Institute for Software Research International
School of Computer Science
Carnegie Mellon

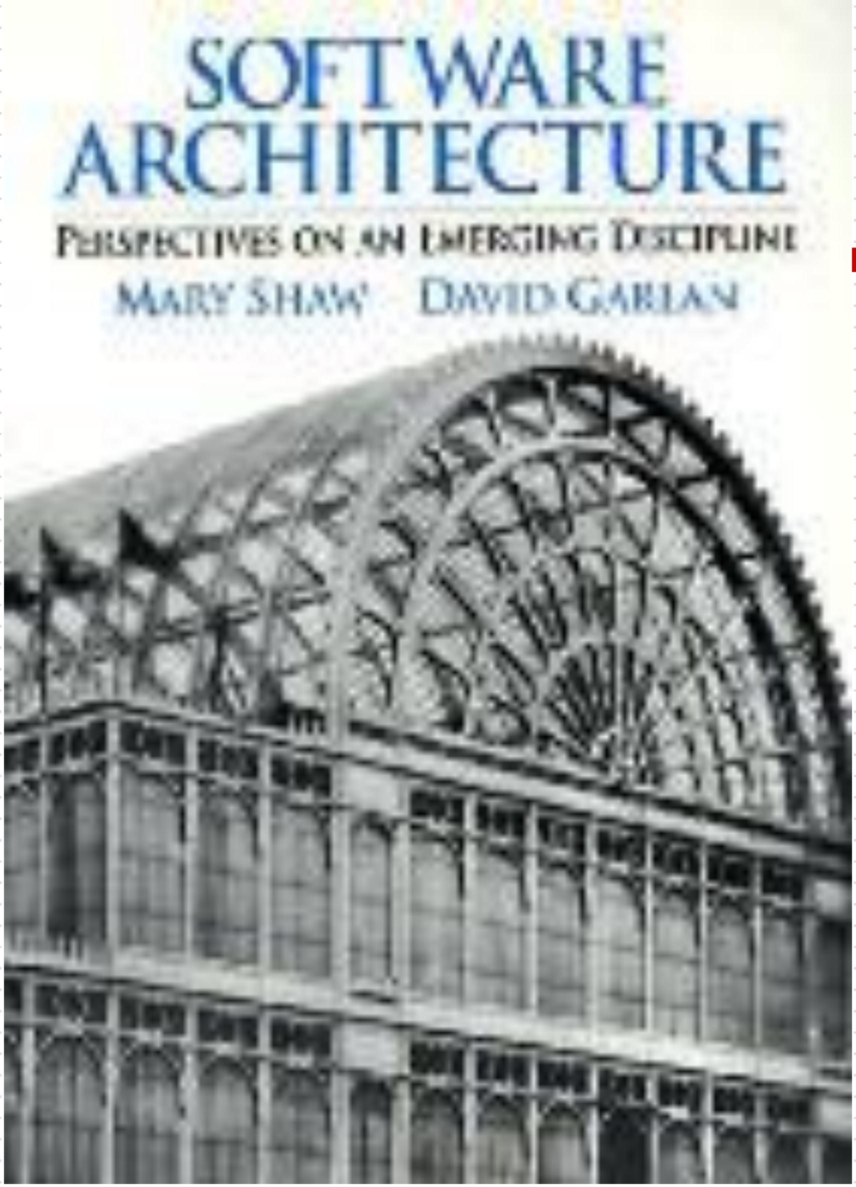
Principles of Software Architecture

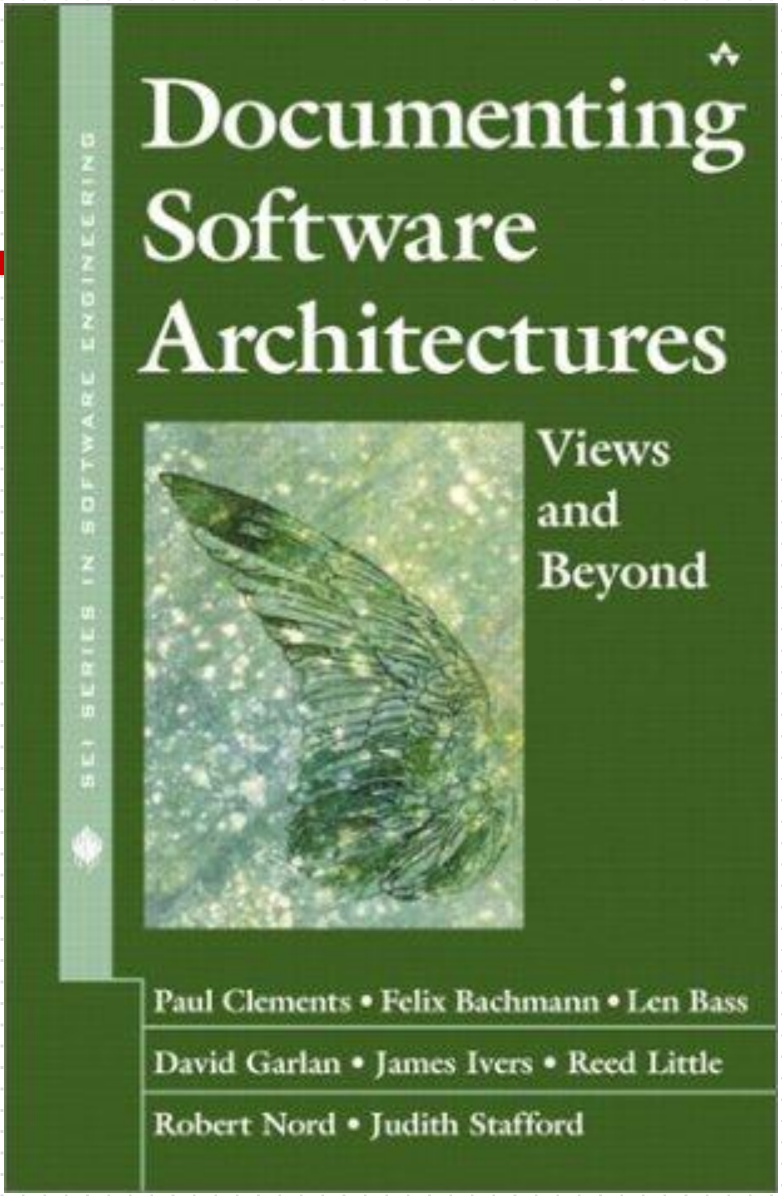
What is Software Architecture?

David Garlan, Carnegie Mellon University

About me

- □ Professor at Carnegie Mellon University since 1990
- □ Worked as software architect in industry before joining academia
- □ Director of Professional Software Engineering Programs
- □ Active researcher and educator in this area
- □ Frequent trips to Korea to work with companies like Samsung Electronics, LG, and others.


SOFTWARE ARCHITECTURE
PERSPECTIVES ON AN EMERGING DISCIPLINE
MARY SHAW
DAVID GARLAN


Documenting Software Architectures
Views and Beyond
SEI SERIES IN SOFTWARE ENGINEERING
Paul Clements • Felix Bachmann • Len Bass
David Garlan • James Ivers • Reed Little
Robert Nord • Judith Stafford

Plan for Lectures - 1

- Part 1: Basic concepts of software architecture
  - What is software architecture?
  - Why is it important?
  - How has it evolved?
  - What are the key concepts involved in it?
  - What should all software engineers know about software architecture?

