Towards Malleable Software

Brian Shearing

Advanced Programming Specialist Group
British Computer Society, 14th April 2005

School of Computer Science & Information Systems
Birkbeck College, University of London, 3rd February 2005

Towards Malleable Software

1940s Machine Codes

18/7/48 — Kilburn Highest Factor Routine (amended) —

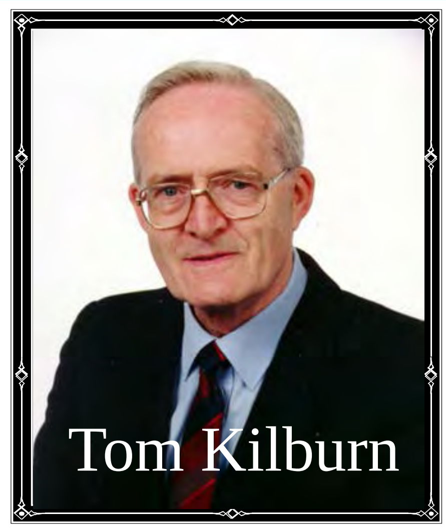
Instrn.	C	25	26	27	line	0 1 2 3 4	13 14 15	
-24 to C	$-b_1$	–	–	–	1	00011	010	
c to 26			$-b_1$		2	01011	110	
-26 to C	b_1				3	01011	010	
c to 27			$-b_1$	b_1	4	11011	110	
-23 to C	a	τ_{n-1}	$-b_n$	b_n	5	11101	010	
Sub 27	$a - ab_n$				6	11011	001	
Test					7	—	011	
Add 20 to Cl					8	00101	100	or 000
Sub. 26	τ_n				9	01011	001	
c to 25		τ_n			10	10011	110	
-25 to C					11	10011	010	
Test					12	—	011	
Stop	0	0	$-b_N$	b_N	13		111	
-26 to C	b_n	τ_n	$-b_n$	b_n	14	01011	010	
Sub. 21	$b_n - 1$				15	10101	001	
c to 27	b_{n+1}			b_{n+1}	16	11011	110	
-27 to C	$-b_{n+1}$				17	11011	010	
c to 26			$-b_{n+1}$		18	01011	110	
22 to Cl.		τ_n	$-b_{n+1}$	b_{n+1}	19	01101	000	

20	-3	10111 etc
21	1	10000
22	4	00100

or 10100

23	$-a$
24	b_1

	init.	final
25	–	τ_N (= 0)
26	–	$-b_N$
27	–	b_N



Towards Malleable Software

1940s Machine Codes

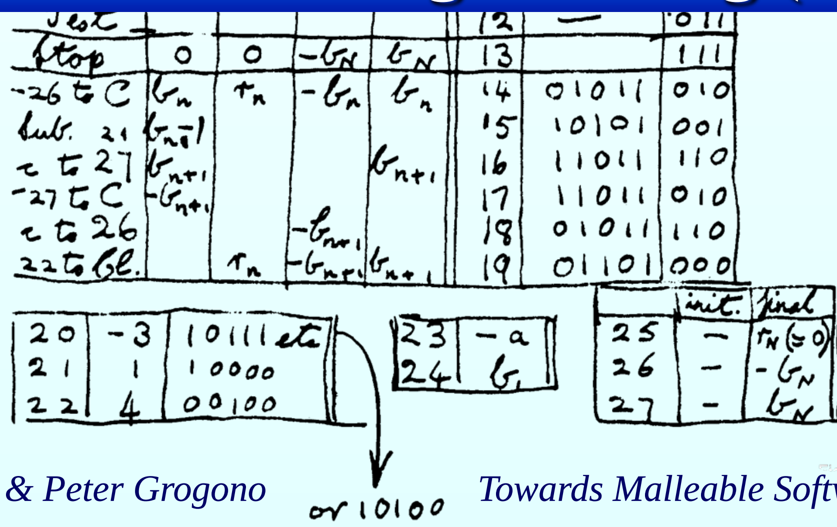
1950s Assembly Languages

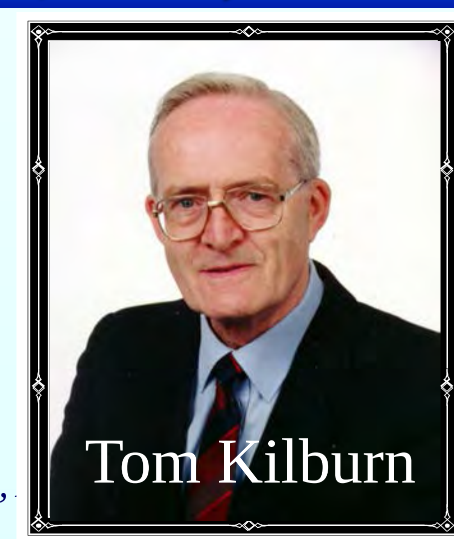
1960s High-level Languages (Fortran, COBOL)

