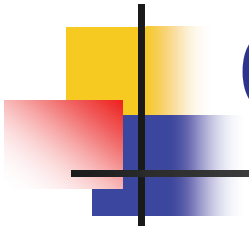
The Power of Abstraction

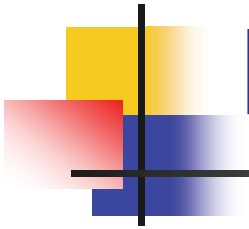
Barbara Liskov

October 2009



Outline

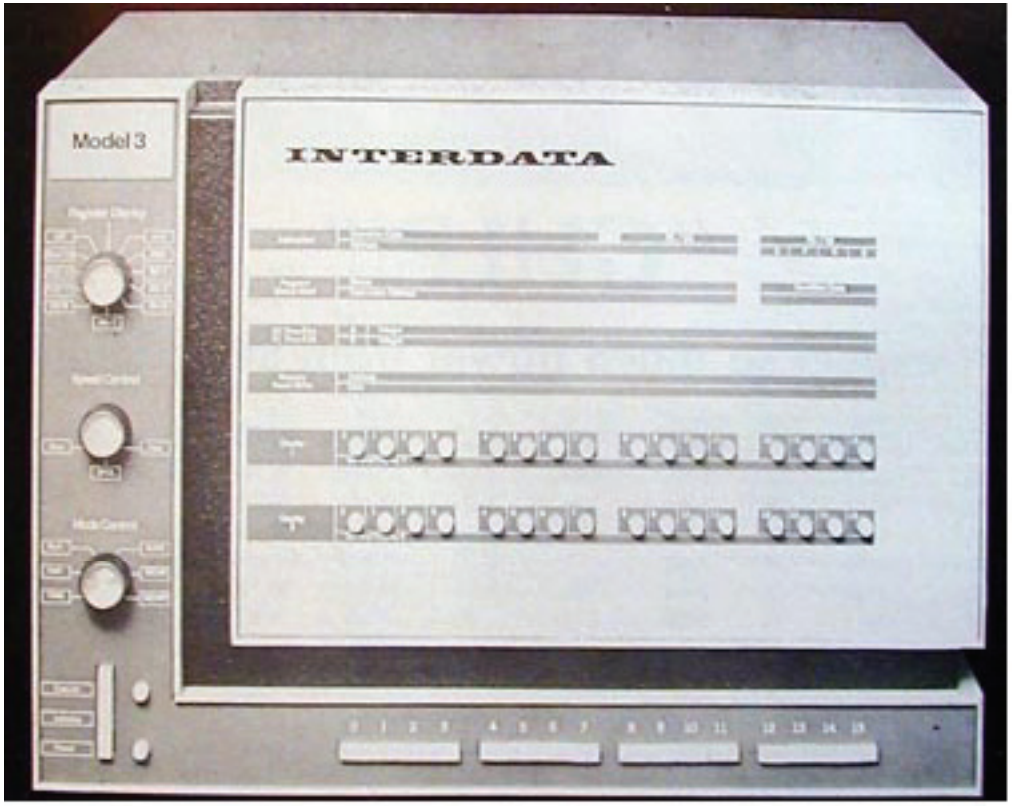
- Inventing abstract data types
- CLU
- Type hierarchy
- What next

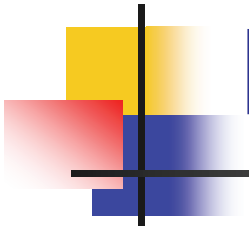


Data Abstraction Prehistory

- The Venus machine

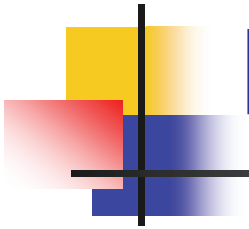
The Interdata 3





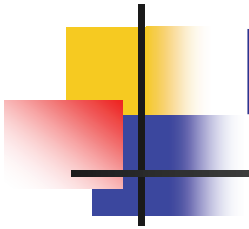
Data Abstraction Prehistory

- The Venus machine
- The Venus operating system



Data Abstraction Prehistory

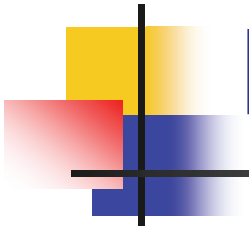
- The Venus machine
- The Venus operating system
- Programming methodology



