

The Conundrum of Systems

Dertouzos Lecture



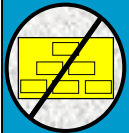
Alfred Z. Spector

Vice President, Services & Software Research

IBM Corporation

aspector@us.ibm.com

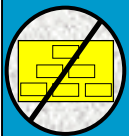
IBM Research Division



Abstract

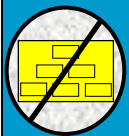
Most software systems are very complicated, by most any metric. As computer scientists, we have delighted in measuring the number of add operations or memory fetches, but the most telling metric would be one that measures the reactions (or pain level) of users and systems administrators, a reaction of frustration, befuddlement, and annoyance. Even programmers often view systems on which they work as ungainly and run amuck. While we computer scientists might like to justify these problems by the newness of our discipline, it is now over 50-years old, and many overly complex systems have been built using the best, widely prescribed techniques of modularity and layering. We try hard to build systems “right,” yet they still turn out too complex.

In this talk, I discuss the need for computer scientists to undertake a new study of complexity, not the one focused on time and space, but rather one that defines (perceived) system complexity (to create, to maintain, and to use), objectively quantifies it, and then seeks to reduce it. I motivate the problem with examples and explain why some well-considered approaches may not have been right. I describe the fascinating challenges in attacking complexity, (1) ranging from agreeing on the meaning of it, (2) learning how to measure it, (3) advancing the science and technology (as, for example, autonomic computing), and perhaps (4) even effecting cultural change within our field. I hope to engender lively discussion that does not end when my lecture is over.

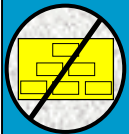


Outline

- Motivation: Complex software systems, built using widely prescribed techniques, often do not delight their constituencies. They are often too complex.
- What is complexity?
- Why are our systems too complex?
- What could Computer Science do to measure and resolve this?
- Should Computer Science undertake this challenge?

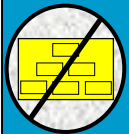


Motivation



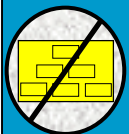
Software Systems Today

- Score high on most metrics:
 - Amount of code
 - # of dependencies
 - # of programmatic interfaces
 - # of layers
 - Administrative interface size & configuration options
 - Non-uniformity
 - Non-orthogonality
 - Defects
 - Documentation
 - # of programmers involved



Anecdotes on Systems

- **Implementation Complexity:**
 - Windows XP Code is tens of millions of lines of code supposedly with circa 10^5 – 10^6 bugs
 - Cisco Routers have support for more than 100 protocols
- **Usage (Administrative) Complexity: Sendmail**
 - Access.db, domaintable.db, local-host-names, mailertable.db, submit.cf, ...
 - More than a page of Features, Defines, etc. in sendmail.cf
 - Longest O'Reilly book, at ~1200 pages
- **Usage (Use) Complexity: New BMW 7 Series**
 - “But why did those Bavarian motor masters have to ruin a wonderful driving machine by filling it with more gadgets and gizmos than you'd find in the cockpit of a space shuttle? The only thing intuitive is how to open the doors,” *Milwaukee Journal Sentinel*, 4/19/2002.



