[redacted]

Professor Nam-Yong Lee, Ph.D,

College of Information Science

Soongsil University

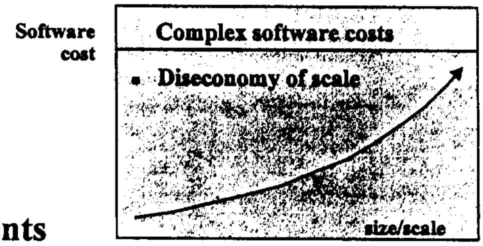
nylee@computing.soongsil.ac.kr

Table of Contents

- **What makes systems complex ?**
- **Three areas of Object-Oriented Landscape**
- **Software cost evolution**
- **Technology state of the art evolution**
- **Megaprogramming**
- **Iterative development**
- **Software production**
- **Resue process and framwork**
- **Discussion**

What Makes Systems Complex?

- Performance constraints
- Time-to-market pressures
- Certification requirements
- Distributed, real-time requirements



Software Costs = E * $(\text{Size})^{P}$

- Size & geographic distribution of the engineering team
- Reliability and fault-tolerance requirements
- Rate of requirements and technology change
- The interplay of these factors

World-Wars of Software Developers

- MICROPROCESSOR WARS
- OPERATING SYSTEM WARS
- PROGRAMMING LANGUAGE WARS
- METHODOLOGY WARS

TODAY's SOFTWARE LANDSCAPE

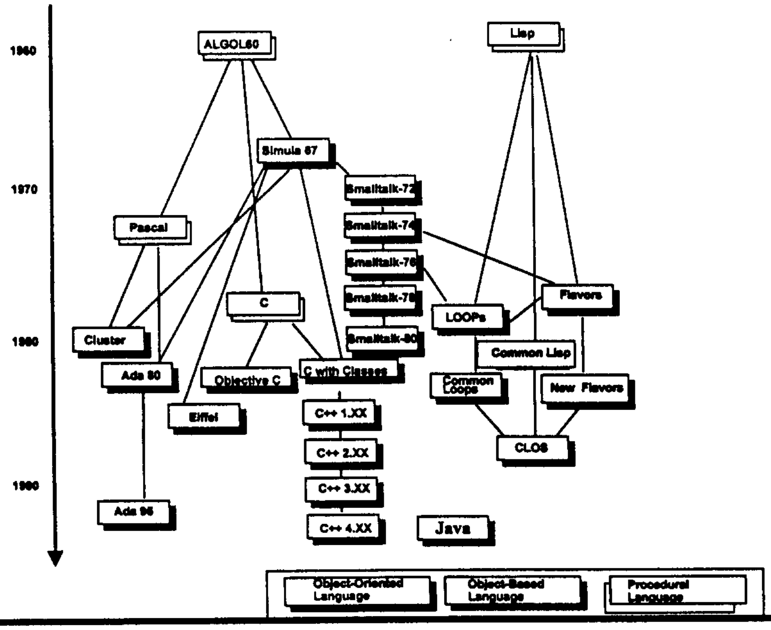
TYPE	CHARACTERISTICS
Off-the-shelf	▪ Codifies some specific horizontal domain ▪ Directed to a large market ▪ Eventually turn into a commonality market
Custom	▪ Architecture along a vertical business line ▪ Typically involves the unique composition of many off-the-shell components ▪ Is inherently complex because of a lack of prior domain models

Three Areas of Object Orientation Landscape

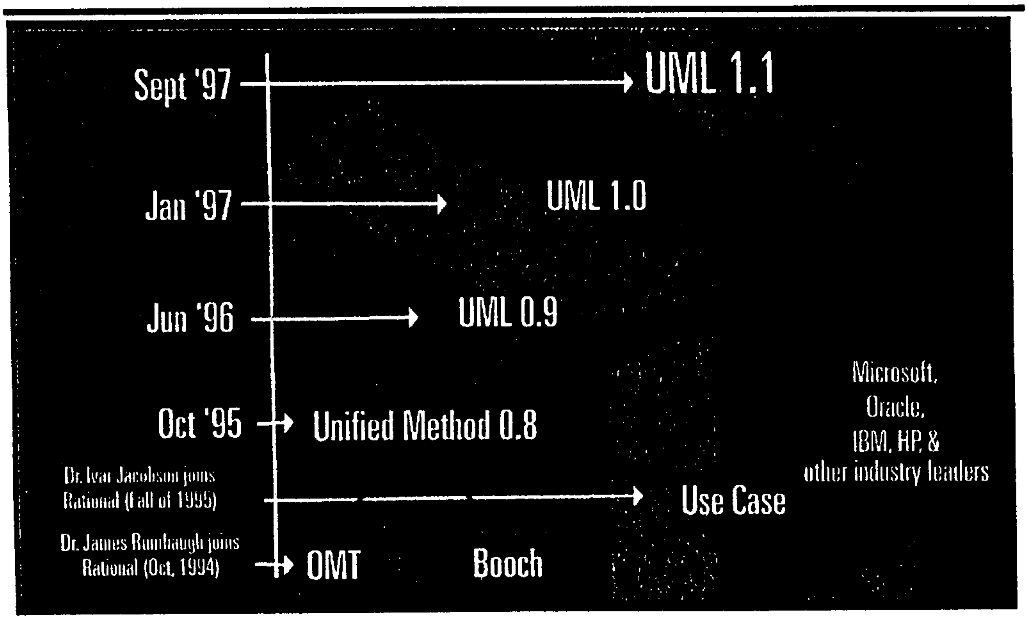
1. Object-Oriented Programming
2. Object-Oriented Methods
3. Object-Oriented Infrastructures

