Minimize the size of abstractions

Lots of Little Pieces

Methods should be small

Median size is 3 lines
10 lines is starting to smell

Classes should be small

7 variables is starting to smell
40 methods is starting to smell

VW 7.6

<table>
<thead>
<tr>
<th></th>
<th>Average</th>
<th>Median</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variables / class</td>
<td>2.1</td>
<td>1</td>
<td>72</td>
</tr>
<tr>
<td>Methods / class</td>
<td>16.6</td>
<td>8</td>
<td>359</td>
</tr>
<tr>
<td>LOC / method</td>
<td>3.0</td>
<td>2</td>
<td>156</td>
</tr>
</tbody>
</table>
Variables Per Class

classes := Smalltalk allClasses reject: [:each | each isMeta]
variablesInClass := classes collect: [:each | each instVarNames size].
average := ((variablesInClass fold: [:sum :each | sum + each] )/
    variablesInClass size) asFloat.
median := variablesInClass asSortedCollection at: variablesInClass size // 2.
max := variablesInClass fold: [:partialMax :each | partialMax max: each]

Note how the above code could use the application of these ideas
Methods Per Class

classes := Smalltalk allClasses reject: [:each | each isMeta]
methodsInClass := classes collect: [:each | each selectors size].
average := ((methodsInClass fold: [:sum :each | sum + each] )/
    methodsInClass size) asFloat.
mean := methodsInClass asSortedCollection at: methodsInClass size // 2.
max := methodsInClass fold: [:partialMax :each | partialMax max: each]

Note how the above code could use the application of these ideas
LOC / Method

methodSizes := OrderedCollection new.
classes
do: [:class |
class selectors
do: [:method | |
| periodCount |
periodCount := (class compiledMethodAt: method) decompiledSource
  occurrencesOf: $..
methodSizes add: periodCount + 1]].
average :=((methodSizes fold: [:sum :each | sum + each] )/
  methodSizes size) asFloat.
max := methodSizes fold: [:partialMax :each | partialMax max: each]
Common Manager Behavior

A project is behind schedule

So to get back on schedule they hire more people
The Result

The project will be even later
Parameters of any Project

Time
   How much time we have for the project

Scope (Size)
   Features of the project
   How much work is to be done

Quality
   The quality of work

Cost
   How many people work
   Tools used
Non-linear Relationships

Size

Development Time

0 1 2 3 4 5 6 7 8 9 10

0 250 500 750 1000
So

Doubling size of project more that doubles the amount of work

Doubling the team does not halve the time
Why adding people slows down projects

Existing people need to help bring new people up to speed
  So get less work done

More people on team makes it harder to communicate
  More meetings
  More documents
  Less work
Small is better
Small is better

But people act like they don't believe it
Survey

1/2 way done with project

Need make support multiple currencies and exchange rates

But have never done that before so don't know how

Option A
Start new project to explore how to do it

Option B
Using existing project to explore how to do it
Which is better

<table>
<thead>
<tr>
<th>Time Required</th>
<th>Option A</th>
<th>Option B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>200</td>
</tr>
<tr>
<td>250</td>
<td>250</td>
<td>500</td>
</tr>
<tr>
<td>500</td>
<td>500</td>
<td>750</td>
</tr>
<tr>
<td>750</td>
<td>750</td>
<td>1000</td>
</tr>
</tbody>
</table>

Size vs. Time Required

Thursday, November 21, 13
ValueWithHistory

Instance methods

value
value: anObject
value: anObject at: aTimestamp
valueAt: aTimestamp
valueFromNow: aDuration
Sample Test

testValueHistory

| test now |
now := Timestamp now.
test := ValueWithHistory on: 5 at: now - 5 days.
test value: 1 at: now - 1 days.
test value: 3 at: now - 3 days.
test value: 2 at: now - 2 days.
test value: -1 at: now + 1 days.
self assert: test value = 1.
self

assert: (test valueAt: now - 1 days) = 1;
assert: (test valueAt: now - 1 days - 1 seconds) = 2;
assert: (test valueAt: now - 3 days + 1 seconds) = 3;
assert: (test valueAt: now - 3 days - 1 seconds) = 5;
assert: (test valueAt: now + 3 days + 1 seconds) = -1;
assert: (test valueFromNow: -3 days) = 3
ValueWithHistory instance methods

initialize

    history := SortedCollection sortBlock: [:a :b | a key > b key].

value

    ^self valueAt: Timestamp now

value: anObject

    self value: anObject at: Timestamp now.
ValueWithHistory instance methods

value: anObject at: aTimestamp

    history add: (Association  key: aTimestamp value: anObject).

valueAt: aTimestamp

    ^history
    detect: [:each | each key <= aTimestamp]
    ifFound: [:each | each value]
    ifNone: [history last value]
ValueWithHistory instance methods

valueFromNow: aDuration

  ^self valueAt: Timestamp now + aDuration
Non-linear Relationships

Development Time

Size
Helper Method?

ValueWithHistory>>value

  ^self valueAt: Timestamp now

ValueWithHistory>>valueAt: aTimestamp

  ^history
  detect: [:each | each key <= aTimestamp]
  ifFound: [:each | each value]
  ifNone: [history last value]
Hinges

Bank Account and Withdrawal

Type of changes

New types of customers

Change fee structure

Change when to apply fee
BankAccount>>withdrawalNormal: aCurrency
  | newBalance |
newBalance isNegative ifTrue: [
  balance := balance - 5.0 asCurrency.
  etc.
]

BankAccount>>withdrawalPreferred: aCurrency
  | newBalance |
newBalance < -1000 asCurrency ifTrue: [
  balance := balance - 3.0 asCurrency.
  etc.
]
Adding New Types of Customers

Requires

New method in BankAccount

Callers need to be changed to call new method
BankAccount>>withdrawal: aCurrency

| newBalance |


balanceLimit := self isNormal
    ifTrue: [0 asCurrency]
    ifFalse: [-1000.0 asCurrency].

overDraftFee := self isNormal
    ifTrue: [0 asCurrency]
    ifFalse: [-5.0 asCurrency].

newBalance < balanceLimit ifTrue: [
    balance := balance - overDraftFee.
    etc.
New Customer types, Fee & Limit Changes

Require
  Editing the method

Other classes still call same method
Using instance variables for balanceLimit & overDraftFee

BankAccount>>withdrawal: aCurrency
    | newBalance |

    newBalance < balanceLimit ifTrue: [
        balance := balance - overDraftFee.
        etc.
    ]
New Customer types, Fee & Limit Changes

Just Data

Types & amounts could be read from file/database

Possible to
Create new customer types
Change fees
Change limits

without changing your code!
Second Example

SomeClass>>someMethod
  blah
  transaction = 'Withdrawal' ifTrue:[ account withdrawal: amount].
  blah.
  transaction = 'Deposit' ifTrue:[ account deposit: amount].
  etc

SomeClassOrDifferentClass>>someOtherMethod
  blah
  transaction = 'Withdrawal' ifTrue:[ amount := data at: 3].
  blah.
  transaction = 'Deposit' ifTrue:[ amount := data at: 4].
  etc
Adding new Transactions

Require
   Find all methods using transactions

Modifying each method
SomeClassOrDifferentClass>>someOtherMethod
  blah
  transaction = 'Withdrawal' ifTrue:[ amount := data at: 3].
  blah.
  transaction = 'Deposit' ifTrue:[ amount := data at: 4].
  etc

SomeClassOrDifferentClass>>someOtherMethod
  amount := transactionObject amount