Example: Credit Card Processing

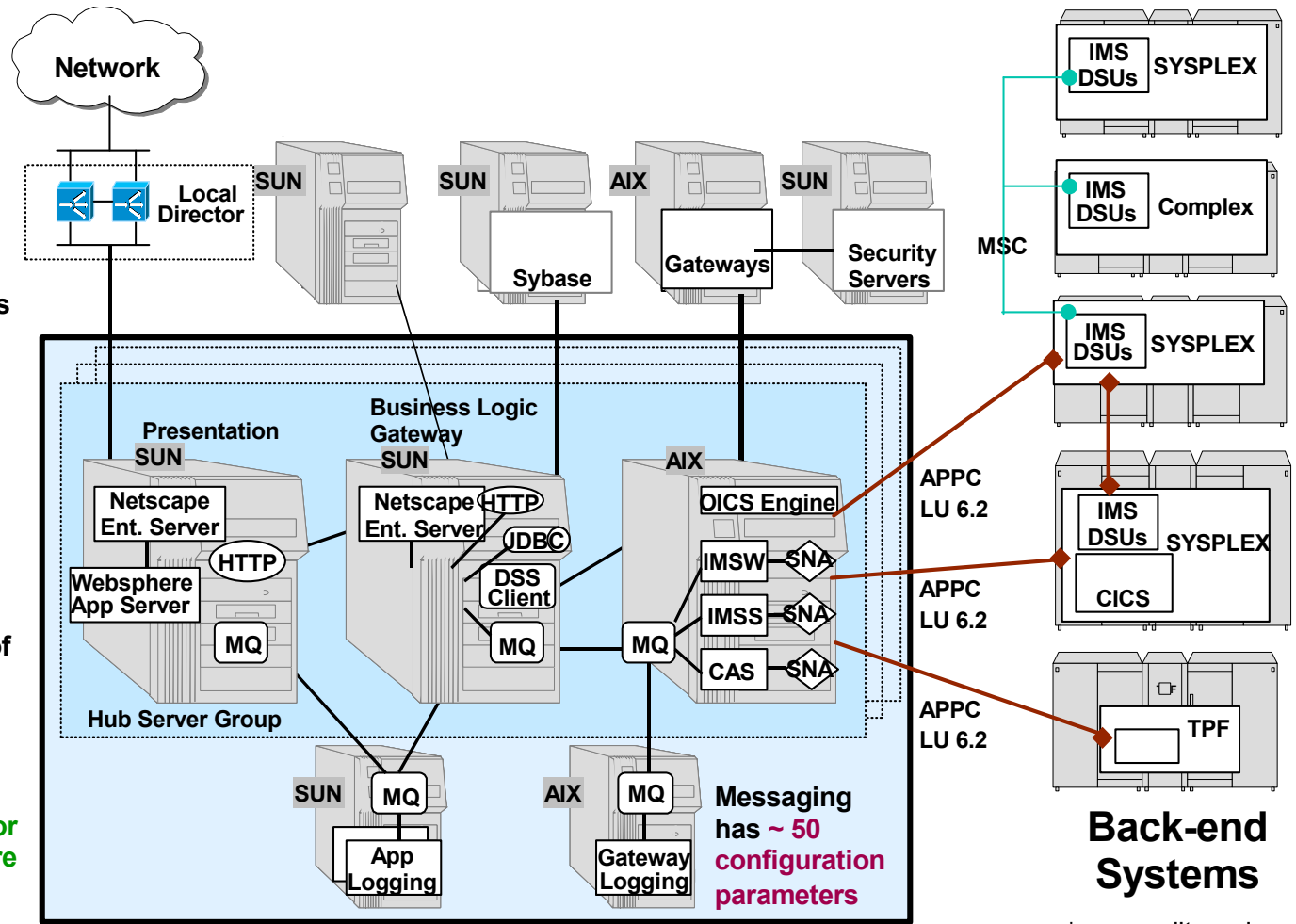
An application server typically supports

- 5 Applications
- 10 EJBs
- Hundreds of servlets
- ~ 100 configuration parameters

A web server typically serves

- Thousands of web artifacts
- ~ 20 configuration parameters

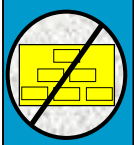
Failure protocols for each component are different: time-out, number of retries, where and what they log, how they fail



Messaging has ~ 50 configuration parameters

Back-end Systems

* my credit card account



RosettaNet Purchase Orders

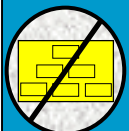
- There are **551** XML fields in the PurchaseOrderRequest
- There are **700** XML fields in the PurchaseOrderConfirmation

