Concept Programming
The Art of Turning Ideas into Code

Christophe de Dinechin,
christophe@dinechin.org
Problem statement
Dealing with Ever Increasing Software Complexity
Exponential Growth

- Software complexity follows Moore’s law
- Driven by customers, not by programmers
- Programmers brains can’t keep up
- Result: periodic paradigm shifts...
- ... obsoleting all the legacy

Graph showing exponential growth with time on the x-axis and complexity on the y-axis.
Exponential Growth

- Software complexity follows Moore’s law
- Driven by customers, not by programmers
- Programmers brains can’t keep up
- Result: periodic paradigm shifts...
- ... obsoleting all the legacy
Exponential Growth

- Software complexity follows Moore’s law
- Driven by customers, not by programmers
- Programmers brains can’t keep up
- Result: periodic paradigm shifts...
- ... obsoleting all the legacy
Exponential Growth

- Software complexity follows Moore’s law
- Driven by customers, not by programmers
- Programmers’ brains can’t keep up
- Result: periodic paradigm shifts...
- ... obsoleting all the legacy
Exponential Growth

- Software complexity follows Moore’s law
- Driven by customers, not by programmers
- Programmers brains can’t keep up
- Result: periodic paradigm shifts...
- ... obsoleting all the legacy
Exponential Growth

- Software complexity follows Moore’s law
- Driven by customers, not by programmers
- Programmers brains can’t keep up
- Result: periodic paradigm shifts...
- ... obsoleting all the legacy
Exponential Growth

- Software complexity follows Moore’s law
- Driven by customers, not by programmers
- Programmers brains can’t keep up
- Result: periodic paradigm shifts...
- ... obsoleting all the legacy
Exponential Growth

- Software complexity follows Moore’s law
- Driven by customers, not by programmers
- Programmers brains can’t keep up
- Result: periodic paradigm shifts...
- ... obsoleting all the legacy
Exponential Growth

- Software complexity follows Moore’s law
- Driven by customers, not by programmers
- Programmers brains can’t keep up
- Result: periodic paradigm shifts...
- ... obsoleting all the legacy
Staying Ahead of Moore’s Law

- Can we integrate new paradigms incrementally?  
  **YES**

- Can we select the best representation independently for any given concept?  
  **YES**
Staying Ahead of Moore’s Law

- Can we integrate new paradigms incrementally?  
  \textbf{YES}

- Can we select the best representation independently for any given concept?  
  \textbf{YES}
Software Complexity

- Scale Complexity
  - Millions of Objects, Billions of Bits

- Domain Complexity
  - Ever Needed “X-Ray Spectrography for Dummies?”

- Artificial Complexity
  - C++ Standard: >700 pages, highly technical

- Business Complexity
  - Deliver this Yesterday, No Budget
The Belief in the Best Paradigm

- “Everything is an object”
  - In Smalltalk, \(2 + 3 \times 5 = 25\), not 17
  - Object 2 gets message + with arg 3

- “Everything is a function”
  - Functional languages: Lisp, OCaml
  - But the computer doesn’t think that way
  - ... and neither do many of us 😊
A Simple Example

How Can We Can Get Stuck so Easily?
Computing a Maximum

- **Mathematical Definition is Well Known**
  - Compares elements with an order relation
  - \( \text{Max}(a_1, a_2, \ldots, a_n) \)

- **Not Exactly a New Problem in Computing**

- **That Ought to be Easy!**
Maximum in C

- Generally Defined as a Macro
  - Something like: `#define max(x,y) ((x) < (y) ? (y) : (x))`
  - Or maybe: `#define max(x,y) ((x) >= (y) ? (x) : (y))`

- Some interesting questions
  - Why all the Parentheses?
  - What About Side Effects in `max(f(a++),c--)`?
  - What about `max(x,y,z,t)`?
Maximum in C

- Generally Defined as a Macro
  - Something like: `#define max(x,y) ((x) < (y) ? (y) : (x))`
  - Or maybe: `#define max(x,y) ((x) >= (y) ? (x) : (y))`

- Some interesting questions
  - Why all the Parentheses?
  - What About Side Effects in `max(f(a++),c--)`?
  - What about `max(x,y,z,t)`?
Maximum in Java (using functions)

- Defined in `java.lang.Math` as overloaded functions
  - You get `max(int,int), max(long, long), ...`

- We got rid of side effects!
  - But what about `max(x,y,z,t)`?
  - What about `max("Hello", "World")`?
  - What about `max(1, 2.5)`?
Maximum in Java (using functions)

- Defined in `java.lang.Math` as overloaded functions
  - You get `max(int, int), max(long, long), ...`
- We got rid of side effects!
  - But what about `max(x, y, z, t)`?
  - What about `max("Hello", "World")`?
  - What about `max(1, 2.5)`?

Failed!
Defined in `java.util.Collections` as generic function

When Java looks up to C++, you get:

```java
public static <T extends Object & Comparable<? super T>>
T max(Collection<? extends T> coll)
```

Hey, we can now compare more than 2 things!

But why can't we write `max(x,y,z,t)`?

