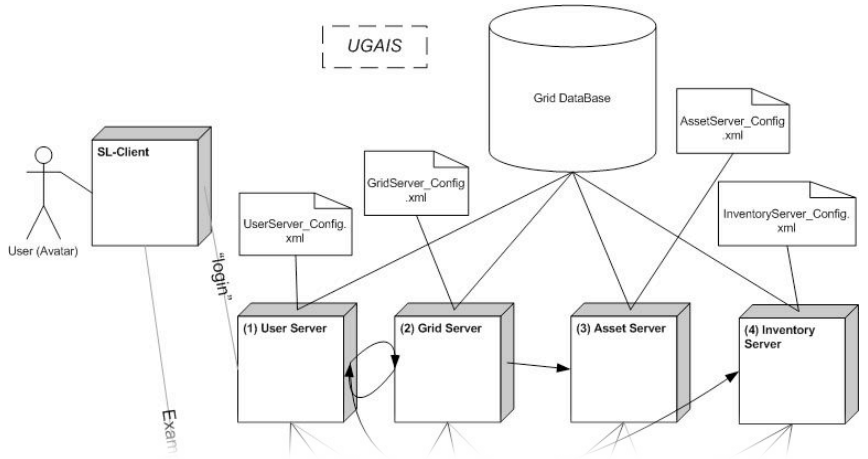
design patterns

```
public vo                ildDetail build, Map paramete
    Ite                  ).iterator();
    while (              (()) {
        try              parameters, (String) iterat
        }
    }
}

void assemblePlugin(BuildDetail build, Map parameters, String
    String className = line.trim();
    if (className.startsWith("#") || StringUtils.isEmpty(clas
        return;
    }
    Class clazz = Class.forName(className);
    Widget digesterService = (Widget) clazz.newInstance();
    mergeParameters(build, parameters);
    build.addPluginOutput(digesterService.getDisplayName(), c
            .getOutput(parameters));
}

private void mergeParameters(BuildDetail build, Map parameter
    parameters.put(Widget.PARAM_CC_ROOT, configuration.getCCH
```