* Grady Booch, Coming of Age in an Object-Oriented World, IEEE Software, 1994

Object-Oriented Programming Languages



Object-Oriented Methods



Sept '97 → UML 1.1

Jan '97 → UML 1.0

Jun '96 → UML 0.9

Oct '95 → Unified Method 0.8

Microsoft,
Oracle,
IBM, HP, &
other industry leaders

Dr. Ivar Jacobson joins Rational (Fall of 1995) → Use Case

Dr. James Rumbaugh joins Rational (Oct. 1994) → OMT Booch

UML Supports Application Development

Objects

Business Objects

Relationships

large scale system

ORDBMS
Oracle

Classes

UNIFIED
MODELING
LANGUAGE

application partitioning

Components
Microsoft

Scenarios

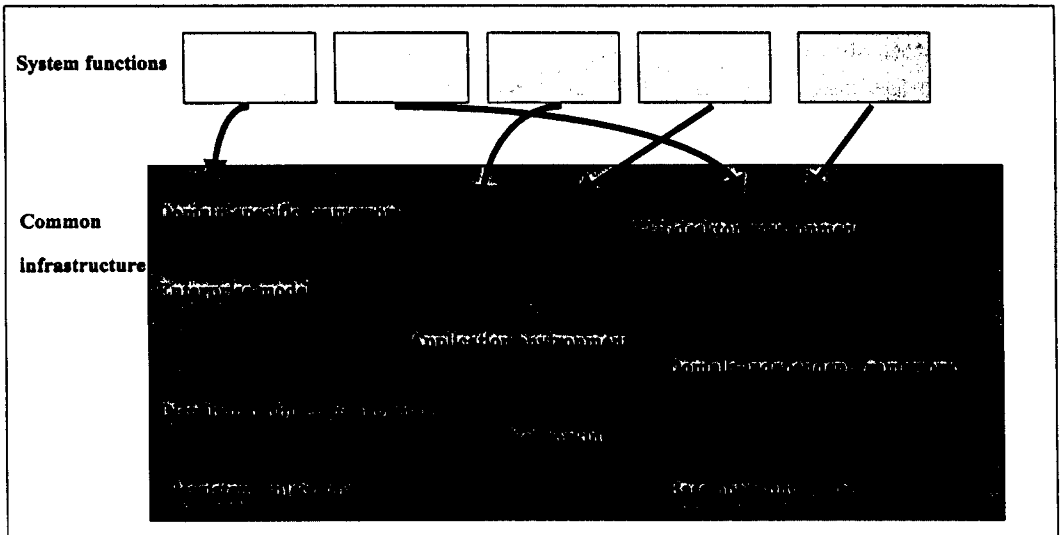
CORBA
OMG

Use Cases

ActiveX/COM
Microsoft

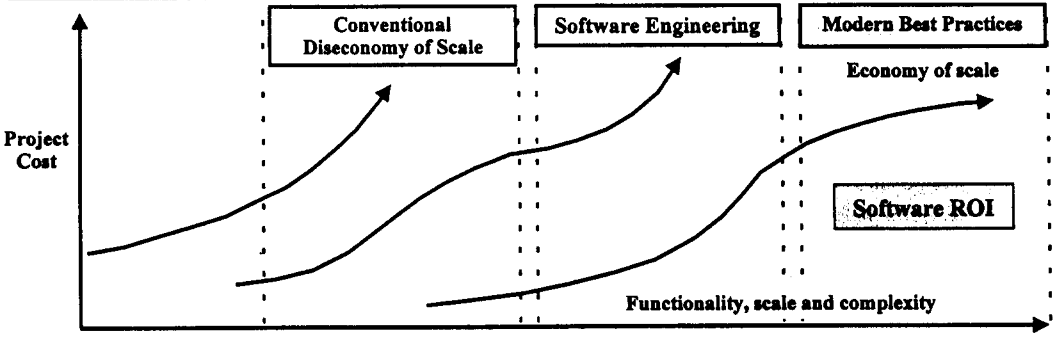
Business Process

Object-Oriented Infrastructures



Elements of an object-oriented architecture

Software Cost Evolution



Era →	60s → 70s	80s → 90s	00s
Design methods →	Functional	Declarative	Object-Oriented
Process →	Waterfall	2167A,7935	ISO12207, MIL-STD-498
Architecture →	Proprietary centralized	Proprietary distributed	Open distributed
Languages →	FORTRAN-COBOL	C-Ada	Ada 95, C++, Java
Risk focus →	Functionality	Performance	Adaptability

Technology State-of-the-Art Evolution

	Conventional	Software Engineering	Target
Environment/Tools	Custom - ad hoc	Separate but off-the-shelf	Off-the-Shelf and integrated
Size	100% Custom	30% Megaprogrammed 70% Custom	70% Megaprogrammed 30% Custom
Process/Team	Ad hoc	Repeatable	Managed & measured
Project Performance	predictable *Always* over budget over schedule	unpredictable *Infrequently* on budget on schedule	predictable Competitive budget & schedule performance

