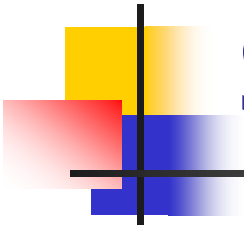
The Power of Abstraction

Barbara Liskov

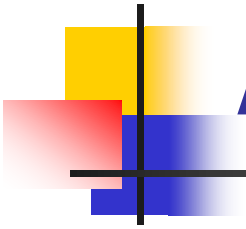
March 2013

MIT CSAIL



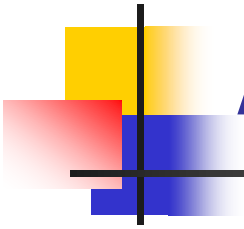
Software is Complex

- Systems are big
- and they do complicated things
- and they may be distributed and/or concurrent



Addressing Complexity

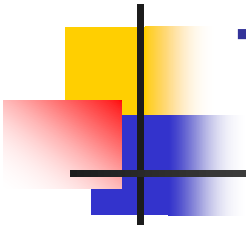
- Algorithms, data structures, protocols



Addressing Complexity

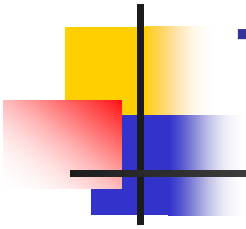
- Algorithms, data structures, protocols

- Programming methodology
- Programming languages



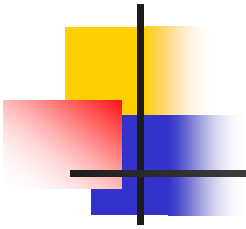
This Talk

- Programming methodology as it developed
- Programming languages
- Programming languages today



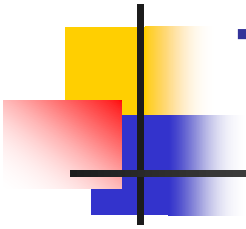
The Situation in 1970

- The software crisis!



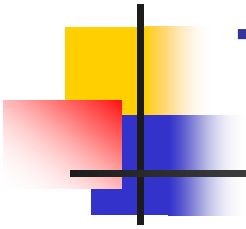
Programming Methodology

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



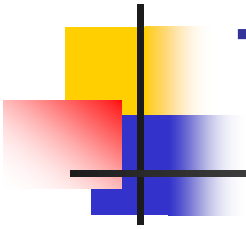
The Landscape

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



The Landscape

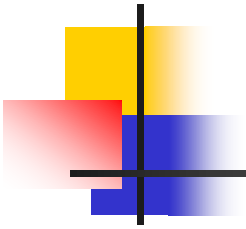
- N. Wirth. Program Development by Stepwise Refinement. Cacm, April 1971



The Landscape

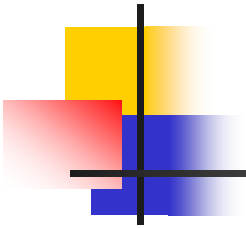
- D. L. Parnas. Information Distribution Aspects of Design Methodology. IFIP Congress, 1971

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



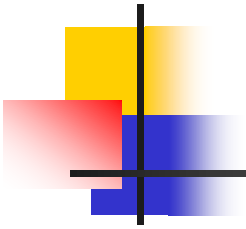
Modularity

- A program is a collection of modules



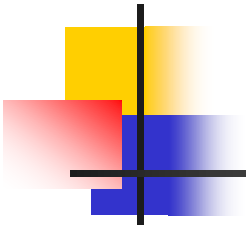
Modularity

- A program is a collection of modules
- Each module has an interface, described by a specification



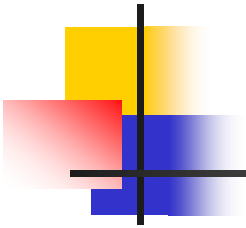
Modularity

- A program is a collection of modules
- Each has an interface, described by a specification
  - A module's implementation is correct if it meets the specification
  - A using module depends only on the specification



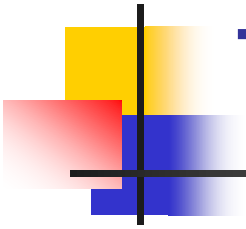
Modularity

- A program is a collection of modules
- Each has an interface, described by a specification
  - A module's implementation is correct if it meets the specification
  - A using module depends only on the specification
- E.g. a sort routine sort(a)



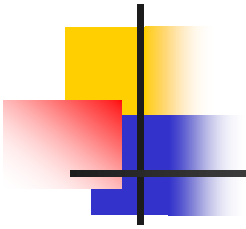
Benefits of Modularity

- Local reasoning
- Modifiability
- Independent development



The Situation in 1970

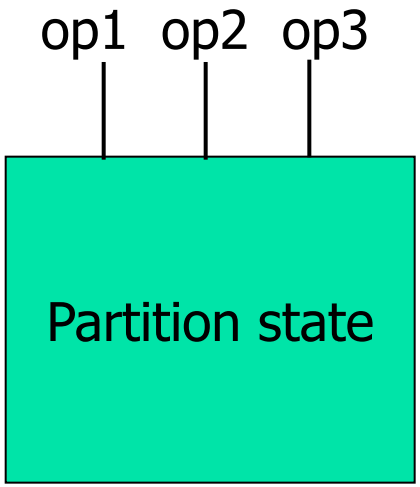
- Procedures were the only type of module
- Not powerful enough, e.g., a file system
- Not used very much
- Complicated connections