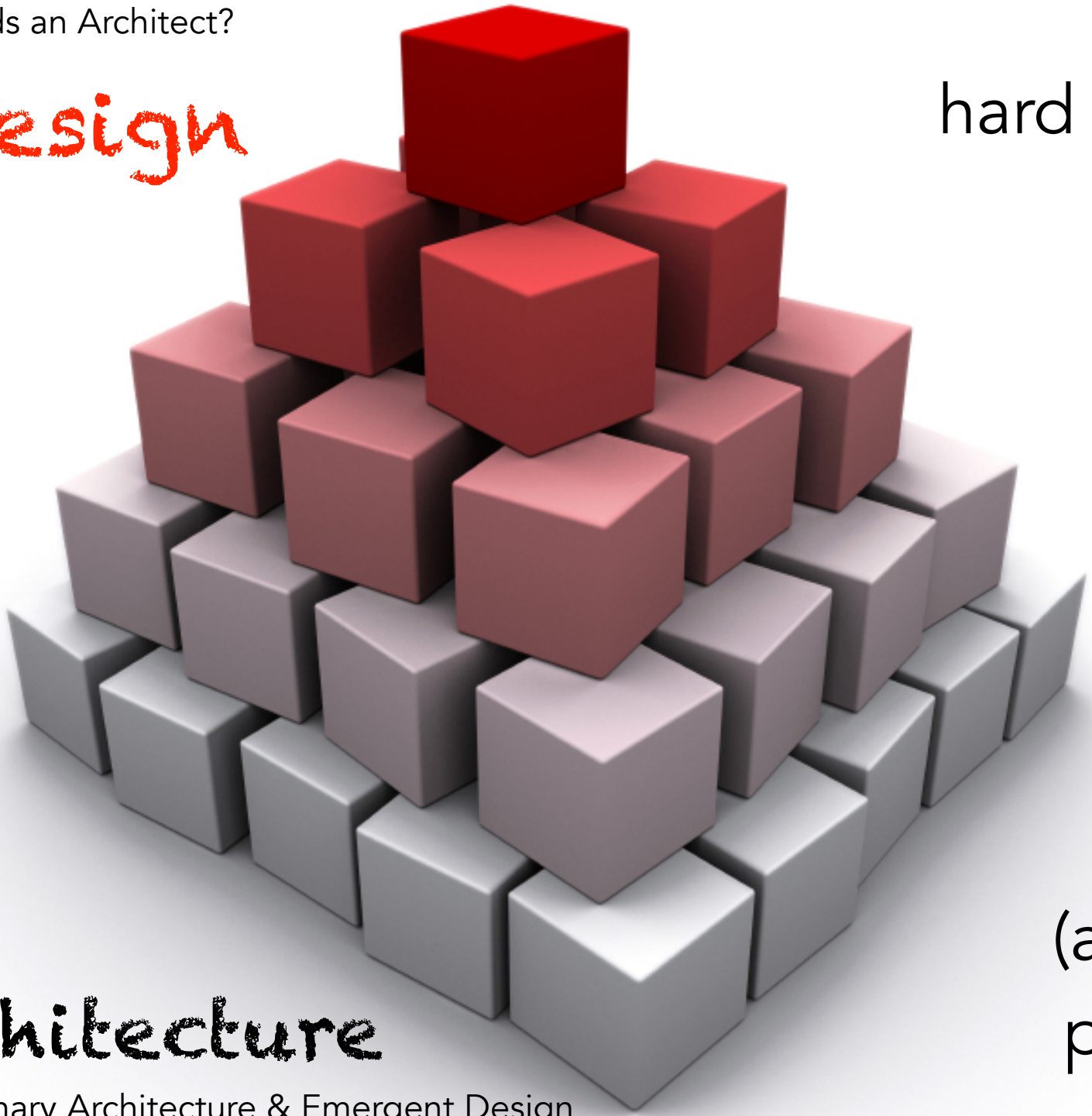


micro <=> macro

(martinfowler.com/ieeeSoftware/whoNeedsArchitect.pdf)
Who Needs an Architect?

design

hard to change later



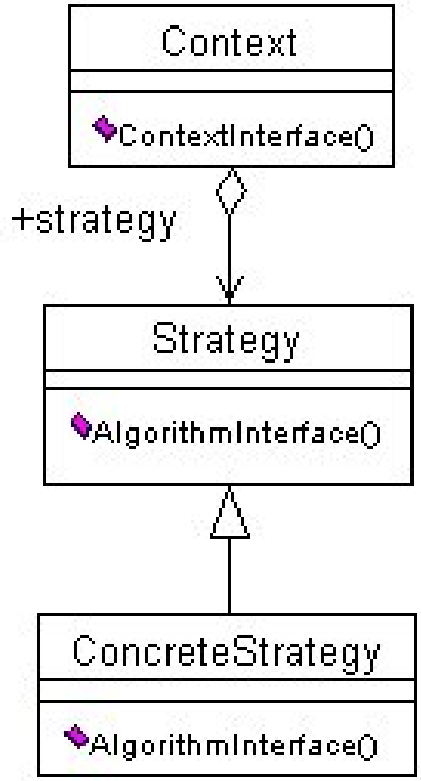
architecture

(as few as possible)

Evolutionary Architecture & Emergent Design
(bit.ly/nf-ead-all)


a sampling of
design patterns


Strategy

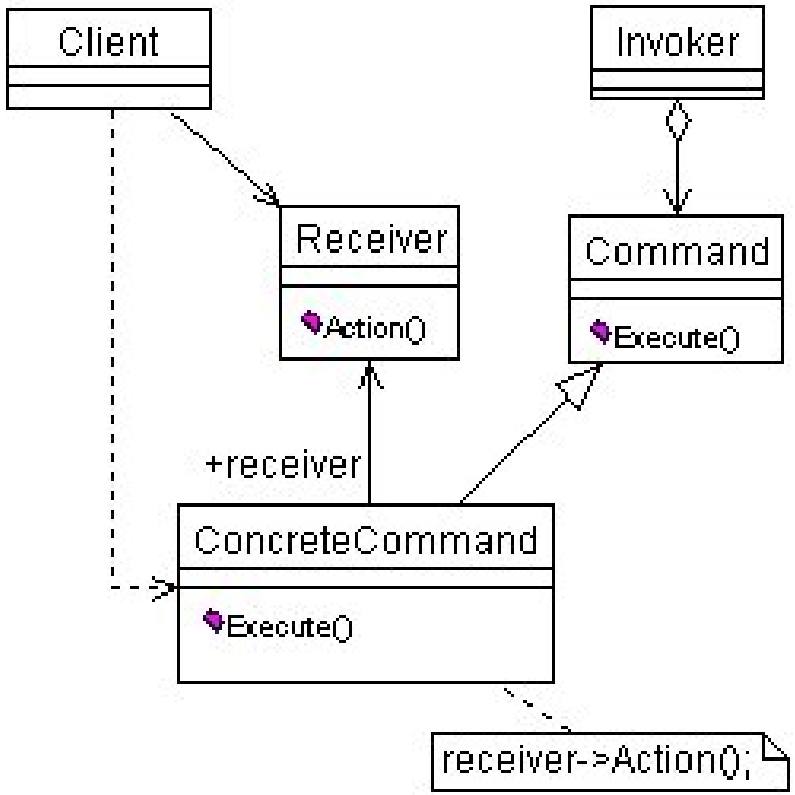


Define a family of algorithms, encapsulate each one, and make them interchangeable.

