Software Model Checking

Background on FM

Moonzoo Kim
CS Dept. KAIST
Fall 2006

Syllabus

- Goal of the class
  - To get a concrete knowledge required to research software model checking
- Topics covered
  - Model-oriented approaches
    - Process algebra
      - Concise syntax and clear semantics
    - Rich modeling language
      - Spin as a simplified C language
  - Code-oriented approaches
    - MS SLAM, Berkeley BLAST, NASA Java PathFinder, etc
    - Run-time verification
    - C code generation from a formal design

Syllabus (cont.)

- Grade policy
  - Seminar presentation 50% (2 presentations)
  - Mid term exam 30%
  - Homework 20%
- Time table
  - 1-3 wk: process algebraic approach – CCS
  - 4-6 wk: programming language-like approach – SPIN
  - 7 wk: mid-term exam
  - 8-13 wk: program code-based verification frameworks
  - 14-15 wk: Esterel – WYPWYE framework

Administrative Stuff

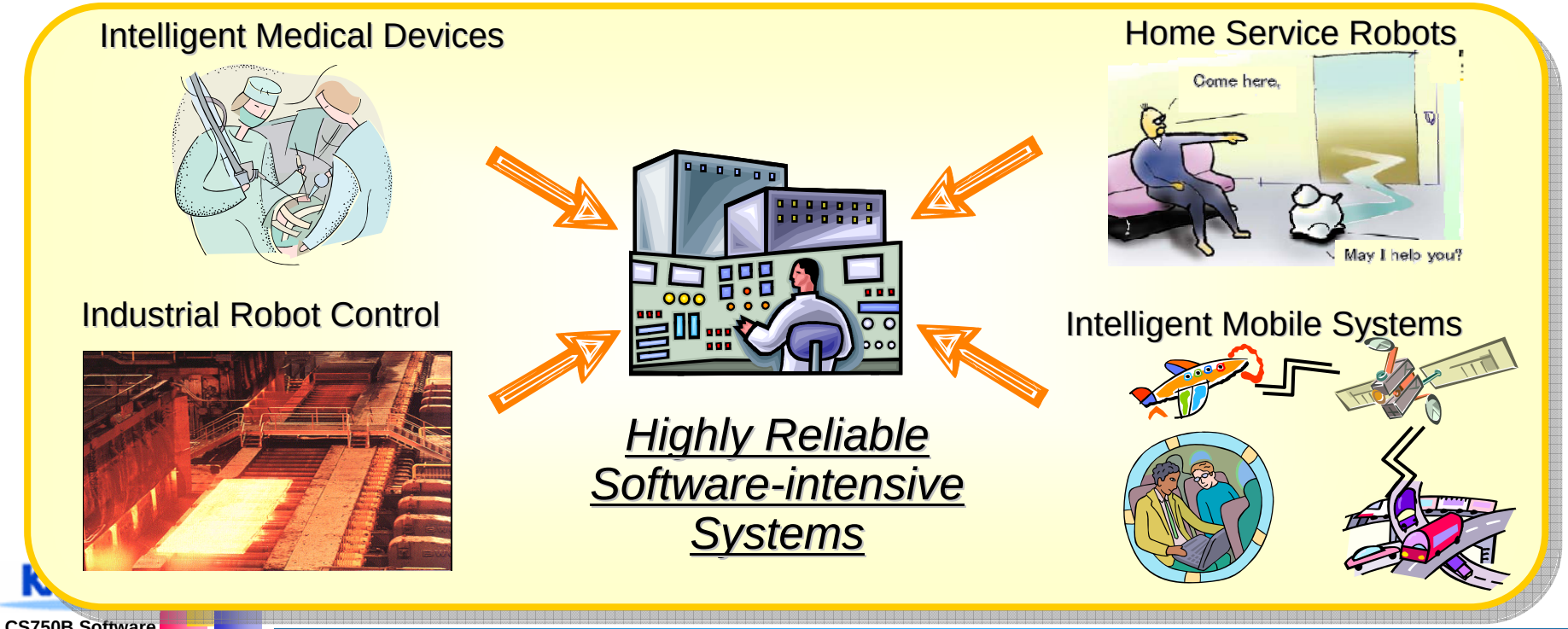
- Instructor
  - Moonzoo Kim
    - moonzoo@cs.kaist.ac.kr
  - Phone #:042-869-3543
  - Office loc: Rm# 2434
- Course home page
  - `http://cs.kaist.ac.kr/~moonzoo/cs750b`
- Class hour: Tues/Thrs 10:30-12:00
- Office hour: Tues/Thrs 1:30 – 3:00
- Note: The official language of this class is English