Programming Methodology

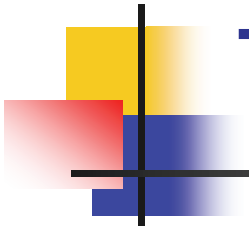
- The software crisis!
  - Machines were getting cheaper
  - And bigger/faster

- E. W. Dijkstra. The Humble Programmer. Cacm, Oct. 1972



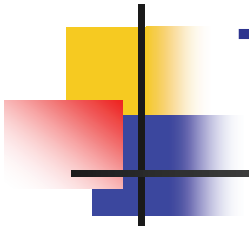
Programming Methodology

- How should programs be designed?
- How should programs be structured?



The Landscape

- E. W. Dijkstra. Go To Statement Considered Harmful. Cacm, Mar. 1968



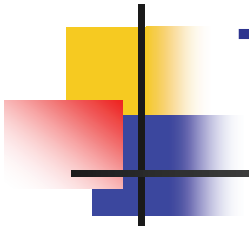
The Landscape

- N. Wirth. Program Development by Stepwise Refinement. Cacm, April 1971

The Landscape

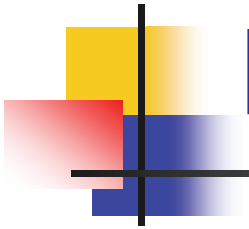
- D. L. Parnas. Information Distribution Aspects of Design Methodology. IFIP Congress, 1971

- “The connections between modules are the assumptions which the modules make about each other.”



The Landscape

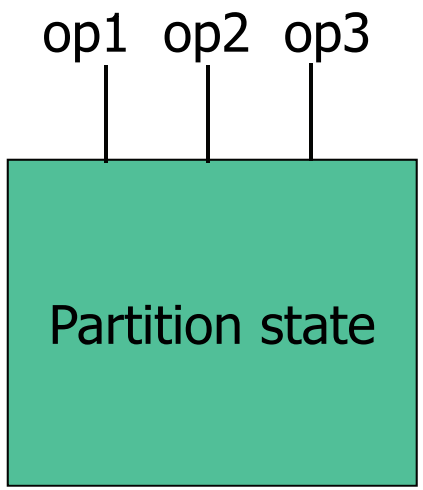
- D. L. Parnas, On the Criteria to be used in Decomposing Systems into Modules. Cacm, Dec. 1972

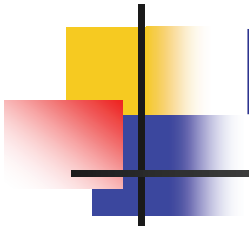


Partitions

- B. Liskov. A Design Methodology for Reliable Software Systems. FJCC, Dec. 1972

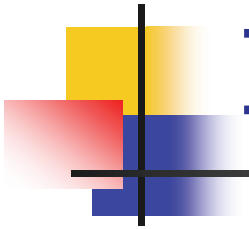
Partitions





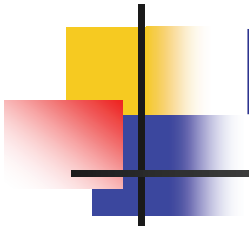
From Partitions to ADTs

- How can these ideas be applied to building programs?



Idea

- Connect partitions to data types



Meeting in Savanah

- ACM Sigplan-Sigops interface meeting. April 1973. (Sigplan Notices, Sept. 1973)
- Started to work with Steve Zilles

The Landscape

- Extensible Languages
  - S. Schuman and P. Jourrand. Definition Mechanisms in Extensible Programming Languages. AFIPS. 1967
  - R. Balzer. Dataless Programming. FJCC 1967

The Landscape

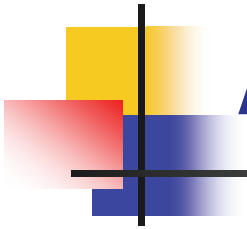
- O-J. Dahl and C.A.R. Hoare. Hierarchical Program Structures. Structured Programming, Academic Press, 1972

The Landscape

- J. H. Morris. Protection in Programming Languages. Cacm. Jan. 1973

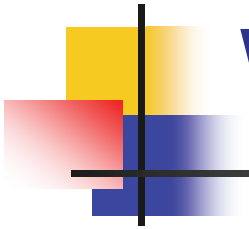
The Landscape

- W. Wulf and M. Shaw. Global Variable Considered Harmful. Sigplan Notices. Feb. 1973.



