

A surreal illustration with a blue monochromatic palette. On the left, a white castle with red-roofed towers sits atop a dark, craggy cliff. A waterfall cascades down the cliff face. On the right, a large, brown hot air balloon is partially visible, with a basket falling from it into the churning blue water below. Several figures are seen in the basket, and long, colorful ribbons trail behind it. The overall scene is dreamlike and evocative of classic children's literature.

“Object-oriented programming done right”

...or what can we learn from the Smalltalk-80 heritage

Dmitry Matveev | 2020

Picture credit © BYTE Magazine, 1981-08

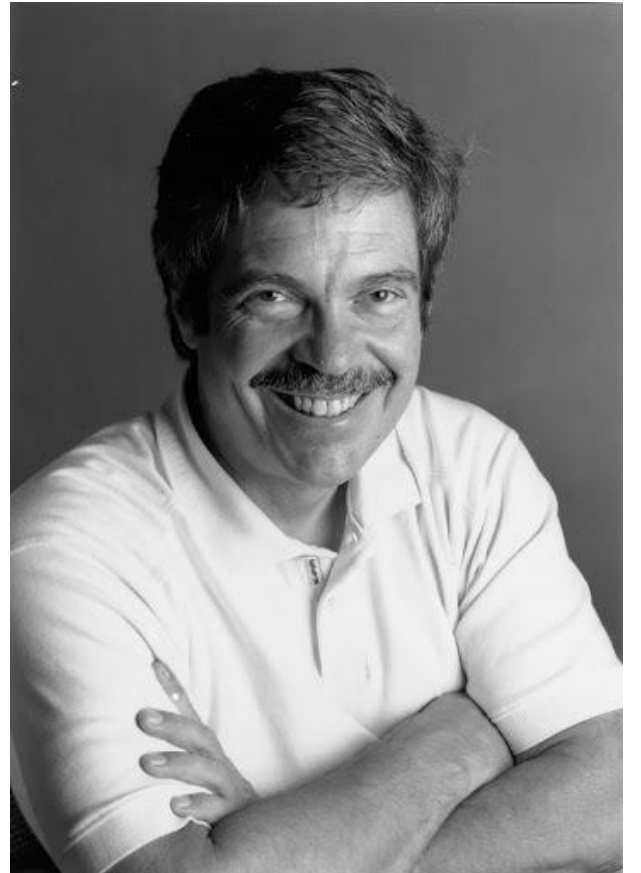
Contents

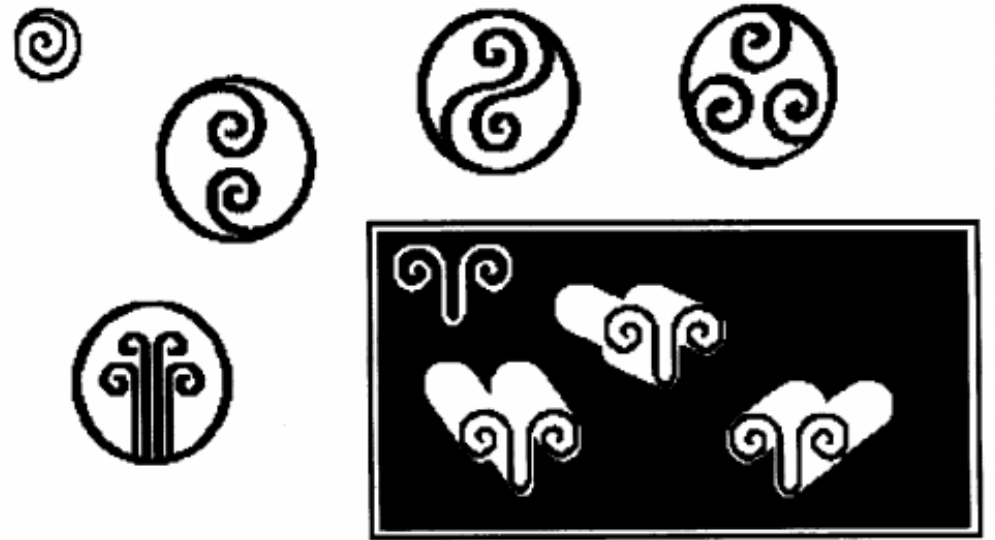
- Intro
- Smalltalk's place in history
- Understanding the language
- Understanding the environment
- Live demo
- Smalltalk today
- Resources on Smalltalk

Intro

“I invented the term ‘object oriented’, and C++ was not what I had in mind”

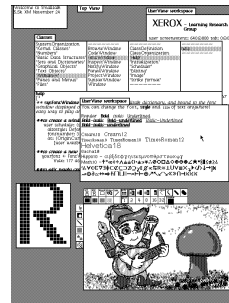
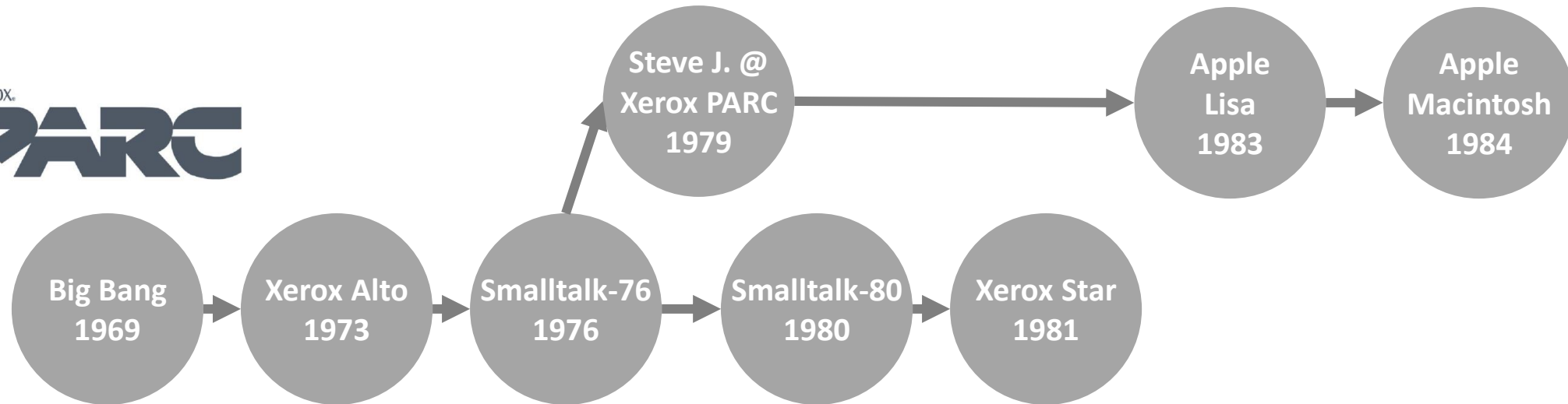
- Dr. Alan Kay



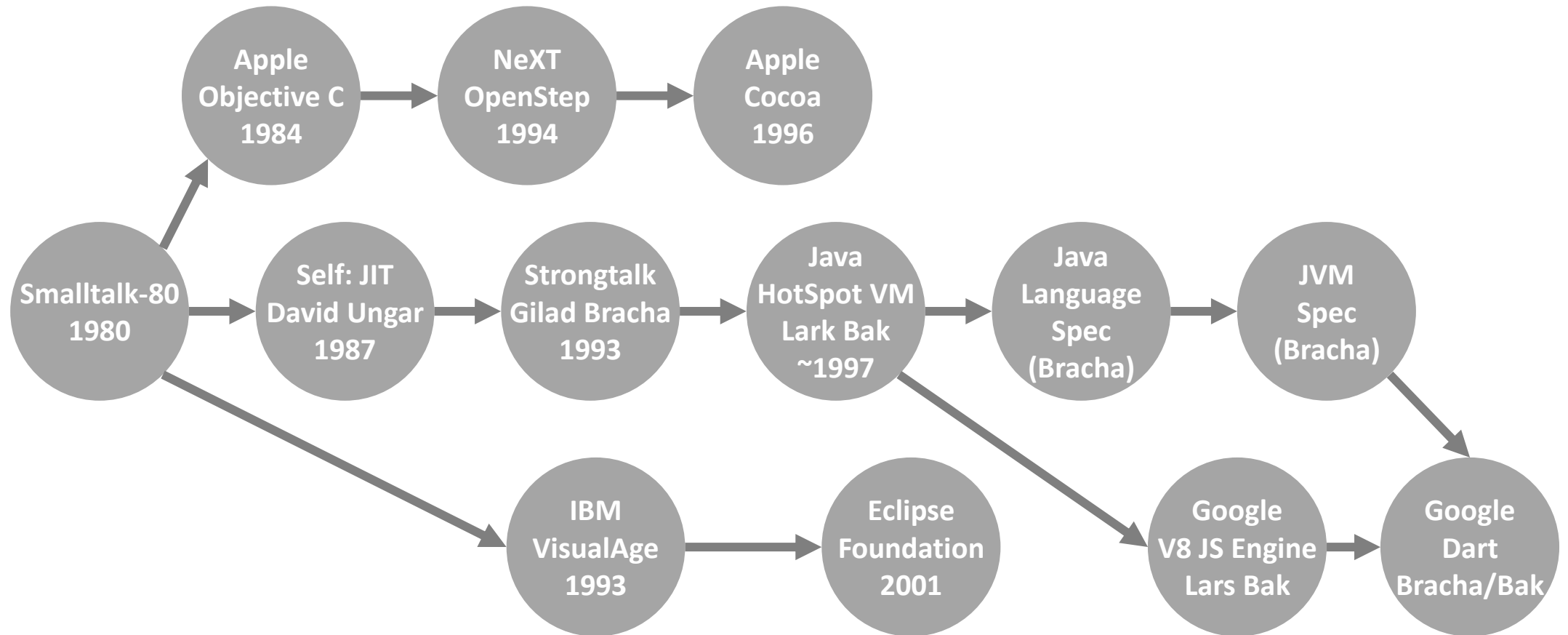


Smalltalk's place in history

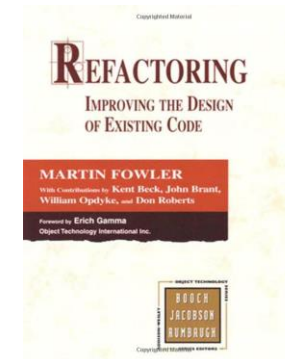
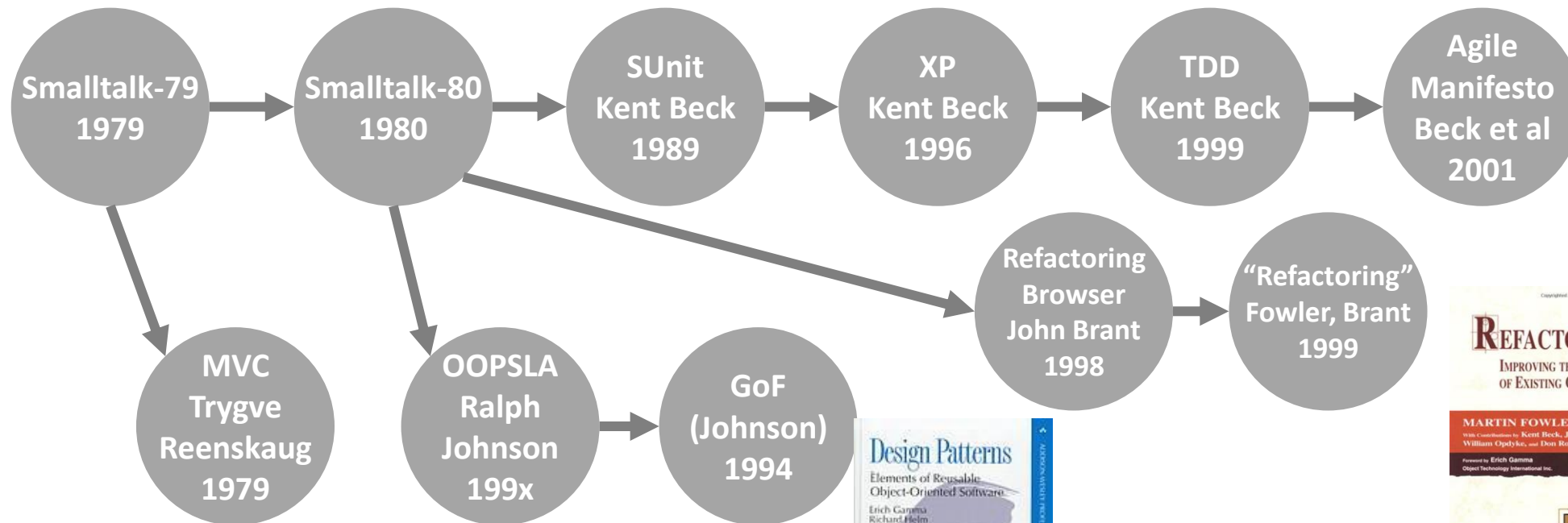
Smalltalk's place in history



