The Changing Software Paradigm

Gio Wiederhold

February 1996

Topic of Study

Program Creation and Maintenance

A central topic for the Software Industry and for Education

Background:

3 years as program manager at ARPA (1991-94)

- **Traditional techniques have topped out**
  - » few new ideas appear in programming
- **New systems are being built differently**
  - » industrial leadership, poor coherence, no science
- **Major potential for innovation**
  - » what is missing, who can contribute

What is changing

Programming is changing

from code generation

to composition of existing codes

Two views of the change:

1. **Logical progression**
2. **Paradigm change**

Effects of the change

- **New business models**
- **Need to adapt education**

1. Logical Progression

- Machine language
  - generation of basic code sequences
- Assembly language
  - shared code & shared data in memory
- Compiled languages
  - standardized code units with parameters
- Database services
  - control ceded to the DB administrator
- Object Libraries for specific Domains
  - domains & solution structure predefined

1945
1955
1960
1975
1990

Loss of autonomy & Cede control
Increase of scale & Benefit from work of others

2. Paradigm Shift

- **Re-allocation of autonomy is gradual, *invisible***
- **In the aggregate, over time, affects**
  - **Technology Paradigm**
  - **Business Practices**
  - **Educational needs**
- **Hypothesis:**

We have to explicitly change from

a *subroutine mentality* (programmer control) to

a *service mentality* (cooperation & mediation)

Evidence

PAST ARGUMENTS

- Hardware architecture
  - Number of bits/word
  - Number of registers
  - Size of address field
- DEC vs IBM vs CDC

CURRENT ARGUMENTS

- Component architecture
  - Description
  - Asynchrony
  - Inheritance
- CORBA vs OLE vs ODE

ARGUING implies VALUE SHIFT from Hardware to Software Componentry

An Early Example ~1980

Databases

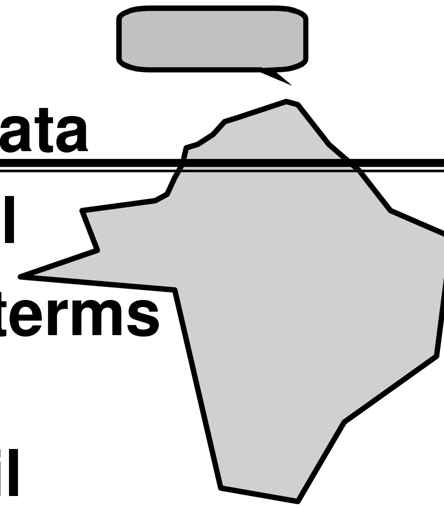
- **Provide a *Service* not under control of the application programmer**
- **Removed about 50% of code from commercial applications**
- **Changed training requirements of programmers**
- **Led to new sub-specialities:**
  - » **database administrators**
  - » **database programmers**
  - » **database educators**
- **Updates in the interface require negotiation**

Database Interfaces

SQL:

