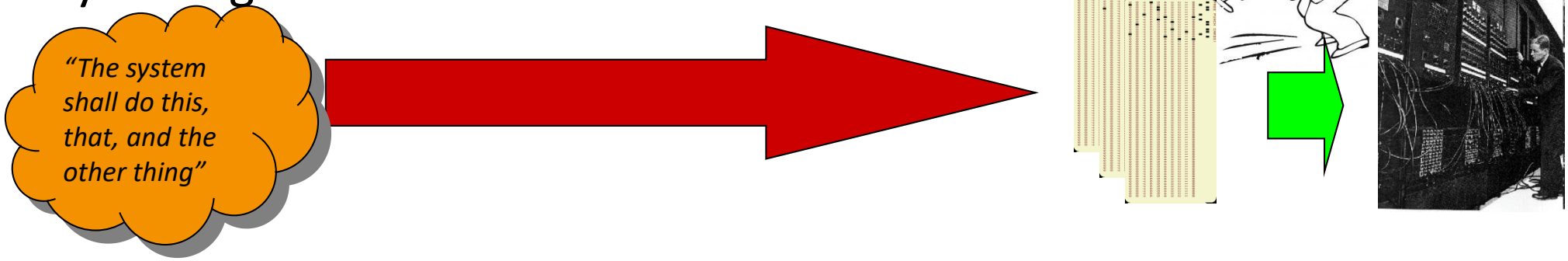
"arrow of pain"

automatic,
cheap,
well-understood

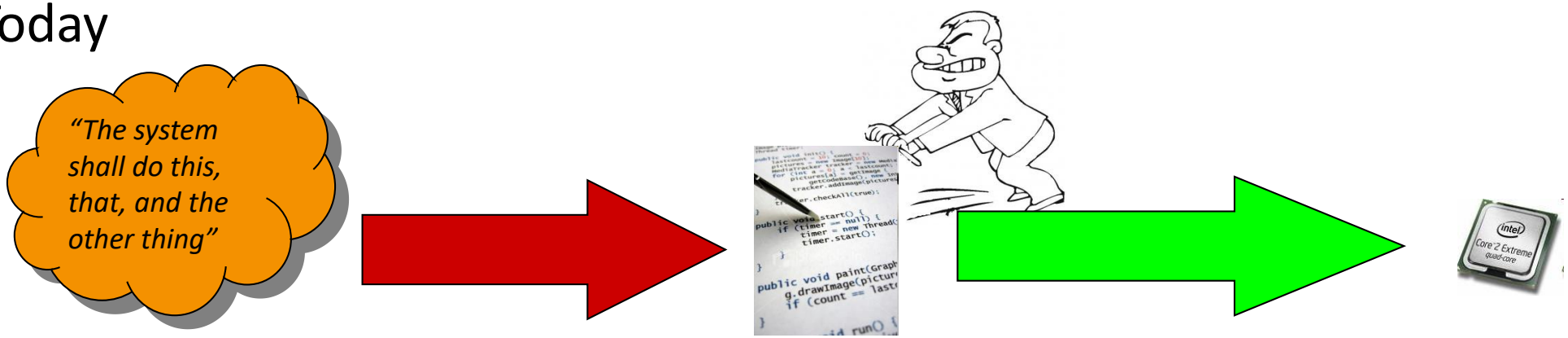
"arrow of joy"