Excerpted First lines of purchase order confirmation:

```

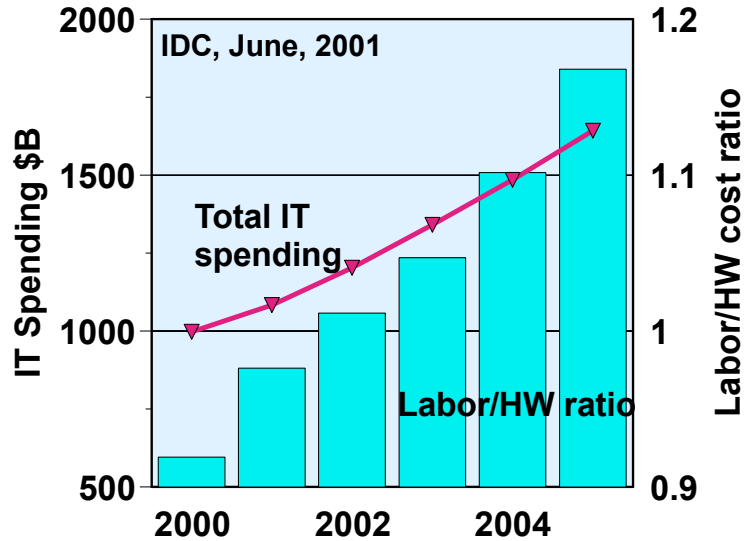
fromRole.PartnerRoleDescription
-- ContactInformation
  -- contactName.FreeFormText
  -- EmailAddress
-- facsimileNumber.CommunicationsNumber
  -- telephoneNumber.CommunicationsNumber
-- GlobalPartnerRoleClassificationCode
-- PartnerDescription
  -- BusinessDescription
    -- GlobalBusinessIdentifier
    -- GlobalSupplyChainCode
  -- GlobalPartnerClassificationCode
GlobalDocumentFunctionCode
PurchaseOrder
  -- AccountDescription
    -- accountName.FreeFormText
    -- AccountNumber
    -- billTo.PartnerDescription
    ...
  -- BusinessDescription
    -- businessName.FreeFormText
    -- GlobalBusinessIdentifier
    -- PartnerBusinessIdentification
      -- ProprietaryBusinessIdentifier
      -- ProprietaryDomainIdentifier
      -- ProprietaryIdentifierAuthority
  -- ContactInformation
    -- contactName.FreeFormText
    -- EmailAddress
    -- facsimileNumber.CommunicationsNumber
  -- PhysicalLocation
    -- GlobalLocationIdentifier
    -- PartnerLocationIdentification
      -- ProprietaryDomainIdentifier
      -- ProprietaryIdentifierAuthority
  ...
  
```

Note: RosettaNet is a consortium of major companies working to create and implement industry-wide, open e-business process standards, that will form a common e-business language, globally aligning processes between supply chain partners. (From RosettaNet Home Page.)



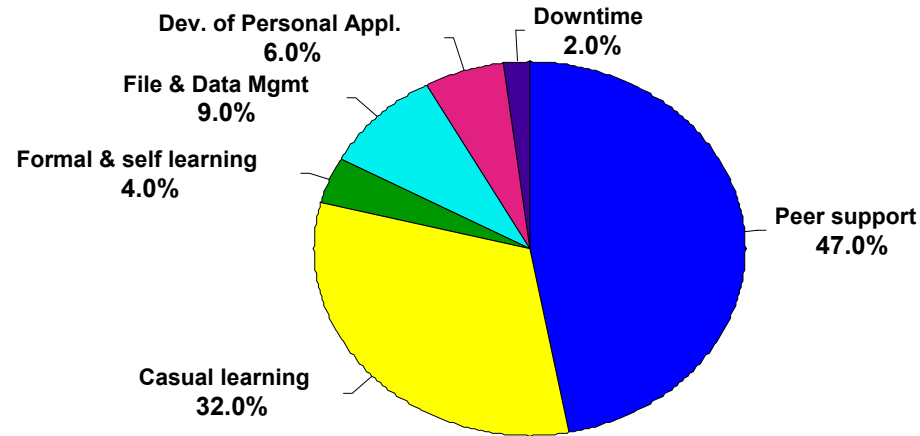
The Cost of Complexity is Large

As hardware costs decrease, labor costs of I/T dominate

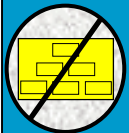


(note: I/T spending data clearly wrong)