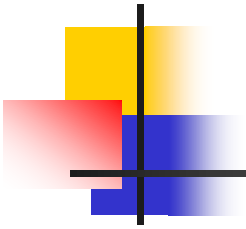


Partitions

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

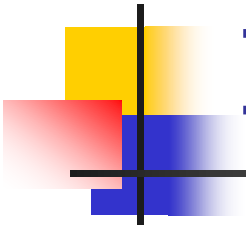
Partitions





From Partitions to ADTs

- How can these ideas be applied to building programs?

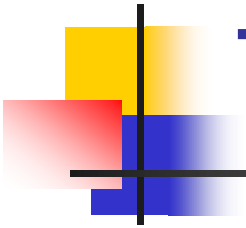


Idea

- Connect partitions to data types

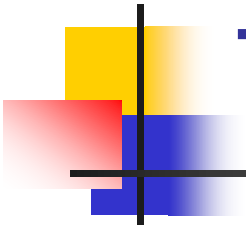
Meeting in Savanah

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



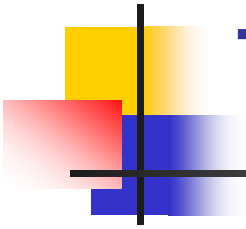
The Landscape

- Extensible Languages
  - S. Schuman and P. Jourrand. Definition Mechanisms in Extensible Programming Languages. AFIPS. 1970
  - R. Balzer. Dataless Programming. AFIPS. 1967



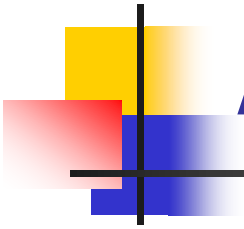
The Landscape

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



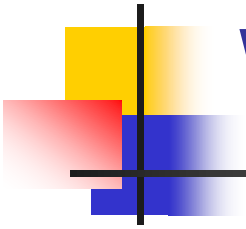
The Landscape

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



Abstract Data Types

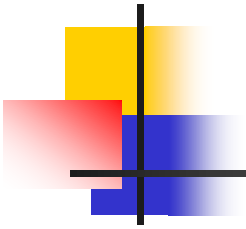
- B. Liskov and S. Zilles. Programming with Abstract Data Types. ACM Sigplan Conference on Very High Level Languages. April 1974



What that paper proposed

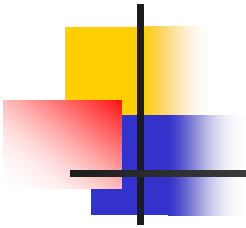
- Abstract data types
  - A set of operations
  - And a set of objects
  - The operations provide the only way to use the objects
- A sketch of a programming language

From ADTs to CLU

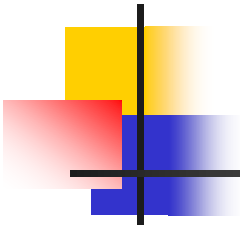


- Participants
  - Russ Atkinson
  - Craig Schaffert
  - Alan Snyder

Why a Programming Language?

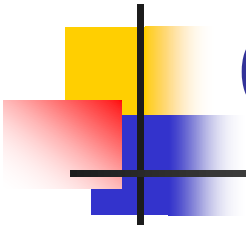


- Communicating to programmers
- Do ADTs work in practice?
- Getting a precise definition
- Achieving reasonable performance



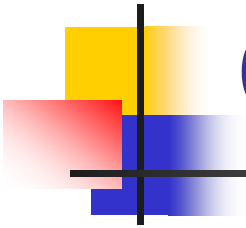
Some Facts about CLU

- Static type checking
- Heap-based
- Separate compilation
- No concurrency, no gotos, no inheritance



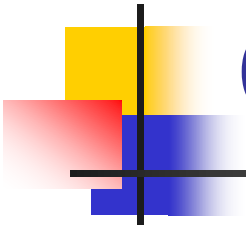
CLU Mechanisms

- Clusters
- Polymorphism
- Iterators
- Exception handling



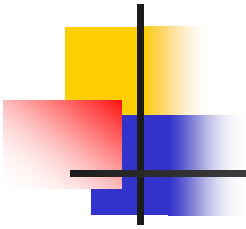
Clusters

```
IntSet = cluster is create, insert, delete, …
    % representation for IntSet objects
    % implementation of the operations
end IntSet
```