Strategy allows the algorithm to vary independently from clients that use it.

```
public interface Calc {
    public int product(int x, int y);
}
public class CalcByMult implements Calc {
    public int product(int x, int y) {
        return x * y;
    }
}
public class CalcByAdds implements Calc {
    public int product(int x, int y) {
        int result = 0;
        for (int i = 1; i <= y; i++)
            result += x;
        return result;
    }
}
```


Command



Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.

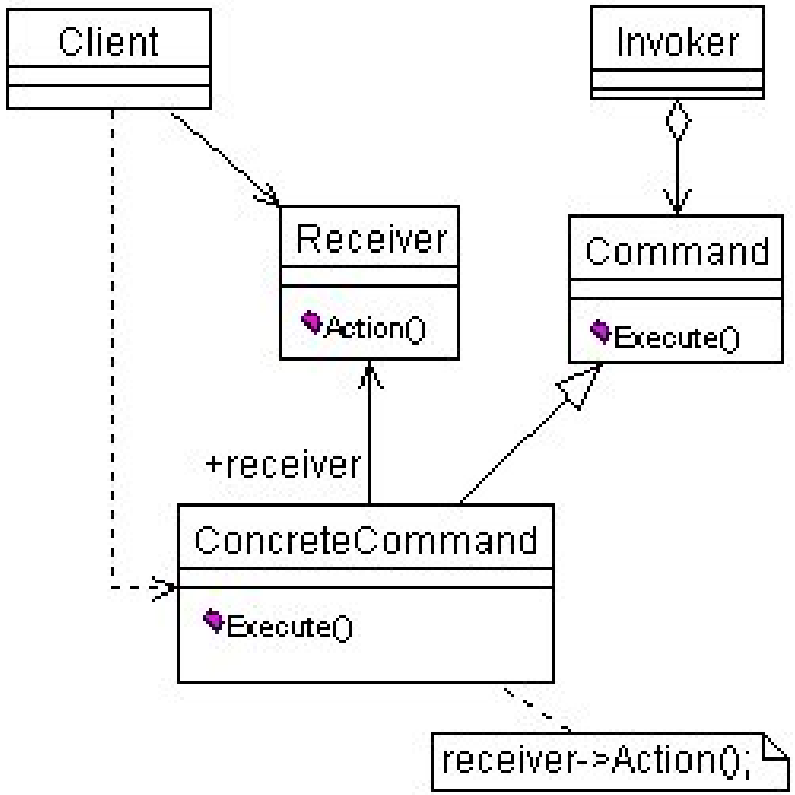
“unit of work”

```
public void addOrderFrom(ShoppingCart cart, String userName,
                         Order order) throws Exception {
    setupDataInfrastructure();
    try {
        add(order, userKeyBasedOn(userName));
        addLineItemsFrom(cart, order.getOrderKey());
        completeTransaction();
    } catch (Exception condition) {
        rollbackTransaction();
        throw condition;
    } finally {
        cleanUp();
    }
}
```

“unit of work”

```
public void wrapInTransaction(Command c) throws Exception {
    setupDataInfrastructure();
    try {
        c.execute();
        completeTransaction();
    } catch (Exception condition) {
        rollbackTransaction();
        throw condition;
    } finally {
        cleanUp();
    }
}

public void addOrderFrom(final ShoppingCart cart, final String userName,
                         final Order order) throws Exception {
    wrapInTransaction(new Command() {
        public void execute() {
            add(order, userKeyBasedOn(userName));
            addLineItemsFrom(cart, order.getOrderKey());
        }
    });
}
```



Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.