Megaprogramming

Software Technology	Language Level	Support Software
Micro programming	Bits: 100, 010 F12, A07, 124, AAF	Machine languages
Low level programming	Instructions: LDR, ADDX, CLA, JMPS	Assemblers, linkers
High level language programming	Lines: IF A then B loop I=I+1	Compilers Operating systems

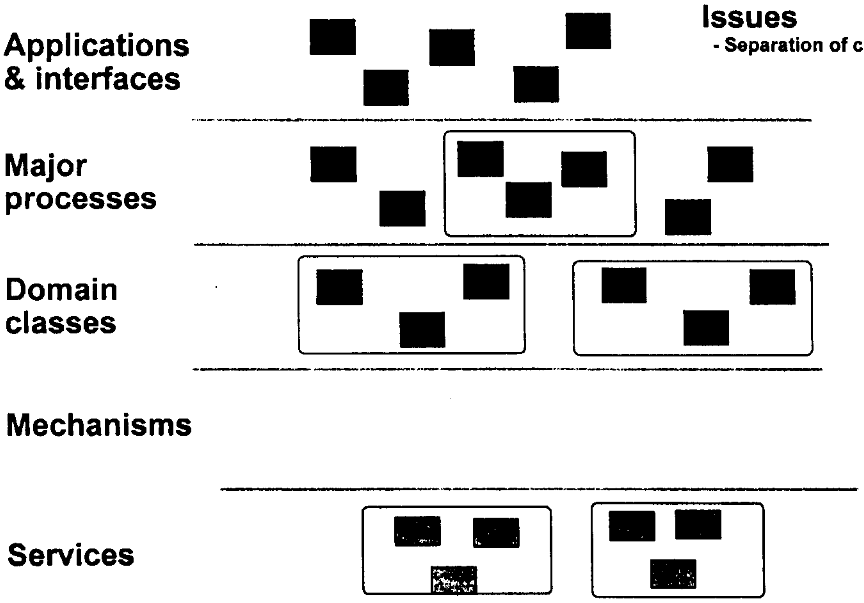
Object based and Object-oriented programming	Objects & packages:
	Type color is (red, yellow, green);
	package traffic_light
	when green go;
		Compilers
		Operating systems
		Runtime libraries

Megaprogramming - Reuse - Automatic coding - COTS components	Components & Services Overlay map with grid; When failure switchover; Shutdown all test processes;	Compilers Operating systems Runtime libraries Networks Middleware CASE tools

Hardware Engineering Analogs to Megaprogramming

	Cobol	C	Ada83	Ada95	C++	Smalltalk	Mega-programming	
System	○	○	○	○	○	○	●	Systems
Rack	○	○	○	○	○	○	●	Layers
Card	○	○	○	○	○	○	●	Categories, Subsystems, Class Libraries
LSI chip	○	○	◕	●	●	●		Classes, Objects,
SSI/ MSI chip	◔	◓	●	●	●	●		Functions, Arguments, Return values, Strong typing
Gate	◓	●	●	●	●	●		Variables, Expressions, Statements

Layered architecture



Issues

- Separation of concerns

00-04-17, p15

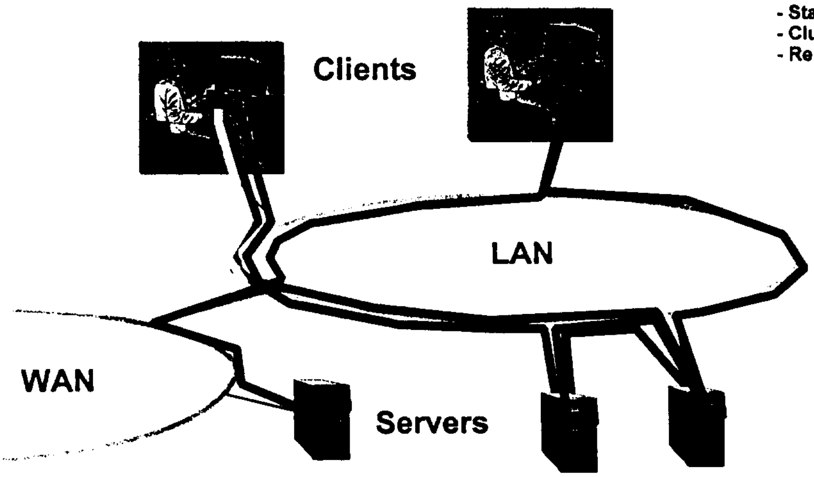
Distributed architecture

Issues

- DCOM vs EJB vs CORBA
- Distribution and migration
- Fine grain/large grain objects
- Stateless vs stateful services
- Clustering
- Replication

