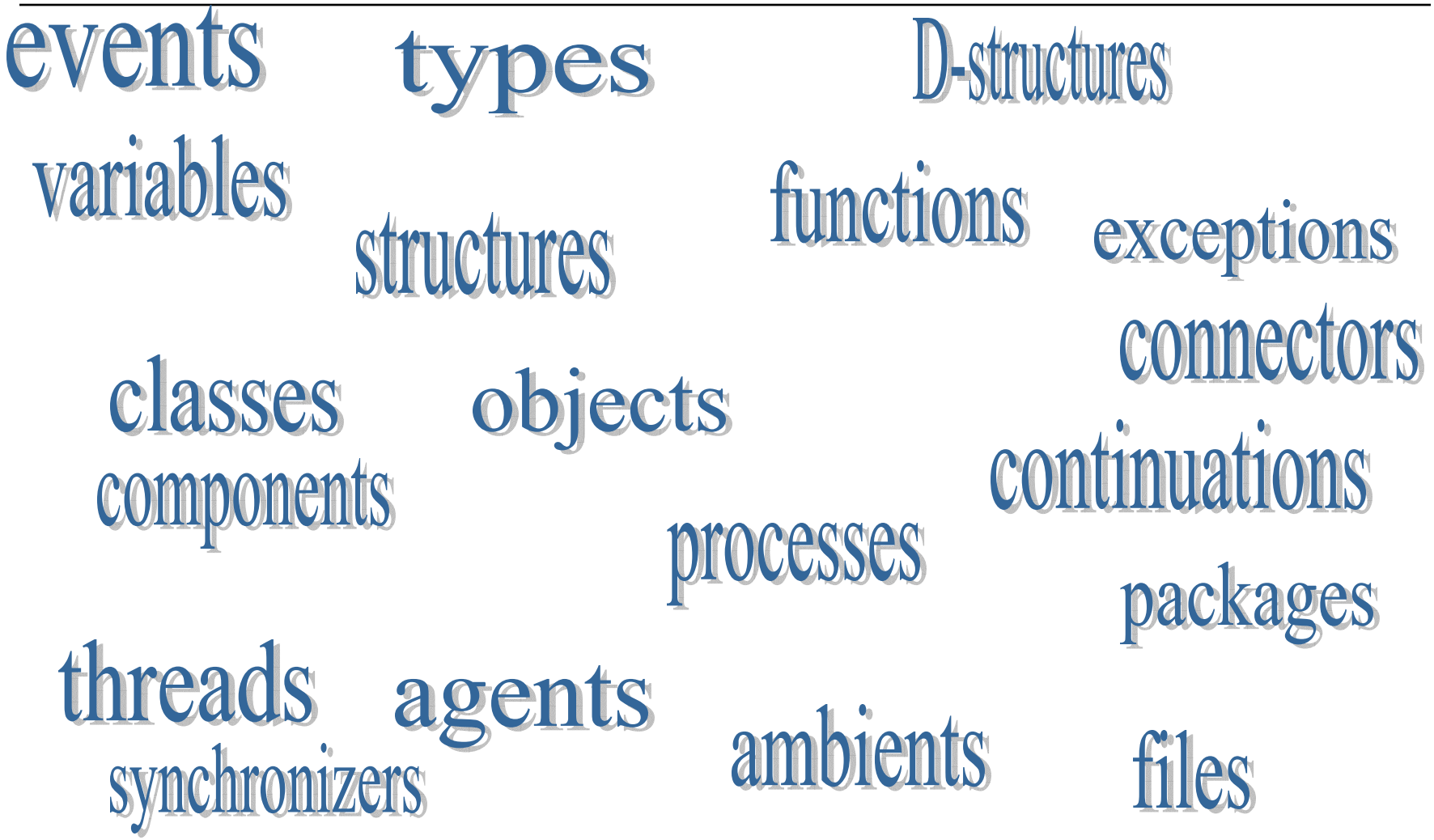
Aspect Orientation: An introduction

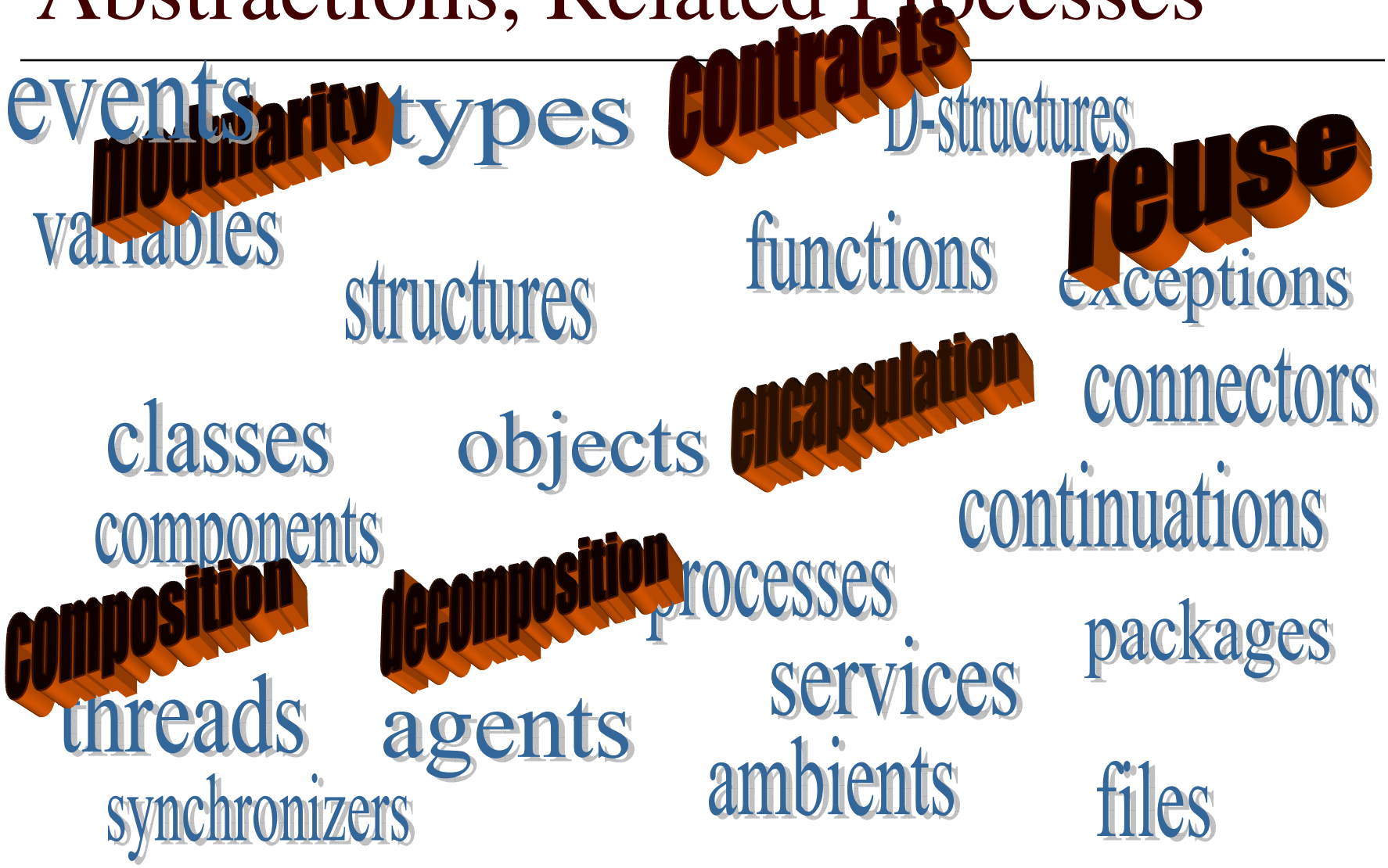
A Tutorial conducted at NCOOT-2004

-R. K. Joshi

Abstractions



Abstractions, Related Processes



Abstractions, Related Processes and Properties

Is this space enough for today's computations?

- Maybe enough
- but …
- Do you have a clean organized view of all aspects of your software that is traceable from architecture to implementations?
- Do you maximize reuse?
- And Eliminate All Redundancies?
  - we may be are asking for too much, but let's make a beginning…

- Separation of concerns?

- Separation of concerns is fine, but…

- Separation of concerns is fine, but…where is the integration?

- The key is to separate but be integrated

Let's Take a look at Some Research Results

- Code redundancies reported (an old research)
  - Application projects: 75%
  - System programs: 50%
  - Telecommunication projects: 70%
- Reengineering projects find redundancies and eliminate them: 20-50%
- A latest study: 60% code in one Java class library was redundant

Is it easy to say ‘Eliminate Redundancies’?

- So how will you eliminate redundancies and still address the concerns handled by them?