1970s Structured Programming (Pascal, C)

1980s Modular Programming (ADA, Modula-2)

1990s Object-oriented Programming (C++, Java)





Towards Malleable Software

Hand wringing	**Today's software is not malleable**
Finger wagging	**What programming should be like**
Chin stroking	**Whence came the inflexibility, and where might malleability be found?**
Arm waving	**A model for malleable software**
Head scratching	**Research: verify or falsify the model**

Towards Malleable Software

Part I

Today's software is not malleable.

– What does this mean?

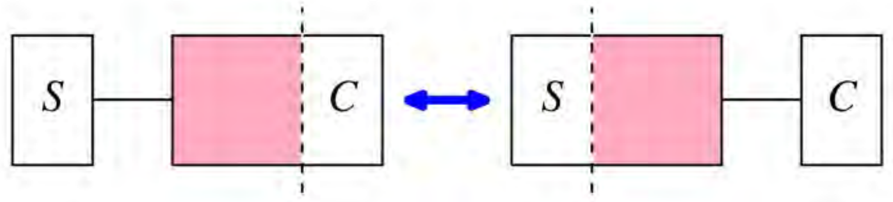
– Why does it matter?

Towards Malleable Software

today's software is not malleable

Some transformations that require much programming

Fat client to or from thin one

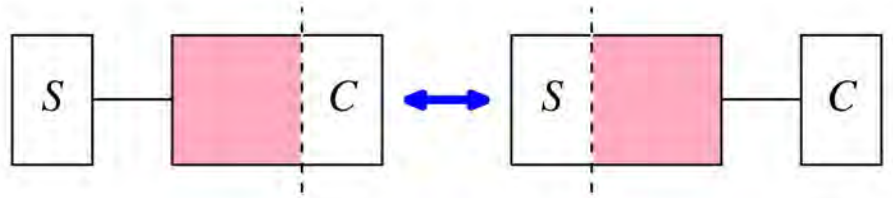


Towards Malleable Software

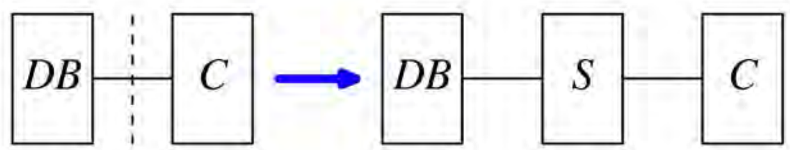
today's software is not malleable

Some transformations that require much programming

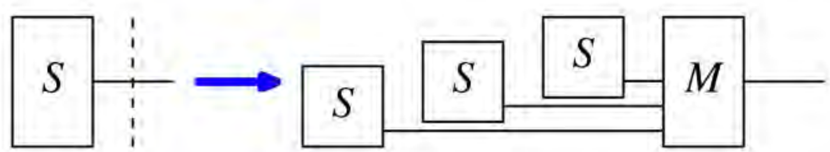
Fat client to or from thin one



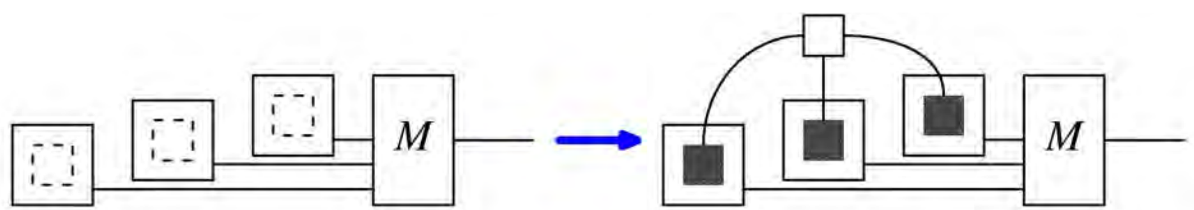
Two-tier to three-tier



Single thread to multiplex



Cache sharing multiplex



Towards Malleable Software

Part II

What programming should be like
– with malleable software
– and tools adapted for malleability