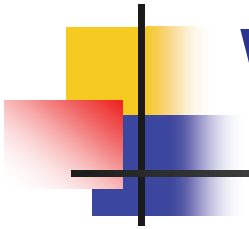
Abstract Data Types

- B. Liskov and S. Zilles. Programming with Abstract Data Types. ACM Sigplan Conference on Very High Level Languages. April 1974



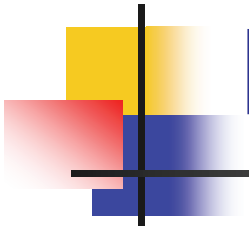
What that paper proposed

- Abstract data types
  - A set of operations
  - And a set of objects
  - The operations provide the only way to use the objects



What that paper proposed

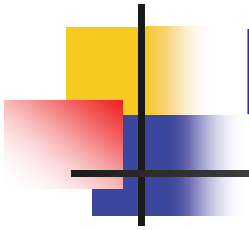
- Abstract data types
  - Clusters with encapsulation
- Polymorphism
- Static type checking (we hoped)
- Exception handling



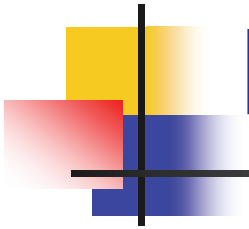
From ADTs to CLU

- Participants
  - Russ Atkinson
  - Craig Schaffert
  - Alan Snyder

Why a Programming Language?

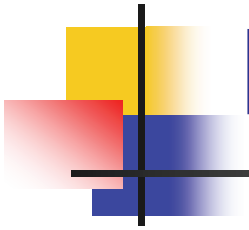


- Communicating to programmers
- Do ADTs work in practice?
- Getting a precise definition
- Achieving reasonable performance



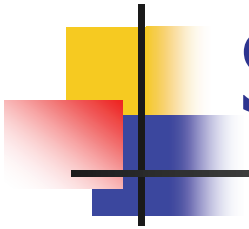
Language Design

- Goals
  - Ease of use
  - Simplicity
  - Expressive power
  - Performance



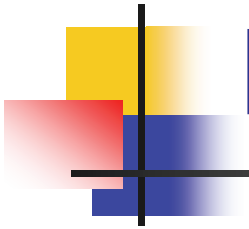
Language Design

- More goals
  - Minimality
  - Uniformity
  - Safety



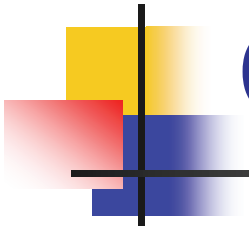
Some Assumptions/Decisions

- Heap-based with garbage collection!
- No block structure!
- Separate compilation
- Static type checking



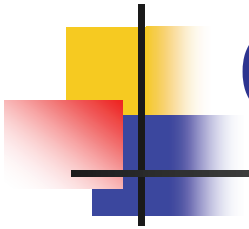
More Assumptions/Decisions

- No concurrency
- No go tos
- No inheritance



CLU Mechanisms

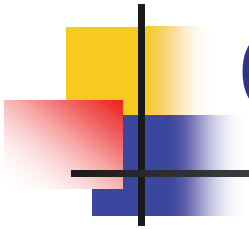
- Clusters
- Polymorphism
- Exception handling
- Iterators



Clusters

IntSet = cluster is create, insert, delete, isIn, ...

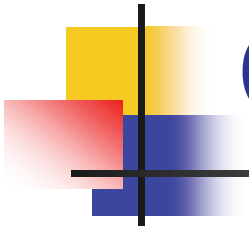
end IntSet



Clusters

```
IntSet = cluster is create, insert, delete, ...
end IntSet
```

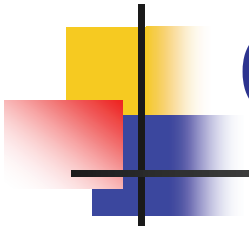
```
IntSet s := IntSet$create( )
IntSet$insert(s, 3)
```



Clusters

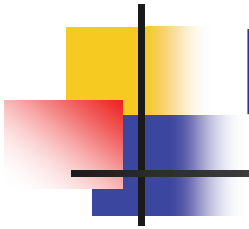
IntSet = cluster is create, insert, delete, ...