Why should we create a collection to start with?

Why `e1.compareTo(e2)<0` and not `e1 < e2`?

Throws `ClassCastException` or `NoSuchElementException`
Maximum in Java (using Objects)

- Defined in `java.util.Collections` as generic function
  - When Java looks up to C++, you get:
    ```java
    public static <T extends Object & Comparable<? super T>>
    T max(Collection<? extends T> coll)
    ```

- Hey, we can now compare more than 2 things!
  - But why can't we write `max(x,y,z,t)`?
  - Why should we create a collection to start with?
  - Why `e1.compareTo(e2)<0` and not `e1 < e2`?
  - Throws `ClassCastException` or `NoSuchElementException`
Maximum in Lisp or Scheme

- Defined as variadic function
  - Scheme: `(define (max . a) (if (null? a) (error) (max-list a)))`

- Much closer to an acceptable definition
  - Syntax is Natural for Lisp: `(max 1 2 3 5)`
  - Still fails at run-time in same cases as Java
Maximum in Lisp or Scheme

- Defined as variadic function
  - Scheme: `(define (max . a) (if (null? a) (error) (max-list a)))`

- Much closer to an acceptable definition
  - Syntax is Natural for Lisp: `(max 1 2 3 5)`
  - Still fails at run-time in same cases as Java

Failed!
That Ought to be Easy! But it's **Hard**

- That simple problem is not solved after 30+ years

There is a gap between:

- **Concepts**, in your head
- **Representations of concepts**, in the code

Concept Programming is all about this gap
General Ideas

Applying Concept Programming
What is Concept Programming?

- **Code represents concepts**
  - Reality: Shape, File, Credit, Shotgun
  - Organization: Function, Visitor, Aspect
  - Focus on concepts relevant to the program

- **Make the code “look like” the concept**
  - Similarity in structure, behavior, locality
  - Principle of least surprise
Domains

- Concept and Code live in separate domains
  - Concepts: Environment, Organization, Algorithms, Pictures
  - Code: Source, Object, Data, Instructions, Bitmaps

- Unlike objects or functions, you won’t find “concepts” in the code, only concept representations
Turning Concepts into Code is a **lossy** conversion
- This is true with any language, any paradigm
- No two people have exactly the same concept in mind

Minimizing the loss remains a **worthy goal**
What is a “Concept”?

- An entity in the problem space...
  - Cars, Error Messages, Connections
  - An object is only one possible representation
  - ... that is relevant to the code space
    - What will it be used for? How do we represent it?
    - Relevant here, irrelevant there
  - The set of concepts is not constrained
Minority Paradigms

- The set of concepts is infinite...
  - Special concepts can make life easier

- Minority paradigms to fill the void
  - Logic programming, design by contract

- To each its (incompatible) language!
  - Prolog, Eiffel

- Not minor in usefulness
  - But the majority can't use them
Limitations of the Tools

- Many notations are difficult to add
  - Symbolic differentiation
  - GUI Elements
  - Debug-only code

- We Need a Concept Programming Language
  - But a lot can be done without
Pseudo Metrics
Identifying Non-Obvious Problems in the Code
Pseudo-metrics

- **Syntactic Noise**
  - Form that doesn’t map to the problem space

- **Semantic Noise**
  - Meaning that doesn’t map to the problem space

- **Bandwidth**
  - How much of the problem space is covered?

- **Signal/Noise Ratio**
  - How much code actually deals with real problems?
Pseudo-metrics

- **Syntactic Noise**
  - Form that doesn’t map to the problem space
  - Useless and potentially distracting visual clutter

- **C:**
  ```c
  if (a == 3) { printf("Hello\n"); }
  ```

- **C++:**
  ```cpp
  list<list<int> > l; // Watch that space!
  ```

- **HTML:**
  ```html
  When N &lt; 0, N is said to be negative
  ```
Pseudo-metrics

- Semantic Noise
  - Meaning that doesn’t map to the problem space
  - Unexpected behavior compared to “native” concept

- C: if (x = 0) y = max(f(), x);
  - Zeroes x, calls f twice

- C++: object.GetBounds(&rect);
  - Exposes two addresses

- Smalltalk: 2+3*5
  - 25 instead of 17
Pseudo-metrics

- **Bandwidth**
  - How much of the problem space is covered?
  - Conditions reuse in different cases

- C: `int max(int x, int y);`
  - vs. macro

- C++: `cout << complex(2.3, 5.2);`
  - vs. printf

- Ada: `accept Help (X : item) do...`
  - vs. pthreads
Pseudo-metrics

- **Signal/Noise Ratio**
  - How much code actually deals with real problems?
  - The rest is mostly useless fluff...