Smalltalk's place in history, continued

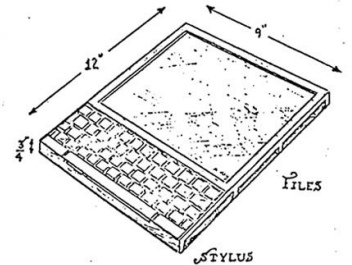
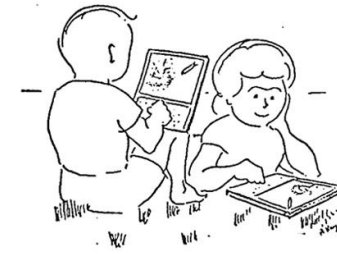


Smalltalk's place in history, continued again

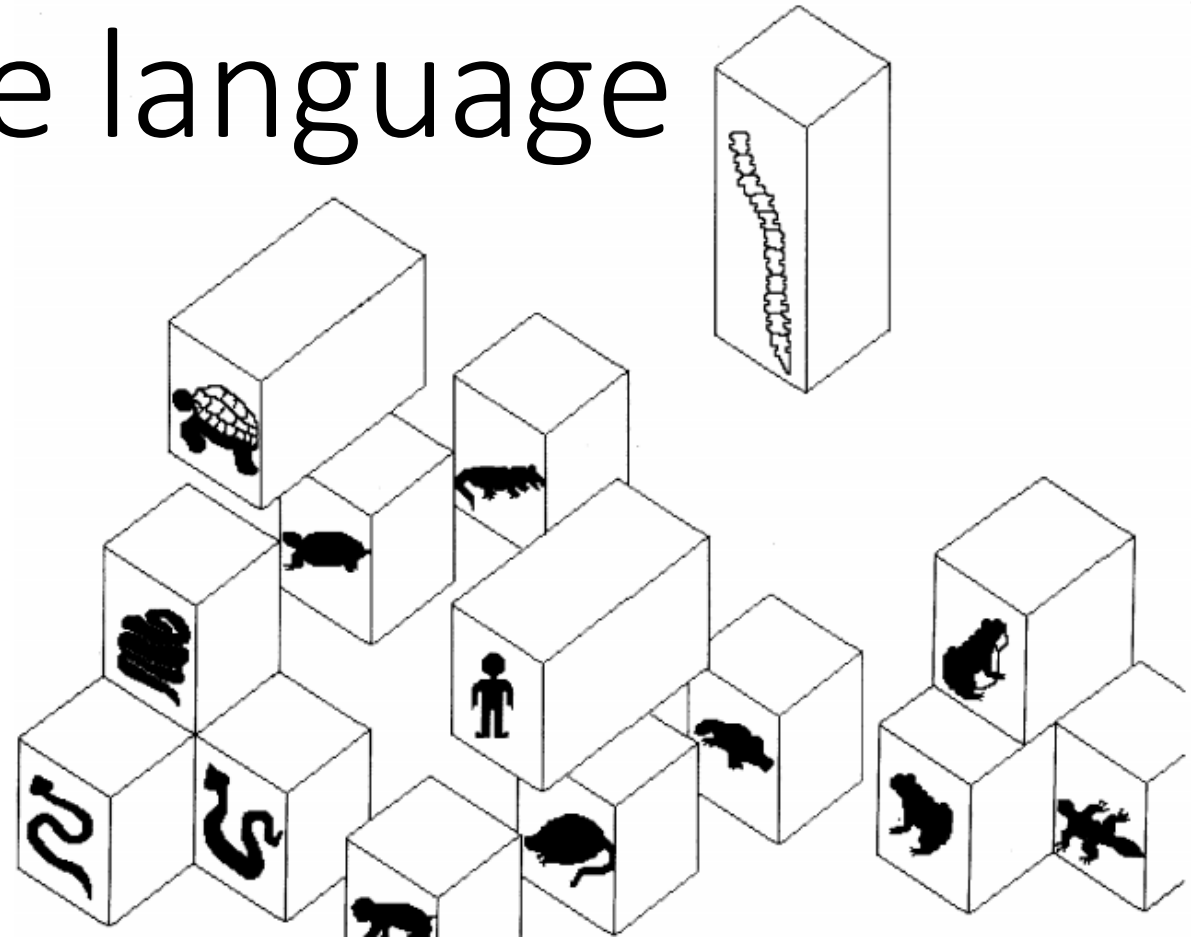


Smalltalk history in one slide

- ~1969: Xerox sets a goal to envision a “workplace of the future”
- 1970: **Alan Kay** joins Xerox Palo Alto Research Parc (PARC)
- 1972: “Dynabook: a personal computer for children of all ages”
- 1972: **Dan Ingalls** joins Xerox PARC
- 1972-80: Smalltalk-72, -74, -76, -79
- 1980-83: Smalltalk-80 (Release)
- 1983-...: Commercial Smalltalk packages
- 1996: Sun releases Java
- 1996-...: Decline in market share, focus on FOSS
- 199x – 200x: Squeak (led by the original authors), 200x – now: Pharo (fork)
- 2003: ACM Turing Award to Alan Kay for pioneering OOP & “fundamental contributions to personal computing”



Understanding the language



Quiz: how complex your programming language is?

class try/catch inheritance
do{}while()
If(){}else
throw
& (bw) If() * (ptr) & (addr)
virtual struct{} union{} SFINAE
ADL ?: C++ If(){}else if(){}
typedef while() * (arith)
casts for(;;) template<> RAI
rvalue multiple inheritance

Smalltalk concepts

1. Everything is **Object**
2. Objects communicate only by sending each other **Messages**

Smalltalk concepts: Objects (Literals)

123	<i>a SmallInteger</i>
16rff	<i>a SmallInteger (255)</i>
'Hello world'	<i>a String</i>
#HelloWorld	<i>a Symbol</i>
#(1 2 3)	<i>an Array</i>

Smalltalk concepts: Messages

2 squared => 4

Unary message

'hello', 'world' => 'helloworld'

Binary message

2 raisedTo: 3 => 8

Keyword message

Transcript

show: 'hello';

show: 'world';

cr.

Cascading

- *keyword*

- *keyword*

- *unary*

Smalltalk concepts: Messages – Quiz!

$$2 + 2 * 2$$



6



8

The correct answer is **8**. There's no arithmetic precedence for operators (as there is no operators except := and ^).
Message precedence order is defined as **unary > binary > keyword**.
2 gets #+: 2, results in 4, 4 gets #*: 2, results in 8.

Smalltalk concepts: Variables & Block closures

<code>[:x x + 1]</code>	<code>=></code>	a BlockClosure (object)
<code> a b </code>	<code>=></code>	declare two variables
<code>a := 3.</code>	<code>=></code>	assignment (literal)
<code>b := [:x x + 1].</code>	<code>=></code>	assignment (lambda)
<code>b value: a.</code>	<code>=></code>	Send keyword message value:

Smalltalk concepts: That's it!

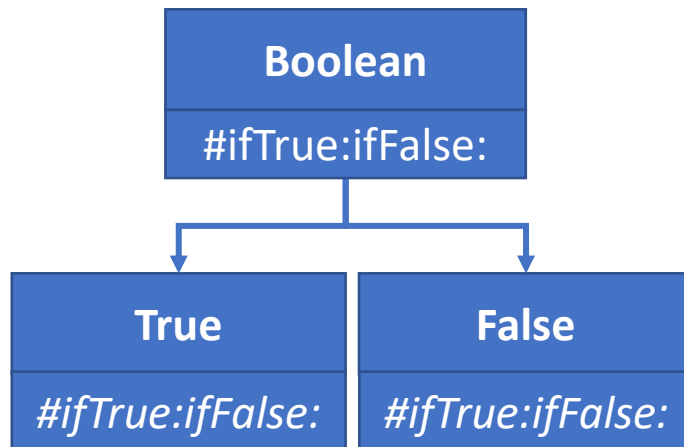
- Now you know the syntax.
- But... What about...
 - Conditionals?..
 - Loops?..
 - Classes?..
 - Overloading?..
 - Exception handling?..
- ...There's no special syntax for those.
- ...Everything is an object, and objects are sending messages!

Smalltalk concepts: Conditionals

(a > 2)

ifTrue: [a + 1]

ifFalse: [a - 1]



1. a `#> 2` -> a Boolean (true or false)
 1. true: a True
 2. false: a False
2. a Boolean **#ifTrue:ifFalse**
 1. True **#ifTrue:ifFalse**, OR
 2. False **#ifTrue:ifFalse**
3. True class: evaluates true block
4. False class: evaluates false block
5. Achievement unlocked: "IF without IF"



Smalltalk concepts: Collections

```
| things |  
things := OrderedCollection new.  
things add: 1.  
things add: 2.  
things add: 'Smalltalk - sila!'.  
|
```

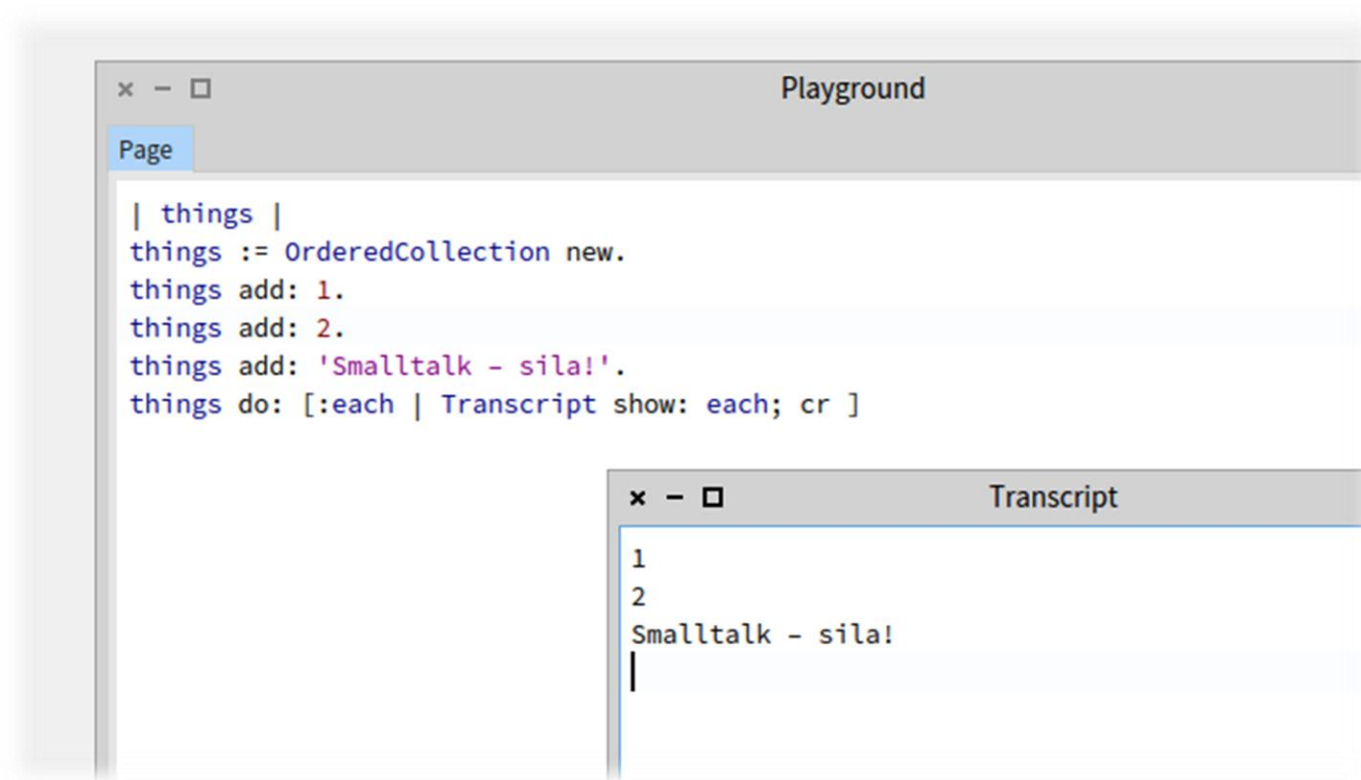
Class is the receiver
Everything is object!