two parts

1. a Non-procedural Query Language
2. a Schema which exposes available data



but hides much detail

The query language is restricted to the terms given in the schema

The schema hides implementation detail

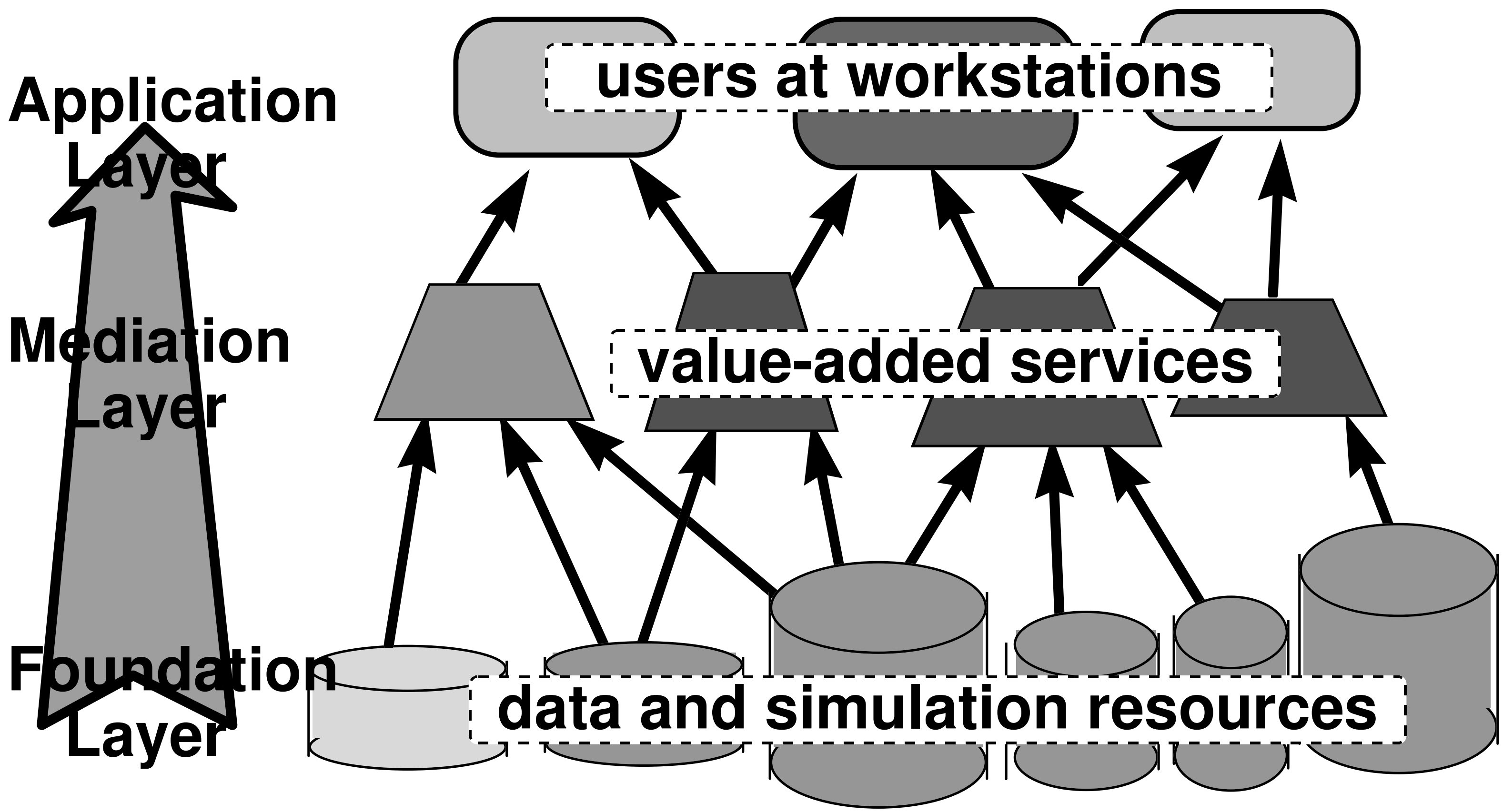
- The Database Management System (DBMS)
  - has the freedom to optimize query processing
  - must optimize to achieve performance