In storage, for example, labor costs already approach 3 times cost of hardware

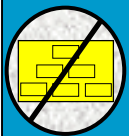


Indirect costs of PCs may soon approach 60% of total cost



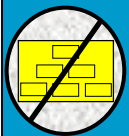
Personal Experiences

- XP:
 - I am uncertain if the ACLs on my home system are right
 - I would guess this is true of many commercial systems, too
 - Many XP applications don't work with security turned on
- Adventures with Linux-based email:
 - 6 rules engines to configure (And, I missed one!)
- Home automation
 - Nice idea, but won't the water leak detector shut-off water during a fire?
 - I've assigned circa 100 X-10 network addresses
- There are more than 150 (!) icons on my screen as I write this talk, with considerably varying behaviors
- The number of requests for PC personal administrative assistance are mind-boggling

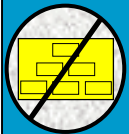


What if The Home PC Were Truly Important

- If the Home PC were “mission-critical”, customers would want:
 - No data loss
 - High availability
 - Freedom from security problems
 - etc.
- What percentage of the population could possibly configure such a system?
- What percentage could afford the time to configure & manage?



Definition



3 Categories of Complexity

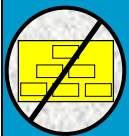
■ Classic Complexity ■ Usage Complexity

- Time
- Space

■ Implementation Complexity

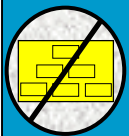
- Logical
- Structural
- Comprehensibility

Task	Pre-Use	Novice	Middle	Expert	Exception
Install					
Configure					
Administer					
Use					



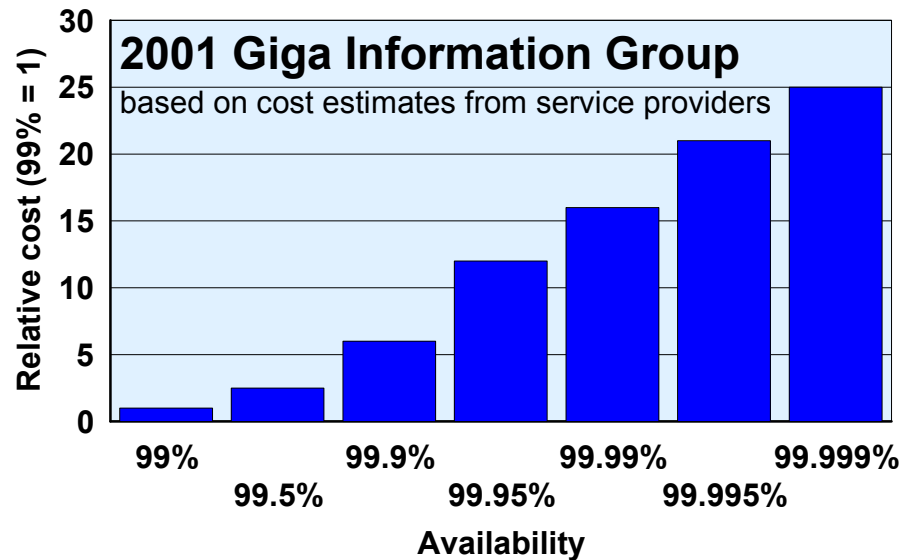
Usage Complexity Must Be Focus

- Consider a humble table (that is, legs and a horizontal surface)
 - *Classic Complexity* is not relevant as defined but there may be parallels
 - *Implementation Complexity* very high
 - Physicists do not fully understand tables, I suspect
 - *Usage Complexity* very low
- While Classic & Implementation Complexity may impact Usage Complexity, they are less important end goals. (In effect, they are tools of Computer Science.)