Outline

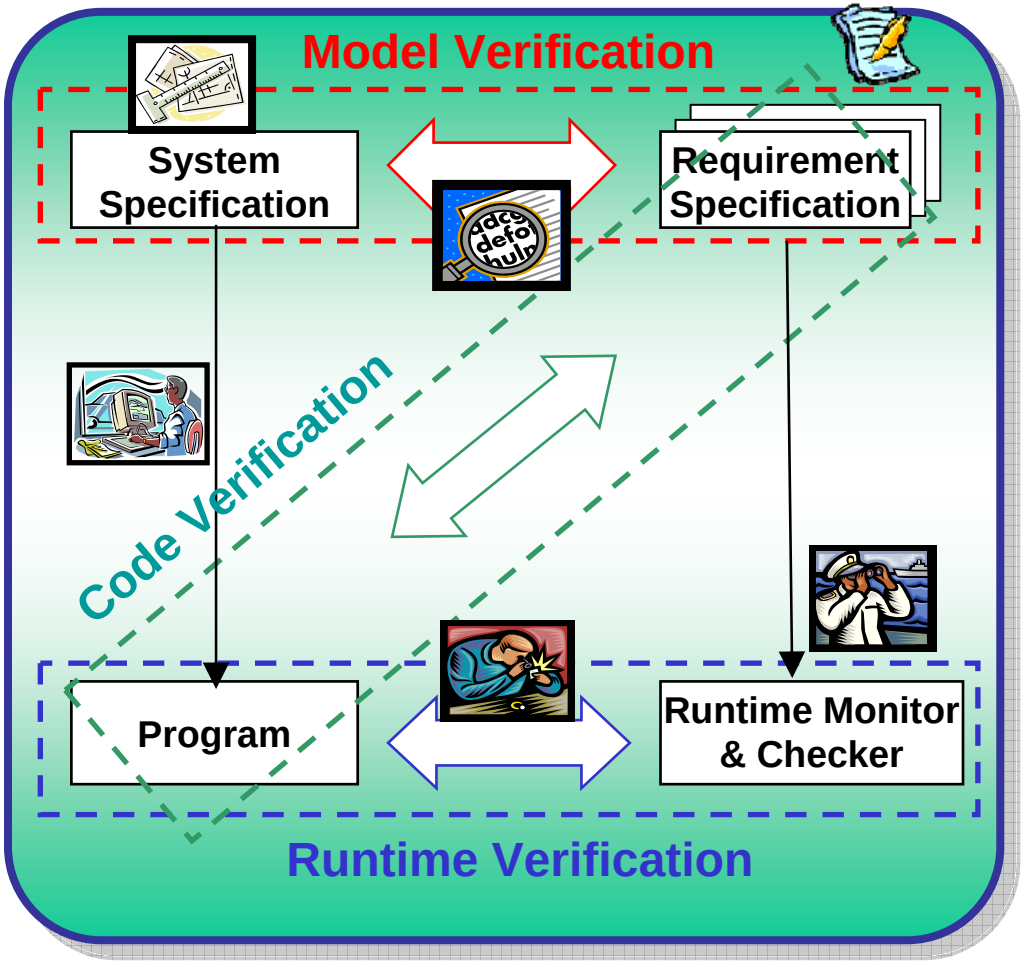
- Research Background
- Safety Critical Systems
- Motivation: Software Crisis
- Formal Methods
- Issues on Formal Methods
- Conclusion

Highly Reliable SW Systems

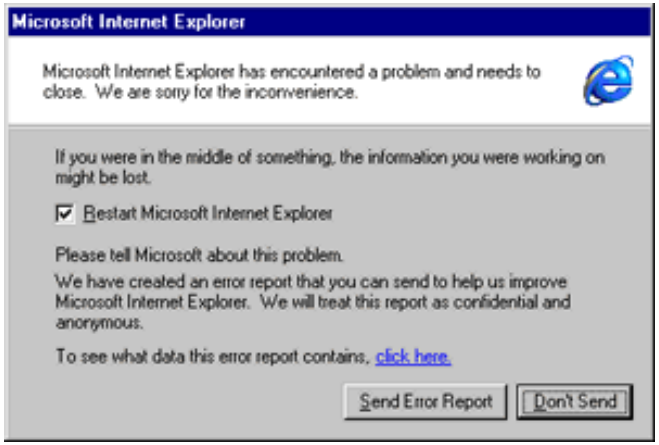
- SW reliability
  - Quality attribute for minimizing malfunctions of systems to reduces damage to human life or valuable properties
- Highly reliable SW technology is a key to the success of industrial products
  - The portion of SW in embedded devices increases continuously

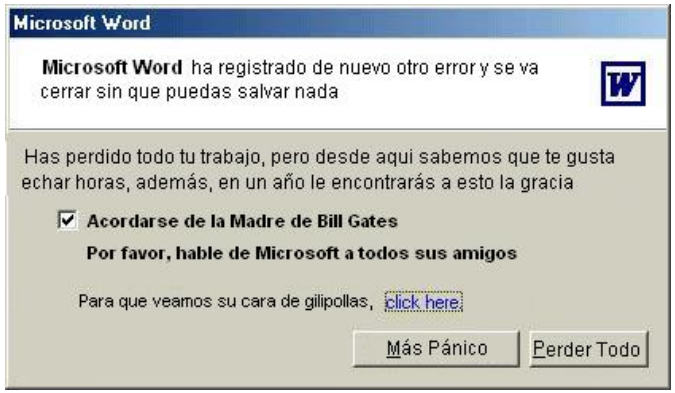


- Unified formal framework of the following three approaches can make synergy
  - Model Verification
    - Targets a system model
    - Req. spec is limited
    - Complete coverage
  - Code verification
    - Targets a real code
    - Extracts an abstract system model from a real code
    - Req. spec is limited
  - Runtime Verification
    - Targets a real code
    - Verifies correctness of current execution run
    - Req. spec can be very expressive