Towards Malleable Software

programming could be like this

Instructions to a compiler of a malleable notation:

- Compile Package *Product.Cache* in its own address space
- Compile *Policies.OvernightBatch* as a remote server using CORBA for communication
- Compile Process *Customer.Trolley* as a middleware component with multiple instances and automatic load-balancing
- Compile Components *Xml.Lexer* and *Xml.Parser* into one address space but as separate processes connected through a buffer with a capacity of 10,000 characters

Towards Malleable Software

programming could be like this

A compiler that knows enough to obey the previous instructions also knows enough to obey these:

Show me the system's structure

Show the dataflows that result from me doing *this*

When I do *that*, where is the time spent?

in words, or in pictures

without having to buy separate tools from specialist vendors

It's easy to do these things for malleable software

Towards Malleable S…re

programming could be like this

A compiler that knows enough to obey the previous instructions also knows enough to obey these:

- Build a reference manual for the system
- Build a book that explains how the system is put together
- Compile and run the system's tests

JavaDoc, Eiffel and Dee show the way

'Literate Programming' and 3R show the way

JUnit shows the way, but test scripts must be in the source

About time too!

Towards Malleable Software

Part III

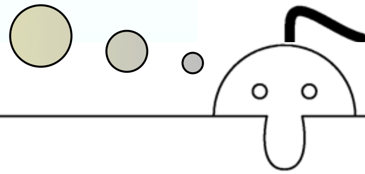
Whence came the inflexibility?

- **What is it about our way of working that impedes malleability?**
- **Where might malleability be found? A look at existing approaches that contribute to malleability**

Towards Malleable Software

What about o-o?

whence came the inflexibility?



✓ Object-oriented programming makes it possible to build systems that are readily adaptable and expandable — *within a given structure*.

✗ It provides little help for altering structures themselves.

What we need is
refactoring in-the-large.

Towards Malleable Software

whence came the inflexibility?

member inner classes

private scope

protected scope

classes

instance methods

It's a mess, and it doesn't work

package scope

local inner classes

interfaces

anonymous inner classes

but no block structure and no nested methods

public scope

static methods

delegates

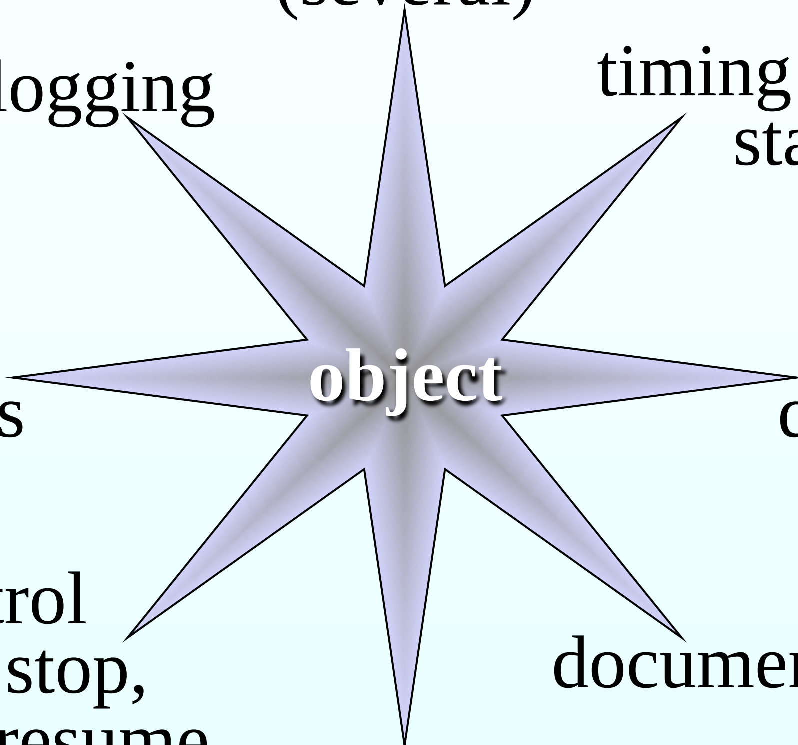
packages

static member inner classes

Towards Malleable Software

whence came the inflexibility?

aspects



Every aspect's interface must be public, even though each aspect should have its own access path.

Modular it isn't! And for restructuring it's hopeless.