Unary message

Keyword message

Smalltalk concepts: Collections

```
things do: [:each | Transcript show: each; cr ]
```



The image shows a screenshot of a Smalltalk environment window titled "Playground". Inside the window, there is a "Page" tab and a code editor containing the following Smalltalk code:

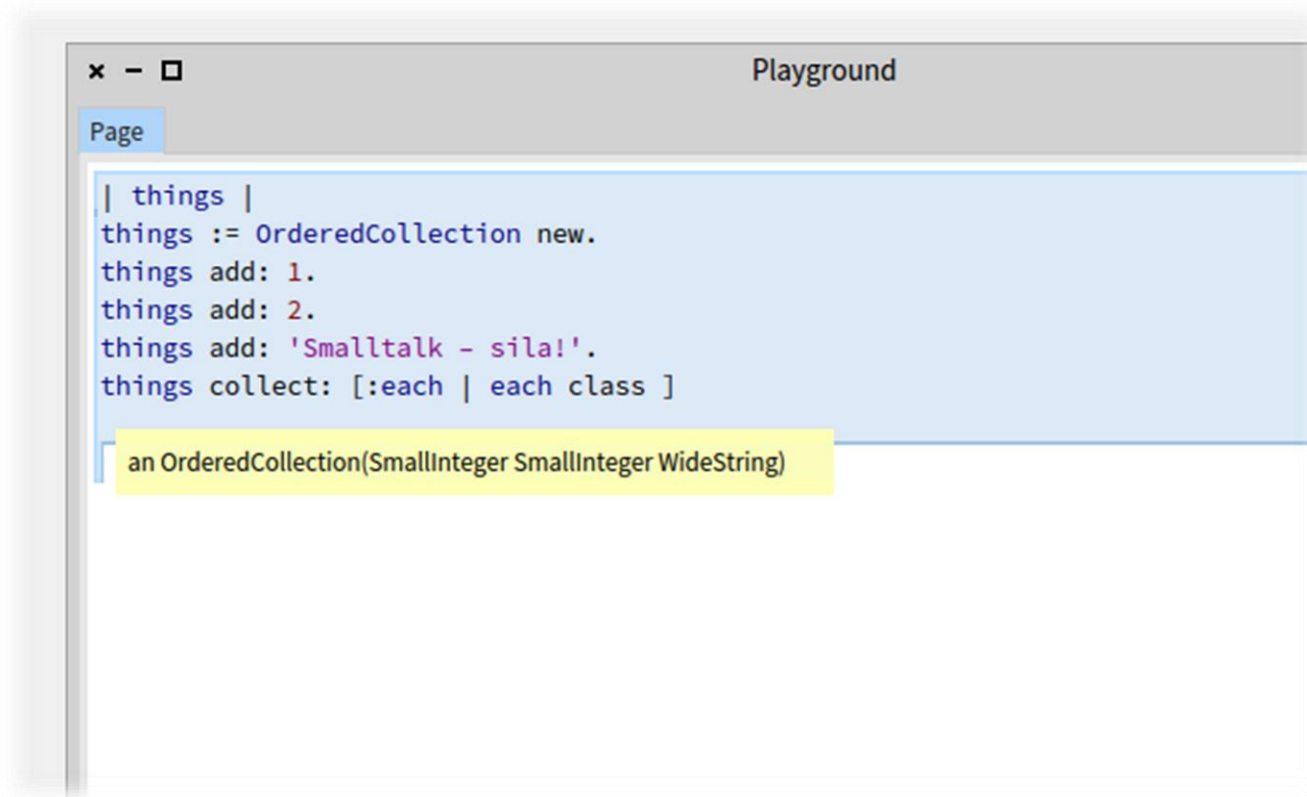
```
| things |  
things := OrderedCollection new.  
things add: 1.  
things add: 2.  
things add: 'Smalltalk - sila!'.  
things do: [:each | Transcript show: each; cr ]
```

Below the code editor, there is a "Transcript" window showing the output of the code execution:

```
1  
2  
Smalltalk - sila!  
|
```


Smalltalk concepts: Collections

```
things collect: [:each | each class]
```



The screenshot shows a window titled "Playground" with a "Page" tab. The code editor contains the following Smalltalk code:

```
| things |
things := OrderedCollection new.
things add: 1.
things add: 2.
things add: 'Smalltalk - sila!'.
things collect: [:each | each class ]
```

Below the code, the execution result is displayed in a yellow box: "an OrderedCollection(SmallInteger SmallInteger WideString)".

Smalltalk concepts: loops (not)

Integer >> timesRepeat: aBlock

```
10 timesRepeat: [Transcript show: 'Cool isnt it?'; space].
```

```
Transcript cr.
```

Integer >> to: aNumber do: aBlock

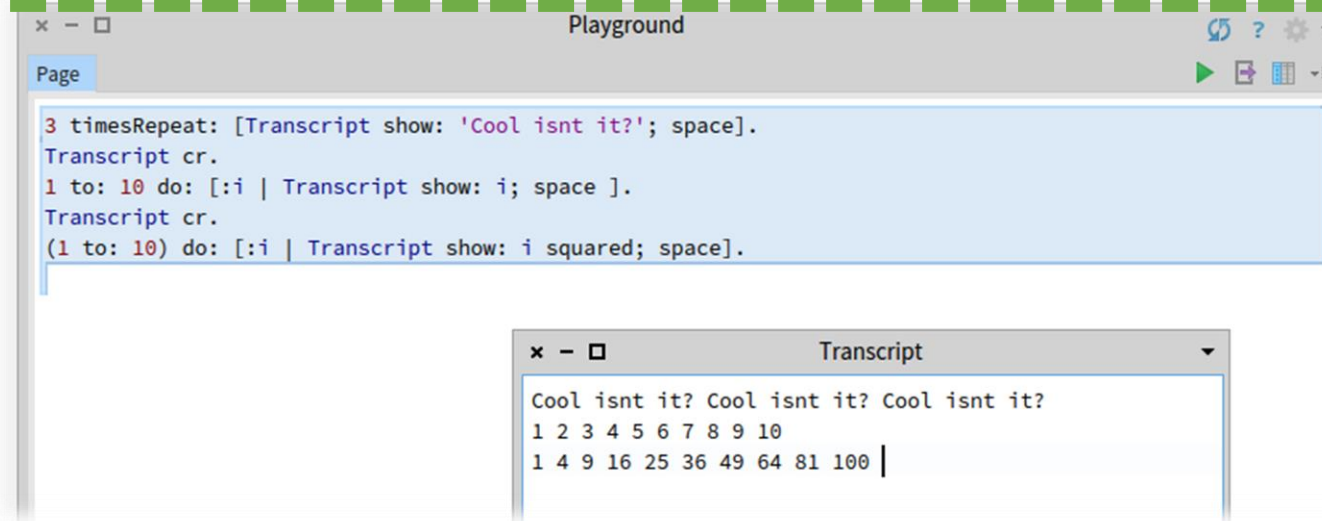
```
1 to: 10 do: [:i | Transcript show: i; space ].
```

```
Transcript cr.
```

Integer >> to: aNumber. Creates Interval

```
(1 to: 10) do: [:i | Transcript show: i squared; space].
```

Interval >> do: aBlock



The screenshot shows a 'Playground' window with a code editor containing the following Smalltalk code:

```
3 timesRepeat: [Transcript show: 'Cool isnt it?'; space].
Transcript cr.
1 to: 10 do: [:i | Transcript show: i; space ].
Transcript cr.
(1 to: 10) do: [:i | Transcript show: i squared; space].
```

Below the code editor is a 'Transcript' window showing the output of the code:

```
Cool isnt it? Cool isnt it? Cool isnt it?
1 2 3 4 5 6 7 8 9 10
1 4 9 16 25 36 49 64 81 100 |
```

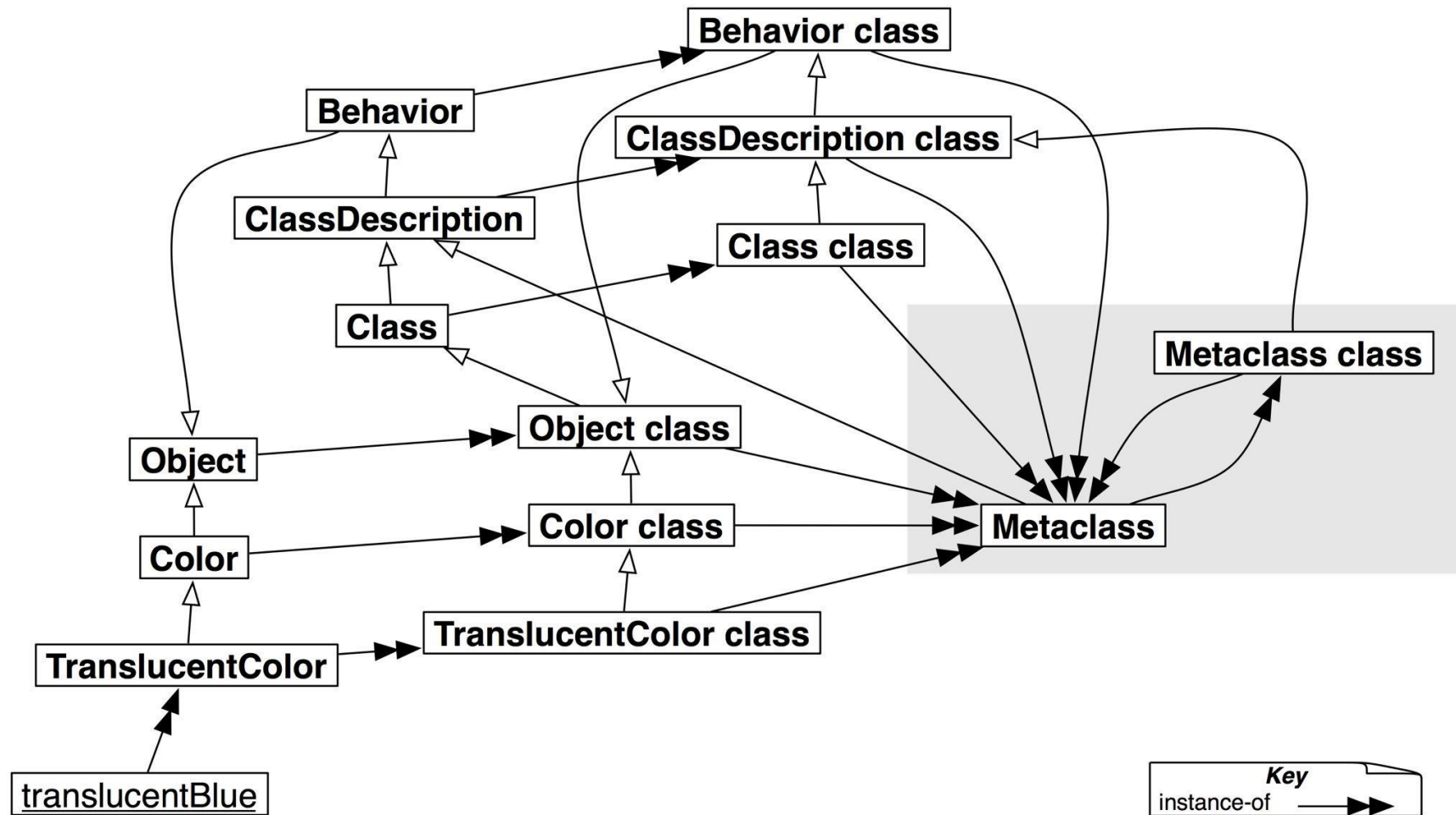
Smalltalk concepts: Classes

```
Object subclass: #Point
  instanceVariableNames: 'x y'
  classVariableNames: ''
  package: 'Kernel-BasicObjects'
```

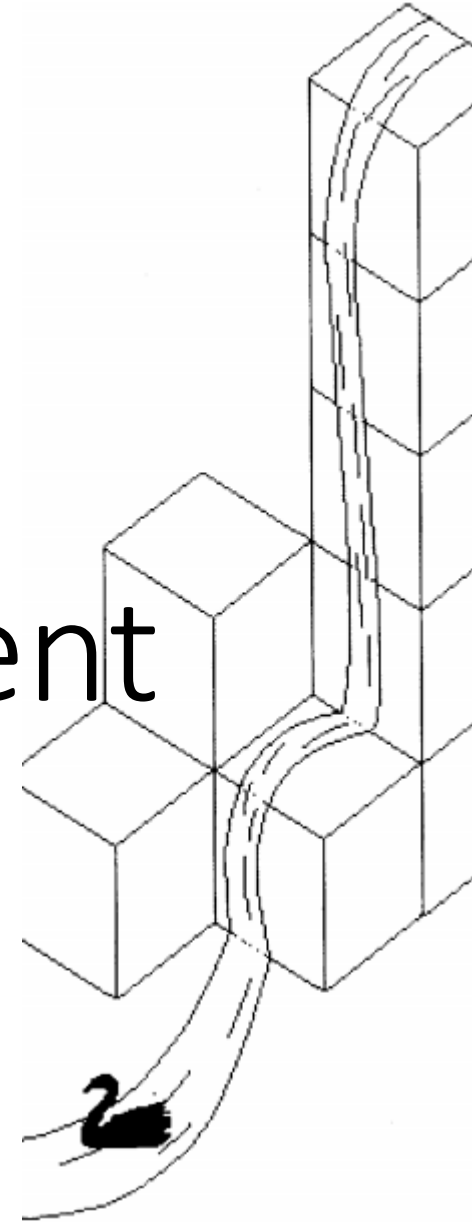
```
Number subclass: #Fraction
  instanceVariableNames: 'numerator
denominator'
  classVariableNames: ''
  package: 'Kernel-Numbers'
```

Inheritance (subclassing) is not a *declaration*.
It is a *message*. A message which is *sent*.
Sent in the *runtime*.