Plan for Lectures - 2

- Part 2: A Quick Tour of Some Software Architecture Techniques
  - Architecture requirements and drivers
  - Evaluation of software architectures
  - Documentation
  - Product lines

Objectives of this Lecture

- Define *software architecture* and explain why you should care about it
- Relate software architecture to *programming*
- Provide a *historical perspective* on software architecture

Examples of Software Architecture

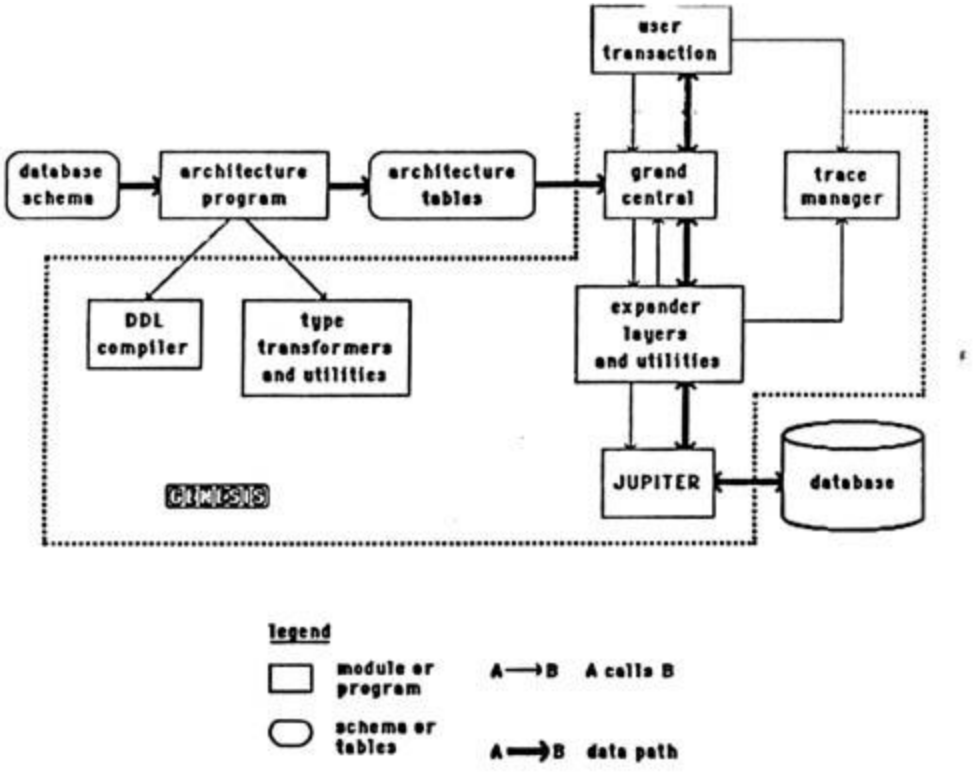


Figure 3.1 The Configuration of the GENESIS Prototype

Genesis: A Reconfiguration Database Management System, D. S. Batory, J.R. Barnett, J.F. Garza, K.P. Smith, K. Tsukuda, B.C. Twichell, T.E. Wise, Department of Computer Sciences, University of Texas at Austin,

More Examples

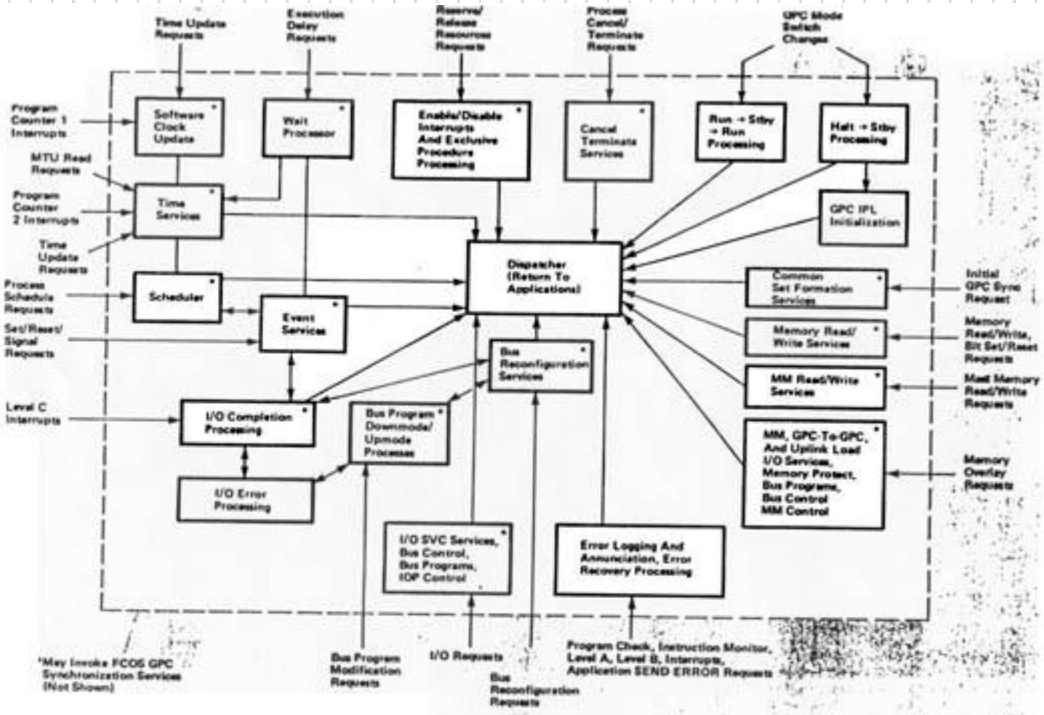

Time Update Requests
Execution Delay Requests
Reserve/ Release Resources Requests
Process Cancel/ Terminate Requests
GPC Mode Switch Changes
Program Counter 1 Interrupts
Software Clock Update *
Wait Processor *
Enable/Disable Interrupts And Exclusive Procedure Processing *
Cancel Terminate Services *
Run → Stby → Run Processing
Halt → Stby Processing
MTU Read Requests
Program Counter 2 Interrupts
Time Services *
GPC IPL Initialization
Time Update Requests
Dispatcher (Return To Applications)
Process Schedule Requests
Scheduler *
Common Set Formation Services *
Initial GPC Sync Request
Set/Reset/ Signal Requests
Event Services *
Memory Read/ Write Services *
Memory Read/Write, Bit Set/Reset Requests
Bus Reconfiguration Services *
MM Read/Write Services *
Mass Memory Read/Write Requests
Level C Interrupts
I/O Completion Processing *
Bus Program Downmode/ Upmode Processes *
MM, GPC-To-GPC, And Uplink Load I/O Services, Memory Protect, Bus Programs, Bus Control MM Control *
Memory Overlay Requests
I/O Error Processing
I/O SVC Services, Bus Control, Bus Programs, IOP Control *
Error Logging And Annunciation, Error Recovery Processing
*May Invoke FCOS GPC Synchronization Services (Not Shown)
Bus Program Modification Requests
I/O Requests
Bus Reconfiguration Requests
Program Check, Instruction Monitor, Level A, Level B, Interrupts, Application SEND ERROR Requests

FIGURE 7. Flight Computer Operating System (The FCOS dispatcher coordinates and controls all work performed by the on-board computers.)

Communications of the ACM, "Architecture of the Space Shuttle Primary Avionics Software Systems," Gene D. Carlow, September 1984, Vol. 27, No. 9, P. 933