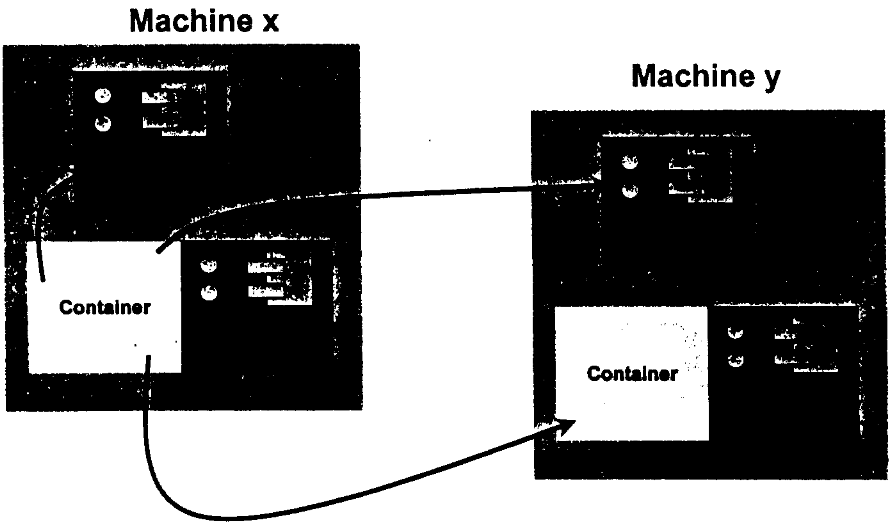
Mechanisms

- RPC
- Transaction
  - Atomicity
  - Consistency
  - Isolation
  - Durability
- Messaging
- Conversation
- Request/response
- Publish and subscribe
- Broadcast