Smalltalk class hierarchy example



Understanding the environment



Smalltalk as an environment

- Smalltalk is not only a language, it is also an environment
- Smalltalk is **more** an environment than a language
- A **live** environment where your objects **live**...
- ...and you, programmer, interact with them
- Creating a new class is a fine example
 - **You** send a message to some class to make a subclass for it
- Adding methods can also be done with a message
 - In fact, this is how it is actually done
 - Smalltalk provides visual tools to do that

What scares beginners in Smalltalk

...or Frequently Asked Questions:

- Can I build an executable file?
 - **NO*** (*limited support by some commercial implementations only)
- Can I use my lovely Vim/Emacs*/Atom/VSCoDe/etc. to write code?
 - **NO*** (*there is a text-based GNU Smalltalk where you can, but it is not “true”)
- Can I use Git, Github, SVN, HG, etc?
 - **YES**, Git since late 2016. Prior attempts were not very successful (~201x)
- Can I use my OS' GUI system (widgets, buttons, etc)?
 - **Still NO** in the open implementations. Emulated in commercial Smalltalks
- Is my code portable across Smalltalk implementations?
 - **NO**, while language is basically the same, *Standard libraries* may differ at API level

Smalltalk environment: Class browser

The screenshot displays the Smalltalk class browser interface for the `ByteString` class. The interface is divided into four main sections, each highlighted with a green dashed box and a corresponding label:

- Class categories (packages):** A tree view on the left showing various packages such as `CodeImport-Tests`, `CodeImport-Traits`, `Collections-Abstract`, `Collections-Atomic`, and `Collections-Sequenceable`.
- Class hierarchy within a package:** A central list showing the hierarchy of classes under the selected package, including `String`, `ByteString` (highlighted), `Symbol`, `ByteSymbol`, `WideSymbol`, and `WideString`.
- Method categories:** A list of method categories for the selected class, including `instance side`, `extensions`, `accessing`, `comparing`, `converting`, `testing`, and `overrides`.
- Methods within category:** A list of methods for the selected category, including `asByteArray`, `asKeyCombination`, `asOctetString`, `at:`, `at:put:`, `beginsWith:`, `byteAt:`, `byteAt:put:`, `byteSize`, `convertFromSystemString`, `findSubstring:in:startingAt:matc`, `fuelAccept:`, and `hasWideCharacterFrom:to:`.

At the bottom of the interface, there is a code editor showing the class definition for `ByteString`:

```
String variableByteSubclass: #ByteString
instanceVariableNames: ''
classVariableNames: 'NonAsciiMap'
package: 'Collections-Strings-Base'
```

Smalltalk environment: Class browser

The screenshot shows the Smalltalk class browser interface. The title bar reads "ByteString>>asByteArray". The left pane shows a package browser with "ByteString" selected. The middle pane shows the class hierarchy with "ByteString" selected. The right pane shows the "instance side" of the class, with "extensions" expanded to show "converting" and "overrides". The "asByteArray" method is selected in the right pane. The bottom pane shows the source code for the "asByteArray" method:

```
asByteArray
| ba sz |
sz := self byteSize.
ba := ByteArray new: sz.
ba replaceFrom: 1 to: sz with: self startingAt: 1.
^ba
```

The bottom status bar shows "1/6 [1]" and "converting extension F +L W".

This is the place where you write code. Have fun!

Smalltalk environment: Class browser

```
emacs@DMITRYMA-MOBL
bool canMerge(const G1IslandNode& Graph &g,
             const adt::NodeHandle a_nh,
             const adt::NodeHandle fslot_nh,
             const adt::NodeHandle b_nh,
             const MergeContext &ctx = MergeContext())
{
    auto a_ptr = g.metadata(a_nh).getFusedIsland().object;
    auto b_ptr = g.metadata(b_nh).getFusedIsland().object;
    GAPI_Assert(a_ptr.get());
    GAPI_Assert(b_ptr.get());

    // Islands with different affinity can't be merged
    if (a_ptr->backend() != b_ptr->backend())
        return false;

    // Islands which cause a cycle can't be merged as well
    // (since the flag is set, the procedure already tried to
    // merge these islands in the past)
    if ((a_ptr->is_user_specified() || b_ptr->is_user_specified())
        && !std::contains(ctx.cycle_causers, std::make_pair(a_ptr, b_ptr)))
        return false;

    // There may be user-defined islands. Initially user-defined
    // islands also are built from single operations and then merged
    // by this procedure, but there is some exceptions.
    // User-specified island can't be merged to an internal island
    if ( ( (a_ptr->is_user_specified() && b_ptr->is_user_specified())
        || (a_ptr->is_user_specified() && !b_ptr->is_user_specified())
        || (!a_ptr->is_user_specified() && b_ptr->is_user_specified())
        )
        return false;
    else if (a_ptr->is_user_specified() && b_ptr->is_user_specified())
    {
        // These islands are different user-specified islands
        // FIXME: today it may only differ by name
        if (a_ptr->name() != b_ptr->name())
            return false;
    }
    // FIXME: add a backend-specified merge checker
    return true;
}

inline bool isProducedBy(const adt::NodeHandle &slot,
                      const adt::NodeHandle &island)
{
    // A data slot may have only 0 or 1 producer
    if (slot->inNodes().size() == 0)
        return false;
    return slot->inNodes().front() == island;
}

inline bool isConsumedBy(const adt::NodeHandle &slot,
                      const adt::NodeHandle &island)
{
    auto it = std::find_if(slot->outNodes().begin(),
                        slot->outNodes().end(),
                        [&](const adt::NodeHandle &nh) {
                            return nh == island;
                        });
    return it != slot->outNodes().end();
}

/**
 * Find a candidate island for merge for the given island nh.
 * @param g Island Model where merge occurs
 * @param nh G1Island node, either IHS or RHS of probable merge
 * @param ctx Merge context, may contain some cached stuff to avoid
 * double-triplet-checking
 * @return Tuple of island handle, data slot handle (which connects them),
 * and a position of found handle with respect to nh (IH/OUT)
 */
std::tuple<adt::NodeHandle, adt::NodeHandle, Direction>
findCandidate(const G1IslandModel& Graph &g,
             const adt::NodeHandle nh,
             const MergeContext &ctx = MergeContext())
{
    using namespace std::placeholders;
    // Find a first matching candidate G1Island for merge
    // among inputs
    for (const auto& input_data_nh : nh->inNodes())
    {
        if (input_data_nh->inNodes().size() != 0)
        {
            // Data node must have a single producer only
            GAPI_Assert(input_data_nh->inNodes().size() == 1);
            GAPI_Assert(input_data_nh->inNodes().front() == nh);
            // ...
        }
    }
}
(Unix) -- exec.cpp 17% (132,0) Git-da/infer-rol (C++11 -1 Abbrev)
```

VS.

The screenshot shows the Pharo 7.0 Smalltalk environment's class browser. The main window displays the class `ByteString` and its `asByteArray` method. The left sidebar shows a package browser with various collections and tests. The right sidebar shows the `asByteArray` method's implementation, which is a `self` message. The bottom status bar shows the current method being viewed: `converting` extension `F +L W`.

OpenCV G-API, subgraph clustering routine

Pharo 7.0, ByteString to ByteArray conversion routine

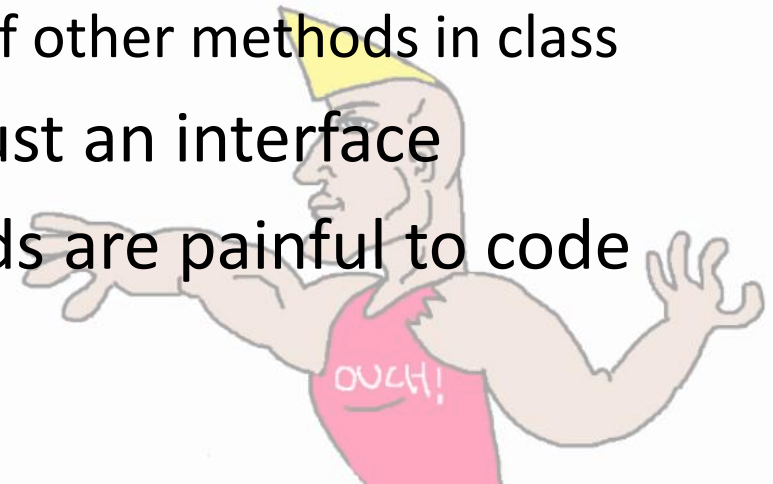
Text editor (“IDE”) vs. Smalltalk Class Browser

Text editor (or “IDE”)

- Gives you a way to edit *text*
 - It is up to you to map it to the object system
- Edit & Compile & Test loop
- You see the whole file
 - May be a white box!
- Details are noise
- You write code “as you wish”

Smalltalk Class Browser

- Gives you a way to edit *class*
 - And it is a real class, not an offline visualization
- Changes applied immediately
- You see one method at time
 - And a *list* of other methods in class
- No details, just an interface
- Long methods are painful to code



On programming in Smalltalk

- You have no “program”
- You always interact with a live, running system
- You change this live, running system
- Your changes to the system are incremental
- You modify classes (and so, objects) on the fly
- You can modify the system classes as well
 - *Extending* standard classes is OK, but *changing* their existing code may be usually hazardous
- There's no source file(s). “Sources” live with objects in the same space
 - Actually, Classes are Objects which keep their sources with them
 - When you modify a method source, the appropriate method is recompiled
 - *Method* is a way to handle *Message*
 - Methods are objects. Messages are objects, too. Everything is object, you know

More on programming in Smalltalk

- Method is your change & compile granularity
 - Smalltalk's VCS used this aspect to track changes, compared to text-based VCS
- In order to compile, your needed needs to be **syntactically** correct
 - This is a very weak requirement given the simplicity of grammar
- It means, that...
 - You can send messages which are not handled or unknown
 - You can refer to variables which don't exist
 - You can refer to classes which don't exist
- So this is how true Smalltalkers write their code!

How Smalltalkers code: Today

The image displays a collage of Smalltalk IDE windows illustrating a testing workflow. A central 'Test Runner' window shows a test suite 'IntelDemo' with 'PolyTest' selected, reporting '1 run, 1 passes, 0 skipped, 0 expected failures, 0 failures, 0 errors, 0 unexpected passes'. A red error message 'Unknown variable' is visible in the background. A 'Stack' window shows the call stack for 'testPolyCreation'. A 'Variables' window shows the state of 'self' (a Poly) and 'points' (an Array [3 items]). A 'Source' window shows the code for 'points := aCollection.'. A 'Debugger' window shows the execution of 'runCase' with a 'from:' message.

