Exploring an existing code base: measurements and instrumentation

by Gerrit Muller       Buskerud University College

e-mail: gaudisite@gmail.com

www.gaudisite.nl

Abstract

Many architects struggle with a given large code-base, where a lot of knowledge about the code is in the head of people or worse where the knowledge has disappeared. One of the means to recover insight from a code base is by measuring and instrumenting the code-base. This presentation addresses measurements of the static aspects of the code, as well as instrumentation to obtain insight in the dynamic aspects of the code.

Distribution

This article or presentation is written as part of the Gaudí project. The Gaudí project philosophy is to improve by obtaining frequent feedback. Frequent feedback is pursued by an open creation process. This document is published as intermediate or nearly mature version to get feedback. Further distribution is allowed as long as the document remains complete and unchanged.

March 6, 2013
status: draft
version: 0.4
**Problem Statement**

*wanted:*
new functions and interfaces, higher performance levels, improvements, et cetera

*given:*
- complex system created by >100 people
- code repository > 1Mloc > 1k files
- document repository > 100 klines > 1k docs
- >100 people left

Exploring an existing code base: measurements and instrumentation

version: 0.4
March 6, 2013
EBMIproblem
### Overview of Approach and Presentation Agenda

<table>
<thead>
<tr>
<th>1 collect overviews</th>
<th>software system</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>2 study static structure</strong></td>
<td></td>
</tr>
<tr>
<td>2A macroscopic fact finding</td>
<td>size, effort relations</td>
</tr>
<tr>
<td>2B microscopic sampling</td>
<td>code reading</td>
</tr>
<tr>
<td>2C construct medium level diagrams</td>
<td></td>
</tr>
<tr>
<td><strong>3 study dynamic behavior</strong></td>
<td>time resources</td>
</tr>
<tr>
<td>3A measurements</td>
<td></td>
</tr>
<tr>
<td>3B construct simple models</td>
<td></td>
</tr>
</tbody>
</table>

4. iterate
**SW Overview(s)**

- **Application**
  - Property editor
  - Session manager
  - Spool server
  - Queue manager
  - Broker
  - Resource scheduler
  - Transparant Communication
  - Configurable pipeline
  - Event manager
  - Compliance profile
  - Abstraction Layer
  - Registry
  - Monitor
  - Hardwar independent format
  - Persistent Storage
  - Plug-in framework
  - Plug & play

**delivery centric**

**mechanism centric**

(over)simplistic

Exploring an existing code base: measurements and instrumentation

Gerrit Muller

version: 0.4
March 6, 2013
EBMInputs
subsystems

control hierarchy

kinematic

physics/optics

exploring an existing code base: measurements and instrumentation

version: 0.4
March 6, 2013

EBMIsystemDiagrams
Case 1: EasyVision (1992)

EasyVision: Medical Imaging Workstation
URF-systems
typical clinical image (intestines)

URF-systems  
EasyVision:  Medical Imaging Workstation

typical clinical image (intestines)
Examples of Macroscopic Fact Finding

> wc -l *.m
72 Acquisition.m
13 AcquisitionFacility.m
330 ActiveDataCollection.m
132 ActiveDataObject.m
304 Activity.m
281 ActivityList.m
551 AnnotateParser.m
1106 AnnotateTool.m
624 AnyOfList.m
466 AsyncBulkDataIO.m
264 AsyncDeviceIO.m
261 AsyncLocalDbIO.m
334 AsyncRemoteDbIO.m
205 AsyncSocketIO.m

version control information:
#new files
#deleted files
#changes per file since ...

package information:
# ... AnyOfList.m
466 AsyncBulkDataIO.m
264 AsyncDeviceIO.m
261 AsyncLocalDbIO.m
334 AsyncRemoteDbIO.m
205 AsyncSocketIO.m

metrics:
QAC type information
# methods
# globals
Histogram of File Sizes EV R1.0

largest file:
4473 lines
DatabaseTool.m

legend
- size OK, sample few
- slightly suspect, sample some
- suspect, have a look

Exploring an existing code base: measurements and instrumentation
Gerrit Muller

version: 0.4
March 6, 2013
EBMihistogram
Example of small classes due to database design;
These classes are only supporting constructs