Recent Examples ~1990

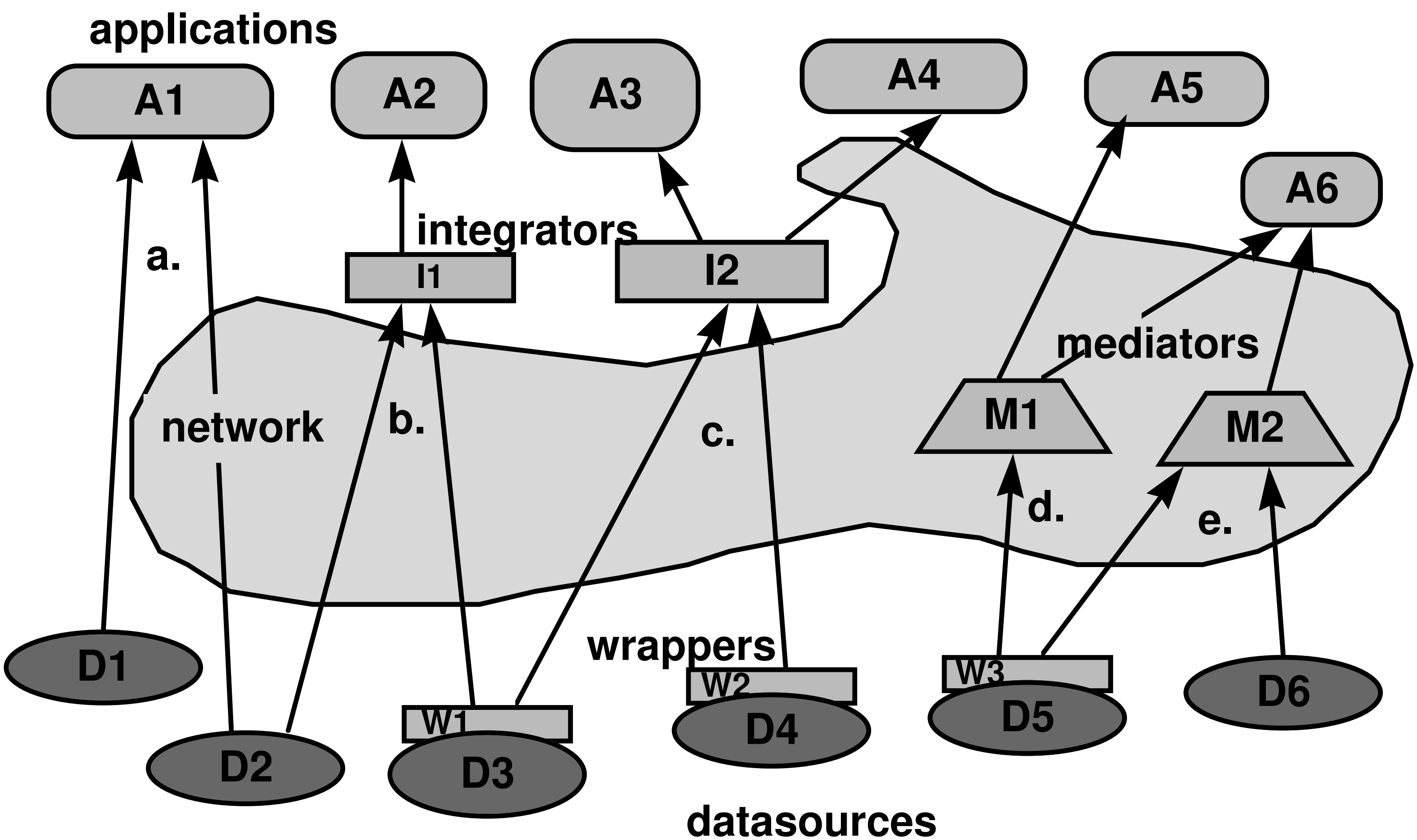
Of removal of code by services:

- Object libraries { Microsoft, KPlib, GNU, NASA, ...}
  - » not autonomous in terms of control
- Graphical Interface (GUI) builders {MOTIF, . . . }
  - » procedural, no optimization
- Application builders {Powerbuilder, NeXTstep, Visual Basic}
  - » high-level programming based on standard interfaces
- Domain Specific Software Architectures (DSSA)
  - » structure, rather than code
- Mediation Architecture {middleware}
  - » intermediate, composable information services