command w/ closures

```
public class OrderDbClosure {
    def wrapInTransaction(command) {
      setupDataInfrastructure()
      try {
        command()
        completeTransaction()
      } catch (RuntimeException ex) {
        rollbackTransaction()
        throw ex
      } finally {
        cleanUp()
      }
    }

    def addOrderFrom(cart, userName, order) {
      wrapInTransaction {
        add order, userKeyBasedOn(userName)
        addLineItemsFrom cart, order.orderKey
      }
    }
}
```

good & bad of patterns

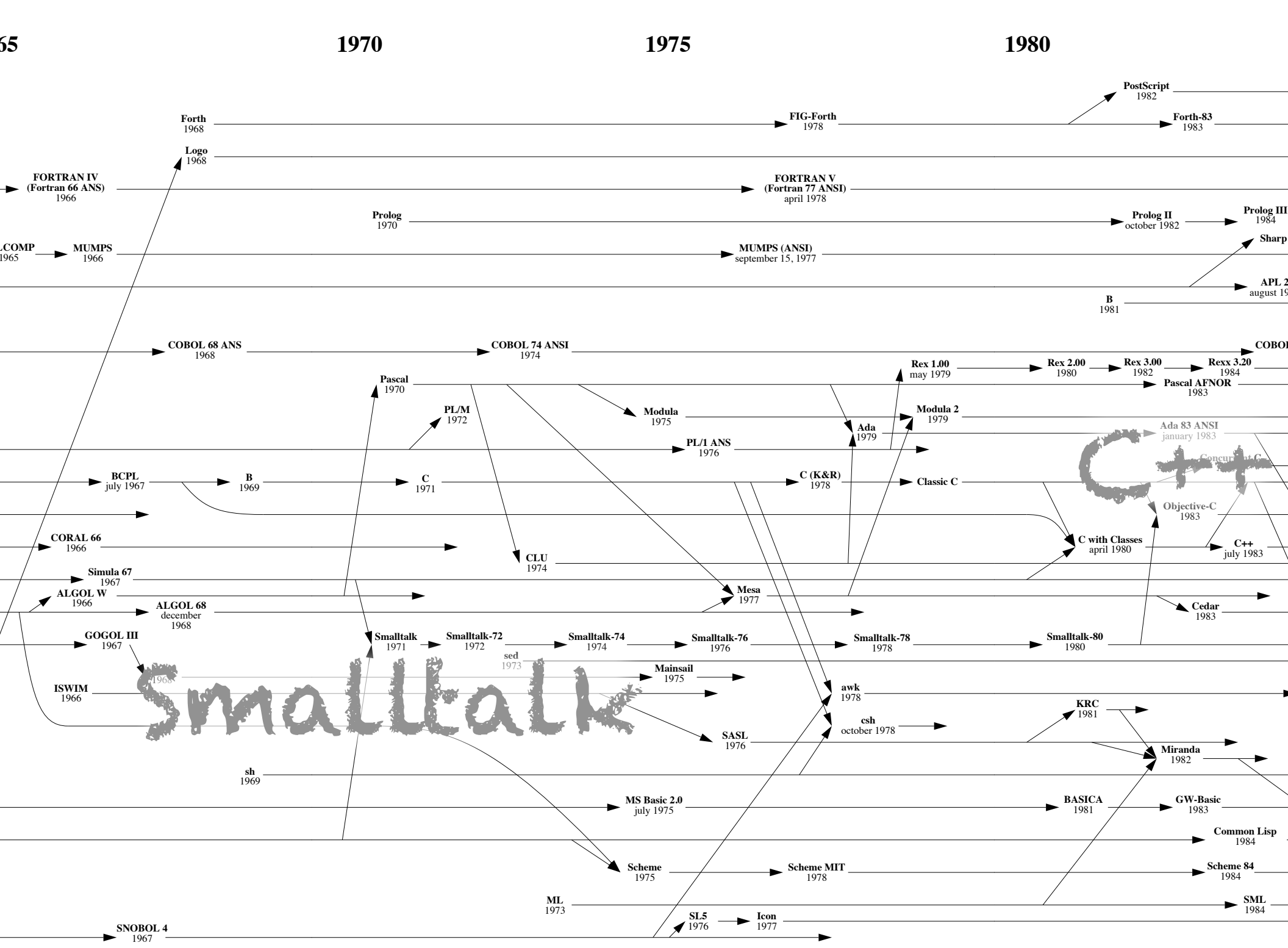
vocabulary

design guide

implementation guide

panacea

religion


65
1970
1975
1980
Forth
1968
FIG-Forth
1978
PostScript
1982
Forth-83
1983
Logo
1968
FORTRAN IV
(Fortran 66 ANS)
1966
FORTRAN V
(Fortran 77 ANSI)
april 1978
Prolog
1970
Prolog II
october 1982
Prolog III
1984
LCOMP
1965
MUMPS
1966
MUMPS (ANSI)
september 15, 1977
Sharp
APL 2
august 19
B
1981
COBOL 68 ANS
1968
COBOL 74 ANSI
1974
COBOL
Pascal
1970
Rex 1.00
may 1979
Rex 2.00
1980
Rex 3.00
1982
Rexx 3.20
1984
Pascal AFNOR
1983
PL/M
1972
Modula
1975
Modula 2
1979
Ada
1979
Ada 83 ANSI
january 1983
PL/1 ANS
1976
Concurrent C
BCPL
july 1967
B
1969
C
1971
C (K&R)
1978
Classic C
Objective-C
1983
CORAL 66
1966
C with Classes
april 1980
C++
july 1983
CLU
1974
Simula 67
1967
ALGOL W
1966
Mesa
1977
Cedar
1983
ALGOL 68
december
1968
GOGOL III
1967
Smalltalk
1971
Smalltalk-72
1972
Smalltalk-74
1974
Smalltalk-76
1976
Smalltalk-78
1978
Smalltalk-80
1980
sed
1973
Mainsail
1975
ISWIM
1966
awk
1978
KRC
1981
csh
october 1978
SASL
1976
Miranda
1982
sh
1969
MS Basic 2.0
july 1975
BASICA
1981
GW-Basic
1983
Common Lisp
1984
Scheme
1975
Scheme MIT
1978
Scheme 84
1984
ML
1973
SML
1984
SL5
1976
Icon
1977
SNOBOL 4
1967