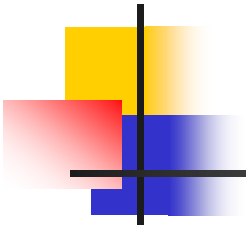
Clusters

```
IntSet = cluster is create, insert, delete, …
    % representation for IntSet objects
    % implementation of the operations
end IntSet


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



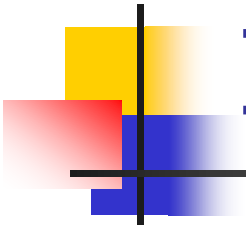
Polymorphism

```
Set = cluster[T: type] is create, insert, …
  % representation for Set object
  % implementation of Set operations
end Set

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



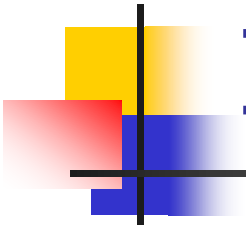
Polymorphism

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



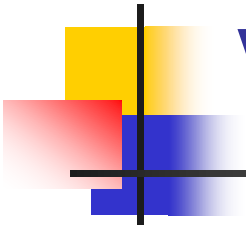
Iterators

- For all x in C do S



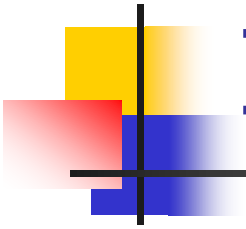
Iterators

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



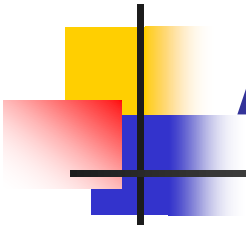
Visit to CMU

- Bill Wulf and Mary Shaw, Alphard
- Generators



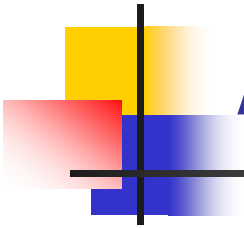
Iterators

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



Also

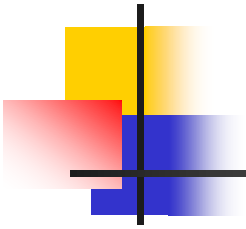
- Exception handling
  - Strong specifications, e.g., IntSet$choose
- First class Procedures and Iterators



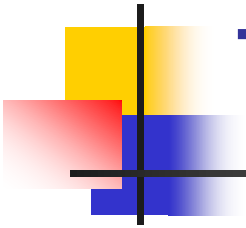
After CLU

- Argus and distributed computing
- Programming methodology
  - Modular program design
  - Reasoning about correctness
  - Type hierarchy

From CLU to Object-Oriented Programming

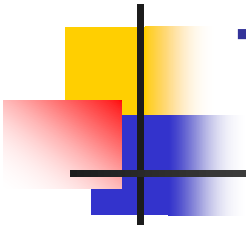


- SmallTalk provided inheritance



The Landscape

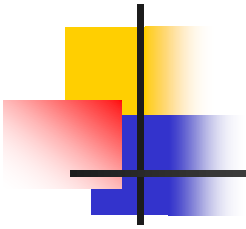
- Inheritance was used for:
  - Implementation
  - Type hierarchy



Type Hierarchy

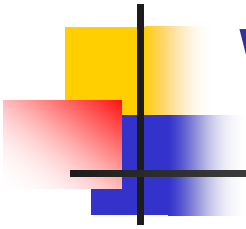
- Wasn’t well understood
  - E.g., stacks vs. queues

The Liskov Substitution Principle (LSP)



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

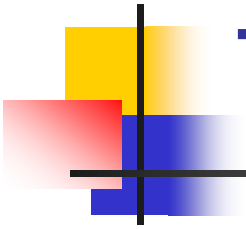
- B. Liskov. Data abstraction and hierarchy. Sigplan notices, May 1988



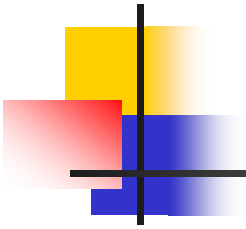
What Next?

- Modularity based on abstraction is the way things are done

Programming Languages Today

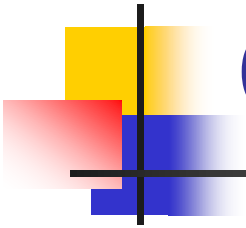


- Languages for experts, e.g., Java, C#



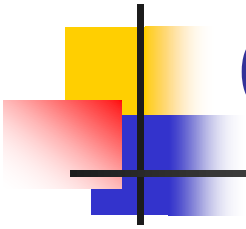
Programming 1A

- E.g., Python



Challenges

- A programming language for novices and experts
  - Ease of use vs. expressive power
  - Readability vs. writeability
  - Modularity and encapsulation
  - Powerful abstraction mechanisms
  - State matters



Challenges

- Massively-parallel computers
  - Programming methodology
  - Programming language support

The Power of Abstraction

Barbara Liskov

March 2013

MIT CSAIL