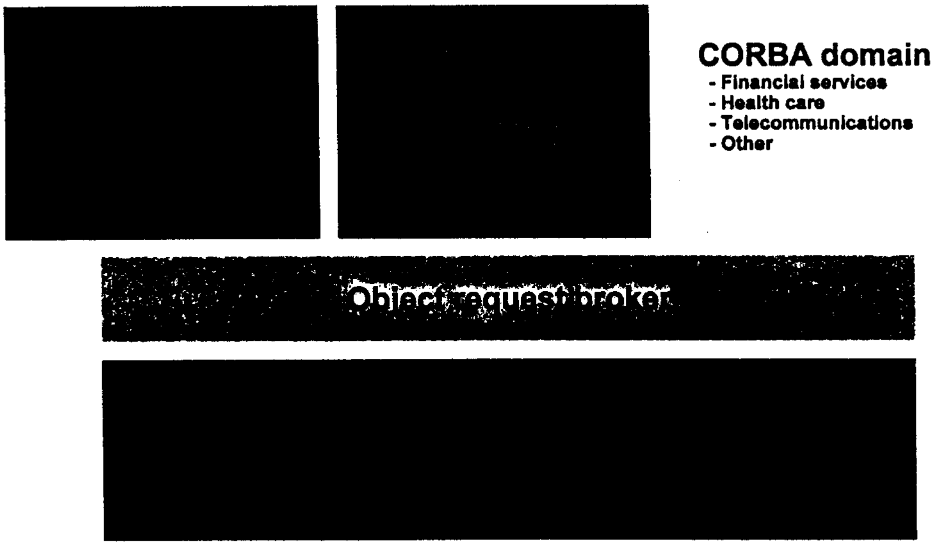
00-04-17, p16

DCOM architecture



Mowbray et al, Inside CORBA

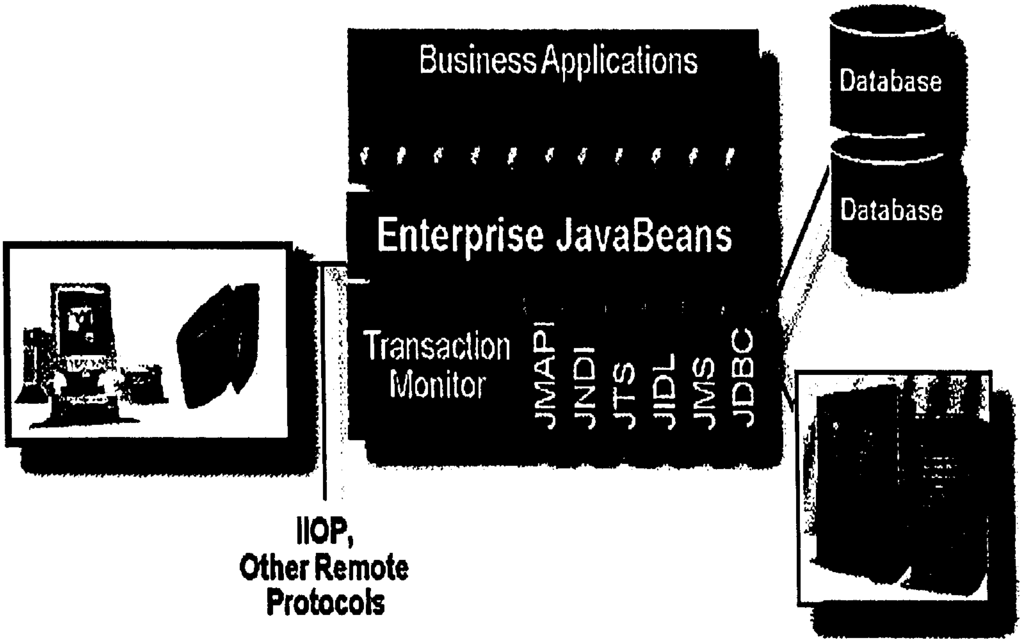
CORBA architecture



CORBA domains
- Financial services
- Health care
- Telecommunications
- Other

Bill Roth, JavaSoft
©1998, Sun Microsystems, Inc

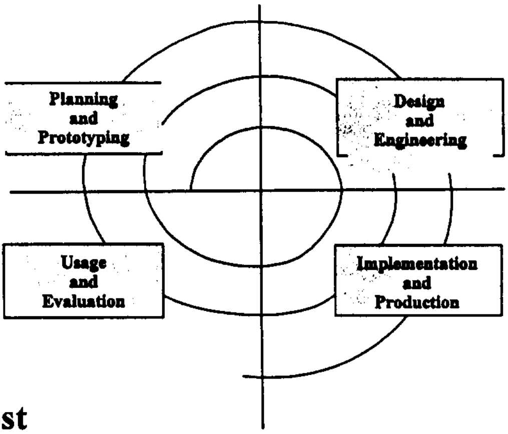
EJB architecture



Iterative Development

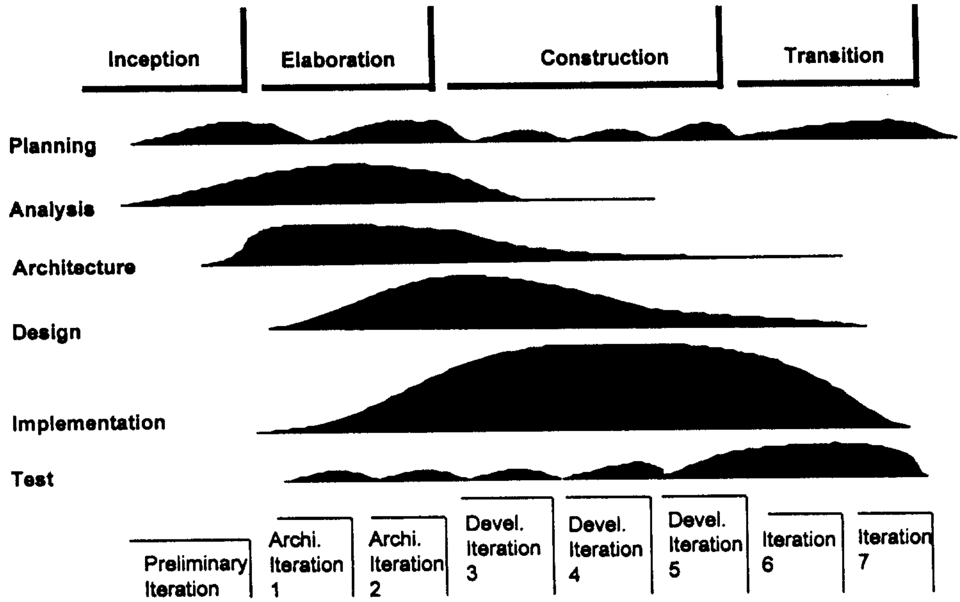
Potential

- Continuous risk management
- “software first” lifecycles
- Higher quality
- Change accommodation
- Substantially reduced lifecycle cost
- Accelerated availability of useful capabilities