A Look at History

40 years ago

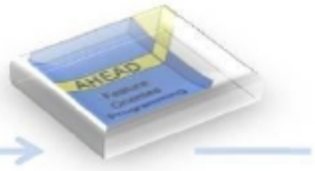


Today

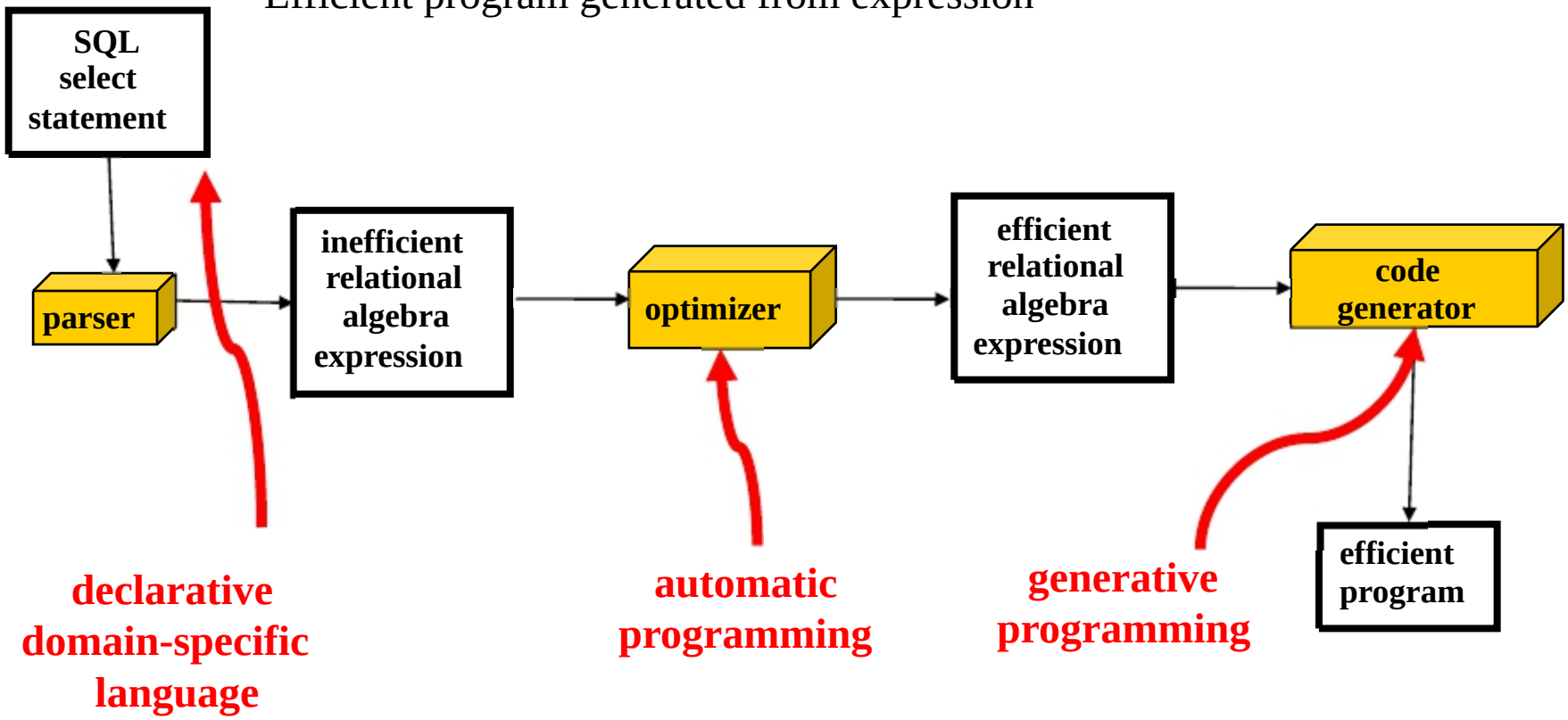


How do we shorten the "arrow of pain" further?