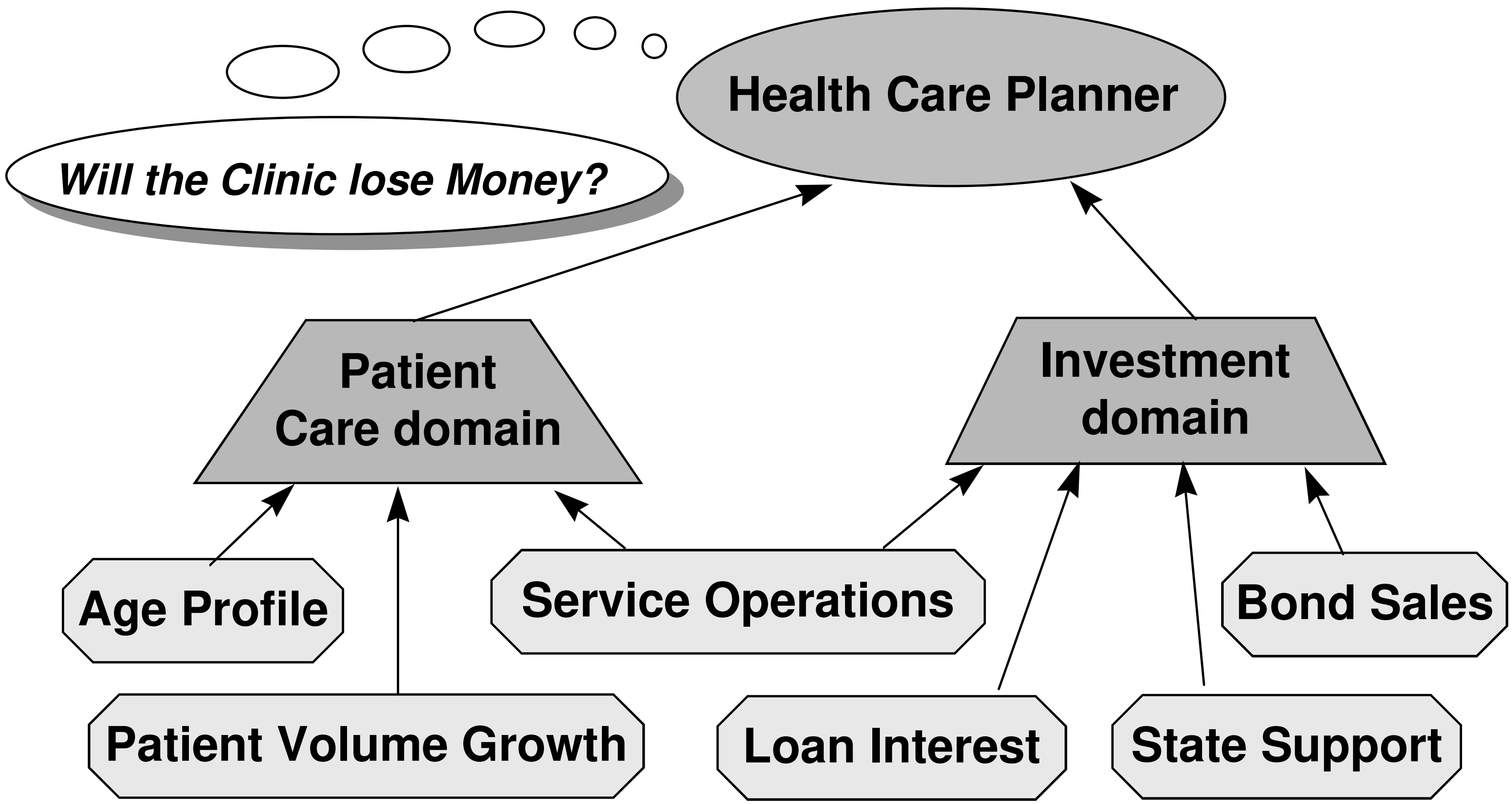
Transform Data to Information



Evolution of mediation

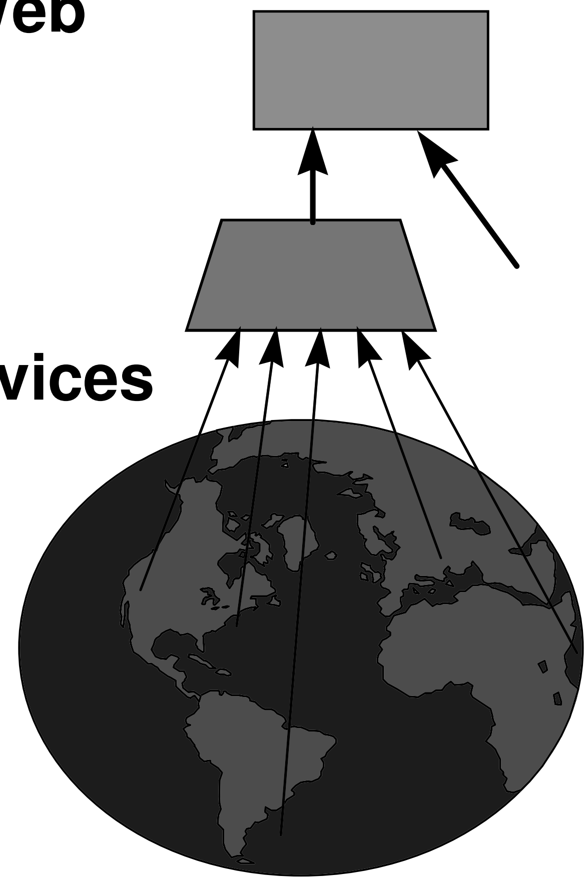


Example in Health Care

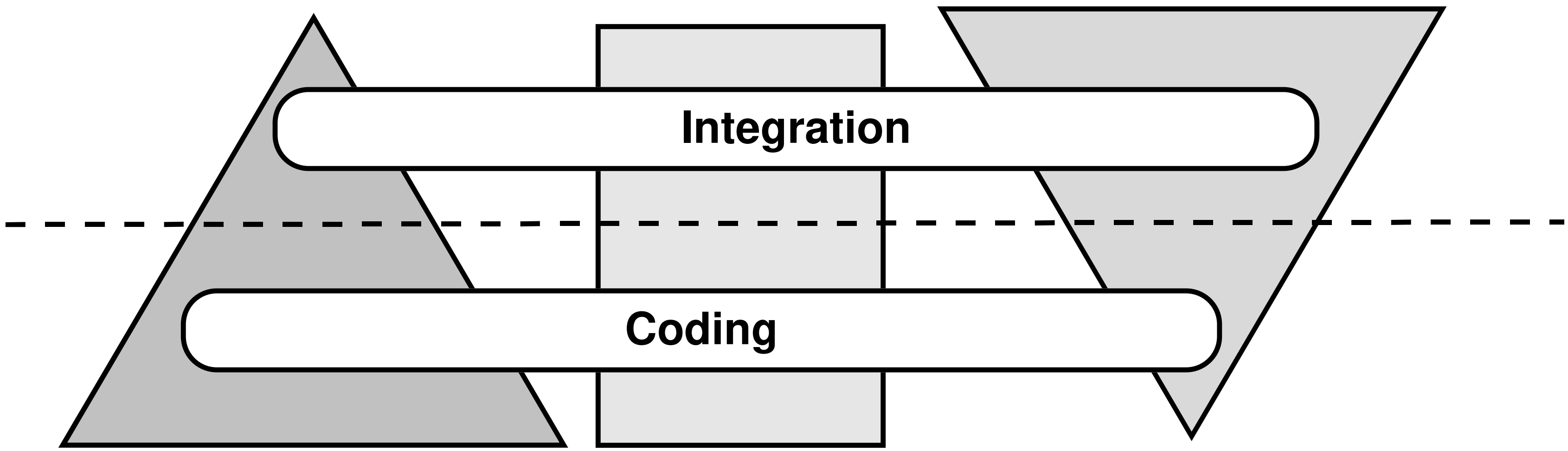


Mediation on the WWW

- Resources on the World-Wide-Web
  - are plentiful
  - autonomous
  - incoherent
- Opportunity for value-added services
  - select best source
  - improve coverage
  - minimize overlap
  - resolve inconsistencies
  - summarize results
- requires low-overhead charge mechanisms