More Examples

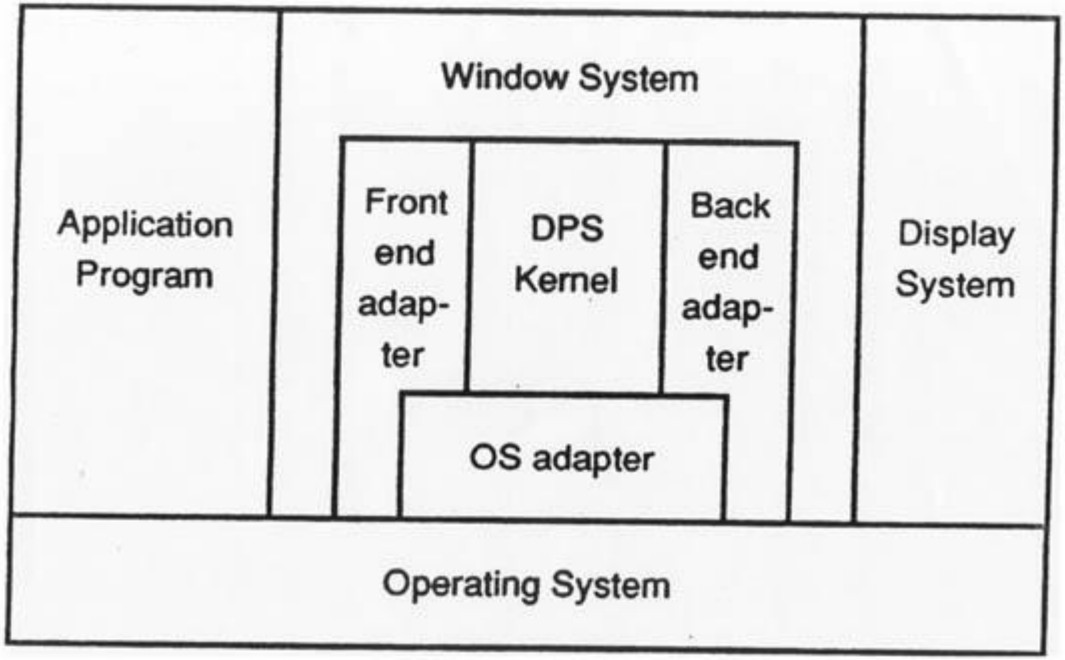
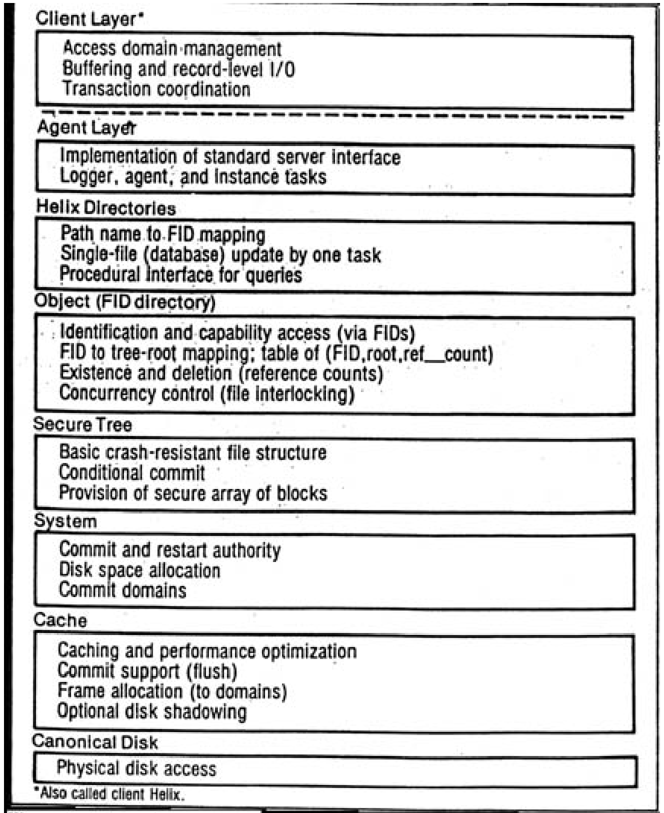


Figure 2. Display PostScript interpreter components.

An Overview of the DISPLAY POSTSCRIPT™ System, Adobe Systems Incorporated, March 16, 1988, P. 10

More Examples



Client Layer*

Access domain management
Buffering and record-level I/O
Transaction coordination

Agent Layer

Implementation of standard server interface
Logger, agent, and instance tasks

Helix Directories

Path name to FID mapping
Single-file (database) update by one task
Procedural interface for queries

Object (FID directory)

Identification and capability access (via FIDs)
FID to tree-root mapping; table of (FID,root,ref__count)
Existence and deletion (reference counts)
Concurrency control (file interlocking)

Secure Tree

Basic crash-resistant file structure
Conditional commit
Provision of secure array of blocks

System

Commit and restart authority
Disk space allocation
Commit domains

Cache

Caching and performance optimization
Commit support (flush)
Frame allocation (to domains)
Optional disk shadowing

Canonical Disk

Physical disk access

*Also called client Helix.

Figure 2. Abstraction layering.

IEEE Software, "Helix: The architecture of the XMS Distributed File System, Marek Fridrich and William Older, May 1985, Vol. 2, No. 3, P. 23

More Examples

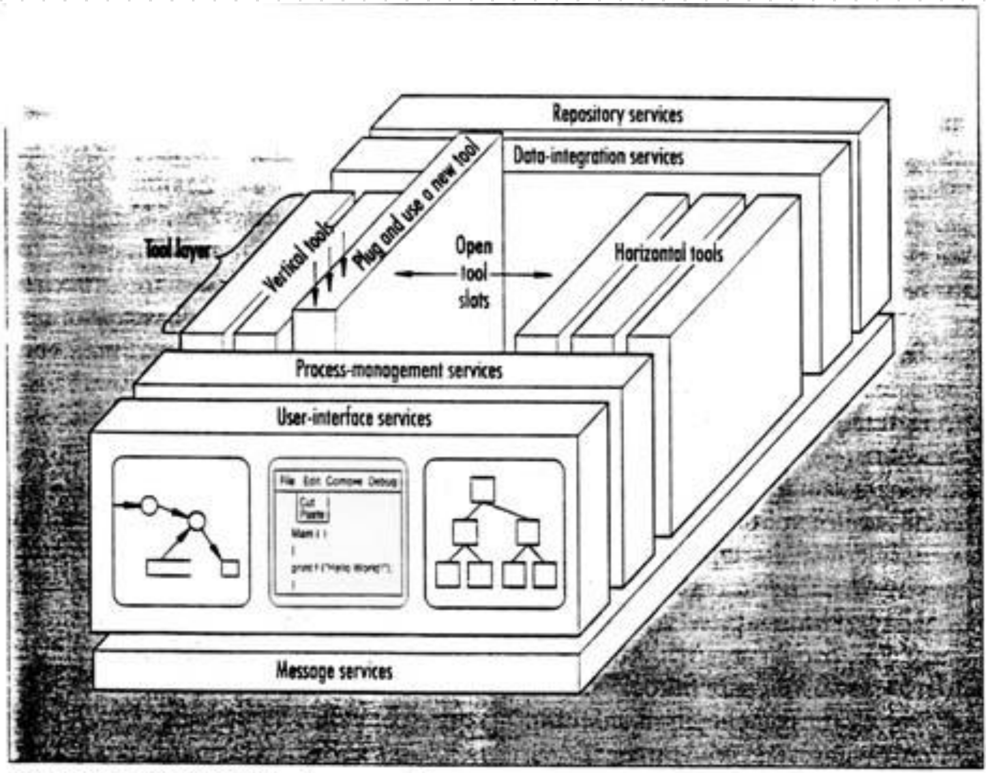
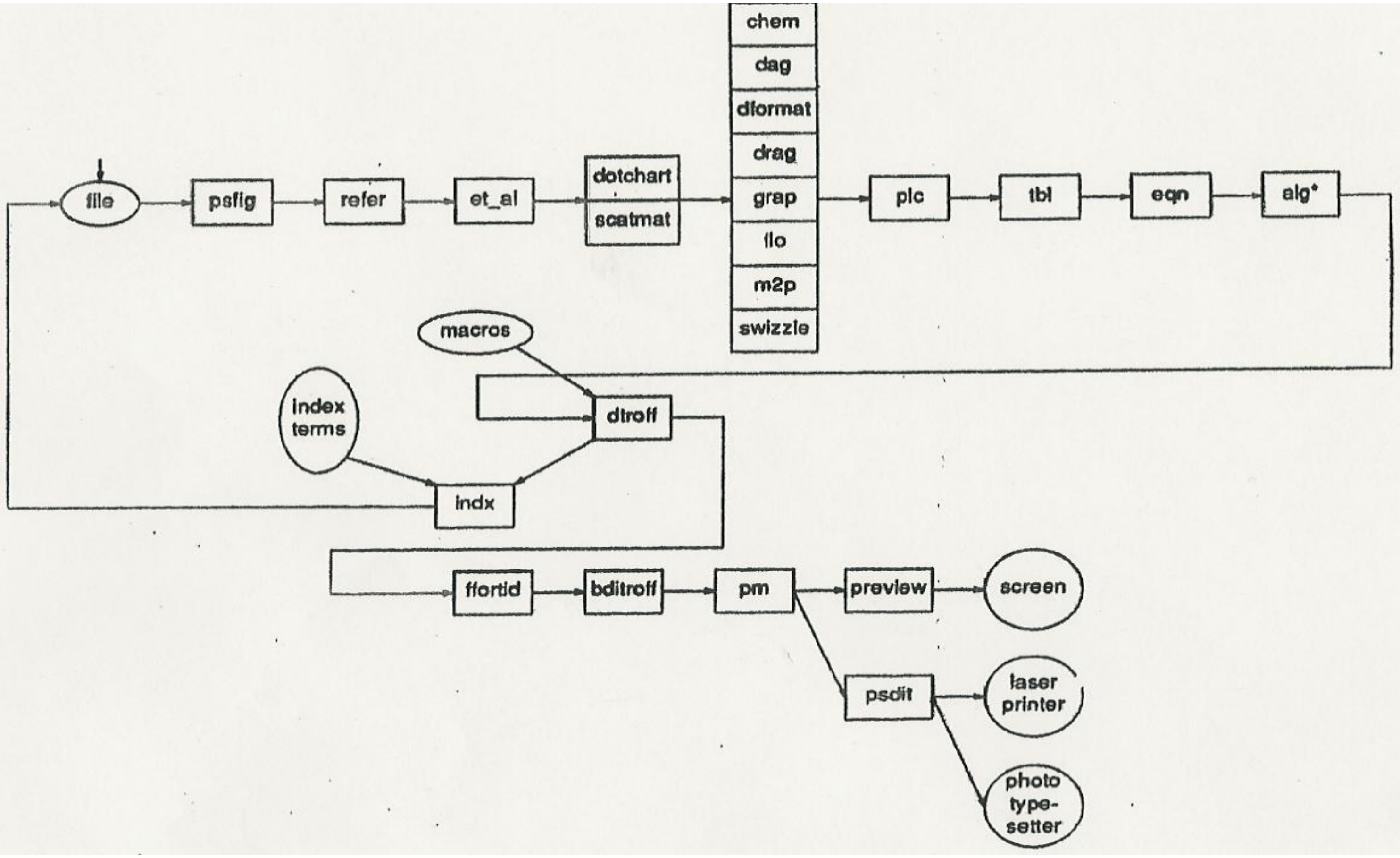


Figure 1. The NIST/ECMA reference model.