Relational Query Optimization (RQO)



- Declarative query is mapped to an relational algebra expression
- Each expression represents a unique program
- Expression is optimized using algebraic identities
- Efficient program generated from expression



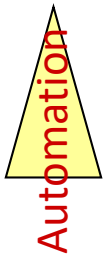
Two Very Important Weapons

Weapon 1: Abstraction

- Put more and more higher-level concepts into programming languages
- Examples:
  - variables, basic data types (bool, arrays)
  - abstract data types (data abstraction)
  - functions and procedures (procedural abstraction)
  - objects
  - semaphores, locks

but what makes this work in practice is

…, Python, C#, Java, VB, C++, Ada, Modula-2, Smalltalk-80, Basic, C, Prolog, PL/1, APL, Cobol, LISP, Algol, Fortran, Assembler, Machine code



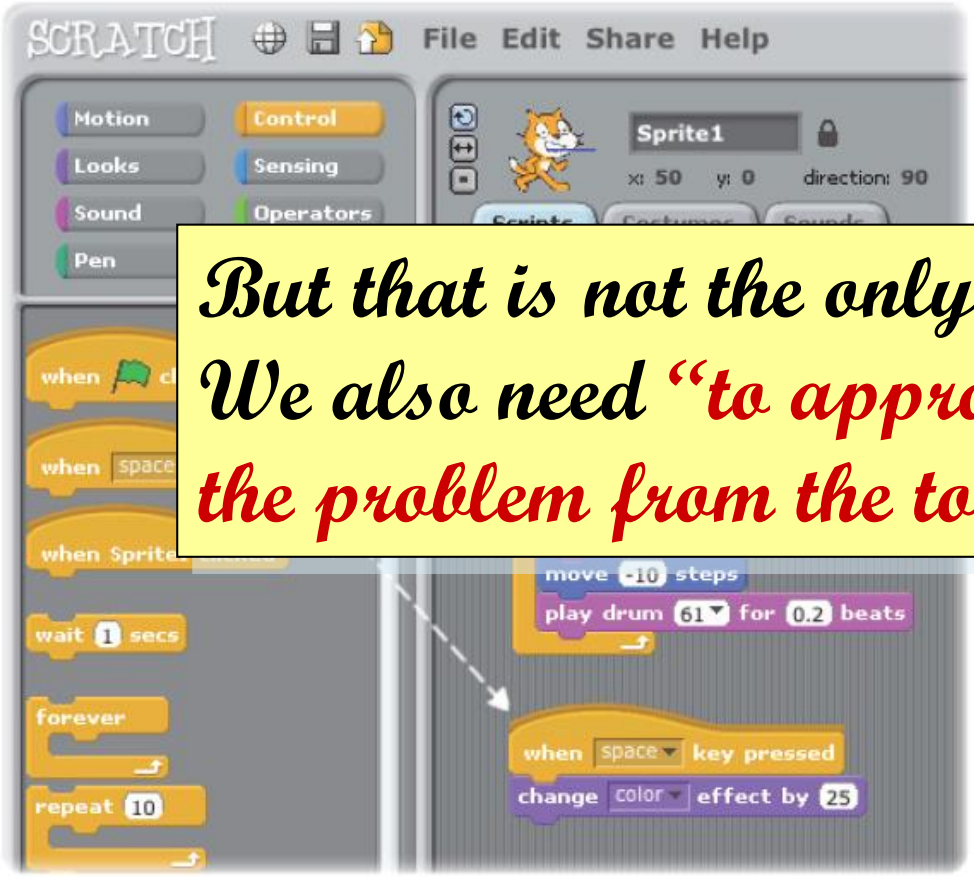
Weapon 2: Automation

- automatically compile high-level concepts into executable code

Better Programming Languages

- E.g., Scratch, Fortress, Go

scratch.mit.edu



But that is not the only thing! We also need "to approach the problem from the top"!

projectfortress.sun.com

Welcome to Project Fortress

Fortress is a new programming language designed for high-performance computing (HPC) with high programmability. Fortress features include:

- Implicit parallelism
- ansactions
- xible, space-aware, mathematical syntax
- atic type-checking (but with type erence)
- finition of large parts of the language in own libraries

```
4 := 0
for k<-1:n do
    a[k] := (1-alpha)b[k]
    sum += c[k] x^k
end
```

$$sum : \mathbb{RG}4 := 0$$
for $k \leftarrow 1 : n$ do
$$a_k := (1 - \alpha) b_k$$
$$sum \mathrel{+}= c_k \, x^k$$
end