Workforce reallocation



Integration originally performed for large systems (> 5Mbyte)

- **by system integration companies:**
  - **Honeywell, IBM federal, Lockheed, SAIC, Andersen ...**

Integration now performed for most systems (> 5Mbyte)

- **by application clients and services:**
  - ***too many to name, including you . . .***

Dealing with Multiple Services

- Multiple resource databases within an enterprise
- Related public data resources
- Processing specialists
  - industry comparison, projection, simulation
- To reduce risks: multiple suppliers

But

- Difficult setup and reimbursements
- Lack of coherence
  - Multiple standards (supplier vs customer interests)
  - No scientific basis (education ignores the changes)
- Security concern (primary barrier for virtual enterprises)

Emerging Mediation

- **Tools**
  - **Open server (Sybase)**
  - **Blades (Informix - Illustra)**
  - **Distributed Object Environment (Sun/Persistence)**
- **Initial Application Areas**
  - **Intelligence systems**
  - **Flexible manufacturing**
- ***Future***
  - ***SimQL to retrieve projections into the future***
    - » ***information provided by experts & simulations***

Maintenance

- **Definition:** ***Unscheduled*** **tasks to fix**
  - » **errors in software** **(often induced by growth in scope)**
  - » **adapt to externally imposed changes** **(tax laws, regulations, interfaces, customer expectations, …)**
  - » **adapt to internally imposed changes** **(mergers, corporate reorganization, new product lines, ...)**
- **Crucial leverage:**
  - **– 70-90% of system costs are SW-maintenance**
    - » **If maintenance costs can decrease 25% we double our capability to develop innovative products**
    - » **If maintenance costs increase by 25% we loose any capability for innovation**
  - **– Affected by paradigm change**

Maintenance Strategy

Good BENEFIT/COST

wrt customer

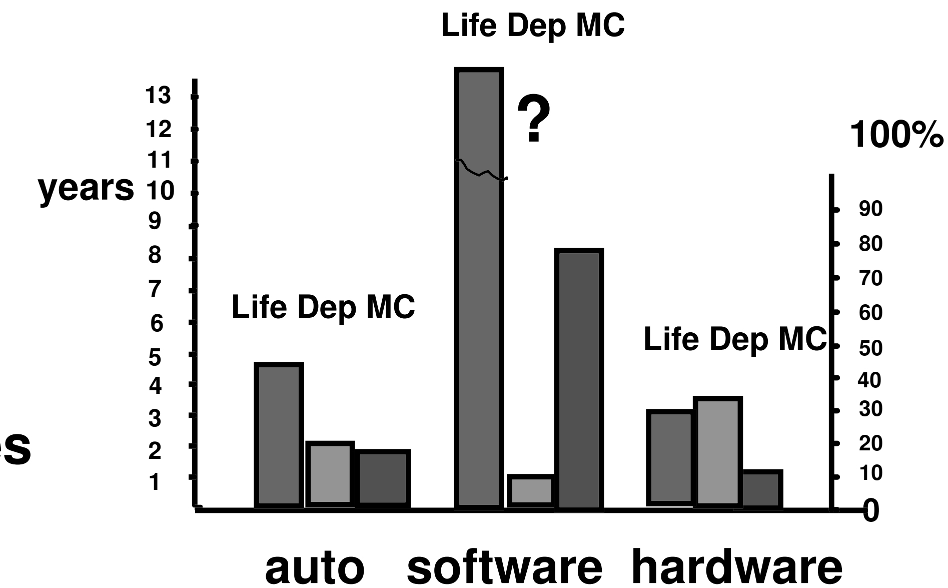
- **Benefit is relevant information, requires constant updating**
- **Avoid cost of change,**

  don't change system, interfaces

We expect SW to be adapatable

- – enables change, growth
- – long liftetimes
- – regular, high maintenance

- **Cost**
  - – initial
  - – maintenance (70-90% for SW)