Version 1 Copyright © 2000 by Dr. Lee, Nam Yong

Activities



Inception

Elaboration

Construction

Transition

Planning

Analysis

Architecture

Design

Implementation

Test

Preliminary Iteration

Archi. Iteration 1

Archi. Iteration 2

Devel. Iteration 3

Devel. Iteration 4

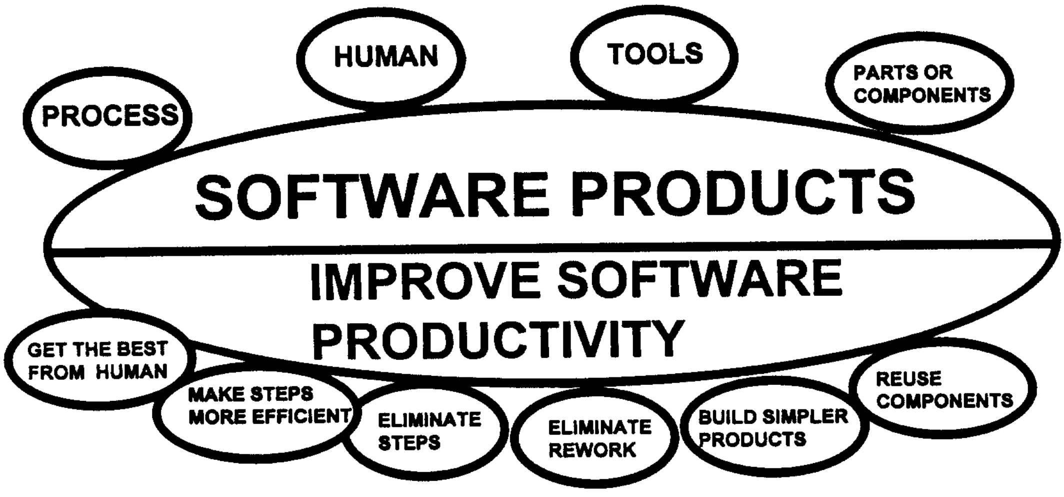
Devel. Iteration 5

Iteration 6

Iteration 7

Software Production

SOFTWARE PRODUCTION FACTORS



HUMAN

TOOLS

PROCESS

PARTS OR COMPONENTS

SOFTWARE PRODUCTS

IMPROVE SOFTWARE PRODUCTIVITY

GET THE BEST FROM HUMAN

MAKE STEPS MORE EFFICIENT

ELIMINATE STEPS

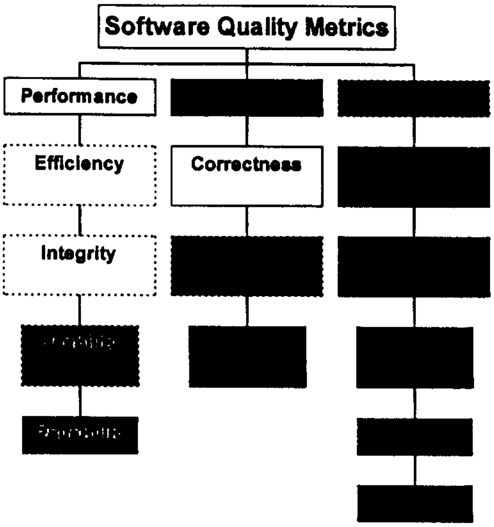
ELIMINATE REWORK

BUILD SIMPLER PRODUCTS

REUSE COMPONENTS

SOFTWARE PRODUCTIVITY IMPROVEMENT FACTORS

Software Quality Metrics



Software Quality Metrics

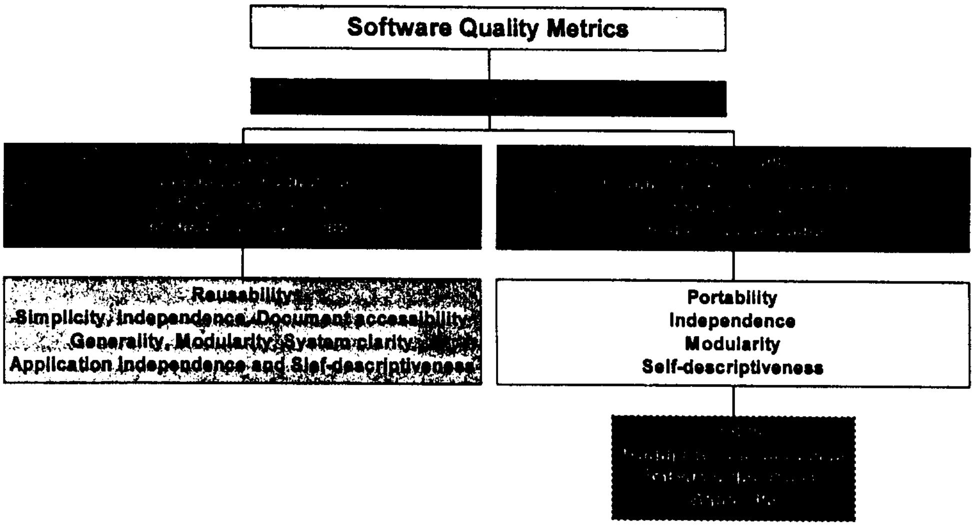
Performance

Efficiency

Integrity

Correctness

Software Quality Metrics

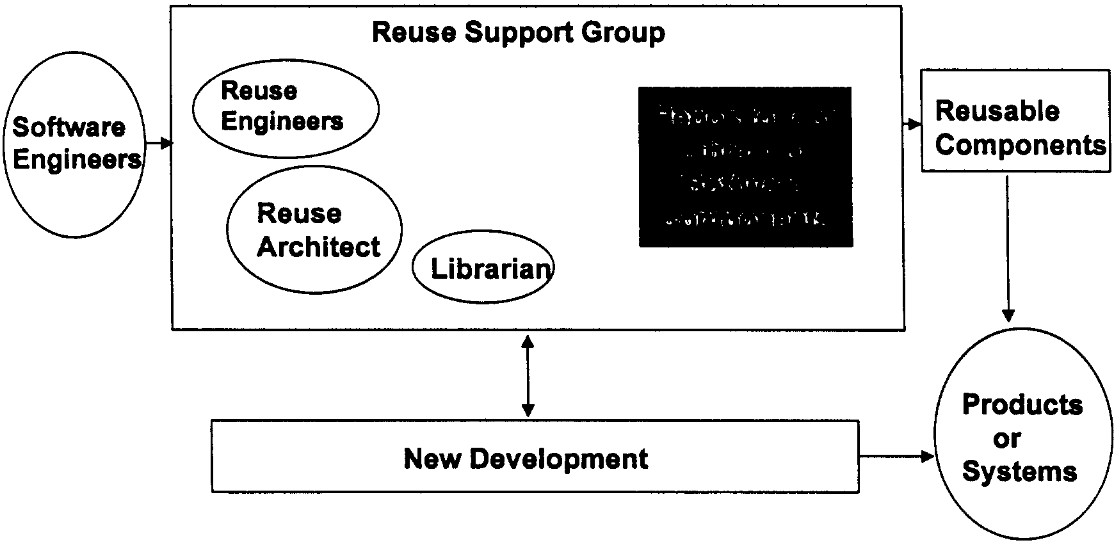


Software Quality Metrics

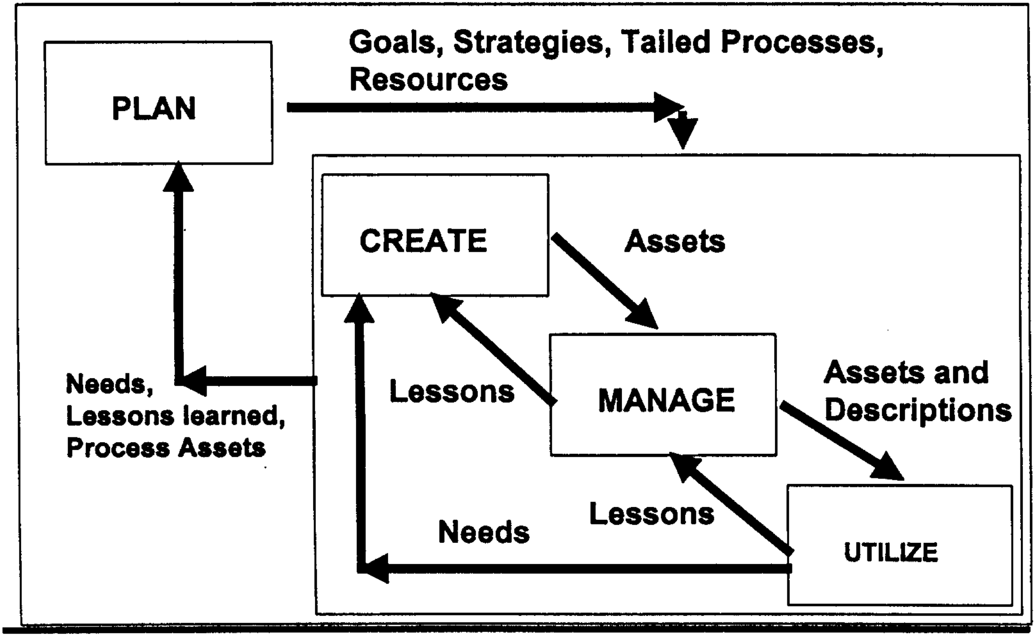
Reusability
Simplicity, Independence, Document accessibility,
Generality, Modularity, System clarity,
Application independence and Slef-descriptiveness

Portability
Independence
Modularity
Self-descriptiveness

Software Reuse Process Model



Process Reuse Framework


PLAN
Goals, Strategies, Tailed Processes, Resources
CREATE
Assets
MANAGE
Lessons
Assets and Descriptions
UTILIZE
Lessons
Needs
Needs, Lessons learned, Process Assets

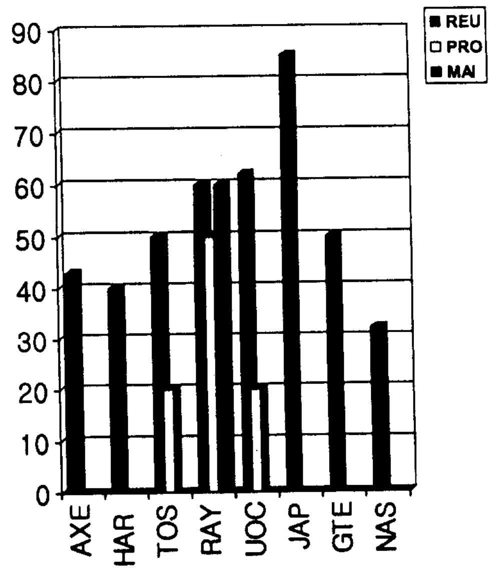