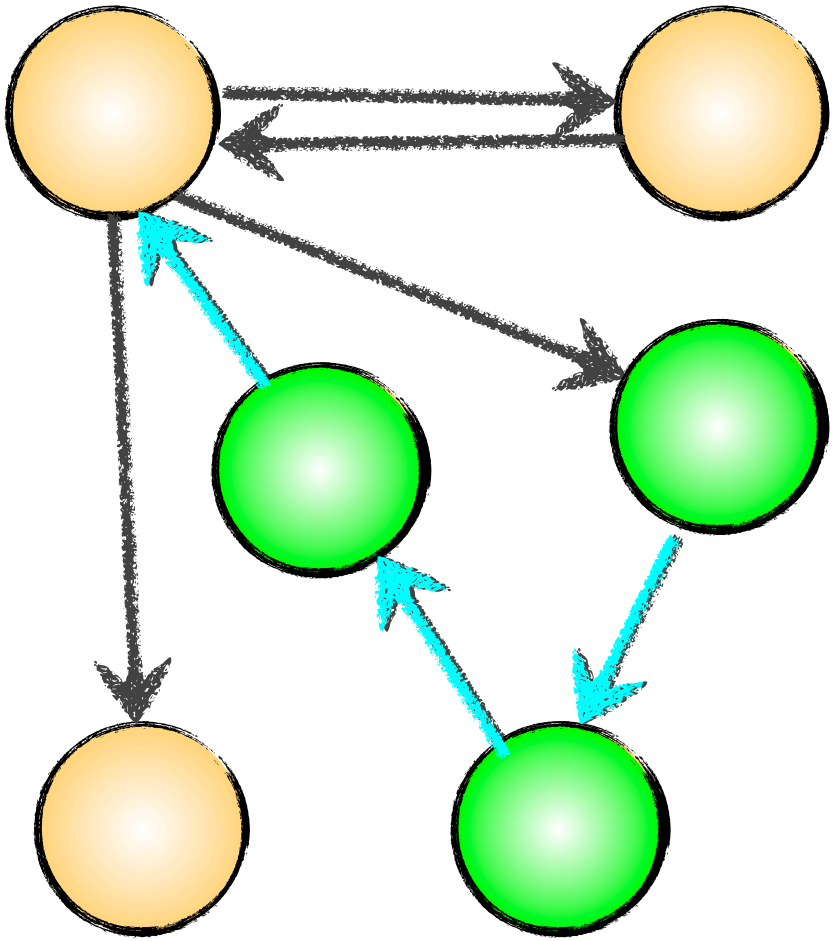
patterns today

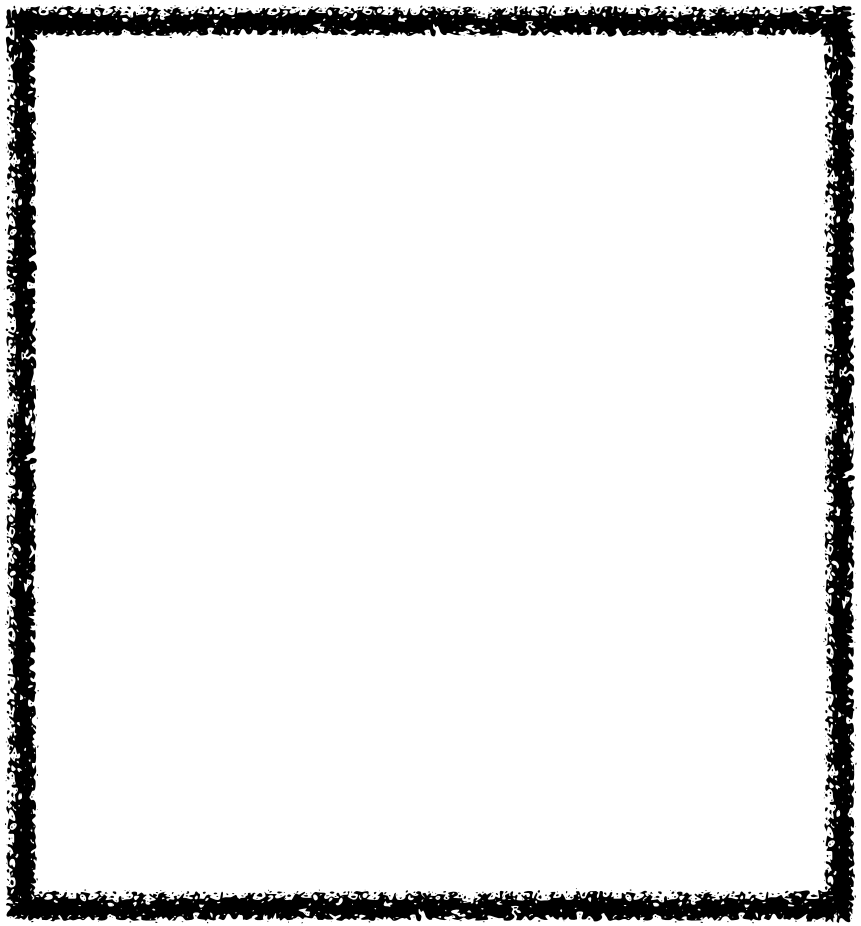
many “classic” GoF patterns encapsulated or deprecated by language evolution

- iterator
- command

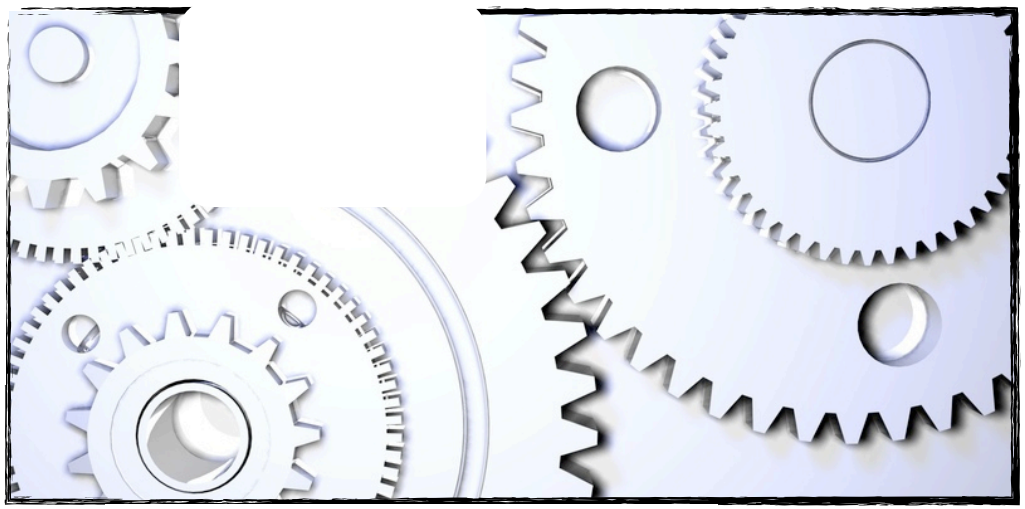
paradigms beyond just object-orientation

OOP & reuse



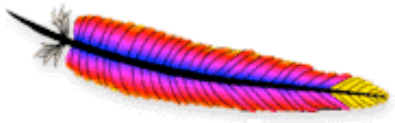


functional reuse

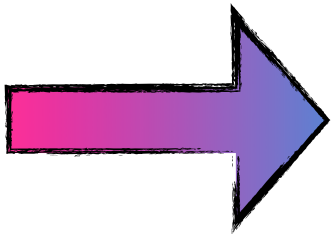


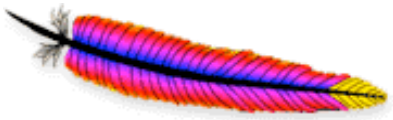

OOPy frameworks

how does “functional” change the structure of my code?