Towards Malleable Software

Principle of managed modularity

An access permission is not a property of a component but of a relationship between components

also known as

The who's-asking principle

or as

The principle of not calling out-the-back

Towards Malleable Software

C. A. R. Hoare

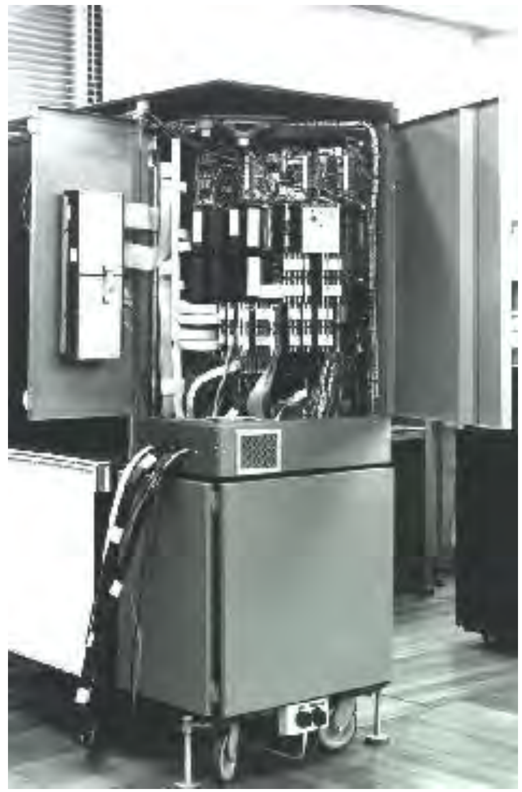
‘The language designer should be familiar with many alternative features designed by others […]
One thing he should not do is to include untried ideas of his own. His task is consolidation, not innovation.’

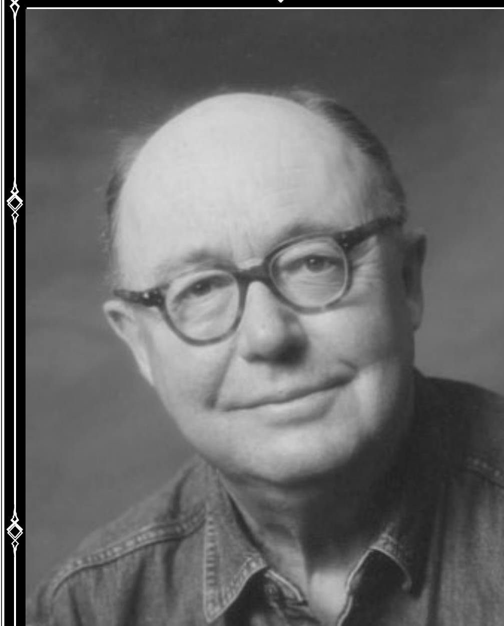
Hints on programming language design
in *Computer Systems Reliability*
State of the Art Report
Vol 20, pp 503–34
Pergamon/Infotech 1974

