15-712:
Advanced Operating Systems & Distributed Systems

A Few Classics

Prof. Phillip Gibbons

Spring 2023, Lecture 2

CS is a Fast Moving Field:
Why Read/Discuss Old Papers?

"Those who cannot remember the past are condemned to repeat it."

- **George Santayana, The Life of Reason, Volume 1, 1905**

See what breakthrough research ideas look like when first presented

Today's Papers

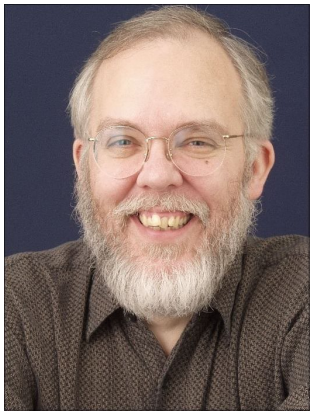
"End-to-End Arguments in System Design"
Jerome Saltzer, David Reed, David Clark 1984

"Hints for Computer System Design"
Butler Lampson 1983

"End-to-End Arguments in System Design"
Jerome Saltzer, David Reed, David Clark 1984

- Jerry Saltzer was a team leader for Multics, advisor of Reed & Clark

- David Reed designed UDP, multiversion concurrency control



- David Clark was chief protocol architect in the development of the Internet (1981-1989)

Q: What award have David Clark, Dennis Ritchie, Ken Thompson, and Richard Hamming all won?

IEEE Richard Hamming Medal

"End-to-End Arguments in System Design"
Jerome Saltzer, David Reed, David Clark 1984

SigOps Hall of Fame citation (2007):

This paper gave system designers, and especially Internet designers, an elegant framework for making sound decisions. A paper that launched a revolution and, ultimately, a religion.

"Choosing the proper boundaries between functions is perhaps the primary activity of the computer system designer."

The End-to-End Argument

- **The function in question can completely & correctly be implemented ONLY with the knowledge and help of the application standing at the endpoints of the communication system.**

- **Therefore, providing that questioned function as a feature of the communication system itself is not possible.**

(Sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement.)

Application of End-To-End Argument

- **Careful file transfer**
  - Enforce desired reliability guarantees only at end points (but each of the steps must be sufficiently reliable)

- **Other reasons for end-to-end vs. low-level?**
  - Other apps using the low-level may not need the same checks
  - Low-level may have too little information to do checks well

Applications of End-To-End Argument

- **Careful file transfer**
- **Delivery guarantees**
- **Secure transmission of data**
- **Duplicate message suppression**
  - Duplicates may be caused by the end-point application (not suppressed since distinct messages)
- **Guaranteeing FIFO message delivery**
- **Transaction management**

Challenges of End-to-End

- **How to identify the "ends"?**
  - Real-time phone conversation vs. leaving voice mail
  - Your application may be "internal" to another application
- **What to still include at the low level?**
  - Sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement.
- **Tension with clean layering, clean APIs**

“Hints for Computer System Design”
Butler Lampson 1983

- Invented personal computer
- Turing Award
- John von Neumann Medal
- NAE, NAS, AAAS Fellow

Xerox Alto

SigOps HoF citation (2005):

A classic study of experience building large systems, distilled into a cookbook of wisdom for the operating systems researcher. As time has passed, the value of these hints has only grown and the range of systems to which they apply enlarged..

"Hints for Computer System Design"
Butler Lampson 1983

- **Designing a computer system is different from designing an algorithm:**
  - The external interface is less precisely defined, more complex, more subject to change
  - The system has much more internal structure, and hence many internal interfaces
  - Measure of success is much less clear

Defining Interfaces

- **Defining interfaces is the most important part of system design**
  - Each interface is a small programming language
  - Conflicting goals:

simple, complete, admit a sufficiently small/fast implementation

Summary of the Slogans

Why?	*Functionality* Does it work?	*Speed* Is it fast enough?	*Fault-tolerance* Does it keep working?
Where?			
Completeness	Separate normal and worst case	Shed load End-to-end Safety first	End-to-end
Interface	Do one thing well: Don't generalize Get it right Don't hide power Use procedure arguments Leave it to the client Keep basic interfaces stable Keep a place to stand	Make it fast Split resources Static analysis Dynamic translation	End-to-end Log updates Make actions atomic
Implementation	Plan to throw one away Keep secrets Use a good idea again Divide and conquer	Cache answers Use hints Use brute force Compute in background Batch processing	Make actions atomic Use hints

Discussion: Summary Question #1

- **State the 3 most important things the paper says.** These could be some combination of their motivations, observations, interesting parts of the design, or clever parts of their implementation.

Functionality: Keep it Simple

"Perfection is reached not when there is no longer anything to add, but when there is no longer anything to take away." – Saint-Exupery

"Everything should be made as simple as possible, but no simpler." – Einstein

Do one thing well

- Don't generalize
- Get it right
- Don't hide power
- Use procedure arguments (to provide flexibility)
- Leave it to the client

Functionality: Keep it Simple

- **Service must have fairly predictable cost.**
- **Interface must not promise more than the implementer knows how to deliver.**
- **Bad choice for interface can lead to inefficiencies.**
  - O(n^2) algorithm for FindNamedField
- **Clients should only pay for power they want (e.g., RISC vs. CISC).**
- **Purpose of abstraction is to hide undesirable properties; desirable ones should not be hidden.**

Functionality: Continuity

- **Keep basic interfaces stable**
- **Keep a place to stand**
  - E.g., backward compatibility package: old interface on new system

Making Implementations Work

"Perfection must be reached by degrees; she requires the slow hand of time." - Voltaire

- **Plan to throw one away**
- **Keep secrets**
  - ability to improve each part separately
- **Divide and conquer**
- **Use a good idea again**
  - instead of generalizing it

Handling All the Cases

"One crash a week is usually a cheap price to pay for 20% better performance."