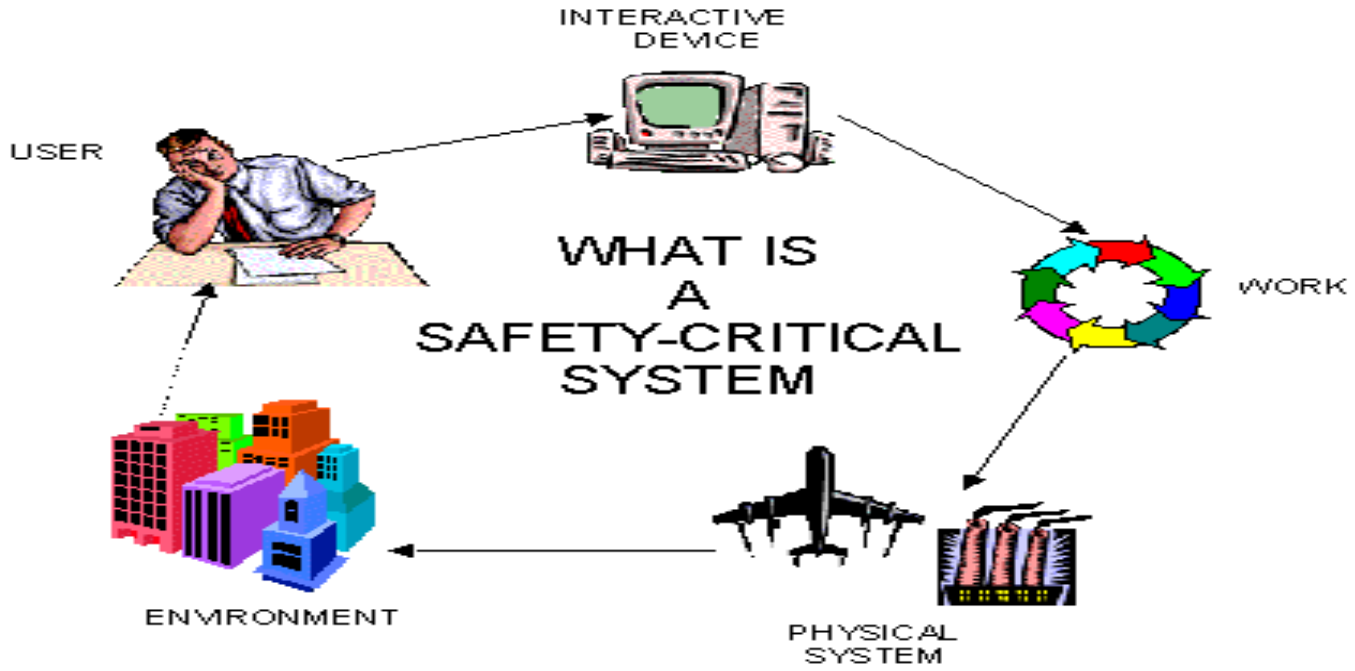
Notorious “Blue Screen”

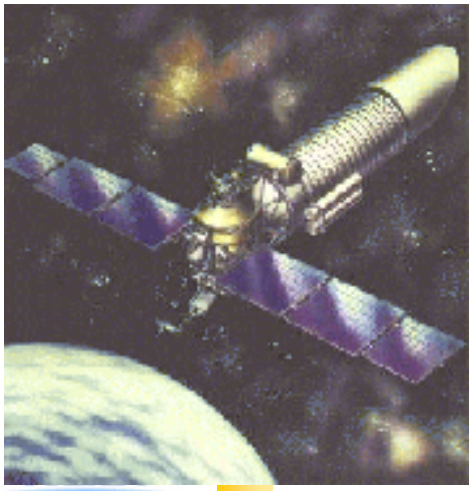


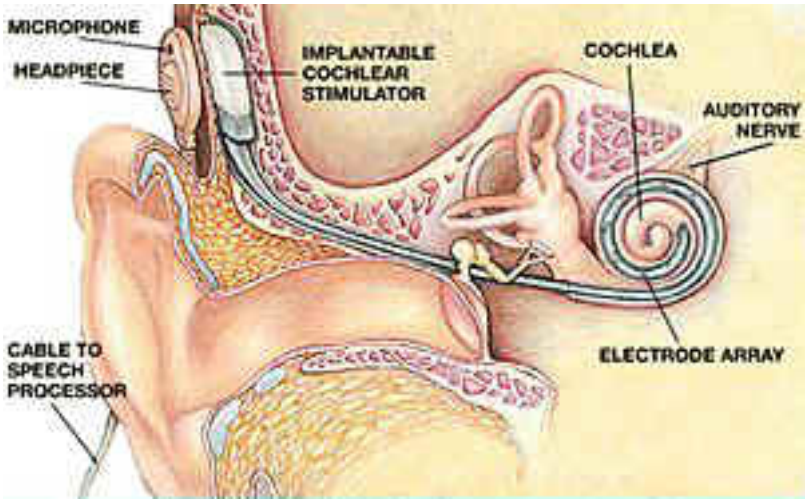


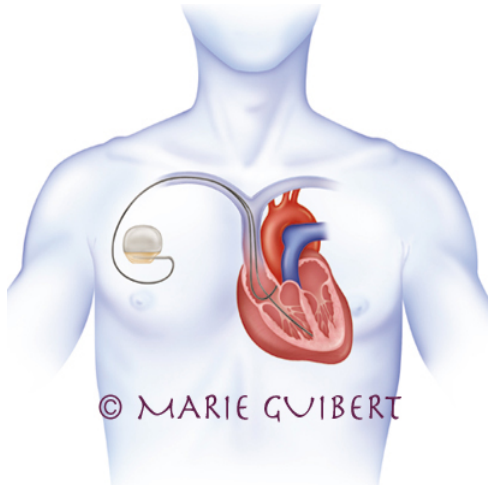

```
*** STOP: 0x0000000A (0x00000000,0x00000002,0x00000000,8038c240)
IRQL_NOT_LESS_OR_EQUAL*** Address 8038c240 has base at 8038c000 - Ntfs.SYS

CPUID:Genuine Intel 6.3.3 irql:1f SYSVER 0xf0000565

Dll Base DateStmp - Name                Dll Base DateStmp - Name
80100000 336546bf - ntoskrnl.exe        80010000 33247f88 - hal.dll
80000100 334d3a53 - atapi.sys           80007000 33248043 - SCSIPORT.SYS
802aa000 33013e6b - epst.mpd            802b5000 336016a2 - Disk.sys
802b9000 336015af - CLASS2.SYS          8038c000 3356d637 - Ntfs.sys
802bd000 33d844be - Siwvid.sys          803e4000 33d84553 - NTice.sys
f9318000 31ec6c8d - Floppy.SYS          f95c9000 31ec6c99 - Null.SYS
f9468000 31ed868b - KSecDD.SYS          f95ca000 335e60cf - Beep.SYS
f9358000 335bc82a - i8042prt.sys        f9474000 3324806f - mouclass.sys
f947c000 31ec6c94 - kbdclass.sys        f95cb000 3373c39d - ctrl2cap.SYS
f9370000 33248011 - VIDEOPORT.SYS       fe9d7000 3370e7b9 - ati.sys
f9490000 31ec6c6d - vga.sys             f93b0000 332480dd - Msfs.SYS
f90f0000 332480d0 - Npfs.SYS            fe957000 3356da41 - NDIS.SYS
a0000000 335157ac - win32k.sys          fe914000 334ea144 - ati.dll
fe0c9000 335bd30e - Fastfat.SYS         fe110000 31ec7c9b - Parport.SYS
fe108000 31ec6c9b - Parallel.SYS        f95b4000 31ec6c9d - ParVdm.SYS
f9050000 332480ab - Serial.SYS

Address  dword dump Build [1314]                                - Name
801afc24 80149905 80149905 ff8e6b8c 80129c2c ff8e6b94 8025c000 - Ntfs.SYS
801afc2c 80129c2c 80129c2c ff8e6b94 00000000 ff8e6b94 80100000 - ntoskrnl.exe
801afc34 801240f2 80124f02 ff8e6df4 ff8e6f60 ff8e6c58 80100000 - ntoskrnl.exe
801afc54 80124f16 80124f16 ff8e6f60 ff8e6c3c 8015ac7e 80100000 - ntoskrnl.exe
801afc64 8015ac7e 8015ac7e ff8e6df4 ff8e6f60 ff8e6c58 80100000 - ntoskrnl.exe
801afc70 80129bda 80129bda 00000000 80088000 80106fc0 80100000 - ntoskrnl.exe

Restart and set the recovery options in the system control panel
or the /CRASHDEBUG system start option. If this message reappears,
contact your system administrator or technical support group.
```