- Can’t I keep one copy of the redundant code and simply use it as a black box using conventional techniques?
  - Not always! – It depends on your technology choices

Programming paradigms influence the way we organize software

- They provide rules on structuring
- They provide varying flexibility
- They provide ways to represent organized logical ideas into organized physical manifestations

- We select paradigms based on our needs
- New paradigms open up new possibilities
  - Opcodes -- assembly language -- fortran – lists – logic programming --- procedural programming – functional programming -- object oriented – distributed – parallel – real time – constraint based – event based – service oriented – component oriented …

Single abstraction languages

- Limitations on what you can express without cross-cutting

- E.g.
  - invariants across objects in OOPLs

  - Whenever function pop() is invoked, print the return value to a file

Most design Pattern implementations cannot be reused as it is

- E.g. singleton:
  - Private constructor
  - A class variable
  - A class method

- Method names are specific and cannot be reused as there is no way to rename method names automatically

Adding persistence to your executing process

- Is it possible to express this requirement on top of an executing process and simply turn it into a persistent image?

- What about an object oriented program executable in Java/C++ runtime environment?

Since Programming paradigms influence the way we organize software

- The problem can be attacked at programming level
  - But that's not done..

There are models too!

- The problem can be attacked at programming level
- And also at modeling level
- Architecture and even at Requirements level

Pattern language Approach

Pattern descriptions in keywords related to patterns

+

Application specific code

→ code for one pattern

But the problem is that patterns do not occur isolated!

The AOP approach

- Express each of the system's aspect in a separate and natural form
  - Capture aspects
- Then automatically combine these forms into one executable form
  - Aspect Weaving

- Thus focus in AOP approaches is on ‘Expression’ : Separate but meant for integration
  - Code tangling
    - (mixup of multiple concerns at one place)
  - Code scattering
    - (a concern is scattered over many places)
- Execution environments take care of the actual interactions/integration

AOP constructs summary

- Join points: A point in a source program
  - Method call
  - Constructor call
  - Variable read/write
  - Exception handler
  - Variable initializer
  - Destructor

AOP constructs summary

- Point Cuts: a set of join points + optionally some of the execution context values
  - Call (void Point.setX(int))
    - A call to a specific function
  - Call (public * Figure.*(..))
    - Calls to all public functions on Figure
  - Pointcut move: call … || call …
    - Any of the above calls
  - !instanceof (X) && call …
    - Call originates not from instance of X and to specified method

AOP constructs summary

- Advices: that is executed at the code at joinpoints for given pointcuts
  - Before advice
    - After reaching a join point, but before the computation proceeds
  - After advice
    - After the computation at join point has completed
  - Around advice
    - Run first. Proceed() inside around advice makes the computation proceed
  - After returning
  - After throwing
- Introductions: add new fields to classes, change relationships

Singletone again

- Is it possible to capture the singleton requirement on many classes at one place?
  - Possible in an AOP paradigm

Some more examples

- Aspects in a distributed objects domain
  - Object's functionality
  - It's location
  - It's itinerary
  - Communication and synchronization
  - Its persistence
  - Its security

Some more examples

- Aspects in an OS kernel
  - Process related attributes
  - Scheduling algorithm
  - Memory allocation policies
  - Memory allocated to a process
  - Locality of reference policies
- Will modifying one need modification of the other?

Early Aspects

- Crosscutting properties at requirements and architectural level
  - Security
  - Deadlines
  - Persistence
  - Mobility
  - Replication

Aspect Orientation in middleware

- Write objects in your application first
- Add on services to the application later
  - Use AOP techniques (interceptor/static transformation) techniques

Feature interaction problem

- Effects of one aspect may interfere with that of another
- Careful ordering of aspect application is important

Product Line Approaches

High level transformation code

+

High level Base code

→ Actual Variant

Base-Meta Separation

- Meta-object protocols
- Reflection
  - Ideas are quite old
  - Some of the recent technologies have discovered them only now!

Filter Objects Approach

- Message based paradigm
- Based on interfaces and capture on messages
- Dynamic and First class aspects
- Pluggability at runtime
- Weaving not possible
- Filter objects for C++/Java/CORBA/COM, patterns, configurations

Open Problems

- Static vs. Dynamic aspects
  - Commercial Tools and Technologies are picking up
- Early aspects and traceability into code
- Aspects in processes
- Large scale applications and actual practice
- Impact on Systems design and software Engineering lifecycle in general
- Impact on Modeling Languages
- Formal Models