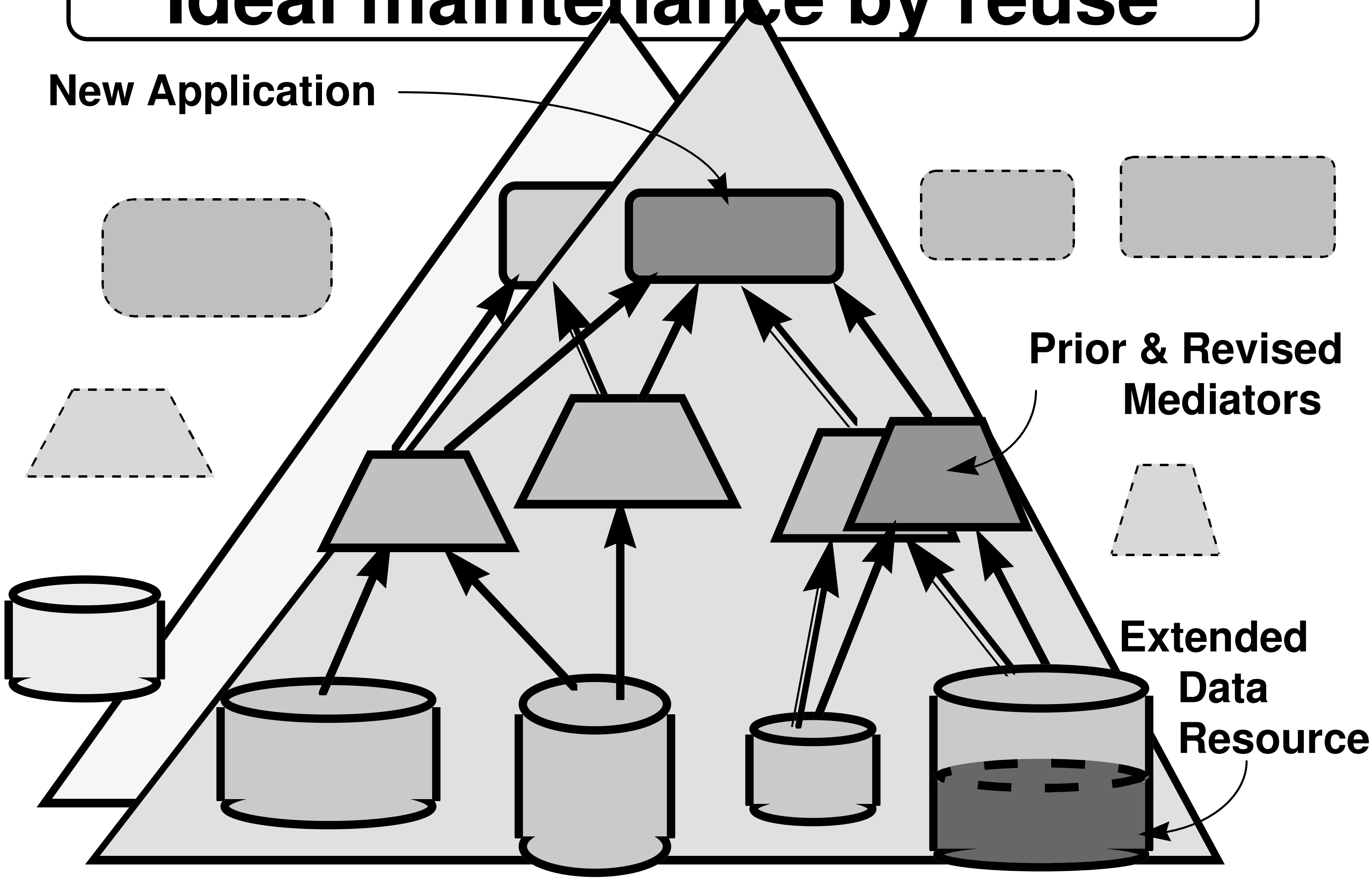


long lifetime

low depreciation ≅ 1 / lifetime

high maintenance cost

Ideal maintenance by reuse



Incremental Maintenance

1. New/Revised application neeeds more data
2. Mediator isolates other applications
3. New application checks out data and function
4. Other applications can join when feasible