Towards Malleable Software

capabilities | **Cambridge CAP computer**

Maurice Wilkes





Roger Needham

Towards Malleable Software

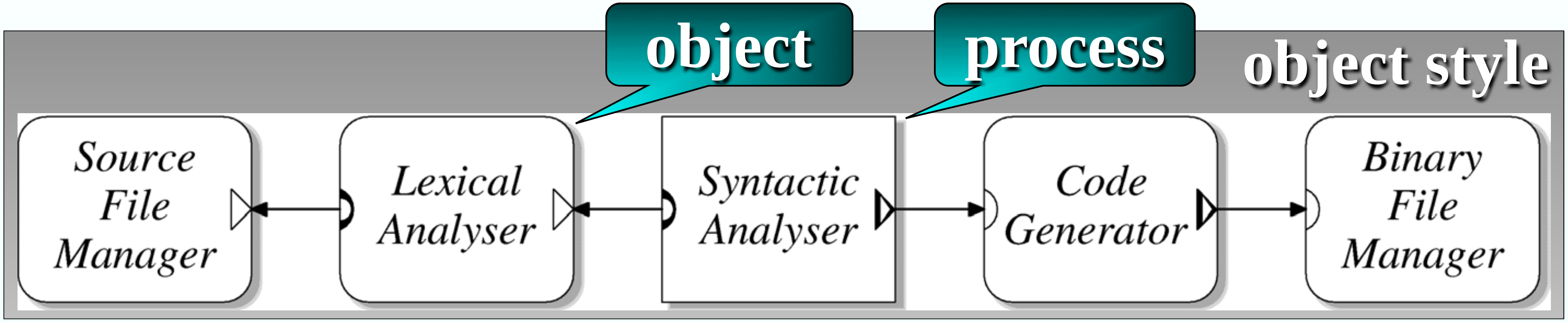
finding malleability

Hermes

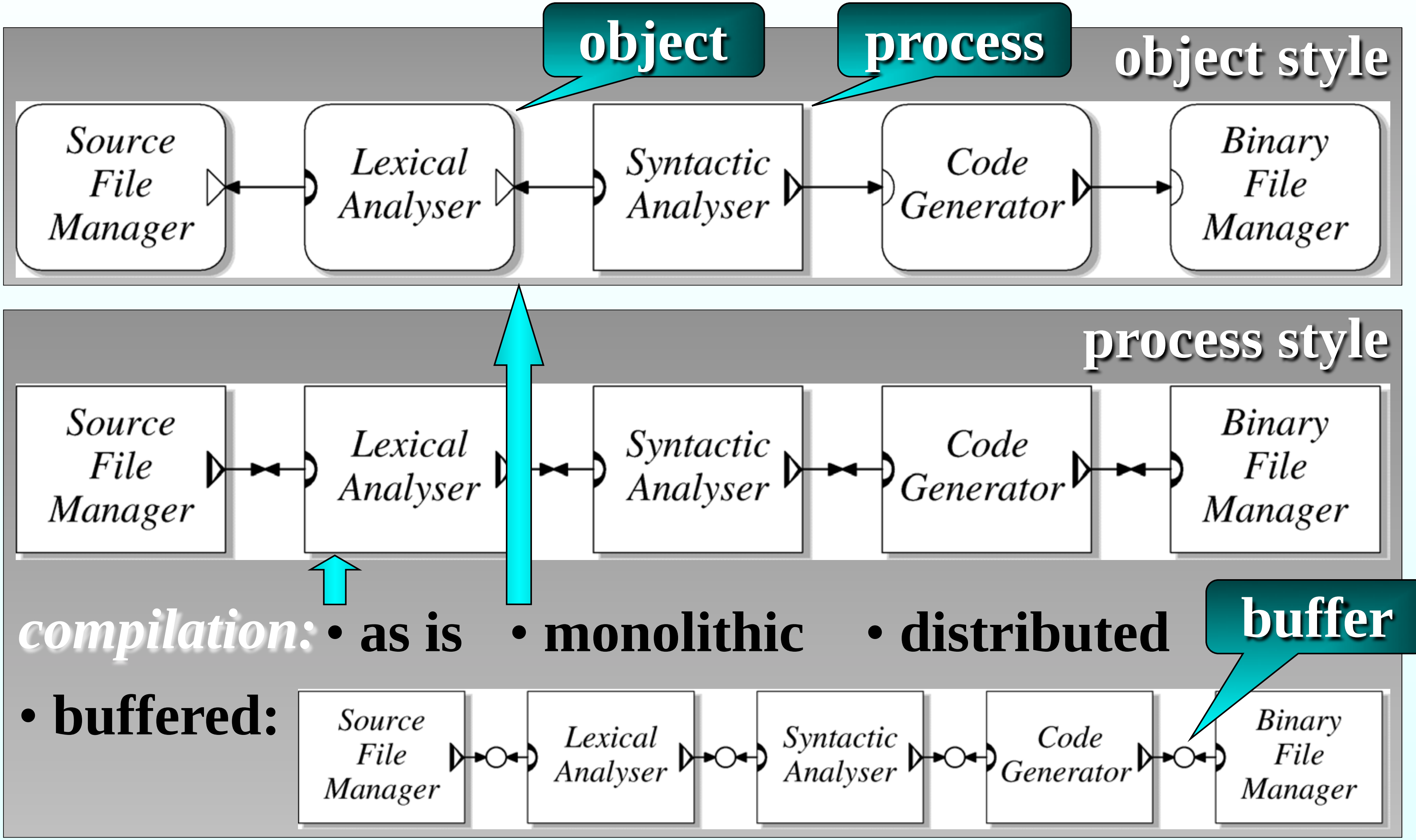
```
HelloWorld:
  using (Standard) process(init: StandardIn)
declare
  Parms: StandardInterface;
begin
  receive Parms from Init;
  call Parms.PutLine("Hello, World!");
  return Parms;
end process
```