Test Runner (Top Left): IntelDemo | PolyTest | 1 run, 0 passes, 0 skipped, 0 expected failures, 1 failures, 1 errors, 0 unexpected passes

Test Runner (Center): IntelDemo | PolyTest | 1 run, 1 passes, 0 skipped, 0 expected failures, 0 failures, 0 errors, 0 unexpected passes

Stack (Top Right): Instance of Poly class did not understand #from: | testPolyCreation

Debugger (Bottom Right): Instance of Poly class did not understand #from: | from: | Dolt

Source (Bottom Left):

```
points: aCollection
points := aCollection.
```

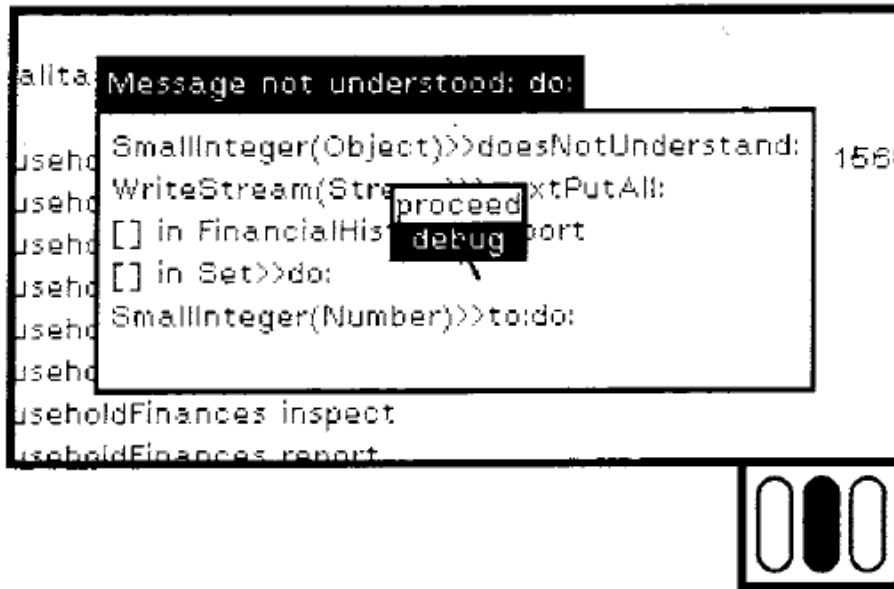
Debugger (Bottom Center): Instance of Poly did not understand #points: | from: | testPolyCreation

Debugger (Bottom Right): Instance of Poly class did not understand #from: | from: | Dolt

Debugger (Bottom Far Right): Instance of Poly class did not understand #from: | from: | Dolt

How Smalltalkers code: 1980

```
Message not understood: do:
SmallInteger(Object)>>doesNotUnderstand: 156
WriteStream(Stream)>>nextPutAll:
[] in FinancialHistory>>report
[] in Set>>do:
SmallInteger(Number)>>to:do:
useholdFinances inspect
useholdFinances report
```



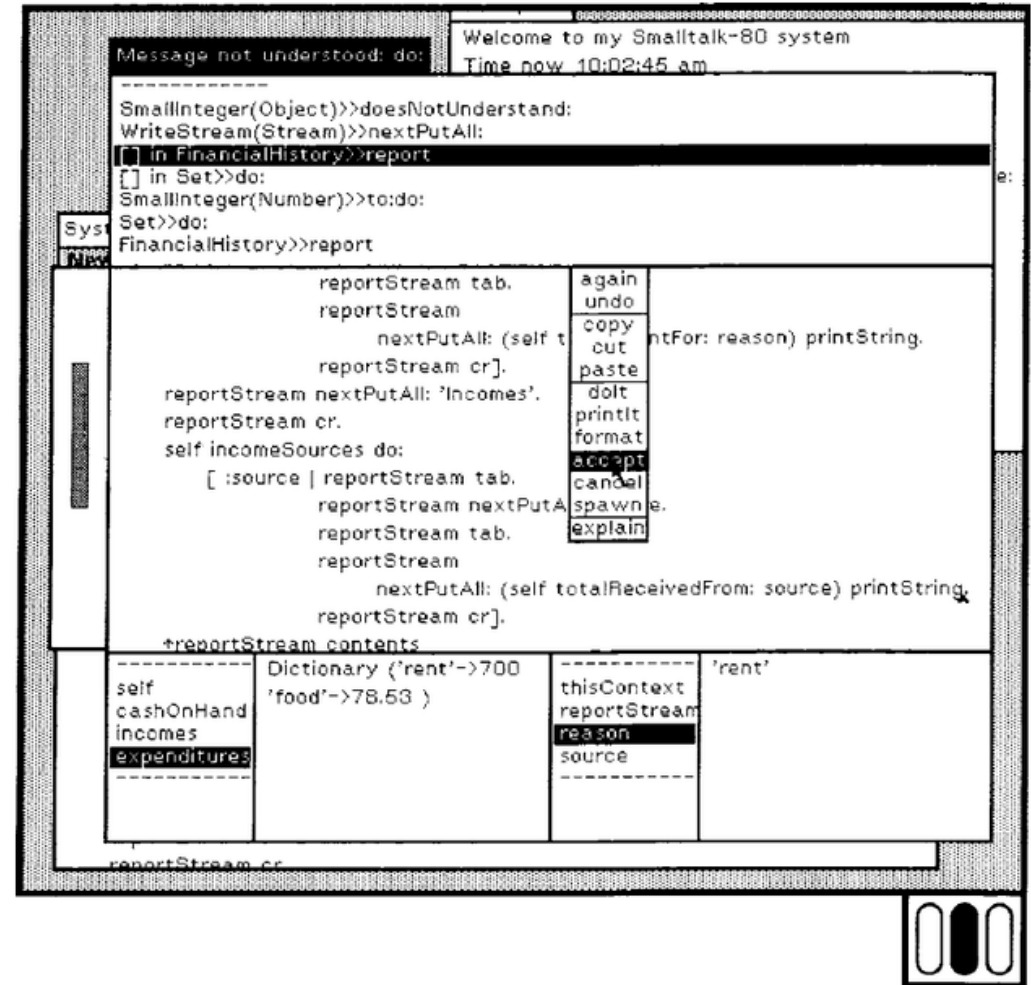
Smalltalkers code in debugger since the early beginning

No wonder now why Test-Driven Development originated in Smalltalk

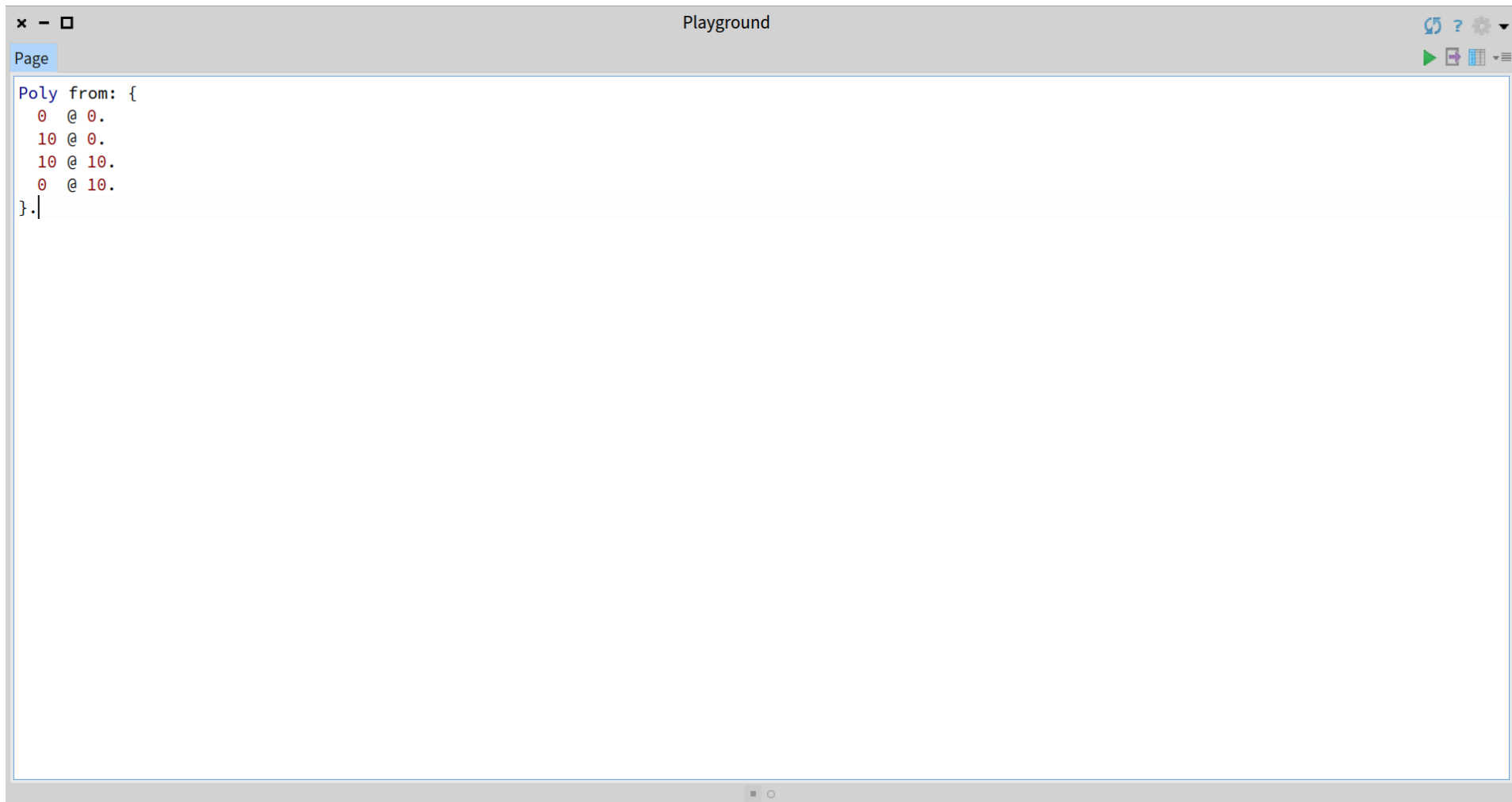
```
Welcome to my Smalltalk-80 system
Time now: 10:02:45 am
Message not understood: do:
SmallInteger(Object)>>doesNotUnderstand:
WriteStream(Stream)>>nextPutAll:
[] in FinancialHistory>>report
[] in Set>>do:
SmallInteger(Number)>>to:do:
Set>>do:
FinancialHistory>>report
```

reportStream tab.	again	
reportStream	undo	
nextPutAll: (self t	copy	ntFor: reason) printString.
reportStream cr].	out	
reportStream nextPutAll: 'Incomes'.	paste	
reportStream cr.	dolt	
self incomeSources do:	printIt	
[:source reportStream tab.	format	
reportStream nextPutA	accept	
reportStream tab.	cancel	
reportStream	spawn	
nextPutAll: (self totalReceivedFrom: source) printString.	explain	
reportStream cr].		
↑reportStream contents		

```
self
cashOnHand
incomes
expenditures
Dictionary ('rent'->700
'food'->78.53 )
thisContext
reportStream
reason
source
```



How Smalltalkers work with objects



The image shows a screenshot of a Smalltalk Playground window. The window title is "Playground". The code area contains the following Smalltalk code:

```
Poly from: {  
  0 @ 0.  
  10 @ 0.  
  10 @ 10.  
  0 @ 10.  
}.|
```

The code defines a list of four objects, each represented as a point with x and y coordinates. The objects are: (0, 0), (10, 0), (10, 10), and (0, 10). The code ends with a cursor at the end of the list.