Safety Critical Systems

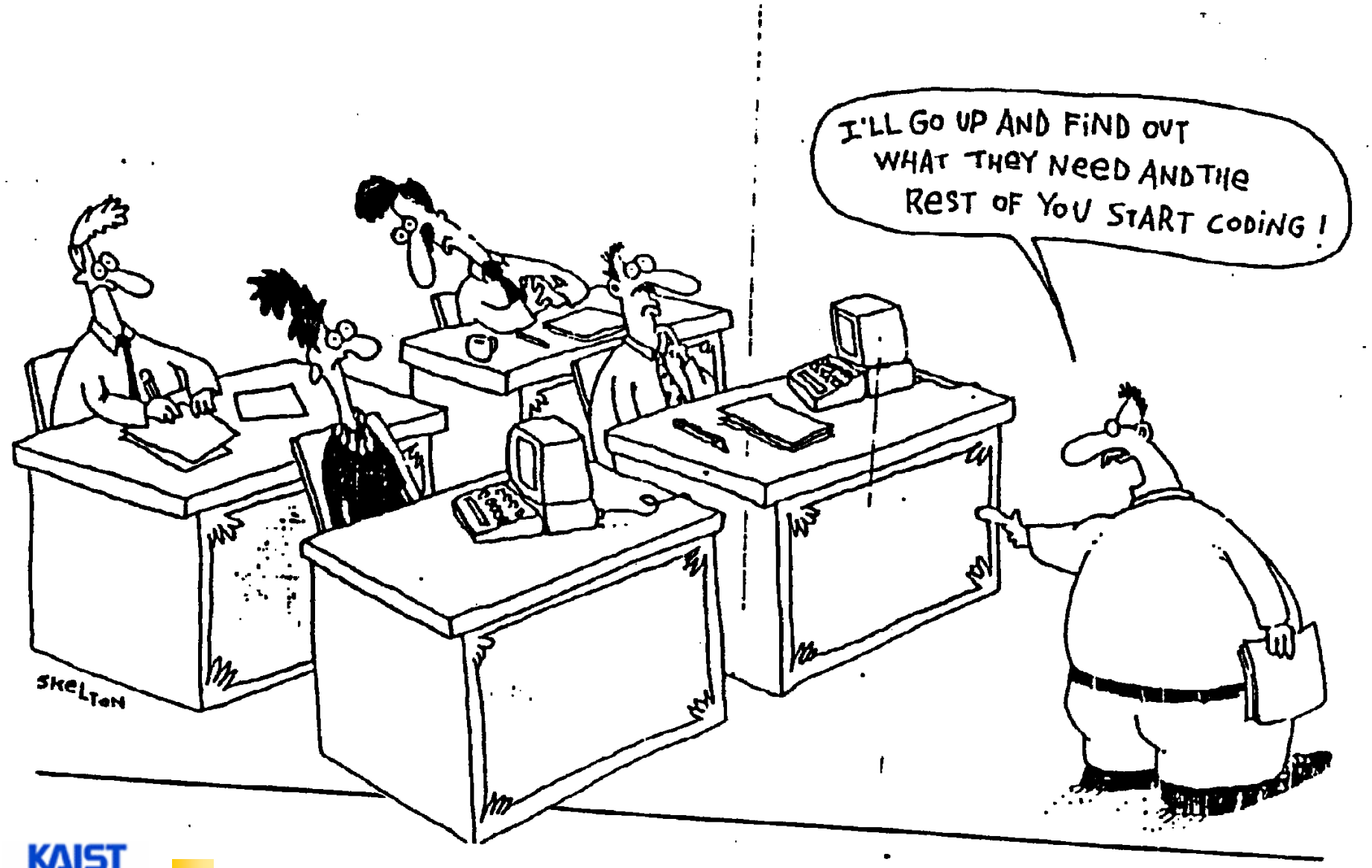




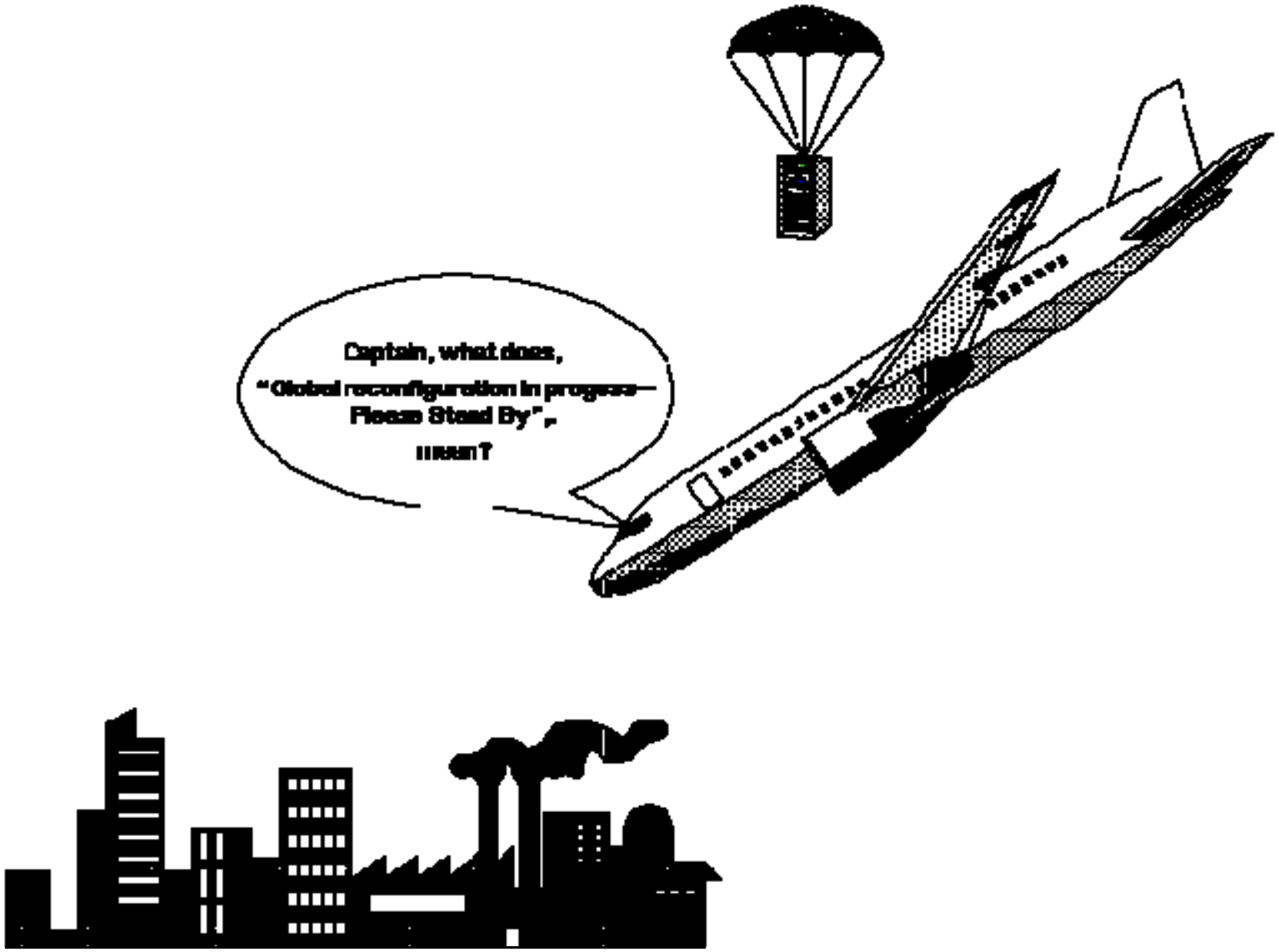




How we deliver SW

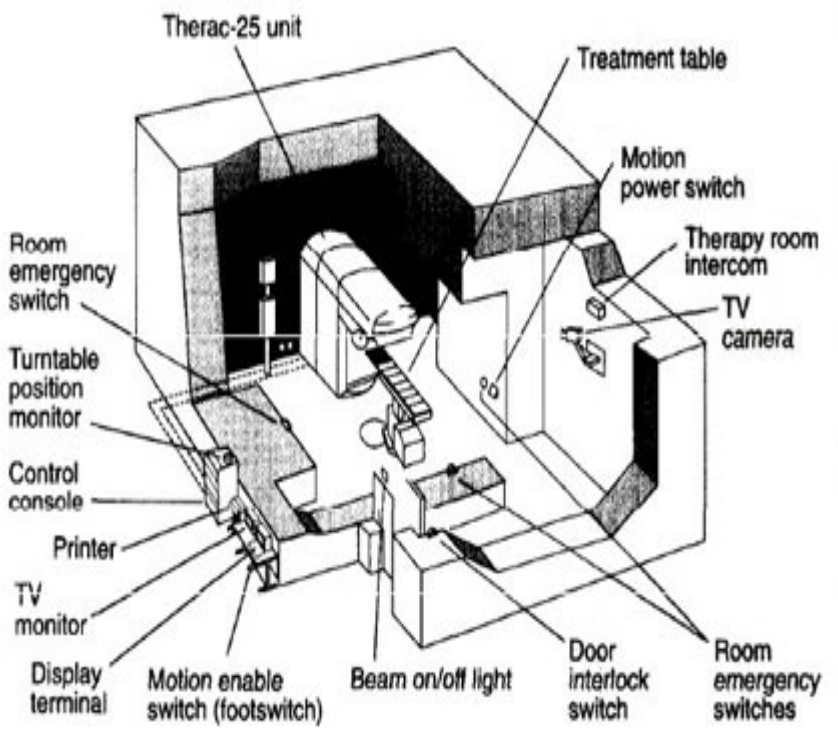


Consequences



Tragic Accidents I

- The Therac-25 Story
  - Between June 1985 and Jan 1987, a computer-controlled radiation therapy machine, called the Therac-25, massively overdosed six people
    - software coding error