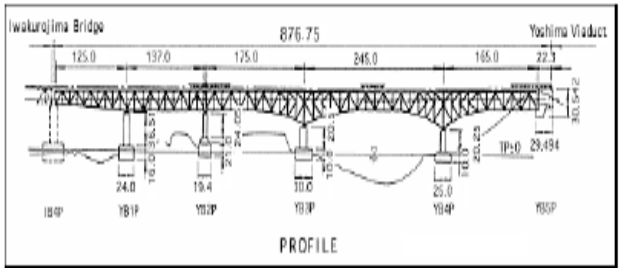
golang.org

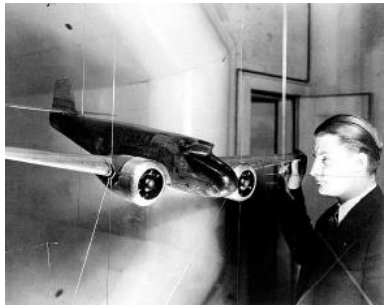
The Go Programming Language

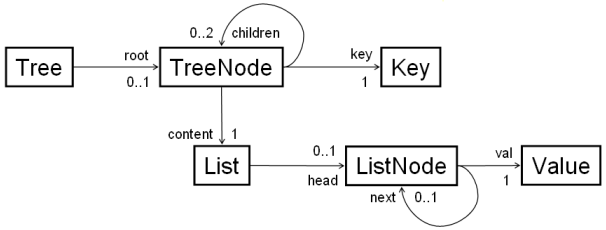
Model-Driven Development

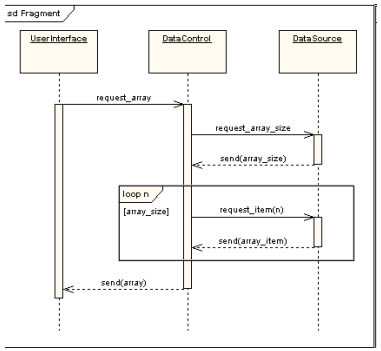
- **Main goal:**
  - increase level of abstraction (weapon 1)
    - through use of models
  - increase degree of automation (weapon 2)
    - e.g., via code generation from models
  - improve analysis capabilities (weapon 3)
    - e.g., through use of models