- **Make normal case fast**
- **Make worse case ensure progress**
  - E.g., reserve resources to free one item

Discussion: Summary Question #2

- **Describe the paper's single most glaring deficiency.** Every paper has some fault. Perhaps an experiment was poorly designed or the main idea had a narrow scope or applicability.

Speed: Interface

- **Split resources in a fixed way**
  - Dedicated is faster than Shared
- **Use static analysis**
  - Discover properties of program that can improve performance
- **Dynamic translation from convenient to fast**

Speed: Implementation

- Cache answers
- Use hints (may be wrong)
- When in doubt, use brute force
- Compute in background
- Batch processing

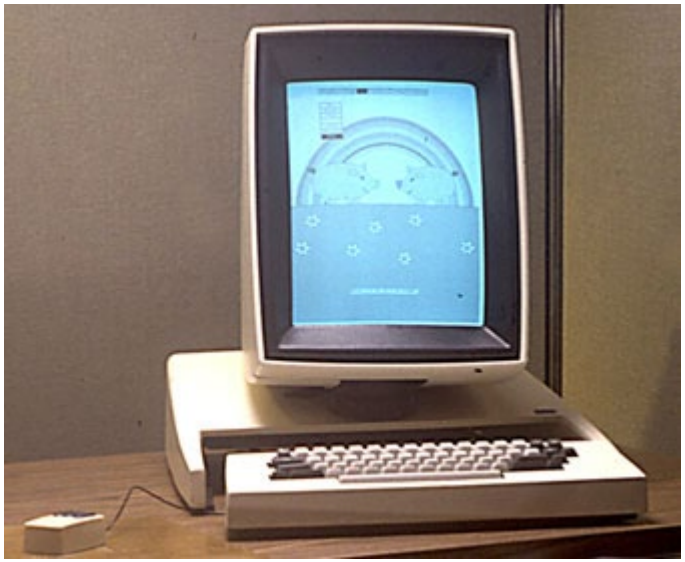
Electronic mail can be delivered and retrieved by background processes, since delivery within an hour or two is usually acceptable.

Speed: Completeness

"Be wary then; best safety lies in fear."

"The nicest thing about the Alto is that it doesn't run faster at night." – Jim Morris (CMU)

- **Safety first**
  - Strive to avoid disaster
  - Paging systems avoid thrashing
- **Shed load**
  - Red button for unhappy users (but allowed to shed the user)



Fault-tolerance

- **"The unavoidable price of reliability is simplicity." - Hoare**
- **End-to-end**
  - For reliability
  - Can add intermediate checks for performance reasons, error codes for visibility reasons
  - Problems: Need cheap test for success. Can mask severe performance defects until operational at scale.
- **Log updates to record the truth about the state of an object**
- **Make actions atomic, or restartable (idempotent)**

Discussion: Summary Question #3

- **Describe what conclusion you draw from the paper as to how to build systems in the future.** Most of the assigned papers are significant to the systems community and have had some lasting impact on the area.

Monday's Paper

"Implementing Remote Procedure Calls"
Andrew Birrell & Bruce Nelson 1984

Guest Lecturer: Val Choung

FURTHER READING

“The UNIX Time-Sharing System”
Dennis Ritchie & Ken Thompson 1974

Q: What award have Lampson, Hamming, Ritchie, and Thompson all won?

ACM Turing Award

SigOps HoF citation (2005):

At a time when operating systems were trending towards complexity, UNIX emerged as a hallmark of elegance and simplicity.

“The UNIX Time-Sharing System”
Dennis Ritchie & Ken Thompson 1974

Key Features of UNIX (according to authors):

- Hierarchical file system incorporating demountable volumes
- Compatible file, device, inter-process I/O
- Ability to initiate asynchronous processes
- System command language selectable on a per-user basis
- Over 100 subsystems including a dozen languages

UNIX

- Only 50K bytes (Linux is 4-8 GBs, Windows 10 is 2+ GB)
- "User-visible locks for the file system are neither necessary nor sufficient"
- i-list system table, indexed by a file's i-number; i-node contains attributes of the file – a unique feature of UNIX

"The success of UNIX is largely due to the fact that it was not designed to meet any predefined objectives."

Our goals:
Building a comfortable relationship with the machine
Exploring ideas and inventions in operating systems

What Influenced the Design

- **Make it easy to write, test, and run programs**
  - Interactive use
  - Interface to the file system is extremely convenient
  - Contents of a program's address space are the property of the program (e.g., no file system control blocks)
- **Severe size constraint on the system & its software**
  - Encouraged economy and elegance of design
- **The system was able to, and did, maintain itself**

Performance Evaluation (?)

- **Timings were made of the assembly of a 7621-line program**
  - "We will not attempt any interpretation of these figures nor any comparison with other systems, but merely note that we are generally satisfied with the overall performance of the system"

- **Statistics on users, number of directories/files, command CPU usage, command access frequencies, reliability (98% uptime)**

Comparison with Related Work (?)

- "UNIX offers a number of features seldom found even in larger operating systems."
- "Perhaps the most important achievement of UNIX is to demonstrate that a powerful OS for interactive use need not be expensive either in equipment or in human effort."
- Influences: "The success of UNIX lies not so much in new inventions but rather in the full exploitation of a carefully selected set of fertile ideas, and especially in showing that they can be keys to the implementation of a small yet powerful OS."