

---

# Performance Optimization

---

# How to make fast software

**Make it work.**

**Make it right.**

**Make it fast.**

## Facts of Life

---

C is 10 times faster than Smalltalk on small benchmarks.

Smalltalk business applications are as fast as C business applications.

## Why is Smalltalk so fast?

---

Smalltalkers can spend more time on optimization.

Well-written Smalltalk programs are easy to optimize.

Most business programs are limited by network and database, not by CPU.

## How to make fast software

---

Make your software correct and easy to understand.

Measure the performance of each part of your program.

Find bottlenecks.

- (80% of the time is spent in 20% of the program)

Fix them.

## Wrong way to make fast software

---

Choose most efficient language possible.

Make each line of code as efficient as possible.

Choose fastest algorithms possible.

## How to measure performance

---

Profilers are in the parcel  
advanced/ATProfiler.pcl

TimeProfiler profile: [3 + 4]

TimeProfiler openView

## Profiling a Server

---

TimeProfiler

profile:[100 timesRepeat:

[| req |

req := WikiClientRequest new.

req

action: 'GET';

...

WikiServer current handleRequest: req]].

100.0 WikiServer>>handleRequest:  
100.0 Wiki>>replyToRequest:  
-----  
94.6 PageRendering>>renderBody  
78.3 PageRendering>>renderCommands  
61.7 WikiRendering>>linkTo:titled:  
34.2 WikiRendering>>putUrlForCommand:  
17.9 WikiRendering>>&=  
16.3 WikiRendering>>encodedPageTitle  
10.9 WikiRendering>>page  
10.9 Wiki>>pageTitled:  
5.4  
CharacterArray>>asUppercase

*Object-oriented Programming and Design - Copyright 1998 by Ralph E. Johnson*

9

## WikiRendering>>page

---

### **page**

^wiki pageTitled: request identifier last

This method is called several times, so  
cache it.

Add an instance variable called “page”.

*Object-oriented Programming and Design - Copyright 1998 by Ralph E. Johnson*

10

## WikiRendering>>page

---

### page

page isNil

if True: [page := wiki page Titled:  
request identifier

last].

^page

Reduced time from .31 seconds to .28

*Object-oriented Programming and Design - Copyright 1998 by Ralph E. Johnson*

11

## Process of improving performance

---

Measure performance.

Make change.

Run tests, measure performance.

**Undo changes if they don't improve  
performance.**

Repeat until performance is good enough.

*Object-oriented Programming and Design - Copyright 1998 by Ralph E. Johnson*

12

## Performance Strategy

---

Make performance targets  
(e.g.<2 seconds)  
Benchmark  
Try out improvements  
Only install ones that help  
Stop when you reach your targets

## Techniques

---

Better algorithms  
Optimize low-level design

- cache
- simplify
- reuse objects
- inline objects and methods
- Smalltalk specific

## Cache

---

Store result in variable and reuse it instead of recomputing it.

- Move expression out of loop.
- Store method result in instance variable.

You must recompute the result when the values it depends on change.

## Simplify

---

Instead of using powerful, easy to use objects, use simple and efficient ones.

- Arrays instead of `OrderedCollection`
- `Symbol` instead of `String`

## Reuse objects

---

Creating and collecting objects can take a lot of time.

Use AllocationProfiler

- 1) Keep a list of unused objects
- 2) Change object rather than make new one

Make sure object is really unused!

## Inline objects and methods

---

Lots of small methods can take time.

Replace messages in inner loops with the bodies of the methods they call.

To eliminate access to an object's instance variables, eliminate object and put its instance variables in the calling class.

**This is ugly -- a step of last resort.**

## Smalltalk specific

---

Make collections of right sizes

Use streams to concatenate

Avoid blocks

Avoid reflective programming

## Make Collections of Right Size

---

Many Collections grow: Set, Dictionary,  
OrderedCollection

Growing requires copying, which takes  
time.

Make Collections large enough that they  
don't have to grow very often.

OrderedCollection new: 20

## Use Streams to Concatenate

---

```
result := "".
names do: [:each | result := result , each].

result := WriteStream on: (String new:
    50).
names do: [:each | result nextPutAll:
    each].
result contents
```

## Avoid Blocks

---

Blocks are used for

- powerful control structures
- parameterizing objects

Alternatives

- in-line the algorithms
- make subclasses

## Inlining blocks

---

Compiler optimizes whileTrue:, to:do:, ifTrue: and does not create a block for them.

```
(1 to: 100) collect: [:each | each * each]
```

```
result := Array new: 100.
```

```
1 to: 100 do: [:each | result at: each put:  
each * each]
```

## Avoid Reflection

---

perform:, doesNotUnderstand: can make programs much smaller and eliminate coding work.

They are slower than a normal message send.

## Scheduling

---

60% - Functionality

25% - Refactoring

15% - Performance tuning

Good performance takes work, so  
schedule time for it.

## Summary

---

Performance depends more on the  
programmer than on the programming  
language.

Functionality and good design are harder  
to achieve than good performance.

Don't let your need for good performance  
prevent you from making a good  
design.