- **Java:**
  ```java
class HelloWorldApp {
    public static void main(String[] args) {
      System.out.println("Hello World!");
    }
  }
```
These are pseudo metrics
- You can’t measure things in the problem space
- Highly subjective metrics
- You can’t write a tool to measure them

Analogy to Music
- Reducing noise is a worthy goal...
- But you cannot completely eliminate it
- Noise to one, music to the other
Abstraction

Fighting Complexity by Reducing it to Tiny Bits
Code is a particular concept abstraction

This abstraction is necessary
  - You can’t run ideas in a computer

But: Abstractions introduce distortions
  - What you think is not what you get
  - Abstraction penalty, inefficiency in generated code
Abstraction Loss: Concept Cast

- Replacing a concept with a related one
  - Often to workaround limits of the tools
  - Example: replace $f(x,y,z,...)$ with $f(\text{list})$
- Too often an **unconscious** decision
  - It works!
- Maybe the most frequent abstraction loss
  - You lose some semantic signal...
  - ... while introducing a lot of noise
Abstractions vs. Complexity

- **Domain**: Equivalence, aka least surprise
  - Programmers read `FILE` and think “file”

- **Scale**: Layering and reuse
  - `FILE` can be reused, e.g. to build `DATABASE`

- **Artificial**: Hide irrelevant details
  - You can safely ignore all the OS magic behind `FILE`

- **Business**: Manageability, predictability
  - `FILE` behavior is reliable, portable, documented
Step by Step

- Define the problem space
- Identify individual concepts
- Document concept behaviors & relations
- Choose notation for each concept
- Select or invent representation
XL: An Extensible Language

Applying Concept Programming to Language Design
Considering Metrics

- **Syntactic Noise**
  
  ```
  if A < 3 then IO.WriteLine "A=", A
  ```

- **Semantic Noise**
  
  ```
  to GetBounds(O : object; out R : rectangle)
  ```

- **Bandwidth**
  
  ```
  function Max(x: ordered; ...) return ordered
  X : integer := Max(1, 3, 7, 2, 4)
  ```

- **Signal/Noise Ratio**
  
  ```
  type complex with
  Re, Im : real
  ```
Extensibility

- **Symbolic differentiation**
  - Standard notation: \( \frac{d}{dx}\sin(x + \frac{1}{x}) \)
  - XL notation: \{differentiation\} d/dx(sin(x+1/x))

- **Compiler plug-ins implement extensions**
  - Plug-in code uses specific extensions:
    
    translation differentiation
    
    when (d/'dvar'('expr')) where BeginsWithD(dvar) then ...
Extensibility benefits

- Represent arbitrary concepts
- Favors "natural" notations in the code
- Unifies "user" and "built-in" entities
- Leaves the computer to do the grunt work
XL Concept-inspired Features

- Expression reduction
- True and validated generic types
- Type-safe variable argument lists
- Iterators and generators

- All used to build “standard” elements
XL Concept-inspired Features

- **Expression reduction**
  - Generalizes operator overloading
  - Efficient matrix linear algebra
    ```python
    function MultiplyAdd(A, B, C : matrix) return matrix
    written A*B+C
    
    function IsIdentity(M : matrix) return boolean
    written M = 1
    ```
  - Easy special cases
XL Concept-inspired Features

- True generic types
  - Make functions implicitly generic
- Array operations
  - function Add (A, B : array) return array written A+B
- Pointer operations
  - function Peek(P : ptr) return ptr.item written *P
  - to Poke(P : ptr; V : ptr.item) written *P := V
XL Concept-inspired Features

- Validated generic types
  - Specify interface of a generic type
  - Type with an order operation
    ```
    generic type ordered where
    A, B : ordered  // Code testing the
    Test : boolean := A < B  // candidate types
    ```

- Makes generic code more robust
  ```
  function Min (X : ordered) return ordered
  Z : complex := Min(Z)  // Error (unlike C++)
  ```
XL Concept-inspired Features

- Type-safe variable argument lists
  - A user-defined Pascal-style `WriteLn`:
    ```pascal
to WriteLn(...) is // ... stand for rest of args
    Write ... // Pass rest of args
    Write new_line
    ```
- Min and max functions that work:
  ```pascal
  function Min(X : ordered; ...) return ordered is
  result := Min(...)
  if X < result then
  result := X
  ```
Iterators and generators

- Define iterator over a range of integers

  ```pascal
  iterator It(var out C: T; L, H: T) written C in L..H is
  C := L
  while C <= H loop
    yield
    C += 1
  end loop
  ```

- Used in for loops (and implements for loops)

  ```pascal
  for K in 3..5 loop
    WriteLn "K=", K
  end loop
  ```
generic type ordered where
   A, B : ordered
   Test : boolean := A < B

function Max (X : ordered) return ordered is
   return X

function Max (X : ordered; ...) return ordered is
   result := Max(...)
   if result < X then
      result := X
Maximum in XL

generic type ordered where
   A, B : ordered
   Test : boolean := A < B

function Max (X : ordered) return ordered is
   return X

function Max (X : ordered; ...) return ordered is
   result := Max(...)
   if result < X then
      result := X
Bridging the Gap: Done?

- Turning Concepts into Code is a lossy conversion
  - This is true with any language, any paradigm
  - No two people have exactly the same concept in mind
- Minimizing the loss remains a worthy goal
- XL does this better 😊
Thank you!

Concept Programming

The Art of Turning Ideas into Code

Christophe de Dinechin,
christophe@dinechin.org