PutLine is a variable

Towards Malleable Software



Towards Malleable Software



Towards Malleable Software

```
process Producer (channel c) is
  loop  // 0
    p;  c ! α;   // 1
    q;  c ! β;   // 2
    r;
  end loop
end Producer
```

```
process Consumer
(channel c) is
  ...
  c ? v;
  ...
end Consumer
```

process style


```
class Producer is
 integer state := 0;
 integer read() is
  loopcase state
  | 0:  p;  state := 1;  return α
  | 1:  q;  state := 2;  return β
  | 2:  r;  state := 0
  end loopcase
 end read
end Producer
```

inversion

```
process Consumer
(Producer c) is
  ...
  v := c.read();
  ...
end Consumer
```

object style

Towards Malleable Software

Principle of process generality

Process-oriented programming
is more general than
object-oriented programming

Towards Malleable Software

Per Brinch Hansen

‘It is astounding to me that Java's insecure parallelism is taken seriously by the programming community, a quarter of a century after the invention of monitors and Concurrent Pascal.

It has no merit.

[...]

Java ignores the last twenty-five years of research in parallel languages.’

Letter to the editor
ACM Sigplan Notices
April 1999

shared variables, weak monitor semantics

Towards Malleable Software

```
type Stream is [int(integer), eos];
```

message protocol

```
process sort(Stream i, o) is
  integer x;
  case
  | i ? eos:  o ! eos
  | i ? int(x):  subsort(x, i, o)
  end case
end sort;
```

Joyce (almost)

Towards Malleable Software

```
type Stream is [int(integer), eos];
```

message protocol

```
procedure subsort(integer x, Stream i, o) is
   Stream s := new Stream;  sort(s, o);
   integer y;
   loopcase
   | i ? eos:
     o ! int(x);  s ! eos;  exit loopcase
   | i ? int(y):
     if x > y then s ! int(x);  x := y else s ! int(y) end if
   end loopcase
end subsort;
```

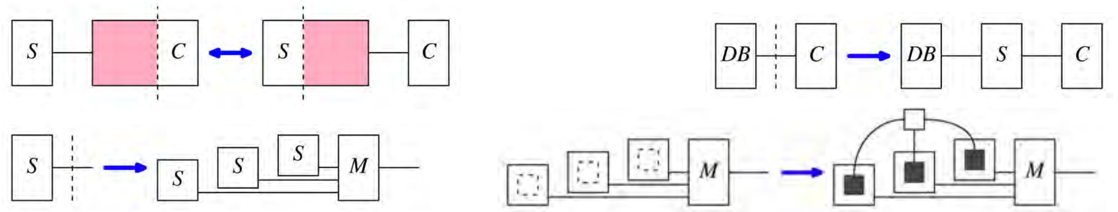
recursive process invocation

```
process sort(Stream i, o) is
   integer x;
   case
   | i ? eos:  o ! eos
   | i ? int(x):  subsort(x, i, o)
   end case
end sort;
```

Joyce (almost)

Towards Malleable Software

Remember the dotted lines



Compositional approach solves only half the problem; it composes but it does not deconstruct

Who can determine the point of discontinuity?

Today's large system is tomorrow's small

Principle of fractal construction

The same notation should be employed at all levels of scale

Towards Malleable Software

Scene: A design session
Dramatis personae:
Simon, a service object
Chloe, a client of Simon

object-oriented programming!

Chloe: Please tell me the value of property p.
Simon: No.
Chloe: (*Taken aback*)
But I really need to know the value of p.
Simon: Why?
Chloe: Because I need to splonge it.
Simon: Just ask me, and I'll splonge it for you.

Principle of property-less objects

Properties are not part of the object model

Towards Malleable Software

Principle of parameterised isolation

Arguments should be passed only by value and not by reference

Passing data by reference has been an unfortunate historical cul-de-sac,
caused by eagerness to optimise too early

Passing data only by value can be done without loss of efficiency

Hermes

occam-pi

Erlang

Deciding if passing by reference is possible, safe, and advantageous is best left to software.