How Smalltalkers work with objects

The screenshot shows a Smalltalk Playground window with the following content:

Page

```
Poly from: {  
  0 @ 0.  
  10 @ 0.  
  10 @ 10.  
  0 @ 10.  
}.
```

a Poly

Variable	Value
self	a Poly
points	an Array [4 items] ((0@0) (10@0) (10@10) (0@10))

"a Poly"
self

How Smalltalkers work with objects

The screenshot shows a Smalltalk Playground window with three panes. The left pane contains the following code:

```
Poly from: {  
  0 @ 0.  
  10 @ 0.  
  10 @ 10.  
  0 @ 10.  
}.
```

The middle pane, titled "a Poly", shows a variable table:

Variable	Value
self	a Poly
{ } points	an Array [4 items] ((0@0) (10@0) (10@10) (0@10))
{ } self	an Array [4 items] ((0@0) (10@0) (10@10) (0@10))
1	(0@0)
2	(10@0)
3	(10@10)
4	(0@10)

Below the table, the object's class and self are shown:

```
"a Poly"  
self
```

The right pane, titled "a Point ((10@0))", shows a variable table:

Variable	Value
self	(10@0)
x	10
y	0

Below the table, the object's class and self are shown:

```
"(10@0)"  
self
```

How Smalltalkers work with objects

The screenshot shows a Smalltalk Playground window with three object inspectors. The leftmost inspector shows the source code for a `Poly` object, which is a list of four points. The middle inspector shows the internal structure of a `Point` object, including its `x` and `y` coordinates. The rightmost inspector shows the value of a `SmallInteger` object, which is 10, and its `squared` method result.

```
Poly from: {
  0 @ 0.
  10 @ 0.
  10 @ 10.
  0 @ 10.
}.
```

Variable	Value
self	a Poly
{ } points	an Array [4 items] ((0@0) (10@0) (10@10) (0@10))
{ } self	an Array [4 items] ((0@0) (10@0) (10@10) (0@10))
1	(0@0)
2	(10@0)
3	(10@10)
4	(0@10)

Variable	Value
self	(10@0)
x	10
y	0

Variable	Value
self	10

"a Poly"
self

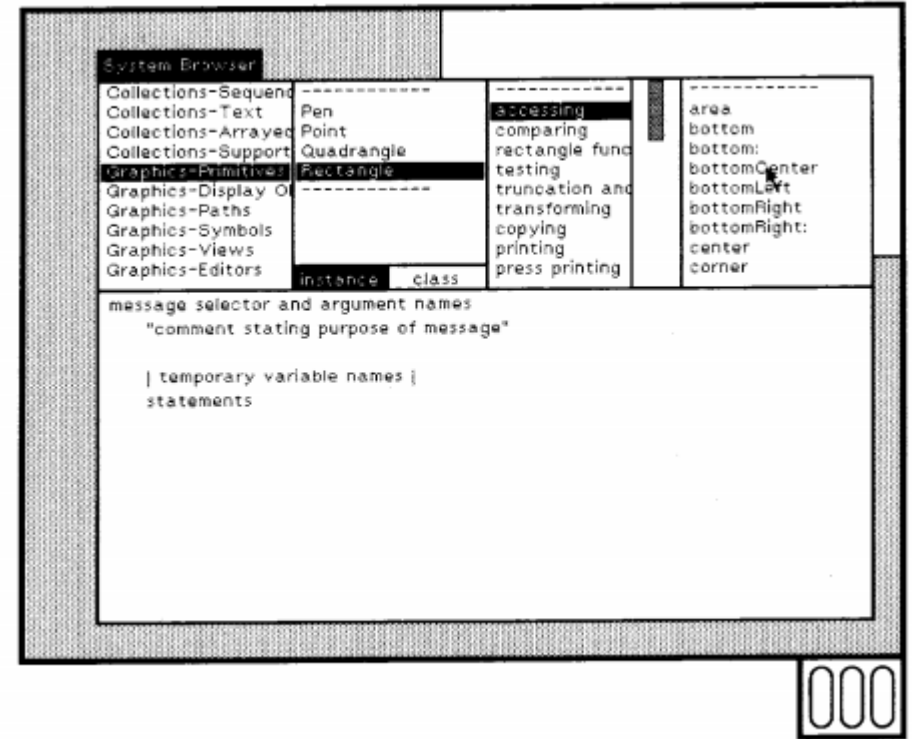
"(10@0)"
self

"10"
self squared

Finally: Image-based persistence

- A physical form of your Smalltalk *system* is an *image*
- Image is a snapshot of object space where your object (and your code) live
- Images are stored to disk (as 1-2 files)
- Images are the deployment model too
- Images capture the execution state, too (it is an object like everything)
- You can save image at any time
 - Even during the debugging session
 - You can get back to your debugging point days (weeks, months, years) later

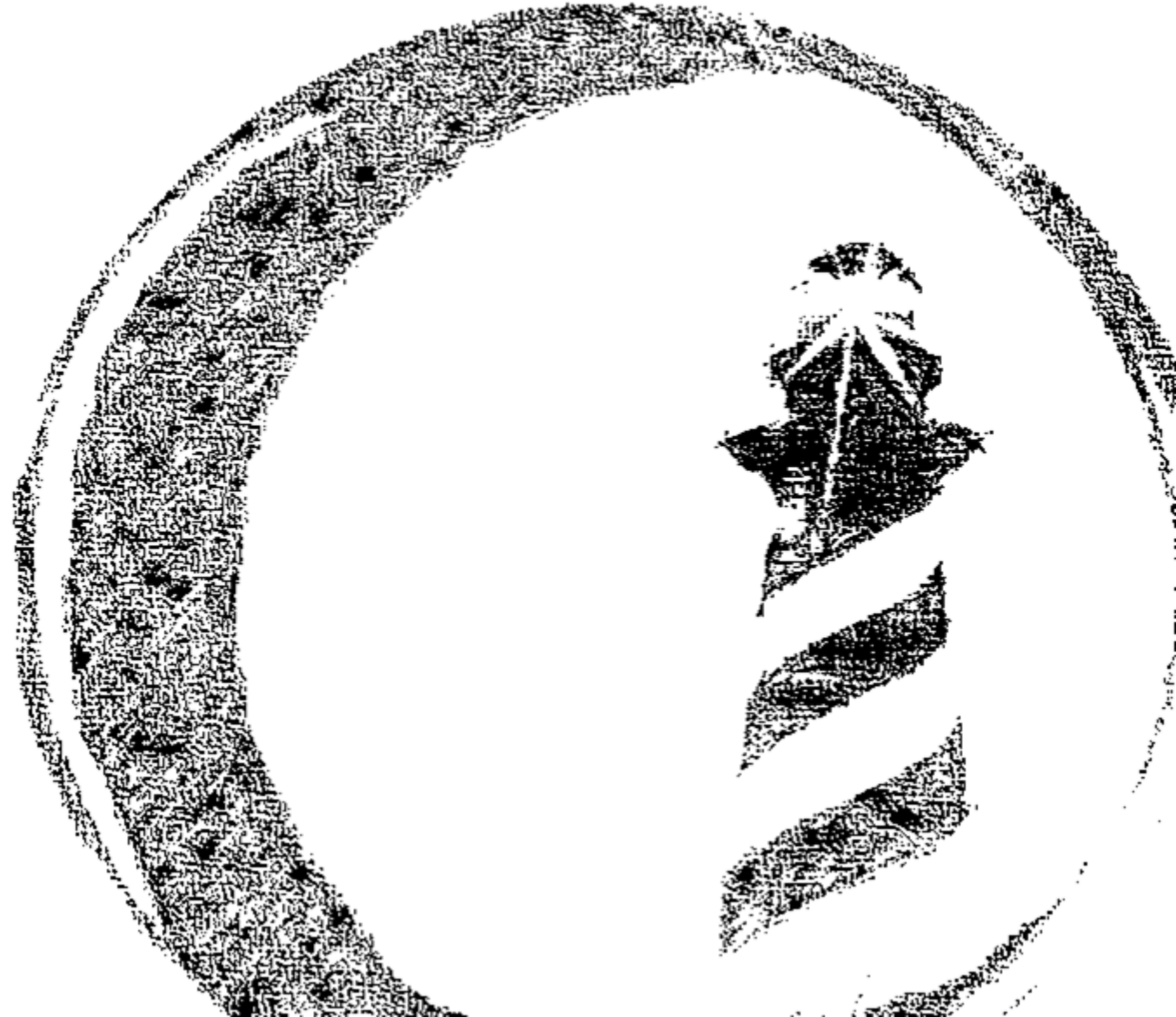
Live demo



What can we learn from Smalltalk

- “Simplicity is the ultimate sophistication”
- Look at your objects like never before
- Look at your classes like never before
- Look at your system like never before
- Fostering live, incremental development
- Fostering focus on interface rather than implementation
- ...Apply all this new knowledge in your daily work

Smalltalk today



Why we're not using Smalltalk every day now?

- I do use, what about you?
- Probably you just didn't know about it. Worth trying!

Why we're not using Smalltalk every day now?

- Overcharged price for its initial hardware, low early adoption
 - Xerox haven't consider itself a "Desktop" company
- Bad marketing and management decisions during the "dawn" era
 - Sun asked to license the language, got a high price tag. Now we have Java
- Initial "low" performance (compared to native C, Pascal, etc)
 - Thanks Moore's law it is not a problem anymore
- Lack of accessible implementations till mid-199x
 - Only commercial or experimental / compact Smalltalks were available
- Lack of the language standard (ANSI Draft of 1993)
 - Incompatible implementations
- Totally different paradigm
 - Own tools and development workflow, hardly compatible/reusable with "de facto"

Why we're not using Smalltalk every day now?

As a result: Chicken and Egg problem

- No jobs
 - There are Smalltalk jobs around the globe, but it is a tiny fraction compared to the “mainstream” languages
- Engineers don't learn Smalltalk since there's no jobs
- Companies don't go with Smalltalk since there's no engineers
 - If there is one and he leaves, where to find a replacement?
- At the same time...
 - All above points make Smalltalk a “secret weapon” for the Smalltalk-oriented teams

Smalltalk today: Implementations

Free / Open

Squeak

- Primary free Smalltalk in 90x-00x
- Defining the modern workflow & tools
- Driven by Alan Kay / Dan Ingalls
- Multi-purpose initially, mainly research now



Squeak!

Pharo

- Started as a Squeak fork (shares VM)
- Goal: clear license & focus on Web
- Primary free Smalltalk today
- All current advances in Smalltalk happen there



GNU Smalltalk

- Most mature from text-oriented Smalltalks
- Many Smalltalkers are unaware of it
- Modest list of supported packages
- Quite alive 10 years ago, last released 7 years ago



Commercial

Cincom VisualWorks

- The most advanced commercial Smalltalk for decades
- Based on the original Smalltalk-80 code as Squeak
- Mainly serves legacy systems now
- Focus on Desktop and Web



Instantiations Visual Age Smalltalk

- Former IBM Visual Age Smalltalk
- Gains new momentum now
- Focus on IoT and Embedded



GemStone/S

- Scalable distributed fail-safe object space
- A continuation of tech acquired by VMWare
- A OODB backend to many commercial Smalltalk deployments



Smalltalk today: Notable deployments

Source: Cincom

Cincom

Overview Goal:

- Leverage JP Morgan's success, which is based on the time-to-market of new products.
- Scale up products in the market to gain significant market share.

Challenge: Provide a development environment that:

- Accommodates the highly complex nature of JP Morgan's derivative products.
- Provides unparalleled productivity to stay ahead of the competition.
- Has the scalability to be deployed to support extremely high trading volumes.

Solution: Cincom Smalltalk

Results:

- The Smalltalk-developed Kapital system has enabled JP Morgan to be the market leader.
- JP Morgan estimates that Smalltalk is three times more productive than other languages, which allows them to consistently beat the competition.
- Revenues from the business that Kapital supports contribute an extraordinary percentage of JP Morgan's total revenues.

PROFILE IN SUCCESS

JP Morgan

JP Morgan Derives Clear Benefits from Cincom Smalltalk™

JP Morgan Chase is a leading financial services firm serving capital markets throughout the world. With assets of \$1.1 trillion and a component of the Dow Jones Industrial Average, capabilities include investment banking, research, private equities, investment management, private banking and treasury and security services.

As one of the world's leading investment banks, it has extensive relationships with corporations, financial institutions, governments and institutional investors worldwide. The firm provides a full range of investment-banking and commercial-banking products and services, including advising on corporate strategy and structure, raising capital in equity and debt markets, sophisticated risk management and market-making in cash securities and derivative instruments in all major capital markets. It also commits the firm's own capital to proprietary investing and trading activities.

One of the key elements of JP Morgan's success is based on one of its differentiators, the time-to-market of new products. Supporting this is a commitment to use information technology to provide this competitive advantage.

"With such a high productivity factor that Smalltalk gives us, reaction times to market changes have enabled us to beat most of our competitors."

– Dr. Colin Lewis
Vice President, JP Morgan

Overview Goal:

- Leverage JP Morgan's success, which is based on the time-to-market of new products.
- Scale up products in the market to gain significant market share.

Challenge:

Provide a development environment that:

- Accommodates the highly complex nature of JP Morgan's derivative products.
- Provides unparalleled productivity to stay ahead of the competition.
- Has the scalability to be deployed to support extremely high trading volumes.

Solution: Cincom Smalltalk

Results:

- The Smalltalk-developed Kapital system has enabled JP Morgan to be the market leader.
- JP Morgan estimates that Smalltalk is three times more productive than other languages, which allows them to consistently beat the competition.
- Revenues from the business that Kapital supports contribute an extraordinary percentage of JP Morgan's total revenues.