More Examples



More Examples

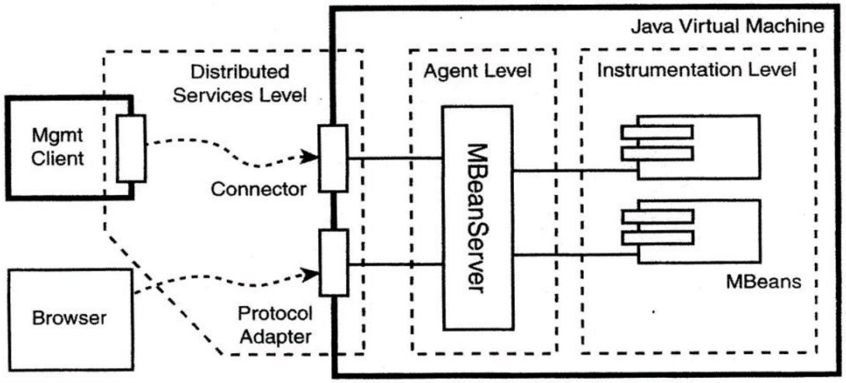
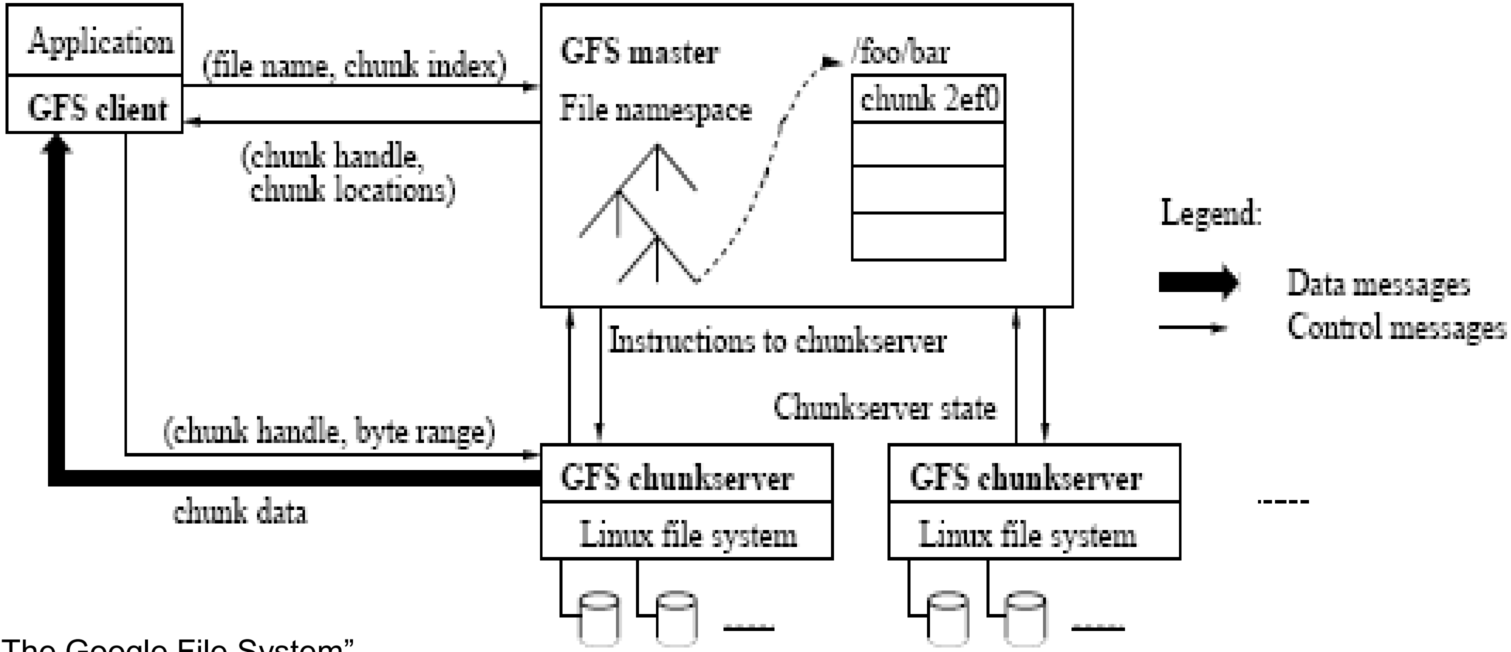


FIGURE 2.1
JMX Management Architecture.

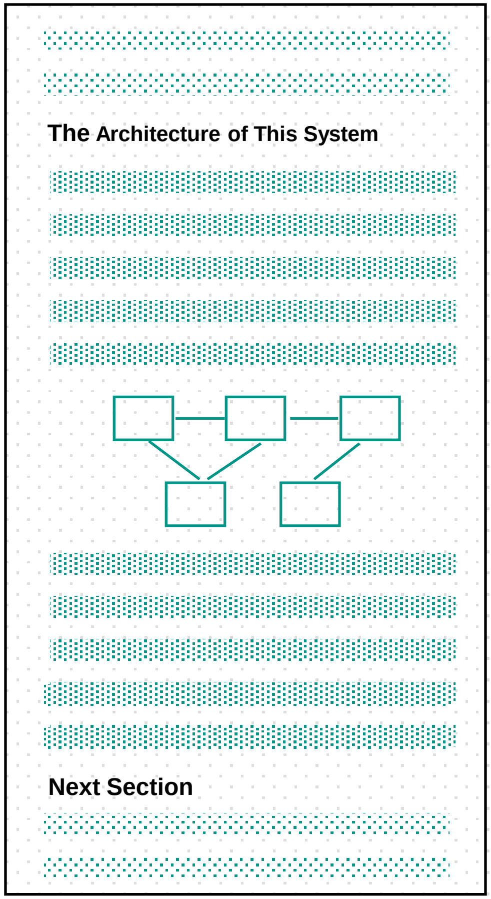
More Examples



Source: “The Google File System”
Sanjay Ghemawat, Howard Gobioff,
and Shun-Tak Leung