Dominant Domain Analysis Techniques

Criteria	FODA	Jaworski	Prieto-Diaz	Synthesis
Definition of Domain	Application area	Business area	Application area	Business area
How to determine problems?	Top-down	Top-down	Bottom-up	Top-down
Products of Domain Analysis(DA)	Canonical Architecture	Domain Knowledge base	Learning more about immature domains, discorvering facts about a domain	Products for application engineering
Permanence of DA results	Permanent	Mutable	Permanent	Mutable
Relation to the S/W Devel. Process	Prerequirements	Prerequirements, Water-fall model	Prerequirements	Meta-process yielding appl. eng. process
Focus of analysis	Decisions	Objectsand operations	Objects and operations	Decisions
Paradigm of problem Space Models	Decision model and generic requirements	Generic requirements	Generic requirements	Decisions model
Purpose and Nature of Domain Models	Specification for software prodects	Repository of domain knowledge	Specofocation for software prodects	Specification for software process, prodects, environment
Oranizational Model of Domains and Projects	Not specified	Not specified	Not specified	Projects are components of a domain organization
Approach to Reuse	Opportunistic	Systematic	Opportunistic	Systematic
Focus of Formalization	Formalizing canonical models	Formalizing canonical medels	Formalizing objects and operations	Formalizing canonical models
Primary Product of Domain Development	Reuse library	Reuse library	Reuse library	Application engineering Process