in software development

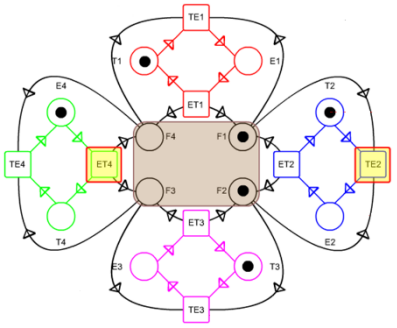






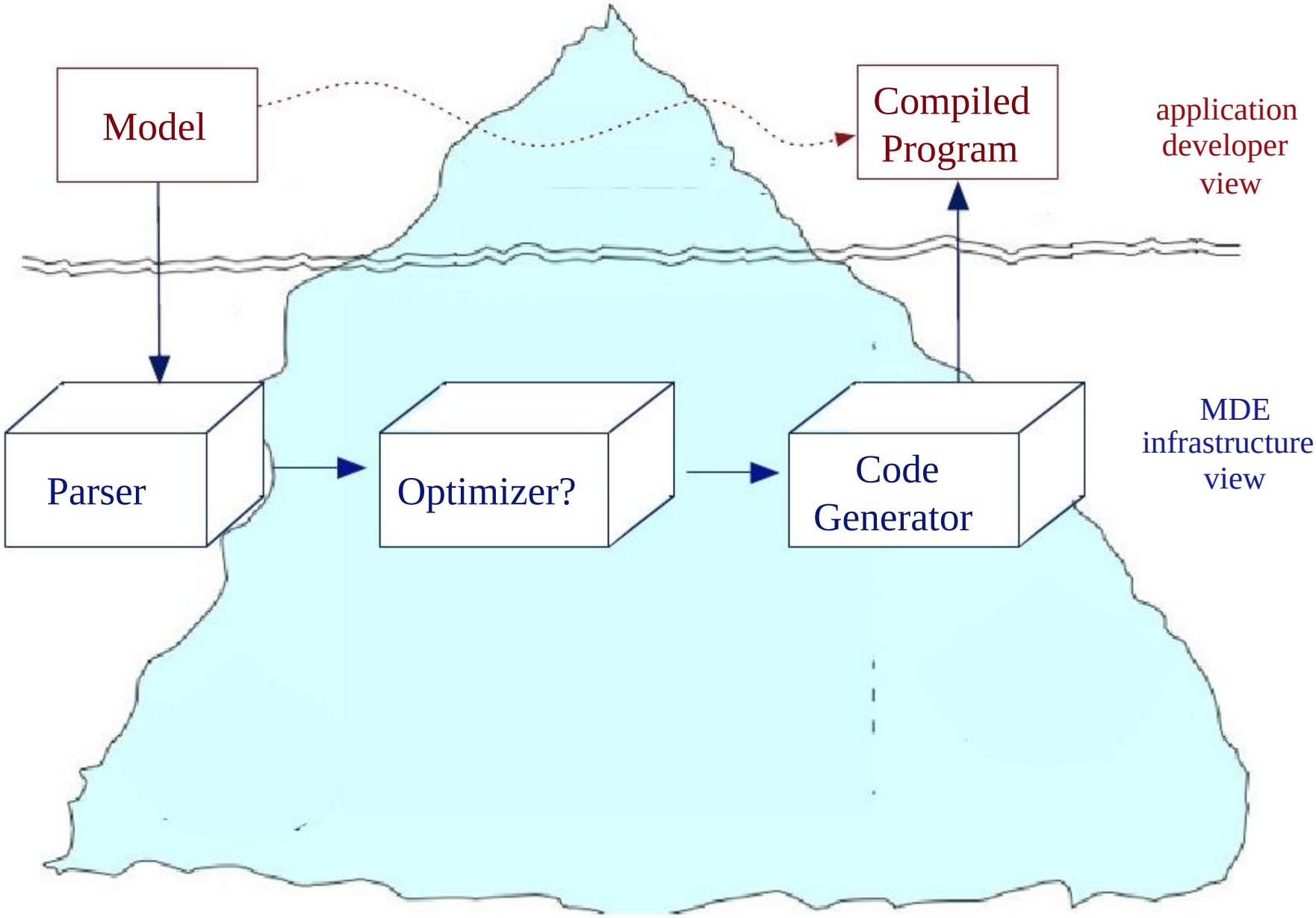



```
game is poker

turns alternate clockwise

Discard means player removes 0..3 cards or 4 cards if Ace
Fold means player loses

2..6 players

game is Shuffle(deck) and Deal(cards, 5) and (bet(money) or Fold)
        and Discard(hand, N) and Deal(cards, 5-N)
        and (bet(money) or Fold) and compare(cards)

StraightFlush is (R, S) and (R-1, S) and (R-2, S) and (R-3, S) and (R-4, S)
FourKind is (R, s) and (R, s) and (R, s) and (R, s)
FullHouse is (R, s) and (R, s) and (R, s) and (Q, s) and (Q, s)
Flush is (r, S) and (r, S) and (r, S) and (r, S) and (r, S)
Straight is (R, s) and (R-1, s) and (R-2, s) and (R-3, s) and (R-4, s)
ThreeKind is (R, s) and (R, s) and (R, s)
TwoPair is (R, s) and (R, s) and (Q, s) and (Q, s)
Pair is (R, s) and (R, s)
HighCard is (R, s)

hands are [StraightFlush, FourKind, FullHouse, Flush, Straight,
           ThreeKind, TwoPair, Pair, HighCard]

hand is five cards
goal is highest(hand)
```