Figure 1: GFS Architecture

Descriptions of Software Architecture



The Architecture of This System

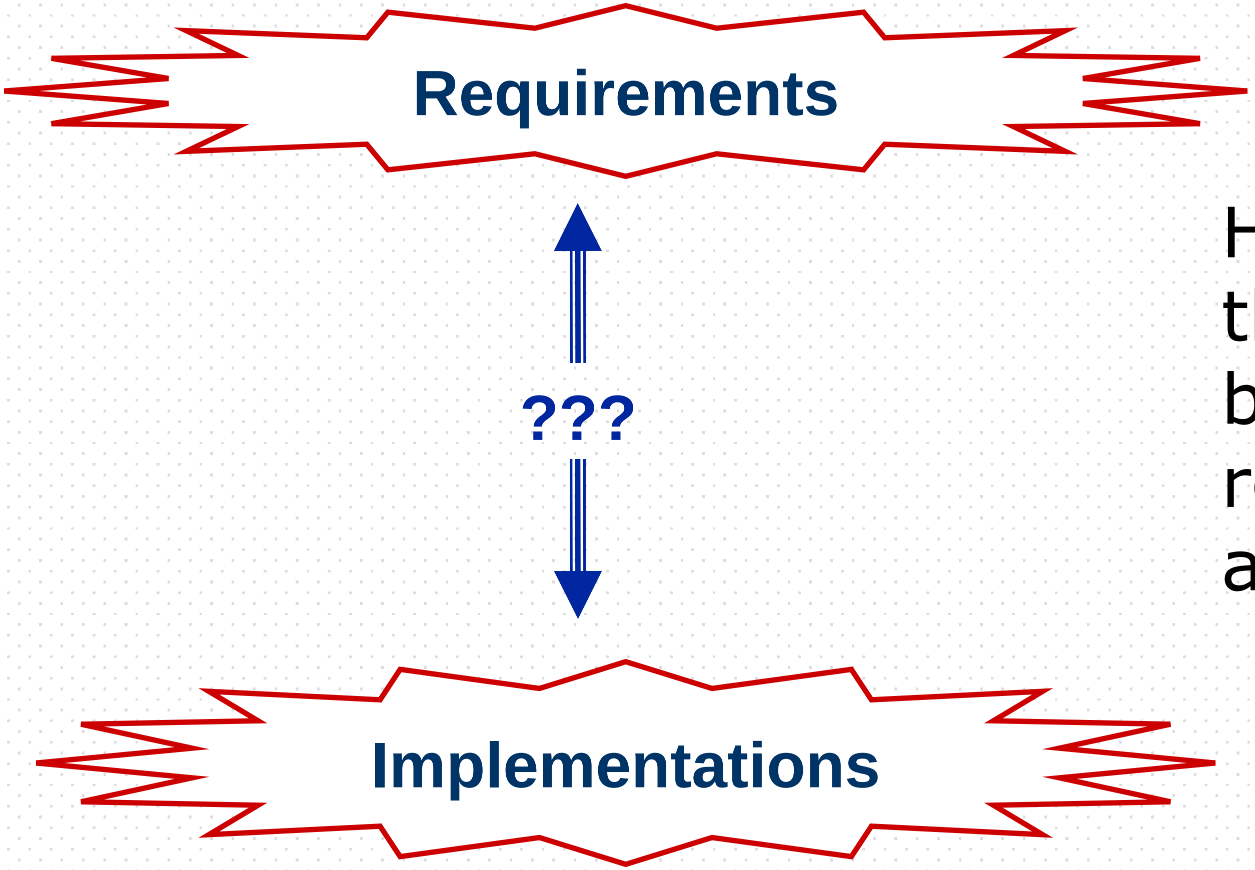
Next Section

- □ Descriptions of software systems often include a section on “the architecture of this system”
- □ Usually informal prose plus box-and-line diagram
- □ Lots of appeal to intuition
- □ Little precision, rarely formal

The Challenge

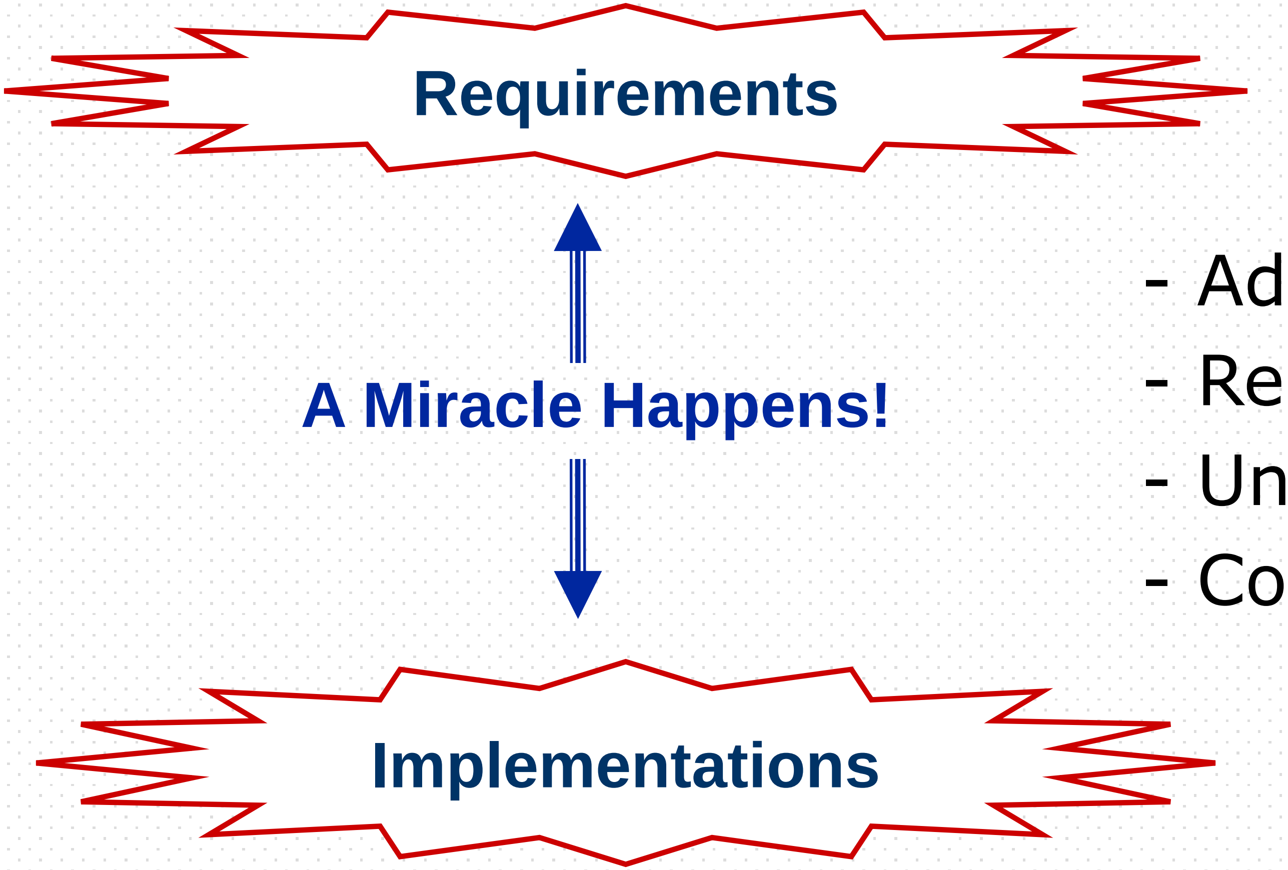
- □ Turn Software Architecture into an *engineering discipline*
  - from ad hoc definition to codified principles
- □ Develop systems *"architecturally"*
  - build systems compositionally from parts
  - assure that the system conforms to the architecture and has the desired properties
  - use standard integration architectures
  - reuse codified architectural design expertise
  - reduce costs through product-lines