rep = array[int]



Clusters

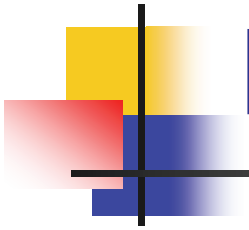
```
IntSet = cluster is create, insert, delete, ...

    rep = array[int]

    create = proc ( ) returns (cvt)
        return (rep$create( ))
        end create
```



Polymorphism

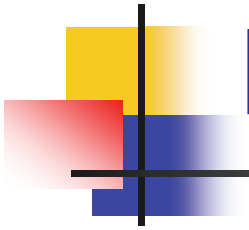
```
Set = cluster[T: type] is create, insert, …
end Set
```

```
Set[int] s := Set[int]$create( )
Set[int]$insert(s, 3)
```



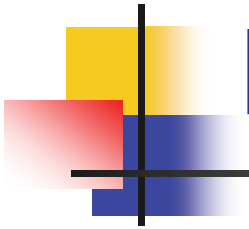
Polymorphism

```
Set = cluster[T: type] is create, insert, …
    where T has equal: proctype(T, T)
        returns (bool)
```



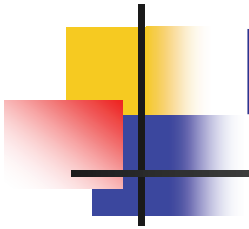
Polymorphism

```
Set = cluster[T: type] is create, insert, …
   where T has equal: proctype(T, T)
      returns (bool)

   rep = array[T]

   insert = proc (x: cvt, e: T)
      … if e = x[i] then …
```



Exception Handling

- J. Goodenough. Exception Handling: Issues and a Proposed Notation. Cacm, Dec. 1975
  - Termination vs. resumption
  - How to specify handlers

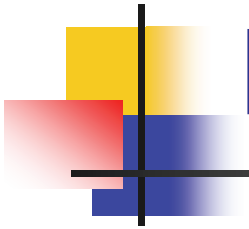


Exception Handling

```
choose = proc (x: T) signals (empty)
    if rep$size() = 0 then signal empty
    ...
```

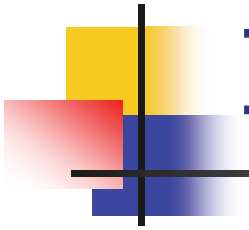
Exception Handling

```
choose = proc (x: T) signals (empty)
    if rep$size() = 0 then signal empty
    ...

set[T]$ choose()
    except when empty: ...
```



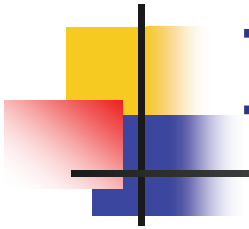
Exception Handling

- Handling
- Propagating
- Shouldn’t happen
  - The failure exception
- Principles
  - Accurate interfaces
  - Avoid useless code



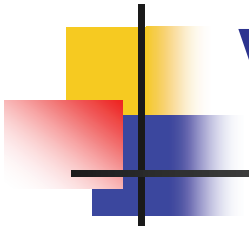
Iterators

- For all x in C do S



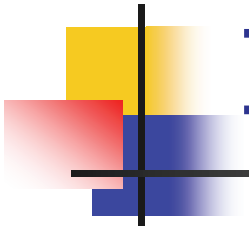
Iterators

- For all x in C do S
  - Destroy the collection?
  - Complicate the abstraction?



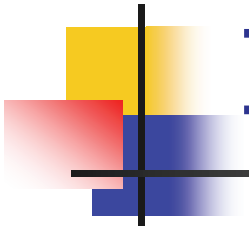
Visit to CMU

- Bill Wulf and Mary Shaw, Alphard
- Generators



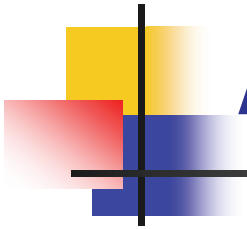
Iterators

```
sum: int := 0
for x: int in Set[int]$members(s) do
    sum := sum + x
    end
```