Towards Malleable Software

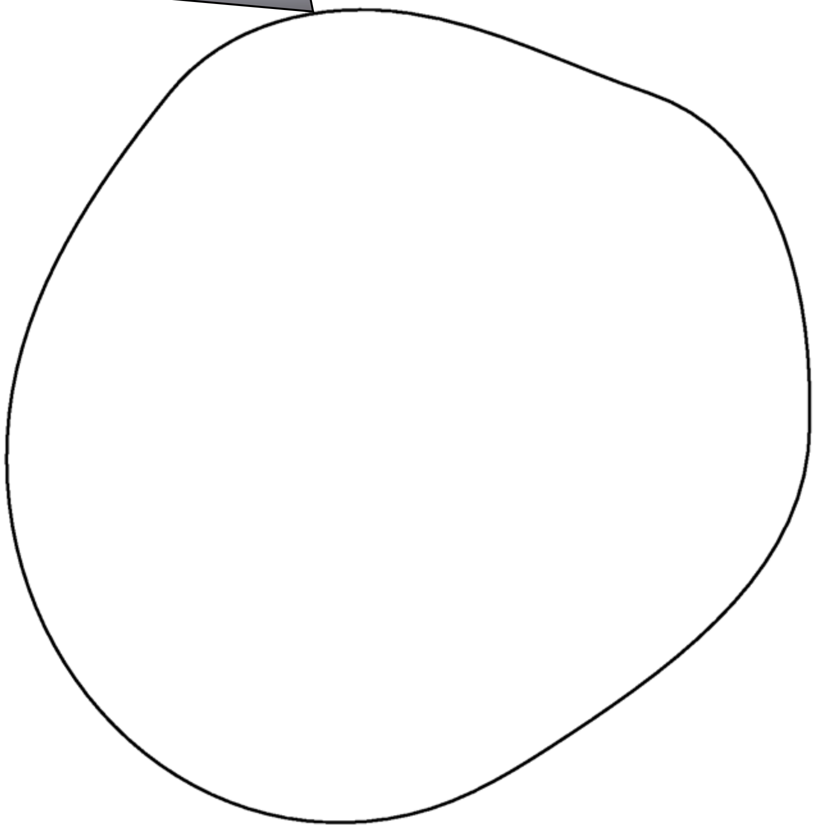
Part IV

A model for malleable software
– putting it all together

Towards Malleable Software

model for malleability

cell (encapsulation only, not relationships and not identity)