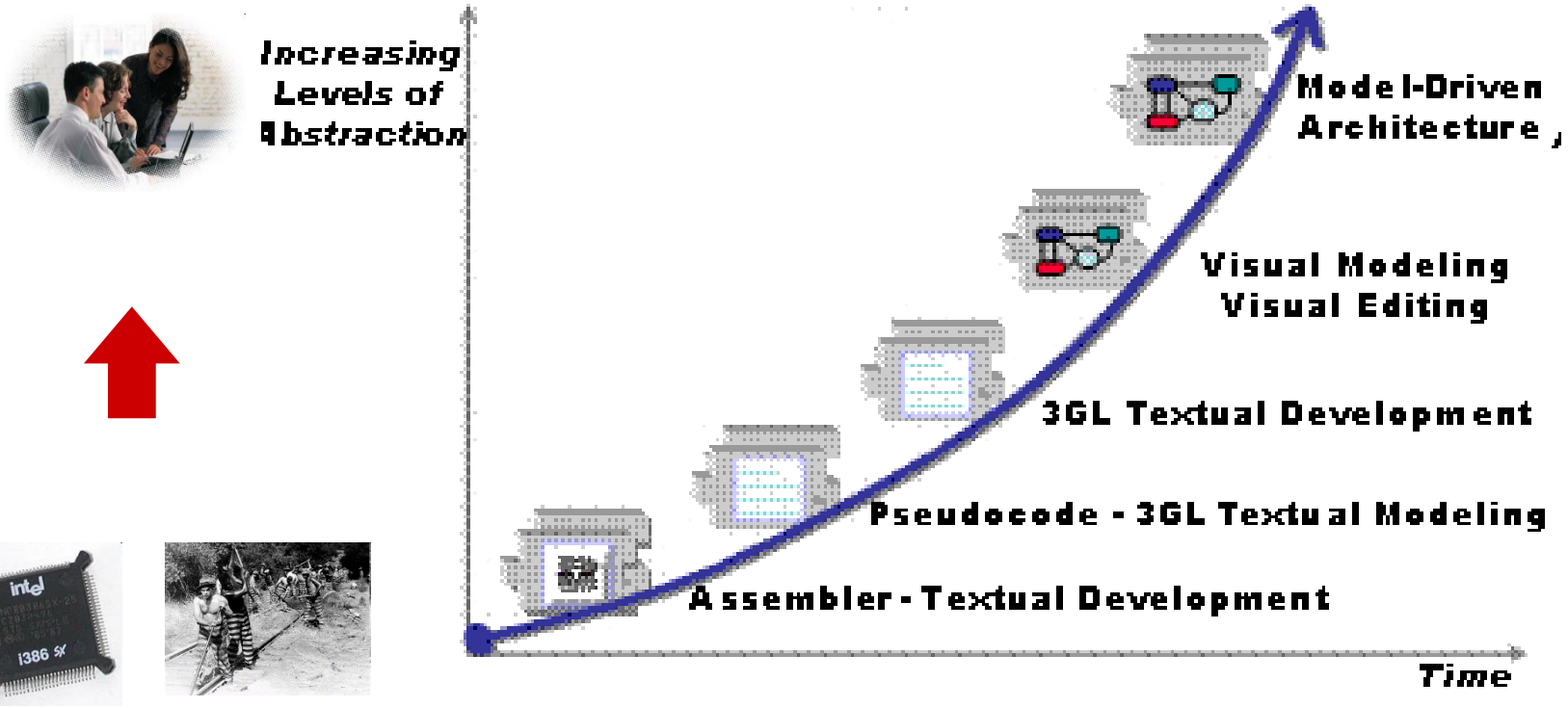

```
process Producer[x]:
  trigger x with ("age", 27) ->
    done

process Consumer[y]:
  when y with message ->
    print message ->
      done
in
  event z in
    par
      Producer[z]
      Consumer[z]
```