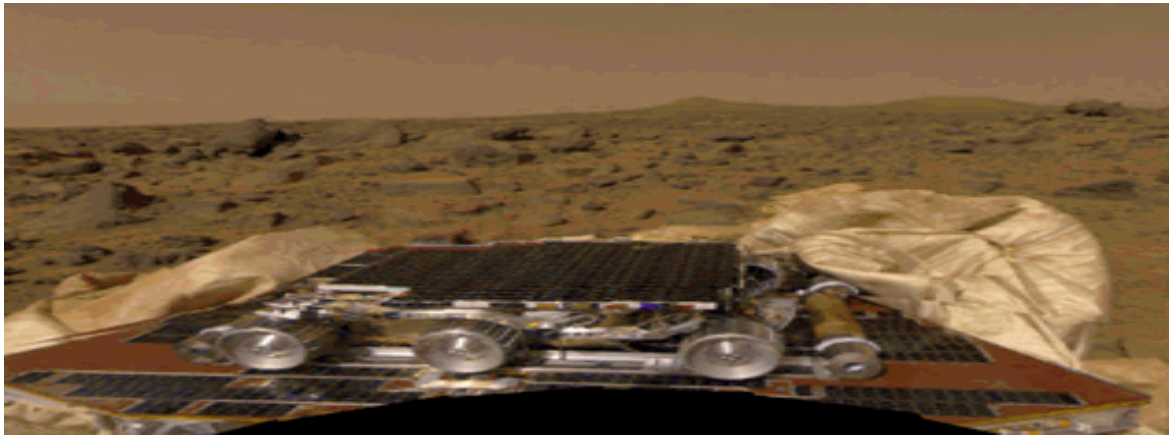
- http://sunnyday.mit.edu/papers/therac.pdf

Tragic Accidents II

- Ariane 5
  - "On 4 June 1996, the maiden flight of the Ariane 5 launcher ended in a failure…The failure of the Ariane 501 was caused by the complete loss of guidance and attitude information …This loss of information was due to specification and design errors in the software of the inertial reference system."
    - Floating number conversion problem
    - http://www.ima.umn.edu/~arnold/disasters/ariane5rep.html

Tragic Accidents III

- NASA Mars Pathfinder (1997)
  - Priority inversion problem led to a system reset and a one-day delay in retransmission of data which wasted valuable mission time.
  - http://www.cis.ksu.edu/~hatcliff/842/Docs/Course-Overview/pathfinder-robotmag.pdf



Software Crisis

Quoted from ``1. Information Technology: Transforming our Society''
President's Information Technology Advisory Committee 1999

"…Furthermore, the Nation needs software that is far more usable, reliable, and powerful than what is being produced today. We have become dangerously dependent on large software systems whose behavior is not well understood and which often fail in unpredicted ways ... We need scientifically sound approaches to software development … "

Quoted from "Science for Global Ubiquitous Computing (GUC)"
A 15 year Grand Challenges for Computing Research
Supported by UK Computing Research Committee 2004

"…unless we offer a mathematically sound methodology to supplant the practice of opportunist software creation there will be consequences of the kind we have illustrated, and a further mass of inscrutable legacy software. These consequences will be greatly more damaging than previously, because the GUC is pervasive, self-modifying and complex in the extreme…"

Hardware v.s. Software

- Flexibility leads to low development cost
  - Minimal costs for HW board manufacturing > 10K$
  - Minimal costs for sofware > 0$
- Growing popularity leads to complex software systems
  - Pentium IV (Willamette): 42 million transistors
  - Windows XP: hundreds million instructions
- **Much harder** to validate/verify (V&V)
  - HW design exploits symmetry, structure, and components
    - Synchronous executions
  - SW design allows maximal degree of freedom/easy construction of programs
    - Asynchronous executions

Formal Methods

- Definition: *(from the Encyclopedia of Software Engineering)*
  - Formal methods used in developing computer systems are **mathematics** based techniques for describing system properties. Such formal methods provide frameworks within which people can specify, develop, and verify systems in a systematic, rather than ad hoc manner.
  - A method is formal if it has a sound mathematical basis, typically given by a **formal specification language**. This basis provides a means of precisely defining notions like consistency and completeness, and more relevant, specification, implementation and correctness.

Example. Mutual Exclusion Algorithm

```
char cnt=0,x=0,y=0,z=0;

void process() {
        char me = _pid +1; /* me is 1 or 2*/
again:
        x = me;
        If (y ==0 || y== me) ;
        else goto again;

        z =me;
        If (x == me) ;
        else goto again;

        y=me;
        If(z==me);
        else goto again;

        /* enter critical section */
        cnt++
        /* assert( cnt ==1); */
        cnt --;
        goto again;
}
```

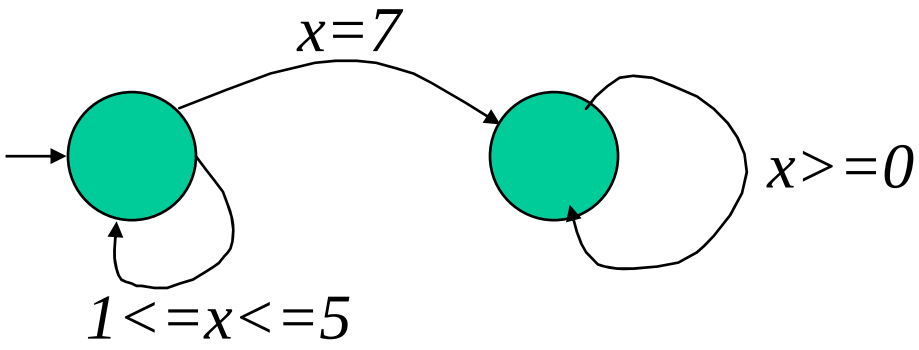
Mutual Exclusion Algorithm

Process 0	***Process 1***
	x = 2
	y==0 || y ==2
	z = 2
	x==2
x = 1	
y==0 || y == 1	
	y=2
	(z==2)
	cnt++
z = 1	
x==1	
y = 1	
z == 1	
cnt++	