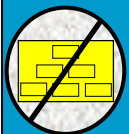


Dual Impact of Reliability

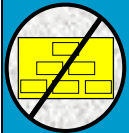
- Reliability requirements often add to complexity, if cost is a surrogate



- When systems are not reliable, anomalous failures drive us crazy



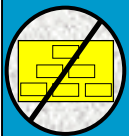
Wherefore Complexity



Causes of Complexity: The Problem Itself

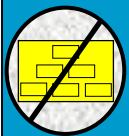
- We aim to automate:
 - Complex problems
 - Ill-defined problems whose space changes over time & where we don't understand fully the primary endpoint
 - Problems requiring human & extra-system adaptation
- Contrast to (structural) bridges¹:
 - Requirements straightforward and derivable from AASHTO specifications

¹Bridge Design and Construction, *CACM* 29(4): 267-283, 4/86, A. Z. Spector and D.K. Gifford



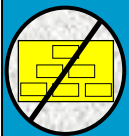
Causes of Complexity: Design Methodology

- 2 related tenets of software methodology
 - Generalization, Encapsulation, Re-use
 - Complex things get more complex if they support re-use
 - E.g., a commercial product is 10x cost of a one-of solution
 - Integration of Components
 - (Don't build new, assemble and connect)
 - This exposes many more interfaces
 - Boundaries are imperfect and “don't connect”
 - Internal interfaces rarely recede into the woodwork
 - Component success may be based on context of use
 - E.g., “Business Process Integration” as a market category
 - These may contribute much to complexity, but it's not clear what choice there is



Causes of Complexity: Taking Time

- Most students and most programmers want to start coding without adequate thought
 - Perhaps, due to a lack of confidence
 - Perhaps, due to the need to prove progress
 - Perhaps, due to the way they are trained or financially motivated
- Time to market considerations of industry
 - It's better to be first than ...
- The “Web Year” deemed a bad thing

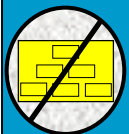


Causes of Complexity: We

- Most computer scientists love complexity
 - We are proud of mastering it.
 - Criteria for my CMU 15-212 final assignments was maximal tenable complexity (I note, 20 years ago.)
 - And, we (!) design, and program systems
- We push what we invent without regard to the goal or primary endpoint
- *We Seem to Misinterpret Complexity as Sophistication*
- Where did ITS, the Incompatible Time Sharing System Originate?

We are the fox guarding the henhouse

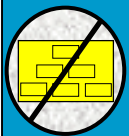
We have met the enemy and he is us



Causes of Complexity:

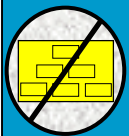
Market Forces & Standards

- Market forces tend to promote “creeping featurism”:
 - Technical reviews & industry analysts do this, partially motivated by a need to have objective comparison metrics
 - Early adopters may revel in complexity
- With the desire to abstract and create reusable abstractions, standards groups tend towards standards that are unions



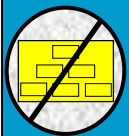
Silicon & Magnetic Technology Scaling

- A boon:
 - Makes everything possible
- A curse:
 - Bloated code?
 - A new layer every every few years
 - Reduced motivation to discard the old
 - Perhaps, poor performance may lead to
- Perhaps, we would benefit from an end to easy scaling



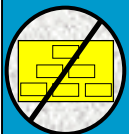
No Clear Idea of Fitness to Task

- I think we are conflicted as a profession
- Perhaps the 150 icons on my screen are a good thing?
 - Perhaps, a “real hammer” is as complex
 - Our friends / colleagues are complex, so should be our computers
 - Good things take time to master
- On the other hand
 - Why should it take days to make computers to have applications perform takes we think we intuitively understand?

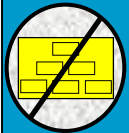


Old Code Does Not Ride Off Into The Sunset

- Automobiles:
 - It took close to 50-years to complete the transition to fuel injectors.
 - I'm aware of no cars that had both carburetors & fuel injectors simultaneously (although there were hybrid devices.)
- But with software, we'd keep both
 - There would be minimal extra expense
 - Except for the user interface & extra control logic
 - Justification:
 - We'd argue there were benefits to both
 - And then, there would be the widget that was dependent on the old interface
 - Admittedly, distributed systems/networks represent a particular dilemma

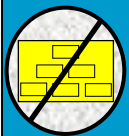


What Could We Do?