Studies on Software Reuse

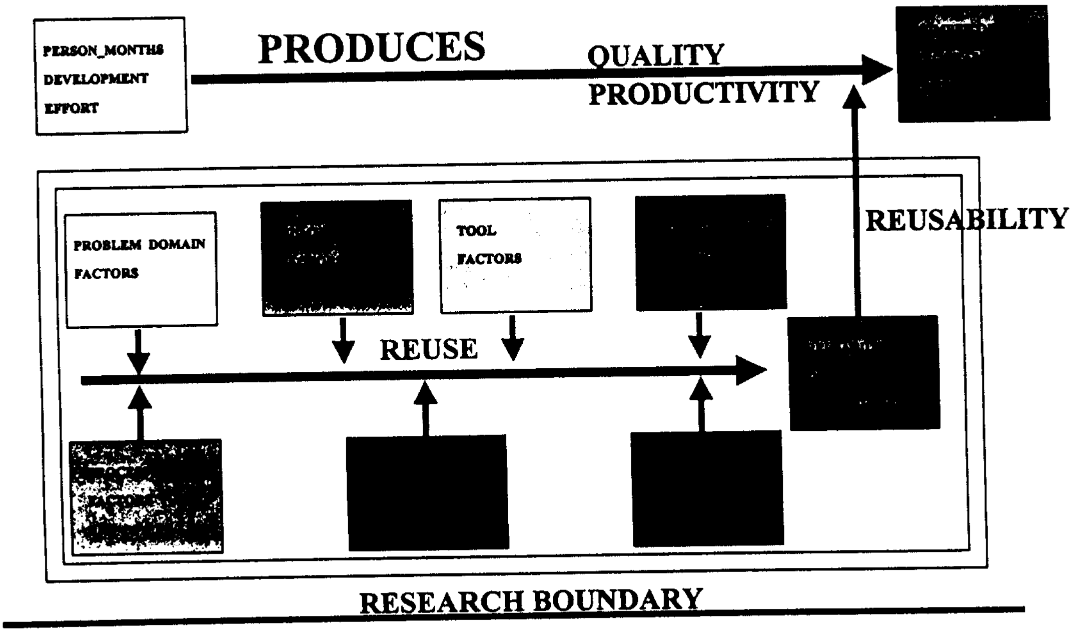
- Software Repository (16%)
- Programming Language (15%)
- Program-Oriented (17%)
- System-Oriented (15%)
- Conceptual Studies (26%)
- Empirical Studies (11%)

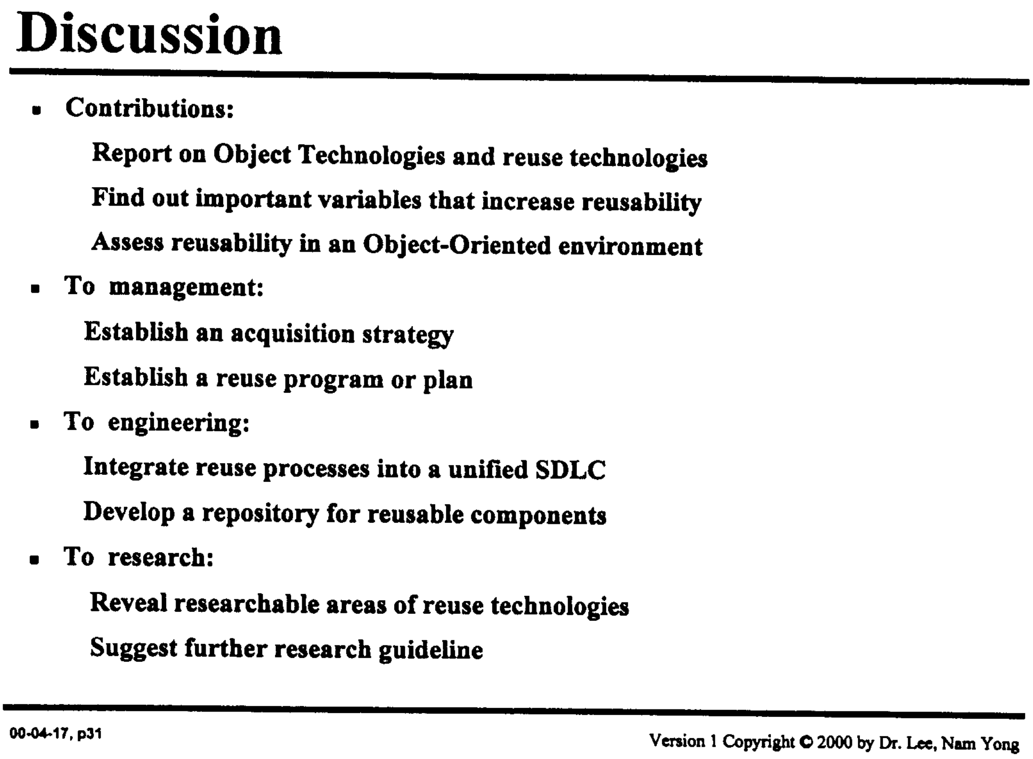
Empirical Studies on Software Reuse



- LEGEND
  - REU: Reusability
  - PRO: Productivity
  - MAI: Maintenability
- GROSSORY
  - AXE: AXE developed by Erisson Telecom in Sweden(Oskarsson, 1983)
  - HAR: Hartford(Cavaliere, 1983)
  - TOS: Toshiba's Software Factory(Matsumoto, 1984)
  - RAY: Raytheon's Missile Systems Division(Lanergan & Grasso, 1984)
  - UOC: University of California at Irvine(Standish, 1984)
  - JAP: Japanese Software Factory(Standish, 1984)
  - GTE: GTE Data Services(McClure, 1992)
  - NAS: NASA(Selby, 1987)

Research Framework





Discussion

- Contributions:

  Report on Object Technologies and reuse technologies

  Find out important variables that increase reusability

  Assess reusability in an Object-Oriented environment

- To management:

  Establish an acquisition strategy

  Establish a reuse program or plan

- To engineering:

  Integrate reuse processes into a unified SDLC

  Develop a repository for reusable components

- To research:

  Reveal researchable areas of reuse technologies

  Suggest further research guideline

00-04-17, p31