The Big Problem



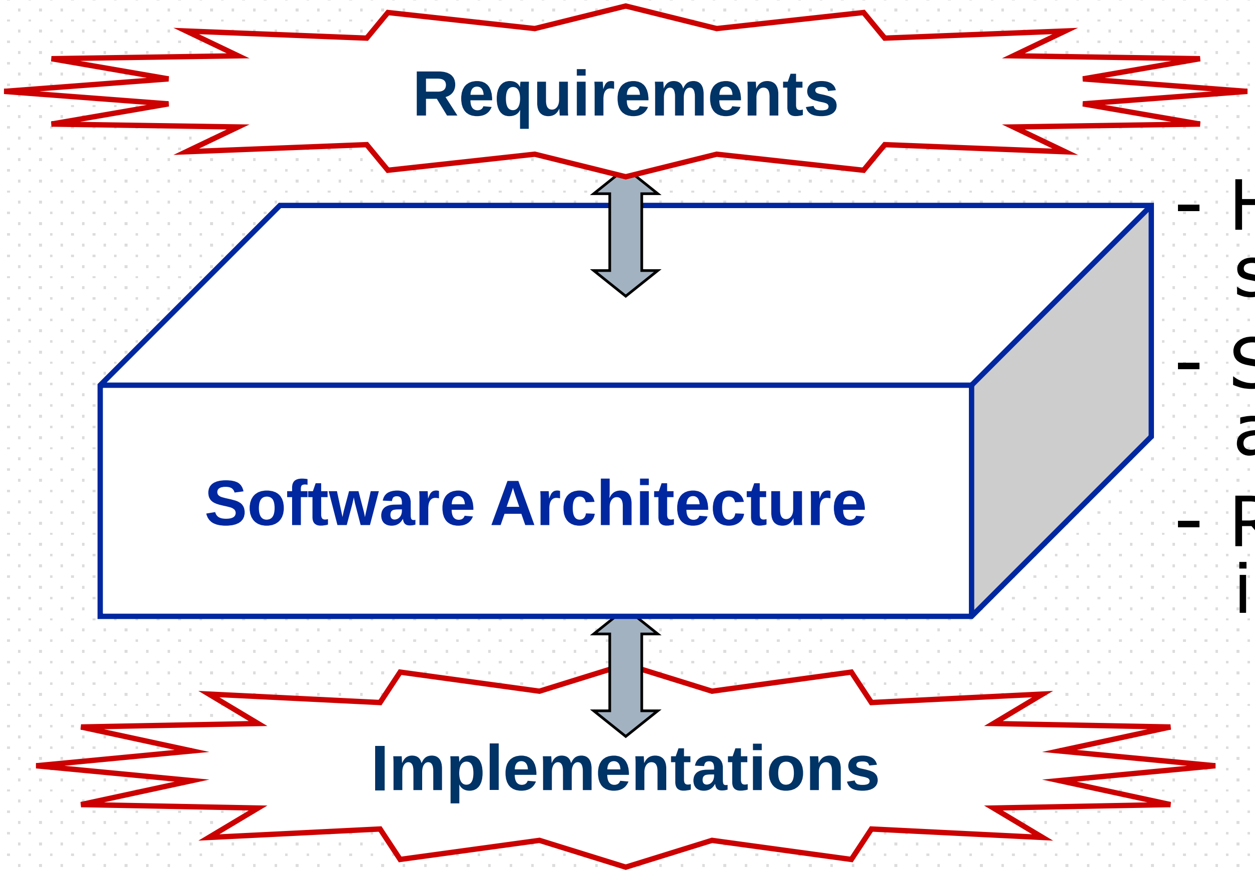
How to bridge the gap between requirements and solutions?

One Possible Answer



- Ad hoc
- Requires gurus
- Unpredictable
- Costly

The Role of Software Architecture



- High level of system design
- System-level abstractions
- Reuse design idioms

What is Software Architecture?

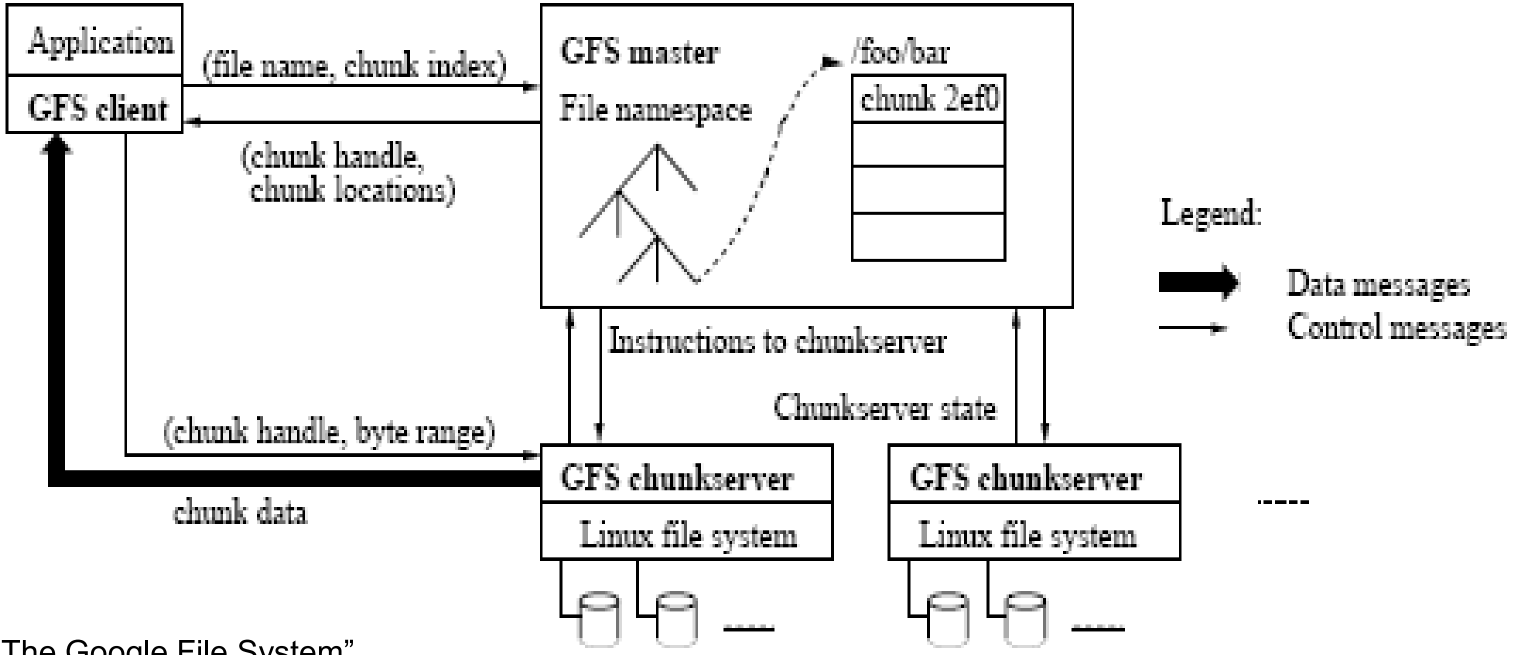
- □ There are many definitions in the literature
  - ■ CMU’s Software Engineering Institute’s web site on software architecture lists over 100 of them.
- □ The definition we like is this:

> The software architecture of a computing system is the set of structures needed to reason about the system, which comprise software elements, relations among them and properties of both.

Issues Addressed by Software Architecture - 1

- Gross decomposition of a system into parts
  - often using rich abstractions for *component interaction* (or system "glue")
  - often using common design *patterns/styles*
- Emergent system properties
  - performance, throughput, latencies
  - reliability, security, fault tolerance, evolvability
- Rationale
  - justifying architectural decisions
- Envelope of allowed change
  - "load-bearing walls"

Google Revisited



Source: "The Google File System" Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

Figure 1: GFS Architecture

The Architectural Design Task

Different issues for architecture & programs

Architecture	*Programs*
interactions among parts	implementations of parts
structural properties	computational properties
declarative	operational
mostly static	mostly dynamic
system-level performance	algorithmic performance
outside module boundary	inside module boundary

Why Do We Care?

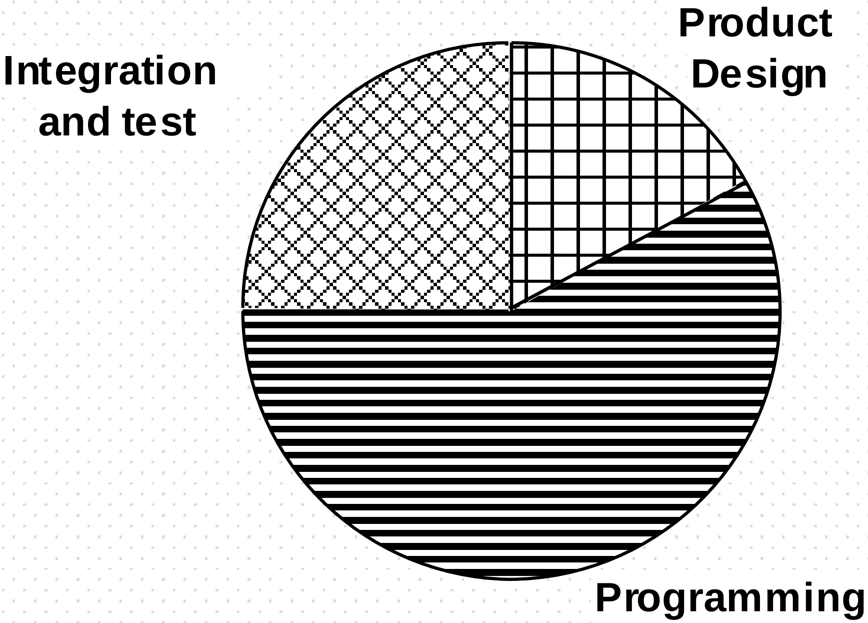
- Reduce development and maintenance costs
  - Reuse of designs
  - Improve understandability
- Improve quality of product
  - Clarify requirements
  - Make principled engineering decisions
  - Early analysis of design flaws