JP Morgan Chase is a leading financial services firm serving capital

Smalltalk today: Notable deployments

Source: Cincom



GemStone and Orient Overseas Container Lines: A Shipping Industry Case Study

1.5 billion data objects.

That's a lot of information to keep track of in an online system; information that is shared and accessed by more than 5,800 people in 150 offices around the globe. It is information that affects departments as diverse as Financial Accounting, Customer Service, Vendor Management, Legal, and Sales. It is information that changes and requires attention from minute to minute, directly affecting revenue opportunities and profitability of operations. Welcome to the world of Orient Overseas Container Lines Ltd. (OOCL).

Hong Kong-based OOCL is an International Container Transport and Logistics service

customer service. The *Integrated Regional Information System*, known as IRIS-2, coordinates all facets of OOCL's core business from the initial order placement to moving goods and reconciling accounting.

OOCL had impressive goals when creating IRIS-2. The software had to be able to coordinate information used by employees and business applications across the entire company. Tracking container movements and costs was necessary in order to make operations more efficient. Enabling rapid customer response was critical in growing revenue opportunities and winning business from competitors.

TECHNICAL SIDEBAR

USER: Orient Overseas Container Line Ltd. (OOCL)
PARENT COMPANY: Orient Overseas (International) Ltd.

DATA MANAGEMENT SOFTWARE:
GemStone/S, from GemStone Systems, Inc.

FRONT END DEVELOPMENT SOFTWARE:
Cincom Smalltalk VisualWorks

HARDWARE: HP and Sun servers
OPERATING SYSTEMS: HP-UX and Solaris

ARCHIVAL DATABASE: Sybase
INTEGRATED THIRD-PARTY APPLICATIONS:
mySAP ERP Financials

SYSTEM LOAD:

- 2700 end users
- 150 offices worldwide
- 1600 concurrent users at peak times
- 4000-5000 new shipment bookings per day
- 10,000 data updates per day
- 50-70 TPS for data reads/searches/updates
- 4-5 commits per second
- 1.9 billion objects persisted in GemStone database

Smalltalk today: Notable deployments

Source: Cincom, Lam RC



Profile in Success: [Rudolph Technologies](#)

Rudolph Technologies Helps Semiconductor Customers Reach Market Faster with Smalltalk-Based ControlWORKS

Goal:

Implement a feature-rich development framework specifically for semiconductor-equipment control.

Challenges:

The framework must reduce the overall cost of control system development and field support by:

- Supporting rapid prototyping for shorter time-to-market
- Being highly reusable and maintainable, increasing development-team productivity
- Providing excellent configuration control for support of multiple product versions

Solution:

ControlWORKS, built on Cincom® VisualWorks®

Key Results:

- Systems implemented in four to eight months, not two to three years
- Time-to-market cut from two to three years to six to nine months
- The control organization reduced by up to 90%
- Maintenance costs cut due to built-in standard processes



Smalltalk today: Notable deployments

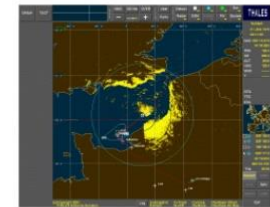
Source: ESUG, THALES

Smalltalk in Thales Brest 

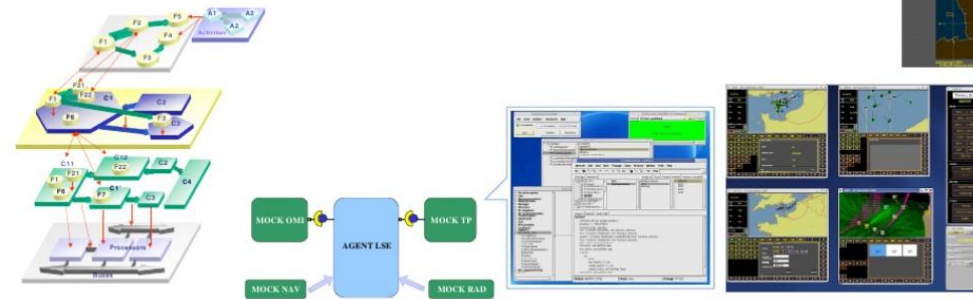


A story of twenty years:

- 1989-1997: Sensor software development
 - MMI development on real time operating system (VRTX), C (TNI) code generator
 - MMI development for embedded workstation, Static typing, C++ (Thales) code generator
 - Automatic test workbench (IEEE488, VxWorks)
- 1996-2000: Training centers
- 1996-2003: MMI workstation for Maritime patrol aircraft demonstrator
- 2002-2009:
 - System Modeling and Mockup
 - Component testing



Reference - data



2 Aéronautique

THALES

<https://www.youtube.com/watch?v=Oq1RSDn2P5Y>

Smalltalk ~~today~~: Notable deployments

The screenshot shows the Scratch IDE interface. The main workspace displays a cat sprite named "Sprite1" with the following script:

```
when green flag clicked
  clear
  set frequency to 0
  set index to 0
  delete all of frequencies
  forever loop
    if index = length of input
      set index to 0
      change index by 1
      set this_input to item index of input
      change frequency by this_input
      if frequencies contains frequency
        say frequency
        stop script
      else
        add frequency to frequencies
```

The right-hand side of the interface shows the "test" window with the following data:

input	
208	-11
209	-10
210	-1
211	-14
+ length: 1004	

frequencies	
21293	105
	11
21294	105
	10
+ length: 21294	

Below the test window, there are three variable monitors:

- index: 210
- frequency: 10510
- this_input: -1

The bottom of the interface shows the "New sprite:" button and a small preview of the cat sprite.

Smalltalk ~~today~~: Notable deployments



The screenshot shows the OSCON PLANNER interface with a calendar view for July 6-12, 2003. The calendar is titled "Portland Perl" and shows sessions scheduled for Sunday, July 6, and Wednesday, July 9. Sessions include "Writing Perl Extensions in C" on Sunday and "One Perl to" on Wednesday. The interface includes a sidebar with filters for "Session", "Keywords" (perl), and "Room is" (Portland). A tip box suggests specifying a home view.

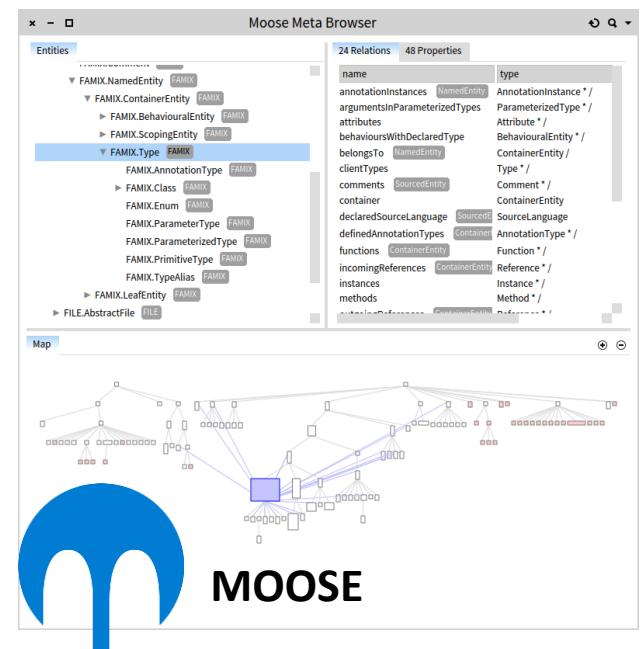
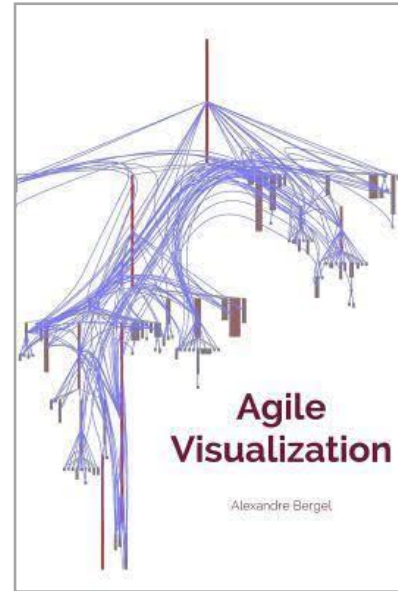
The screenshot shows the OSCON PLANNER interface with a list view of sessions. The list is titled "Unsaved View" and shows sessions categorized by room: Columbia, Eugene, Oregon Ballroom, and Portland. Each session entry includes a checkbox, session name, room, presenter name, and a summary. A tip box suggests saving the view.

	Room	Presenter Name	Summary
<input type="checkbox"/>	Columbia	Shane Warden	Type: Tutorial, Description: Done well, unit testing can improve code quality and increase...
<input type="checkbox"/>	Eugene	Marty Pauley	Type: Tutorial, Description: This one-day tutorial provides a hands-on introduction to Ext...
<input type="checkbox"/>	Eugene	Tim Bunce	Type: Tutorial, Description: Learn how the DBI works and how to get the best out of it, in...
<input type="checkbox"/>	Eugene	Phil Tomsson	Type: Session, Description: It's often valuable to learn a new programming language becau...
<input type="checkbox"/>	Oregon Ballroom		Type: Plenary, Description: Larry Wall's annual address on what's new in the world of Perl...
<input type="checkbox"/>	Portland	Andy Lester	Type: Session, Description: Perl's testing tools have been traditionally aimed at modules,...
<input type="checkbox"/>	Portland	Jeff Horwitz	Type: Session, Description: extproc_perl extends an Oracle database by allowing stored pro...
<input type="checkbox"/>	Portland	Dave Rolsky	Type: Session, Description: It's often better to give up in a controlled manner than to fo...
<input type="checkbox"/>	Portland	Merijn Broeren	Type: Session, Description: Perl is used extensively at Morgan Stanley to intensively mana...
<input type="checkbox"/>	Portland	Randal L. Schwartz	Type: Session, Description: Randal and co-conspirator Phoenix offer an informative (and hu...

- “Secret weapon” success story
- Successful project developed rapidly (months) and ran by just a few people
- Acquired by Twitter in 2010, closed in 2011