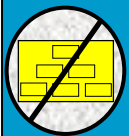
There are steps to take

- Meaning
- Measuring
- Methodology
- System Architecture
- Science and Technology
- Acknowledgment, Legal & Cultural Change



Meaning

- Computer scientists instantly know about time and space bounds
- It is just as important to arrive at clear **definitions** of all forms of complexity
- There has been work in this area, but we are likely to arrive at something like:
 - Classic Complexity
 - Implementation Complexity
 - Usage Complexity
- I note this topic is a very small part of the CS curriculum today



The Unmeasured Life Is Not Worth Leading

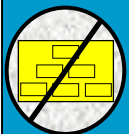
- If we can reach some definitions, we should try to create metrics

- Minimization or Maximization adds focus and fun
- Where metrics have existed in the field
 - Latency/Throughput
 - Word accuracy
 - Recall & precision
 - Translation quality

More progress has been made

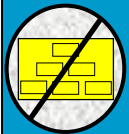
- There are risks to measuring things (you get what you measure)

- I think metrics could be the strongest weapon against complexity



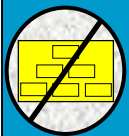
Methodology

- User-centered methods
- Ethnography
- Product Lines
- Increased use of metrics
- Component-based



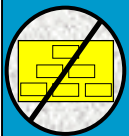
Focus on The Right Function

- What do our user communities really want?
 - Can we more directly provide exactly that and dispense with distracting and wasteful items
 - Can we focus on the breadth of the problem and provide a solution to it, perhaps with incrementally more function
 - Perhaps, either directed or automatic adaptation to usage community required



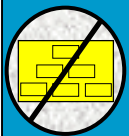
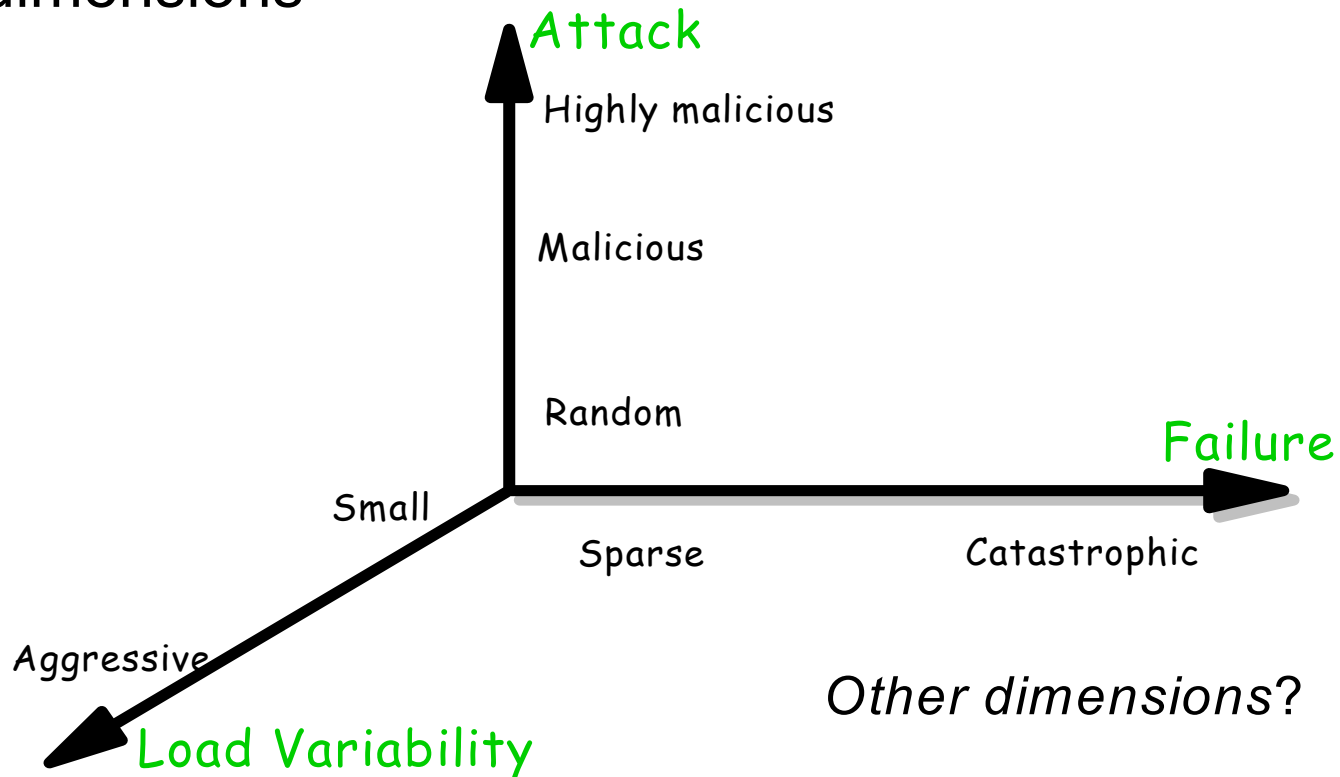
System Architecture