Reducing Costs - Example

- Design Reviews at ATT, Lucent, Avaya, Millennium
  - Architecture Review Board
  - Supports architectural reviews for projects
- Results
  - Reviewed over 700 projects from 1989-2005
  - “A correct architecture has the largest single impact on cost and quality of the product” (Maranzano 1995)
  - “We estimate that projects of 100,000 non-commentary source lines of code have saved an average of US$1 million each by identifying and resolving problems early.”*

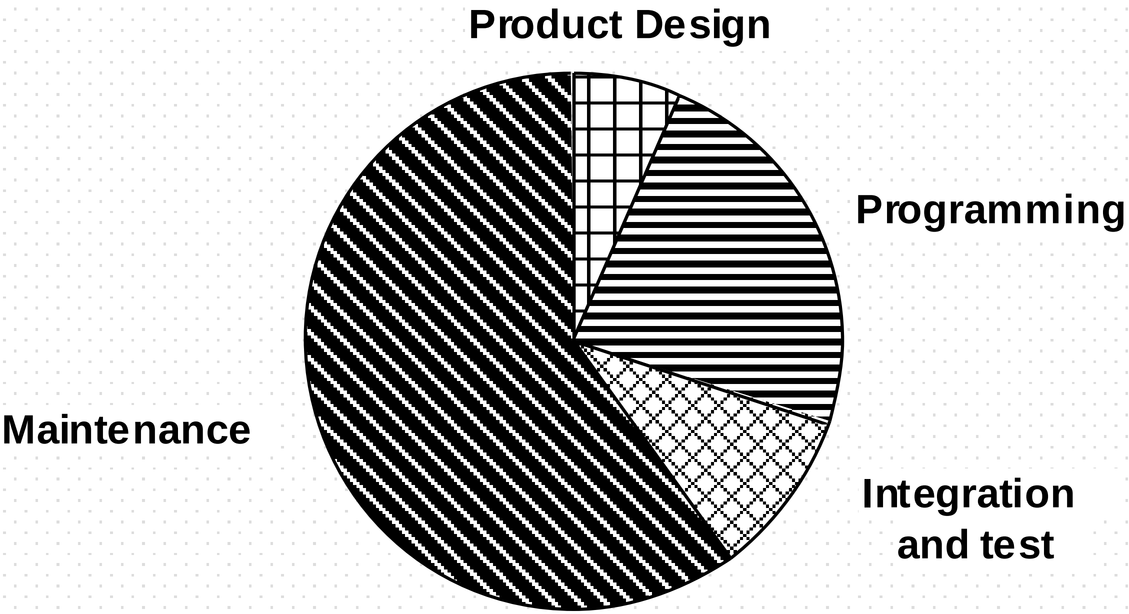
*”Architecture Reviews: Practice and Experience.” J. Maranzano, et al., *IEEE Software*, April 2005

Distribution of Software Development Costs

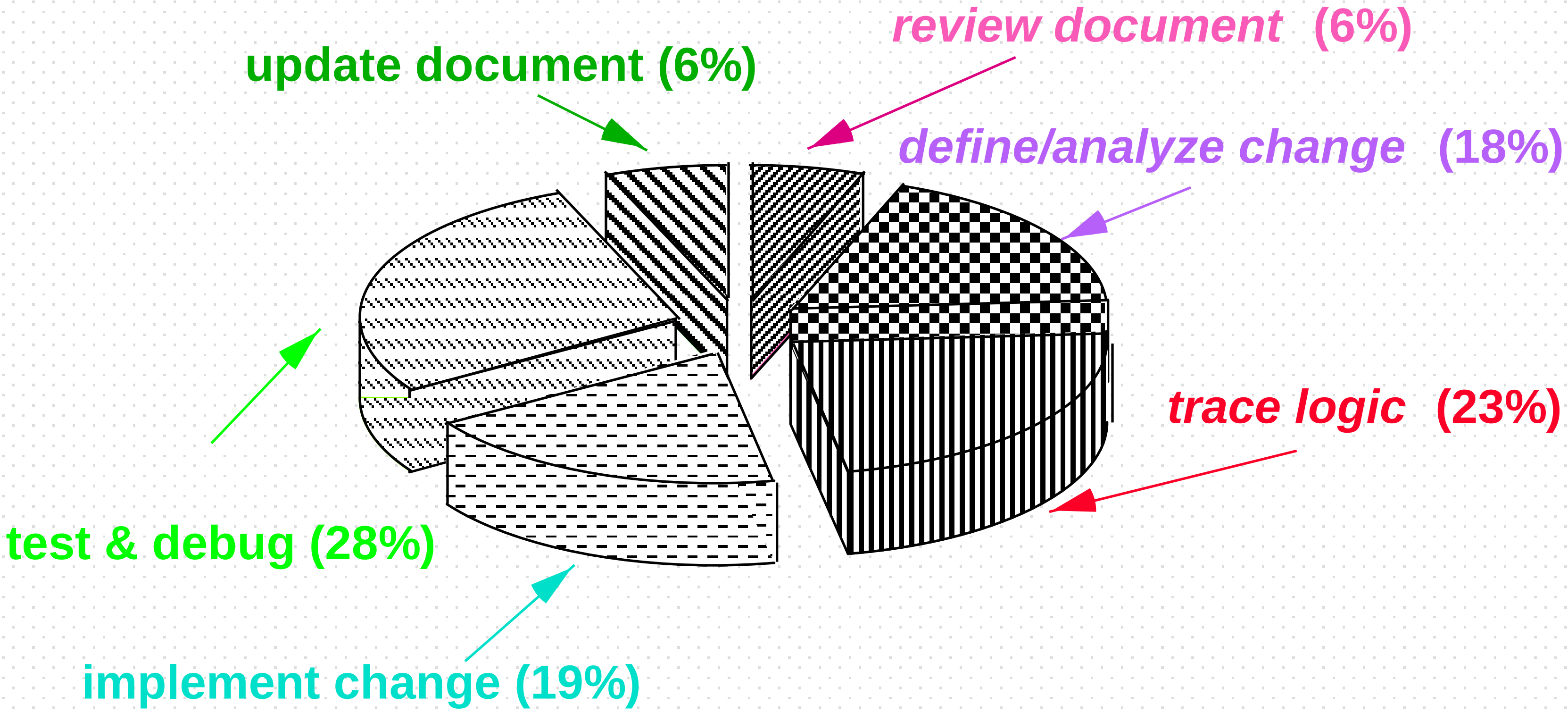


What's wrong with this picture?