versus

1. New/Revised application neeeds more data
2. Determine affected applications
3. Change affected codes, create test scaffolding
4. Plan for system-wide release at *half-year* cycle
5. Batch all other changes for that cycle
6. Perform critical, risky release of batch

New Metrics

- Productivity

  Lines of Code --> Time to Asssemble a Product

- Time to Update Product

  Several years --> 3/6 months

- Satisfaction of

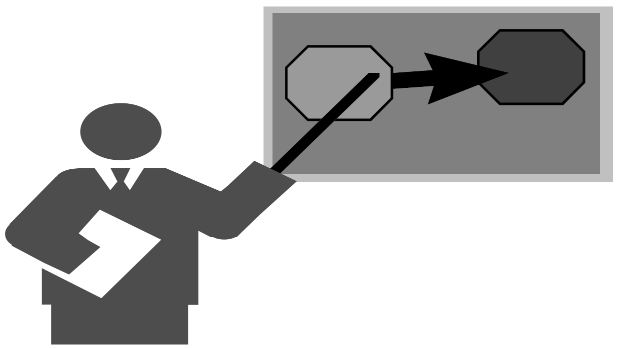
  System Administrator --> Customer

  – -- *missing:*

- Performance based on

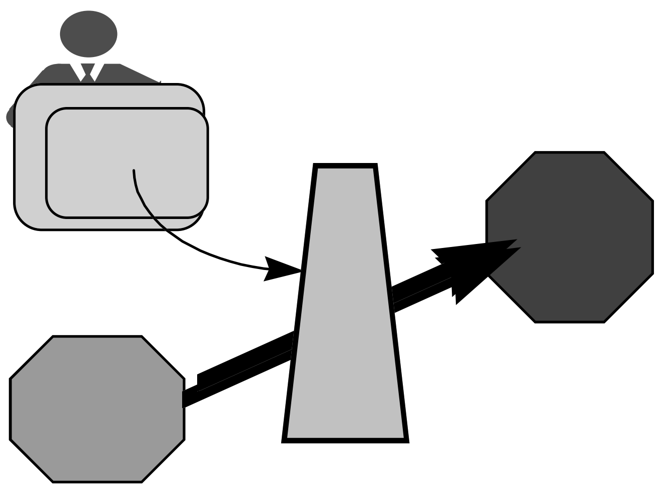
  Detailed data --> Component / Service specs

New Role for Consultants



Old

- Used at Design Time
  and
- To Explain Failures



Future

- Available as a Service
- Responsible for Knowledge Maintenance

Conclusion

- **Change in software paradigm to *Composition* is occurring, nearly invisibly**
- **Flexibility is becoming more important than satisfying all wishes wrt to performance and features**
- **Mediation provides incremental, responsive, and less risky maintenance at less cost**
- **Old habits and metrics of success are becoming obsolete and a hindrance**

Acceptance of Service Paradigm

Dissemination

- New Experimental Course: CS446, Fall 1995, evolving
  - large-scale software composition
  - develop principles for education
- Arrangements to develop course materials
  - support discussed with Hewlett-Packard Co.
- Book on Intelligent Integration of Information,
  - with Kluwer, due to appear early 1996 containing papers that focus on infrastructure research
- Monograph to cover Mediation Technology and Effect
  - under discussion, with two coauthors, for 1997
- Web page on Large-scale Integration & Composition
  - leads to many papers