Apache Commons
http://commons.apache.org/
`indexOfAny`

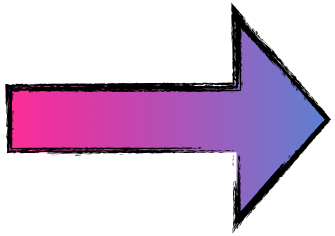




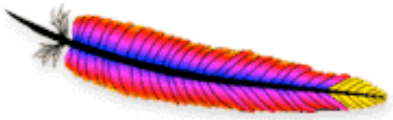
Apache Commons

http://commons.apache.org/

indexOfAny

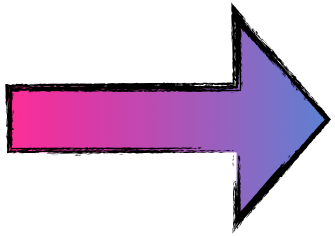


```
StringUtils.indexOfAny(null, *)                = -1
StringUtils.indexOfAny("", *)                  = -1
StringUtils.indexOfAny(*, null)                = -1
StringUtils.indexOfAny(*, [])                  = -1
StringUtils.indexOfAny("zzabyycdxx",['z','a']) = 0
StringUtils.indexOfAny("zzabyycdxx",['b','y']) = 3
StringUtils.indexOfAny("aba", ['z'])           = -1
```

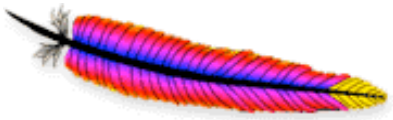


Apache Commons
http://commons.apache.org/

indexOfAny

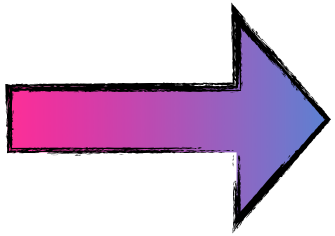


```
// From Apache Commons Lang, http://commons.apache.org/lang/
public static int indexOfAny(String str, char[] searchChars) {
  if (isEmpty(str) || ArrayUtils.isEmpty(searchChars)) {
    return -1;
  }
  for (int i = 0; i < str.length(); i++) {
    char ch = str.charAt(i);
    for (int j = 0; j < searchChars.length; j++) {
      if (searchChars[j] == ch) {
        return i;
      }
    }
  }
  return -1;
}
```



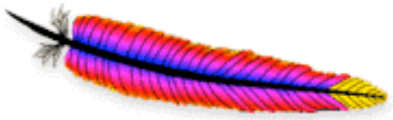
Apache Commons
http://commons.apache.org/

indexOfAny



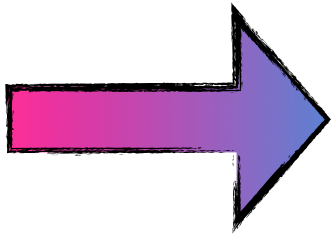
```
public static int indexOfAny(String str, char[] searchChars) {
  when (searchChars)
    for (int i = 0; i < str.length(); i++) {
      char ch = str.charAt(i);
      for (int j = 0; j < searchChars.length; j++) {
        if (searchChars[j] == ch) {
          return i;
        }
      }
    }
  }
}
```

simplify corner cases



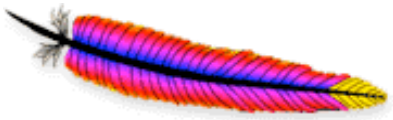
Apache Commons
http://commons.apache.org/

indexOfAny



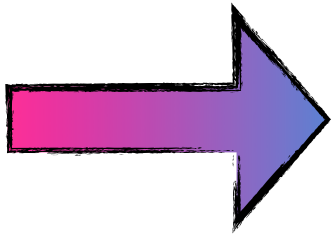
```
indexOfAny(str, searchChars) {
  when (searchChars)
    for (i = 0; i < str.length(); i++) {
      ch = str.charAt(i);
      for (j = 0; j < searchChars.length; j++) {
        if (searchChars[j] == ch) {
          return i;
        }
      }
    }
  }
}
```

— type decls



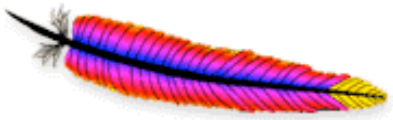
Apache Commons
http://commons.apache.org/

indexOfAny



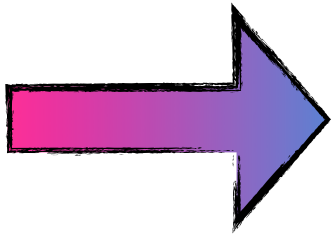
```
indexOfAny(str, searchChars) {
  when (searchChars)
    for (i = 0; i < str.length(); i++) {
      ch = str.charAt(i);
      when searchChars(ch) i;
    }
  }
}
```