- We need to have higher standards
- Example:
 - In systems today, we have disks, partitions, volumes, logical volumes, file systems, and directories structures
 - Do all of these still need to be visible interfaces?
 - Why not have a configuration option to set MBTF to Low, Medium, or High?
 - With ACLs, there could be far more useful profiles established
- What about increased use of classic AI techniques?



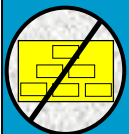
Science and Technology

- Autonomic computing concept: Making systems robust in the presence of stimuli occurring in different dimensions



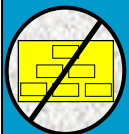
Autonomic Computing

- Subsystem design improved to eliminate manual control
- Core techniques:
 - Control theory
 - Increased use of rules systems; perhaps, with inference & common sense
 - Negotiation
- Standardization of event reporting to provide opportunities for data mining, statistical machine learning, and more feedback control
- Architecture

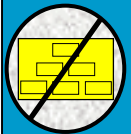


Acknowledgment, Legal, & Cultural Change

- We should take public responsibility
- We should increase university and research focus
 - Education curriculum
 - Research agenda
 - Opportunity to broaden university collaborations
- We need to debate role of legal system
 - As we ever-more depend on computers, how do customers/society evaluate risk?
- Systems builders need to return more to artistry

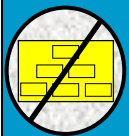


Should We Do This?



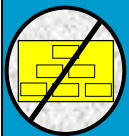
Should We Attack Complexity?

- Is the problem important
- Is the problem tractable and scientifically viable?
- Is the problem fun?



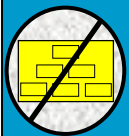
Economic Importance: PC's & Minivans

- Minivans
 - Human Factors
- PCs
 - Mhz, expansion slots, AGP4
- I believe we are quickly moving towards the minivan. Without the right human factors, we may not sell.



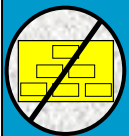
Tractability & Viability

- Work in this area has been done by relatively isolated sub-communities
 - Much new work feasible
 - Also, existing work may have rapid impact integrated into the mainstream
- Viability:
 - Synthetic work feasible
 - Analytic work feasible
 - Experimental work feasible



The Conundrum of Systems

- Complexity grows despite all we have done in computer science, from Simon's *Sciences of the Artificial* to modern programming languages & software engineering techniques
- There is valuable, rewarding, and concrete work for Computer Science in combating complexity:
 - Meaning
 - Measuring
 - Methodology
 - System Architecture
 - Science and Technology
 - Acknowledgment, Legal & Cultural Change
- This area of work could prove as valuable as direct functional innovation: It requires focus



Thanks for listening.

