A Few Classics

Prof. Phillip Gibbons

Spring 2020, Lecture 2
Today’s Reminders / Announcements

• Summaries are to be submitted via Canvas by class time

• Announcements and Q&A are via Piazza (please enroll)

• Course home pages are still homeless

• Project presentations now May 4 & 5

• TA Andrew Chung’s office hours: Wednesdays 3-4 pm in CIC 2214

• SDI Seminar Series (usually Thursdays at noon)
  – https://www.pdl.cmu.edu/SDI/index.shtml
AI for Application Modernization

Enterprises who want to take advantage of the Cloud are struggling to modernize their legacy applications. These applications can be old enough that the original architects are no longer present with the company and there may be a limited documentation available. In this talk, I will discuss the application modernization process, and focus on two key areas where AI can play a significant role in transforming it. Firstly, the talk will address how AI can improve the application portfolio assessments, and automate recommendations for application modernization (e.g. containerize app, refactor app, etc.) by considering business, technical and operational properties of the application. Secondly, the talk will present an AI driven semi-automated utility for transforming JEE monoliths to microservices. The approach formalizes the concept of ‘data dependency’ graphs obtained from static code analysis. For a monolith, ML and DL are used on runtime traces corresponding to various use cases along with the data dependency graph to generate stateless microservices. The talk will also discuss research challenges in the application modernization domain.

Maja Vuković
IBM Research

Maja Vuković is a Research Manager and a Principal Research Staff Member at IBM Research. Her team is responsible for building AI solutions for application modernization, AIOps, and compliance in hybrid cloud environments. Maja is a member of IBM Academy of Technology, an IBM Master Inventor and a Senior Member of IEEE, and was awarded Women in Services Computing Award by IEEE. Maja has received her Ph.D. from the University of Cambridge, for her work on context-aware service composition using AI planning.

VISITOR HOST: Phil Gibbons, gibbons@cs.cmu.edu
For more information or questions:
Karen Lindenfelser, 8-6716, karen@ece.cmu.edu
http://www.pdl.cmu.edu/SDI/
“End-to-End Arguments in System Design”
Jerome Saltzer, David Reed, David Clark 1984

“Hints for Computer System Design”
Butler Lampson 1983

“The UNIX Time-Sharing System”
Dennis Ritchie & Ken Thompson 1974
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
Jerry Saltzer was a team leader for Multics, advisor of Reed & Clark

David Reed designed UDP

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
The End-to-End Argument

• The function in question can completely and 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.)

“Choosing the proper boundaries between functions is perhaps the primary activity of the computer system designer.”
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 these checks
  – Low-level may have too little information to do checks well
Applications 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)

• Delivery guarantees

• Secure transmission of data

• Duplicate message suppression
  – Duplicates may be caused by the end-point application

• 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

- 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
Do one thing well

- Don’t generalize
- Get it right
- Don’t hide power
- Use procedure arguments
- 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.
# Summary of the Slogans

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

• **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.

• **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.

• **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.
Functionality: Continuity

• Keep basic interfaces stable

• Keep a place to stand
  – Backward compatibility package: old interface on new system
  – World-swap debugger
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
Speed: Interface

• Split resources in a fixed way
  – Dedicated is faster than Shared

• Use static analysis

• 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
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

• Safety first
  – Strive to avoid disaster
  – Paging systems avoid thrashing

• End-to-end

• Shed load
  – Red button user interface idea
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

• Log updates to record the truth about the state of an object

• Make actions atomic or restartable (idempotent)
Q: What award have Lampson, Hamming, Ritchie, and Thompson all won?

ACM Turing Award

- 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**
Discussion: Summary Questions

• **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.

• **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.

• **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.
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.”
Friday’s Paper

“Implementing Remote Procedure Calls”
Andrew Birrell & Bruce Nelson 1984