+ when clause



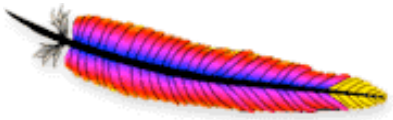
Apache Commons
http://commons.apache.org/

indexOfAny

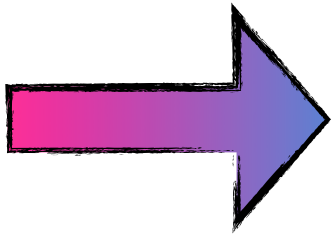


```
indexOfAny(str, searchChars) {
  when (searchChars)
    for ([i, ch] in indexed(str)) {
      when searchChars(ch) i;
    }
  }
}
```

+ comprehension



Apache Commons
http://commons.apache.org/
indexOfAny



```
(defn index-filter [pred coll]
  (when pred
    (for [[idx elt] (indexed coll) :when (pred elt)] idx)))
```

Lispify

which version is simpler?

	imperative	functional
functions	1	1
classes	1	0
internal exit points	2	0
variables	3	0
branches	4	0
boolean ops	1	0
function calls*	6	3
total	***18***	***4***

which is more general?

```
; idxs of heads in stream of coin flips
(index-filter #{:h}
[:t :t :h :t :h :t :t :t :h :h])
-> (2 4 8 9)

; Fibonaccis pass 1000 at n=17
(first
  (index-filter #(> % 1000) (fibo)))
-> 17
```

which is more general?

imperative	**functional**
searches strings	searches *any sequence*
matches characters	matches *any predicate*
returns first match	returns *lazy seq of all matches*

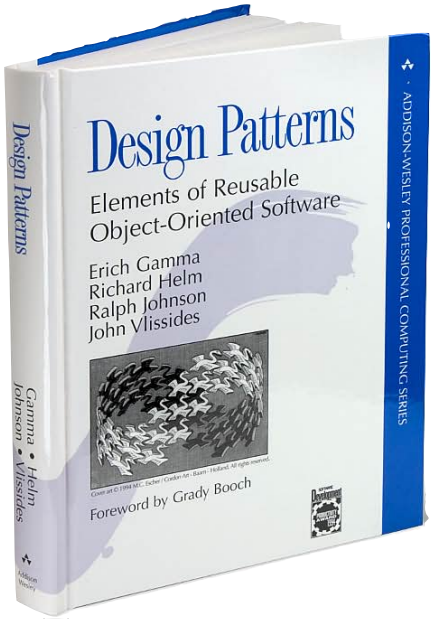
patterns exist in all languages

understand the idiomatic patterns for your language/platform

question dogma

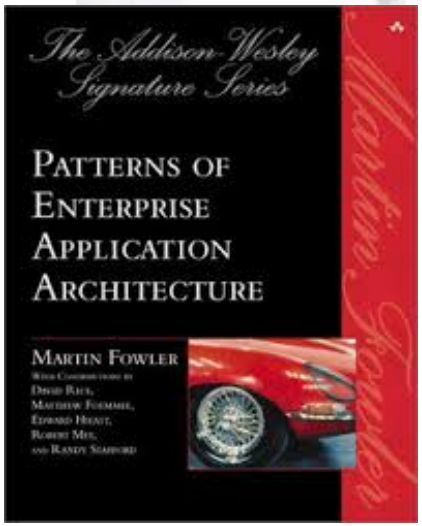
clarity vs obscurity

for more information



Design Patterns
Gamma et al

Martin's "bliki"
http://martinfowler.com/bliki/



Patterns of Enterprise Application Architecture
Martin Fowler

?'s

Mark Richards

Independent Consultant

Hands-on Enterprise / Integration Architect
Published Author / Conference Speaker

http://www.wmrichards.com
http://www.linkedin.com/pub/mark-richards/0/121/5b9

Published Books:

Java Message Service, 2nd Edition
97 Things Every Software Architect Should Know
Java Transaction Design Strategies

Neal Ford

Director / Software Architect /
Meme Wrangler

ThoughtWorks®

2002 Summit Blvd, Level 3, Atlanta, GA 30319, USA
T: +1 40 4242 9929 Twitter: @neal4d
E: nford@thoughtworks.com W: thoughtworks.com