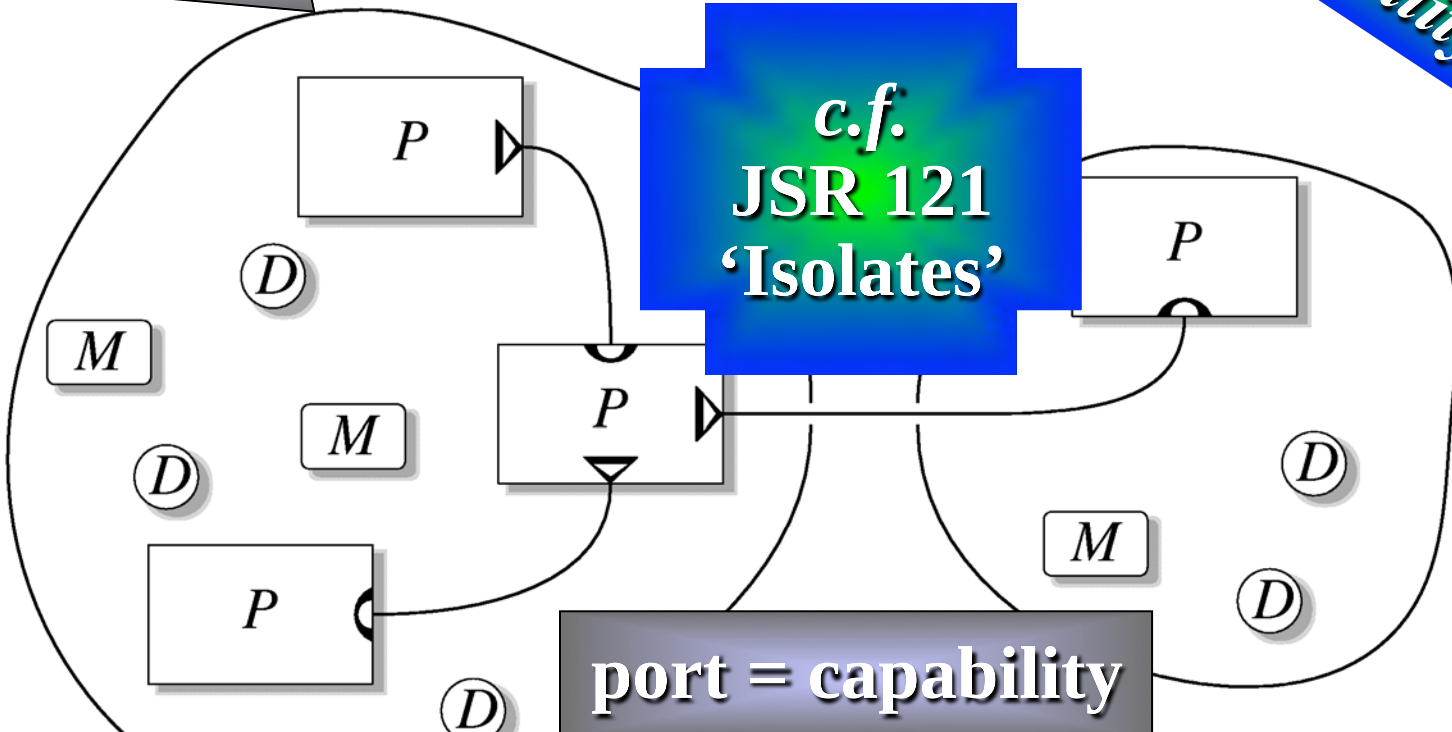


Towards Malleable Software

model for malleability

cell (encapsulation only, not relationships and not identity)

c.f. JSR 121 'Isolates'



port = capability

data, methods and processes all private	only ports can be imported	one program counter per cell; processes within a cell are coroutines

Towards Malleable Software

Part V

Research to verify or falsify the model.

– **Develop and prove the flexibility, simplicity and efficiency of the model**

Towards Malleable Software

project and criteria

complete first draft of model

demonstrate mapping of business models

model => abstract syntax => concrete syntax

simple compiler and interpreter for notation

configurability — process-oriented library, inc. GUI

code generator — IDE, with visualisation tools

Towards Malleable Software

project and criteria

complete first draft of model
demonstrate mapping of business models
model => abstract syntax => concrete syntax
simple compiler and interpreter for notation
configurability
code generator

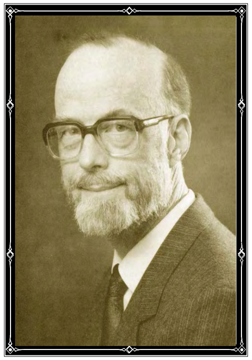
process-oriented library, inc. GUI
IDE, with visualisation tools

essential for performance
e.g. communication: Java 10,000i, Hermes 9i

10 years?

build complete system as is, monolithic, distributed ...
make random cuts; put extracts on remote machines
compiler (in malleable notation) compiles at 1,000 ipc

Towards Malleable Software



1940s Machine Codes

1950s Assembly Languages

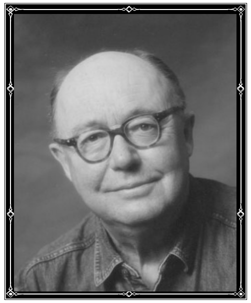
1960s High-level Languages

1970s Structured Programming

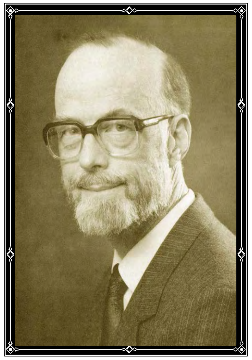
1980s Modular Programming

1990s Object-oriented Programming

2000s ???



Towards Malleable Software



1940s Machine Codes

1950s Assembly Languages

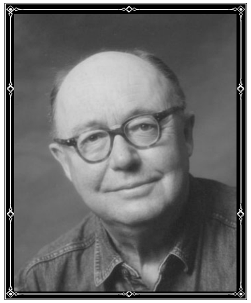
1960s High-level Languages

1970s Structured Programming

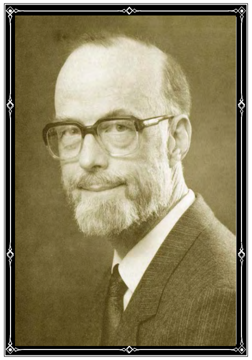
1980s Modular Programming

1990s Object-oriented Programming

2000s Process-oriented Programming



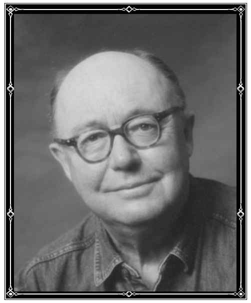
Towards Malleable Software

1940s Machine Codes

1950s Assembly Languages

1960s High-level Languages

1970s Structured Programming

1980s Modular Programming

1990s Object-oriented Programming

2000s Process-oriented Programming

2010s Malleability: *Modular Concurrency*