Distribution of Total Software Costs

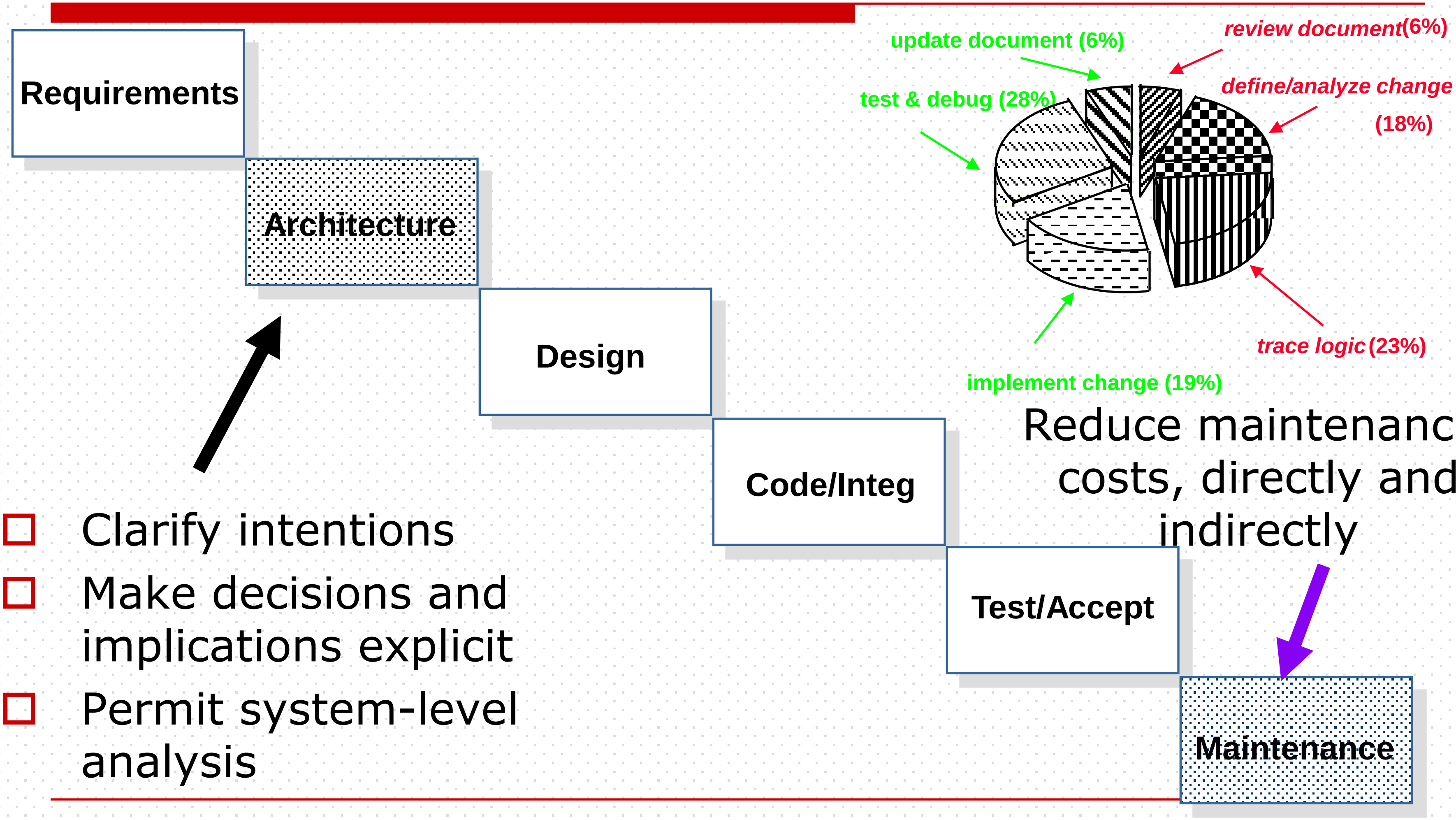


Allocation of Available Time



Up to 50% of a maintenance programmer's time is spent in analyzing and understanding existing code and documentation

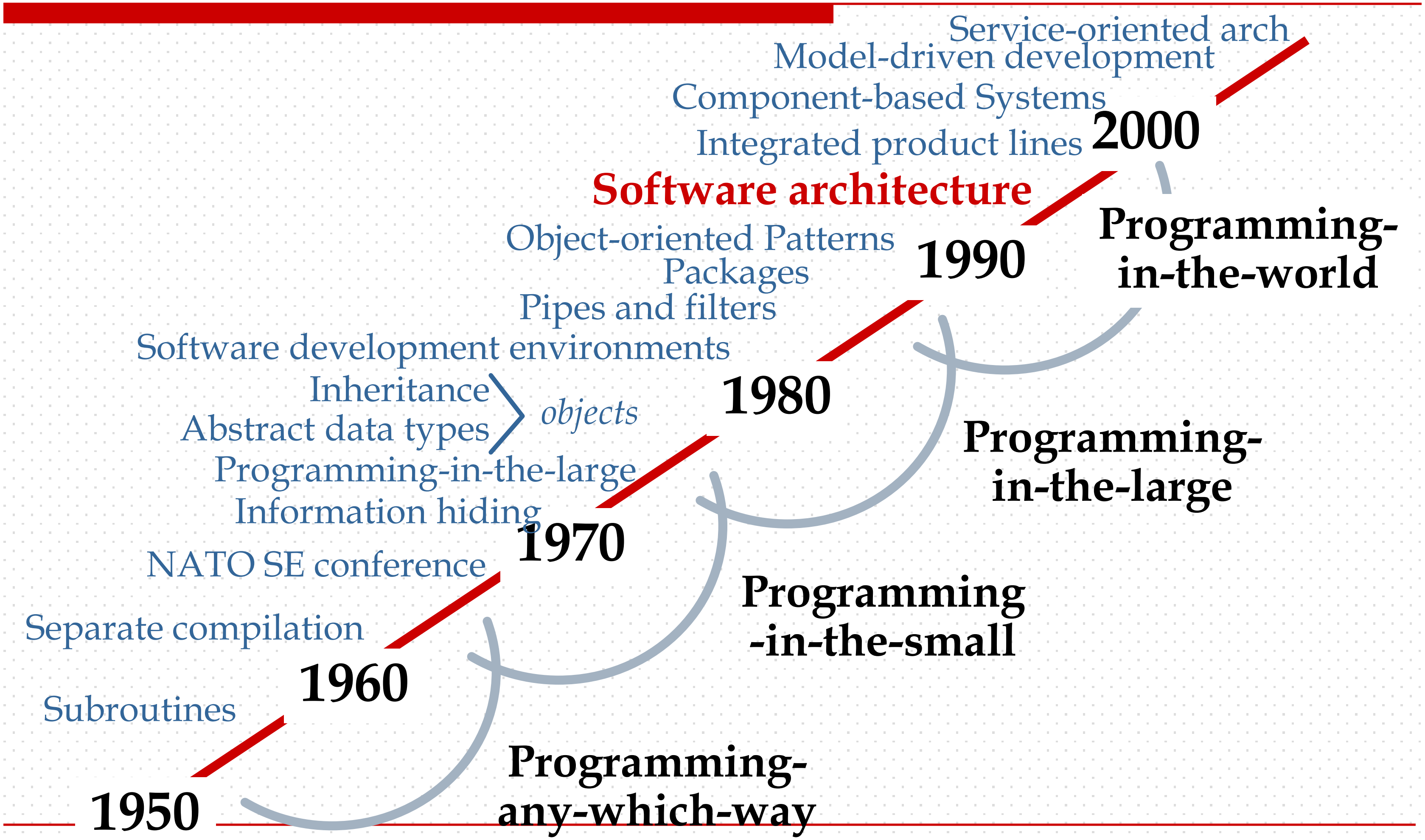
Anticipated Benefits



- □ Clarify intentions
- □ Make decisions and implications explicit
- □ Permit system-level analysis

Reduce maintenance costs, directly and indirectly

Software Architecture in Context



Service-oriented arch

Model-driven development

Component-based Systems

Integrated product lines

2000

Software architecture

Object-oriented Patterns

Packages

Pipes and filters

1990

Programming-in-the-world

Software development environments

Inheritance

Abstract data types

objects

1980

Programming-in-the-large

Information hiding

Programming-in-the-large

1970

NATO SE conference

Programming-in-the-small

Separate compilation

1960

Subroutines

Programming-any-which-way

1950

© David Garlan

Evolution of the Field of Software Architecture – 1980’s

- Informal use of *box and line diagrams*
- Ad hoc application of architectural expertise
- Diverse, uncodified use of architectural patterns and styles
- No identified “architect” on most projects

1990's

- □ Recognition of the value of *architects* in software development organizations
- □ *Processes* requiring architectural design reviews & explicit architectural documentation
- □ Use of *product lines*, commercial architectural *standards*, component *integration frameworks*
- □ *Codification* of vocabulary, notations & tools for architectural design
- □ *Books/courses* on software architecture

2000’s

- Incorporation of architectural notions into mainstream *design languages* and *tools* (e.g., UML-2)
- *Methods* based on architectural design and refinement (e.g., Model-Driven Design)
- Some architecture *analysis tools*
- Architectural *standards* for Enterprise Systems (e.g., RM-ODP, TOGAF)
- Architectural *frameworks* (e.g., SOA)

What should software engineers know? -1

- Part 1: General Concepts
  - What is software architecture
  - Basic concepts: views, styles, patterns
- Part 2: Principles of Architecting
  - Understanding architectural requirements
  - Architecture styles and tactics
  - Product lines and integration frameworks

What should software engineers know? -2

- Part 3: Architecture in Practice
  - Evaluating architectural designs
  - Handling architectural problems
  - Documenting a software architecture
  - Presenting an architecture to others

Summary

- Software Architecture is a critical area for modern software engineering
- The field has evolved so that it is now possible to do principled software architecture
- Software engineers should understand the value and techniques of software architecture

Book References

- □ *Software Architecture in Practice*, *2nd Edition* Bass, Clements, Kazman. Addison Wesley, 2003.
- □ *Software Architecture: Perspectives on an Emerging Discipline*, Shaw & Garlan Prentice-Hall, 1996.
- □ *Documenting Software Architecture: Views and Beyond*, Clements et al., Addison Wesley, 2003.