Counter Example

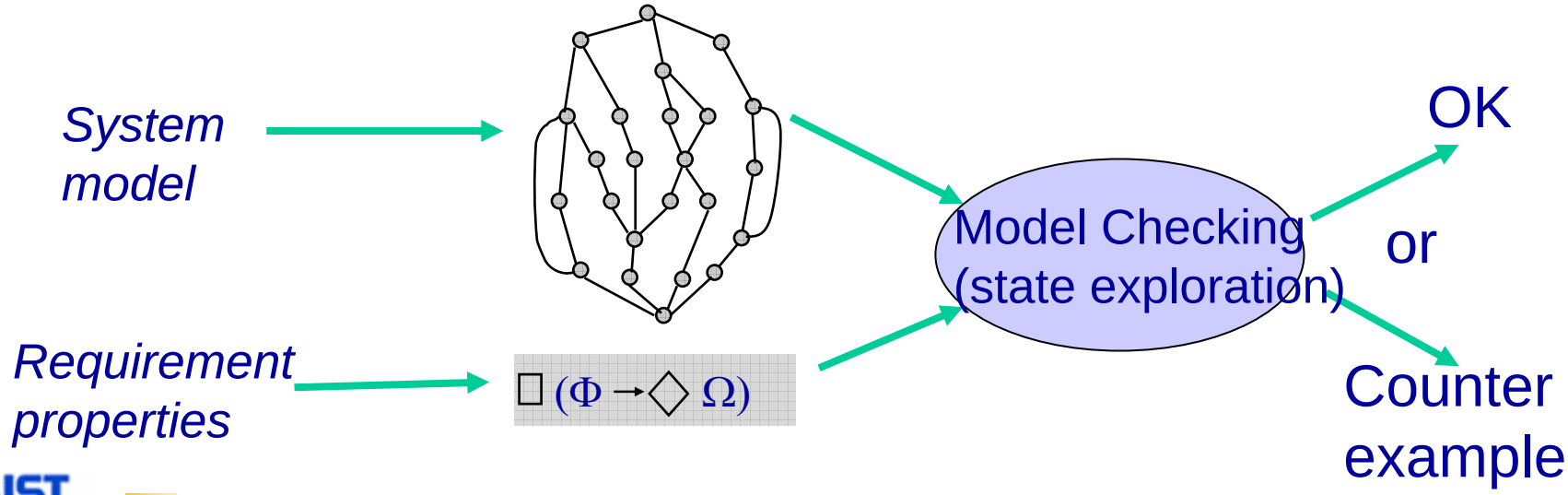
Specification: Informal, textual, visual

- Informal specification
  - The value of x will be between 1 and 5, until some point where it will become 7. In any case it will never be negative.
- Temporal logic specification
  - *((1<=x && x<=5) U x=7) /\ [] x>=0*
- Visual Specification



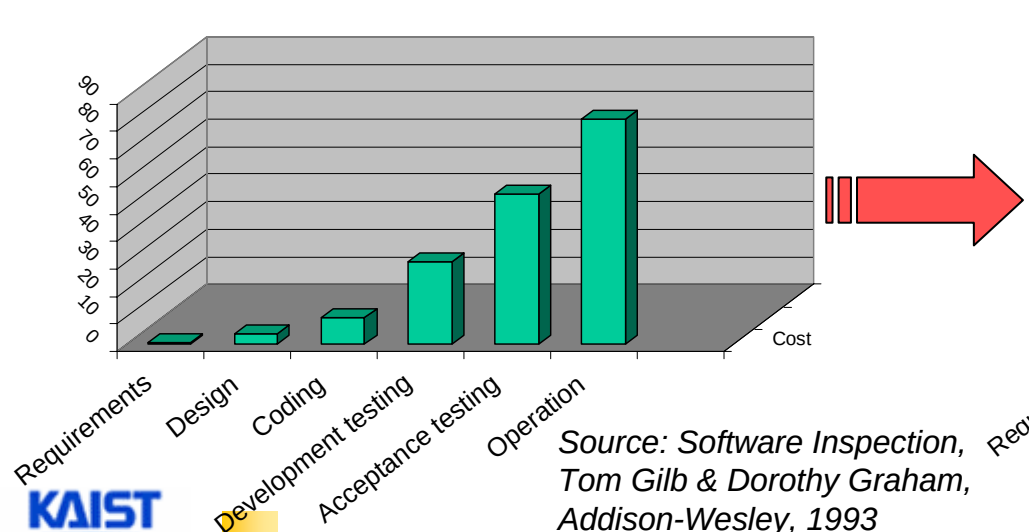
Verification: State Exploration Method

- Model checking
  - Specify requirement properties and build system model
  - Generate possible states from the model and then check whether given requirement properties are satisfied within the state space
    - On-the-fly v.s. generates all
    - Symbolic states v.s. explicit state
    - Model based v.s. code based

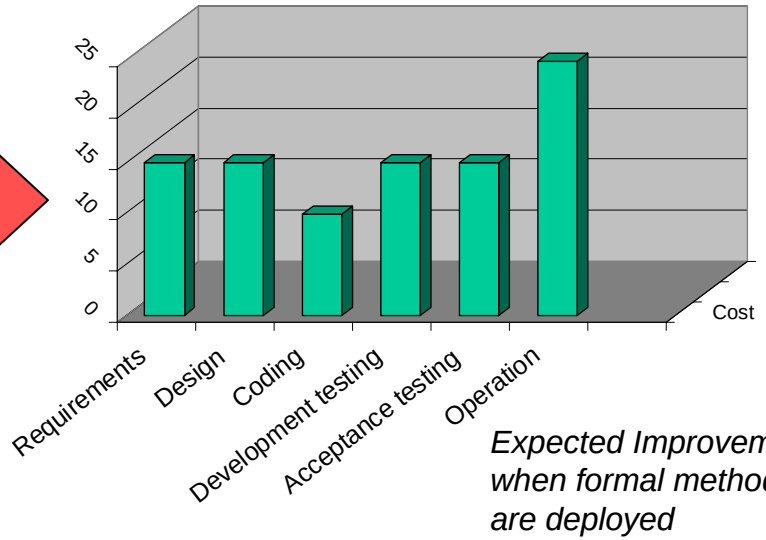


Too Expensive?