Enabling MDE



Software modeling is just continuing a trend

- Models as the result of an abstraction
- Abstraction has been key to many advances in CS
- So, software modeling is just continuing that trend



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

Course Topics

Modelling Notations

Software models, domain-specific notations, meta-modelling

Model Management

Relationships between models, model operations (e.g., merge, match, slice, diff), mega-modeling

Analysis and Verification

Consistency and completeness, simulation, constraint solving, model checking, transformation verification

Model Transformations

Model-to-model transformations, code generation, model synthesis.
Maybe: generative programming, model visualization

Other topics

Safety and security, modeling and reasoning about product lines, models@runtime, biological systems, real-time and embedded systems, combining modeling and machine leaning

Assumed Background

Undergraduate course in software engineering

- Software development activities (e.g., requirements, design, testing)
- Modularity, information hiding
- Software modelling (e.g., UML)
- Sets, functions, relations, mathematical logic
- Knowledge of Eclipse (and EMF) is helpful

Workload and Evaluation

CS2125 is a seminar course that will cover roughly 3 research papers per week.

Workload

- Course readings
- Class participation: 10%
- Paper presentations (2-3): 25%
- Paper reviews (5-7): 15%
- Term project: 50%
  research problem or implementation project (on top of MMTF)

Paper Presentations

Presentations:

- ~30 papers to be presented by students, up to three presentations per week
- Normally 50 minutes per paper: 25 minute presentation, followed by presenter-led discussion
- Evaluated by the class and me (form is on course web page).
  - 65% by the instructor
  - 35% by your classmates

Reviews:

- Plan on reviewing 5-7 papers
- Review form is on the course web page

Project

Types of projects

- work on an open research problem (individual)
- develop / implement / verify modeling notation/relationship/transformation (e.g., using MMINT)

See course Web site for details

Project timeline

- Feb 5: 1 page project proposals due
- April 2?: project papers/reports due
- April 9?: project presentations in class

Reading List

Reading list and schedule are on-line:

www.cs.toronto.edu/~chechik/courses18/csc2125/readings.htm

- most paper links lead to ACM, IEEE, or Springer web pages from which the paper can be retrieved (from on-campus machines)

- I am willing to consider alternative papers, if you have suggestions.

Schedule

Week	Topic, papers, deliverables	Presenter	Reviewers
1. Jan. 8	Introduction, motivation, course organization	Marsha	
2. Jan. 15	Modeling	Marsha	
3. Jan. 22	Modeling notations		
4. Jan. 29	DSLs and DSMLs		
5. Feb. 5	Meta-Modeling 1-page project proposals due		
6. Feb. 12	Model Transformations and their analysis (Marsha out of town, so TBD)		
Feb. 19	Family day. No class		
7. Feb. 26	Model Analysis		
8. Mar. 5	Model Evolution and Management		
9. Mar. 12	Product lines, Model Transformation Testing		
10. Mar. 19	Applications I		
11. Mar. 26	Applications II		
12. April 2	Applications III Project write-ups are due. DISCUSS: April 9?		
April 9	Project presentations (in class). DISCUSS: April 16?		

Next Steps

Send e-mail by Sunday, January 14 to chechik@cs.toronto.edu

`SUBJECT: CSC2125 Paper Selections`

Message body should include

- your name
- your preferred e-mail address
- your research area
- titles of papers from the reading list that you would prefer to present or review. Choose up to 10 papers and prioritize your choices.

Make sure some are from foundations, some from transformations and analysis, and some from applications.

I will try to have paper assignments on-line by January 18.