Smalltalk today: Community

FAST



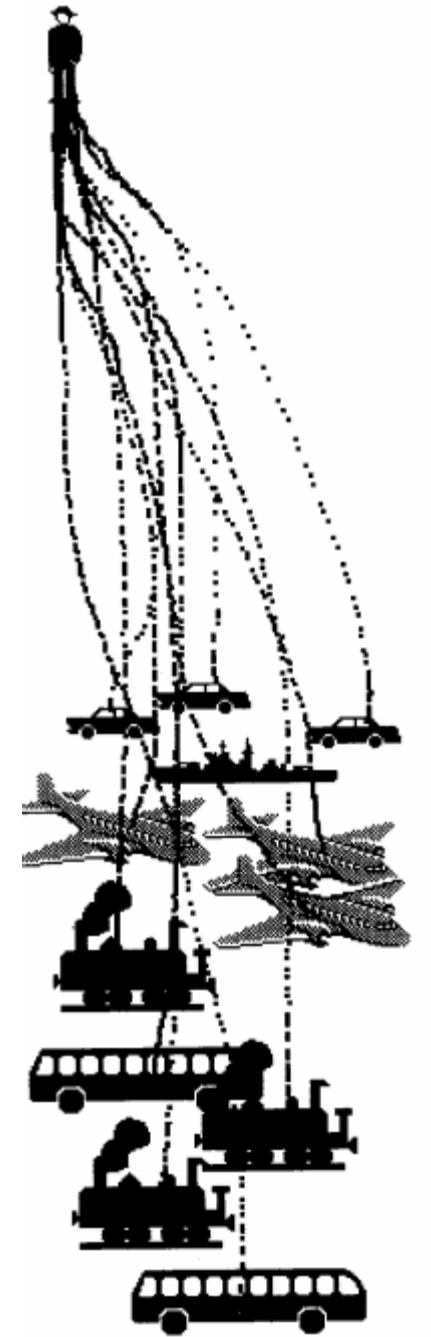
seaside 

The framework for developing sophisticated web applications in Smalltalk

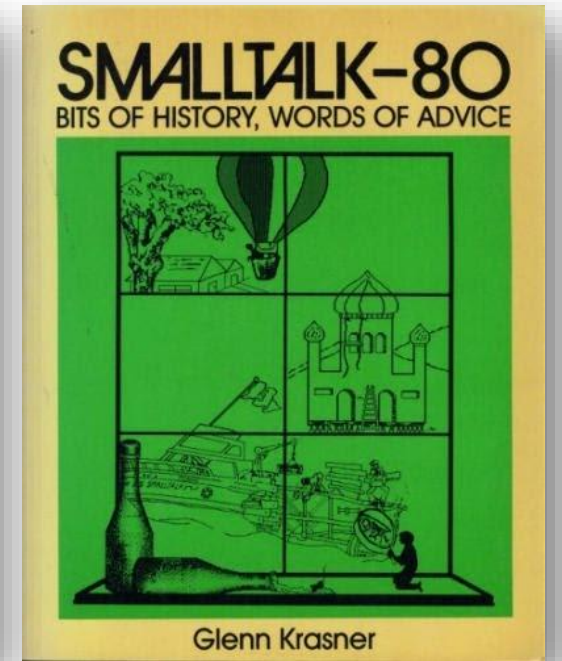
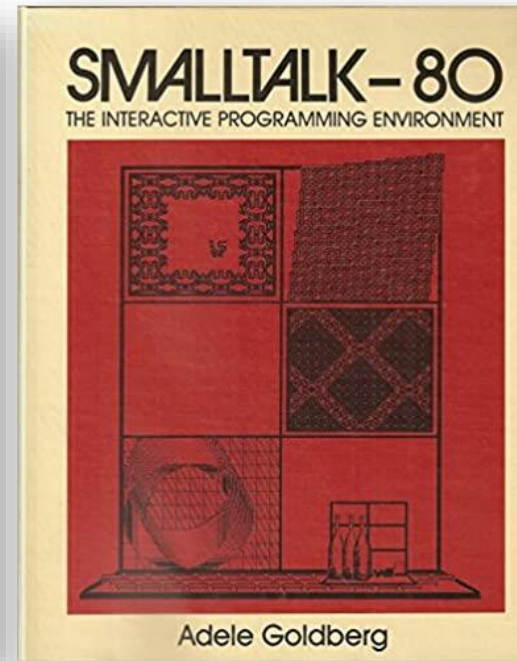
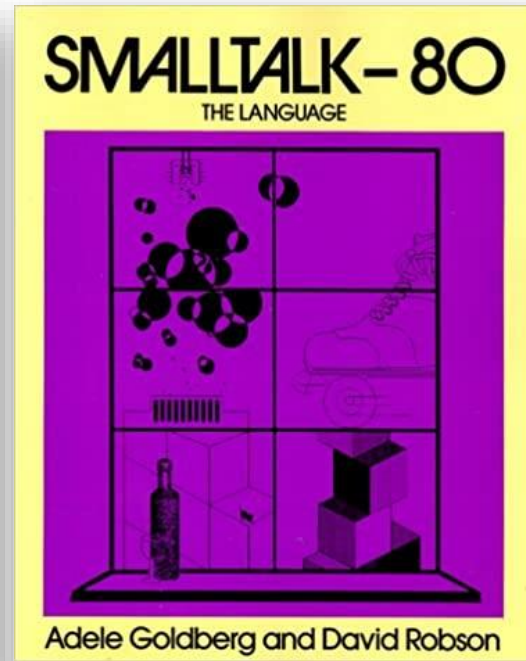
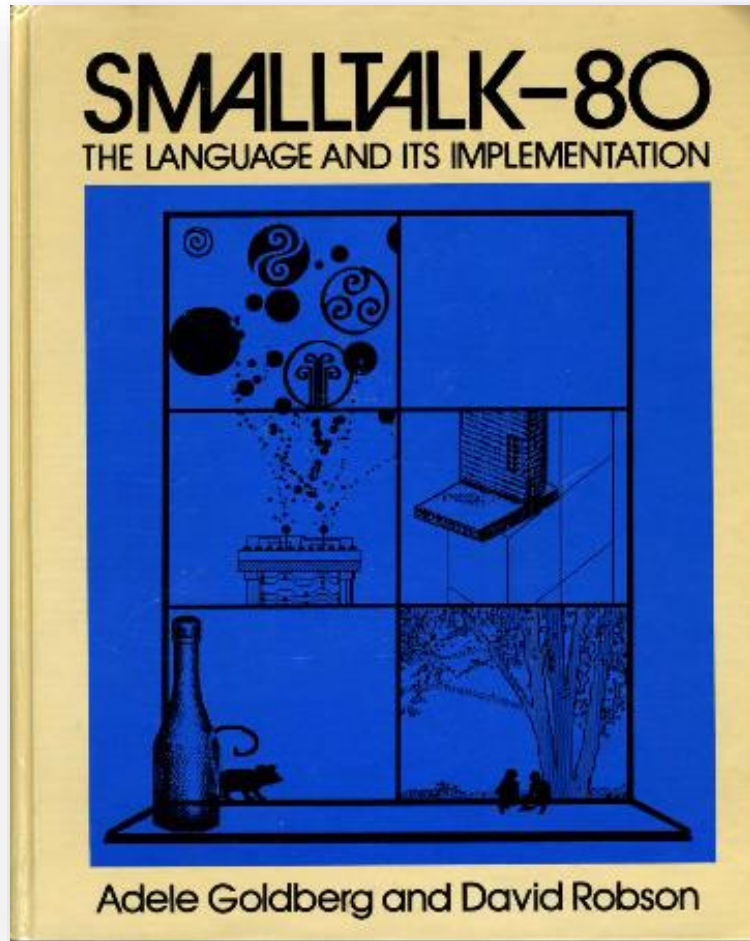


feenk

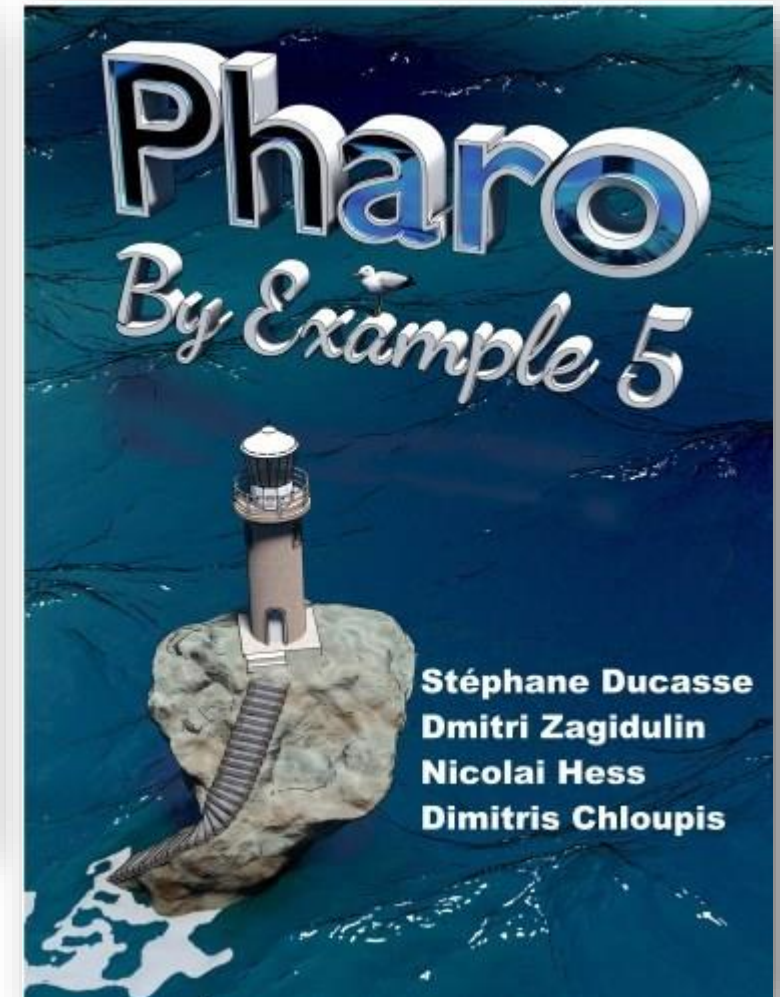
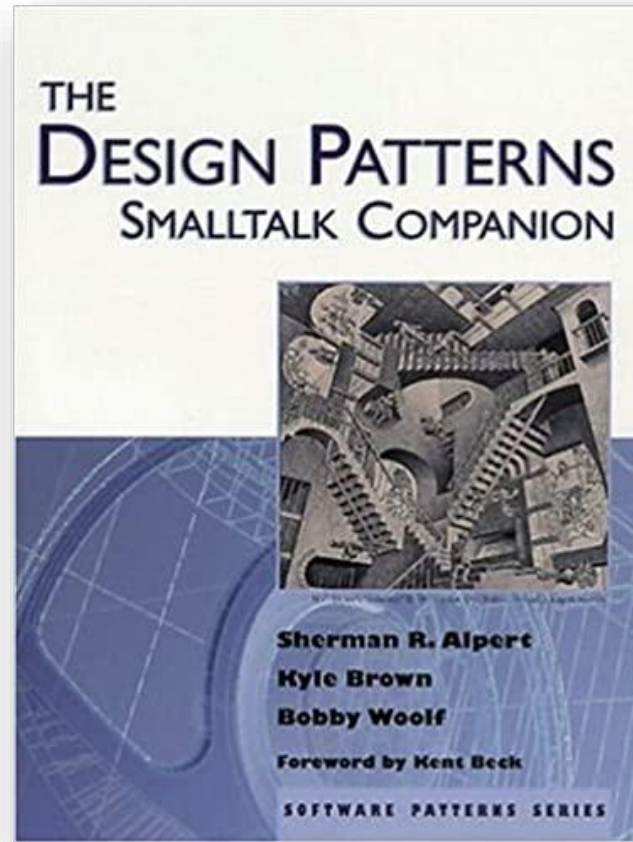
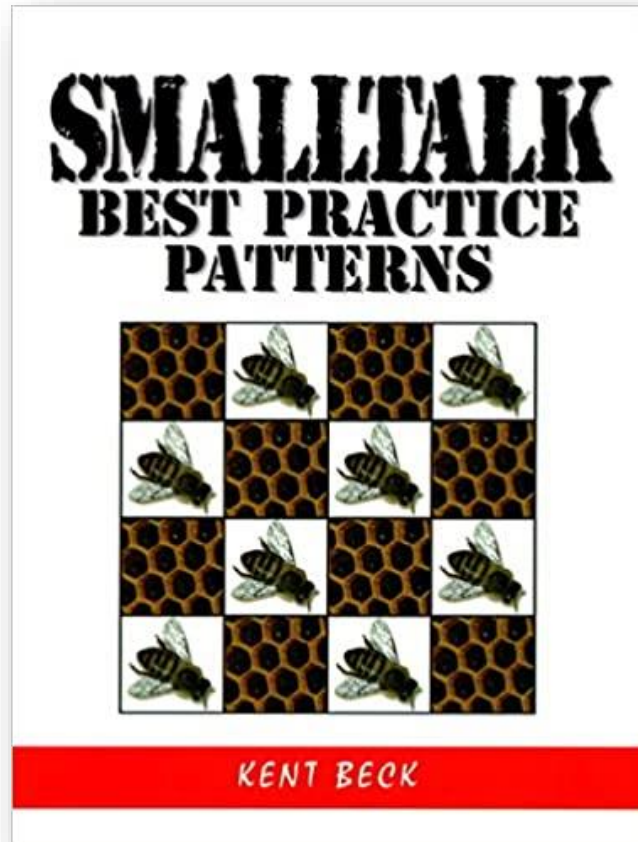
Resources on Smalltalk



Classic Smalltalk Books



Practical Smalltalk Books



Other resources

- Free online books
 - <http://stephane.ducasse.free.fr/FreeBooks.html>
- The Evolution of Smalltalk: by Dan Ingalls
 - <https://dl.acm.org/doi/pdf/10.1145/3386335>
- Why Smalltalk failed: Opinion by Gilad Bracha
 - <https://gbracha.blogspot.com/2020/05/bits-of-history-words-of-advice.html>

Other resources

- Early Smalltalk TV cut (**early 1980s**)
 - <https://www.youtube.com/watch?v=AuXCc7WSczM>
- Lecture on OOP by Dan Ingalls (**1989**)
 - <https://www.youtube.com/watch?v=Ao9W93OxQ7U>
- Smalltalk-76 demo on historic Xerox Alto by Dan Ingalls (**2017**)
 - https://www.youtube.com/watch?v=NqKyHEJe9_w

Thanks!