- Formal methods *do* involve time consuming training, initially causing longer development time which customers would be unhappy to pay for
- But *in the long run* they generate less costly systems
  - System costs are high in the early life cycle stages, but far lower in the later testing and maintenance stages
  - Problems are discovered early when least damage has been done and least expense has been incurred



Source: Software Inspection, Tom Gilb & Dorothy Graham, Addison-Wesley, 1993

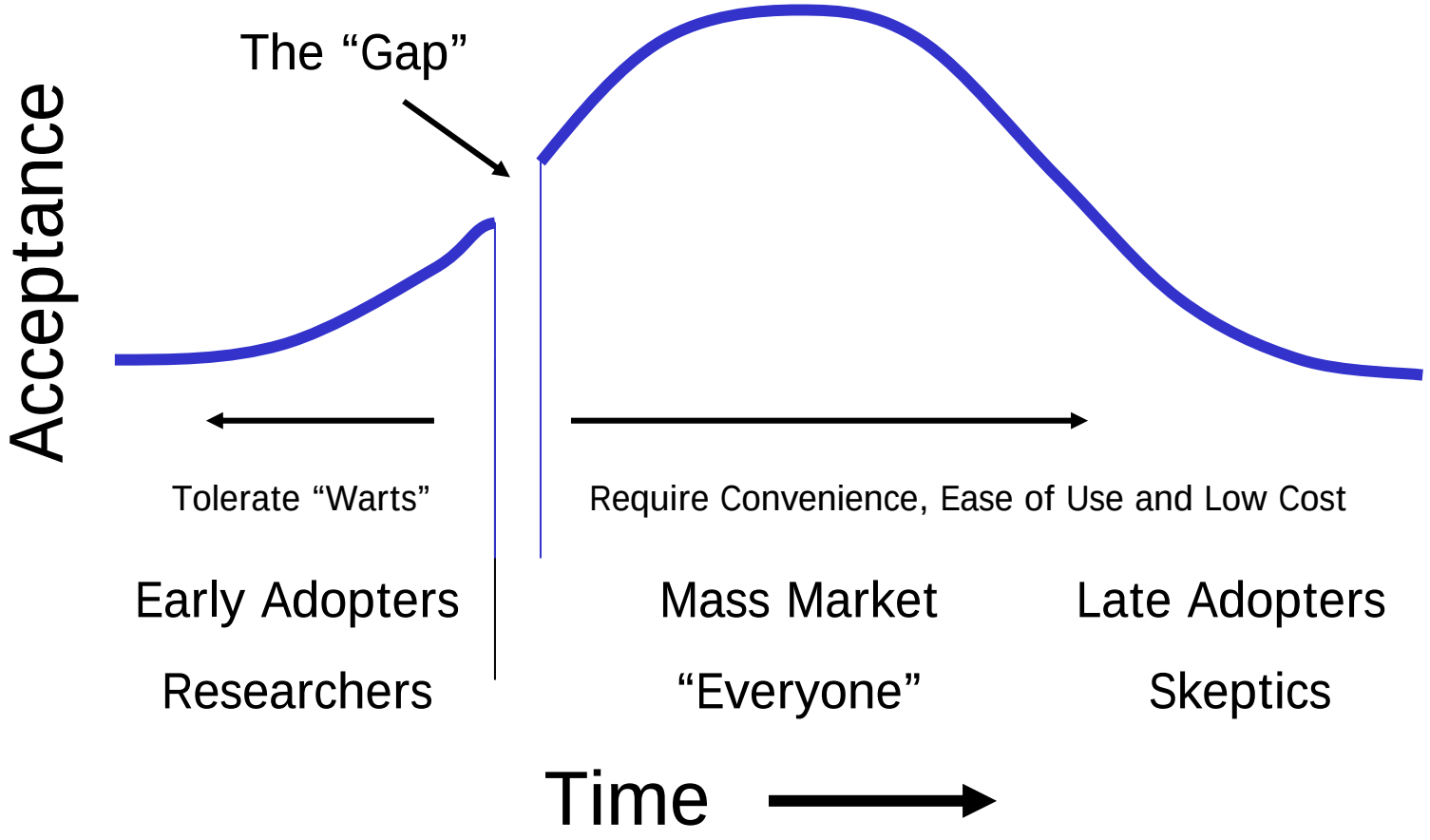


Expected Improvement when formal methods are deployed

Still a Research Area in Ivory Tower?

- It is *not* true that formal methods use complex mathematics, only simple maths is involved, mainly set theory and elementary logic
- Also, tools are becoming more common, e.g. to help develop and understand formal specifications, and to translate them directly into a first attempt at source code
- Opponents claim that formal methods still belong to the world of research
  - They claim that it is not a mature, widely used software engineering technique, that it is confined to ivory towers and not tested in the real world
  - This is untrue. It is (becoming) pragmatic in some areas such as safety critical systems and several hundred relatively large systems have been formally verified.

Adoption



Source: Ken Arnold, HiTech Equipment Corp.

Conclusions

- Software has flexibility as its strong point at the cost of validation/verification difficulty
- Use of formal methods should be encouraged as they produce higher quality software systems
- Formal methods are particularly appropriate for safety critical systems, providing the most effective way of achieving a sufficient level of confidence in the developed software