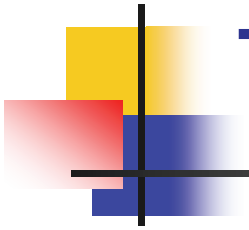
Iterators

```
Set = cluster[T] is create, ..., members, ...

      rep = array[T]

      members = iter (x: cvt) yields (T)
         for z: T in rep$elements(x) do
            yield (z) end
```



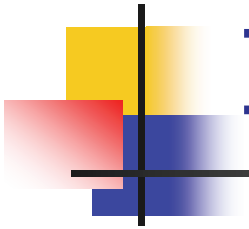
After CLU

- Argus and distributed computing
- Type Hierarchy



The Landscape

- Inheritance was used for:
  - Inplementation
  - Type hierarchy



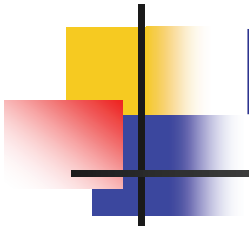
Implementation Inheritance

- Violated encapsulation!

Type hierarchy

- Wasn’t well understood
- E.g., stacks vs. queues

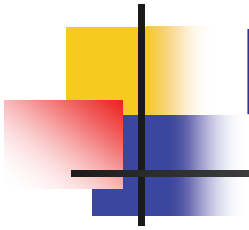
The Liskov Substitution Principle (LSP)



- Objects of subtypes should behave like those of supertypes if used via supertype methods

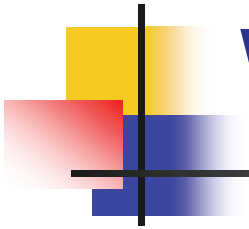
- B. Liskov. Data abstraction and hierarchy. Sigplan notices, May 1988

Polymorphism



- where T has ... vs.
- where T subtype of S

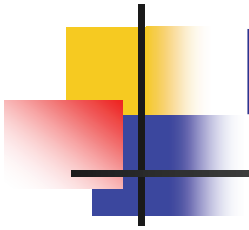
- Proofs happen at different times!



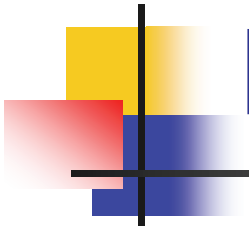
What Next?

- Modularity based on abstraction is the way things are done

Modern Programming Languages



- Are good!



Missing Features

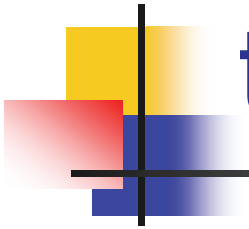
- Procedures are important
- And closures and iterators
- Exception handling
- Built-in types

- Can’t we do better for “serialization”?

The state of programming

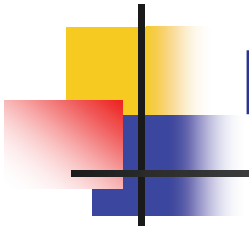
- Isn’t great!
- Especially web services and browsers
- Return of globals

An aside: persistent storage typically violates abstraction

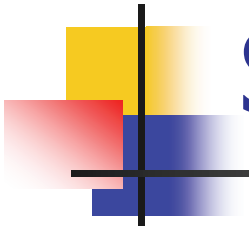


- E.g., a file system or a database
- It's a big space of globals!
- Without support for encapsulation
- An object store would be much better
  - Automatic translation
  - Type preservation

Programming language research

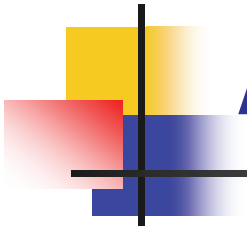


- New abstraction mechanisms?
- Concurrency
  - Multi-core machines
- Distributed systems?



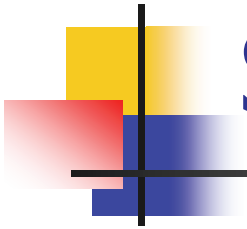
Systems research

- Has done well
  - Distributed hash tables
  - Map-reduce
  - Client/server
  - Distributed information flow
  - …



A Concern

- Performance isn't the most important issue
  - vs. semantics
  - vs. simplicity
- E.g., one-copy consistency
  - Failures should be catastrophes



Systems Problems

- Internet Computer
  - Storage and computation
  - Semantics, reliability, availability, security
- Massively parallel machines

The Power of Abstraction

Barbara Liskov

October 2009