13 IndexBtree.m
12 IndexInteriorNode.m
13 IndexLeafNode.m
13 ObjectStoreBtree.m
12 ObjectStoreInteriorNode.m
13 ObjectStoreLeafNode.m

Example of large classes due to inherent complexity;
some of these classes are really suspect

1541 GenericRegion.m
1415 GfxArea.m
1697 GfxFreeContour.m
4095 GfxObject.m
1714 GfxText.m
1374 CVObject.m
1080 ChartStack.m
1127 Collection.m
1651 Composite.m
1725 CompositeProjectionImage.m
1373 Connection1.m
1181 Database1.m
3707 DatabaseClient.m
3240 Image.m
1861 ImageSet.m

Example of large classes due to large amount of UI details

4473 DatabaseTool.m
1291 EnhancementTool.m
1106 AnnotateTool.m
1291 EnhancementTool.m
3471 GreyLevelTool.m
1639 HCConfigurationTool.m
1007 HCQueueViewingTool.m
1590 HardcopyTool.m
Changes Over Time

- redesign by mature designer
- partial redesigns
- failed in retrospect

<table>
<thead>
<tr>
<th>time</th>
<th>#changed</th>
<th>lines</th>
</tr>
</thead>
</table>

*hot spots*

- ever changing files e.g.: systemConstants.h
- ShakylImplementation.m

Exploring an existing code base: measurements and instrumentation
version: 0.4
March 6, 2013
EBMChangesOverTime
The real layering diagram did have >15 layers
Quantification helps to \textit{calibrate} the \textit{intuition} of the architect

\textbf{Macroscopic} numbers related to \textit{code level} understanding provides insight

+ relative complexity
+ relative effort
+ hot spots
+ (static) dependencies and relations
Dynamics ≫ Static

Exploring an existing code base: measurements and instrumentation

March 6, 2013

Gerrit Muller
Layered Benchmarking

typical values
interference
variation
boundaries

CPU
cache
memory
bus
.. (computing) hardware
typical values
interference
variation
boundaries

duration
footprint
interrupts
task switches
OS services
CPU time
footprint
(cache)

network transfer
database access
database query
services/functions

end-to-end
function

duration
services
interrupts
task switches
OS services
CPU time
footprint
(cache)

applications

services

operating system

latency
bandwidth
efficiency

interrupt
task switch
OS services

locality
density
efficiency
overhead

tools

Exploring an existing code base: measurements and instrumentation

version: 0.4
March 6, 2013
EBMibenchmarkStack

Gerrit Muller
Example: Processing HW and Service Performance

- spatial enhancement
- interpolate
- Look up table (invert contrast / brightness)
- graphics merge
- colour LUT
- monitor

Legend:
- SW
- HW

Image from database → spatial enhancement → interpolate → Look up table → graphics merge → colour LUT → monitor

Bi-linear, bi-cubic

Output:
- brightness
- contrast

Input:
- brightness
- contrast

From database.

Exploring an existing code base: measurements and instrumentation

version: 0.4
March 6, 2013
MICVpresentationPipeline
Exploring an existing code base: measurements and instrumentation

Gerrit Muller

version: 0.4
March 6, 2013
MICVprocessingTimes
Resource Measurement Tools

\[ t_{n-2} \quad \text{preamble to remove start-up effects} \quad t_{n-1} \quad \text{use case} \quad t_n \quad \text{time} \]

<table>
<thead>
<tr>
<th>oit</th>
<th>( \Delta ) object instantations heap memory usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ps</td>
<td>kernel CPU time</td>
</tr>
<tr>
<td>vmstat</td>
<td>user CPU time</td>
</tr>
<tr>
<td>kernel resource stats</td>
<td>code memory</td>
</tr>
<tr>
<td></td>
<td>virtual memory</td>
</tr>
<tr>
<td></td>
<td>paging</td>
</tr>
</tbody>
</table>

heapviewer (visualise fragmentation)
Object Instantiation Tracing

<table>
<thead>
<tr>
<th>class name</th>
<th>current nr of objects</th>
<th>deleted since (t_{n-1})</th>
<th>created since (t_{n-1})</th>
<th>heap memory usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsynchronousIO</td>
<td>0</td>
<td>-3</td>
<td>+3</td>
<td></td>
</tr>
<tr>
<td>AttributeEntry</td>
<td>237</td>
<td>-1</td>
<td>+5</td>
<td></td>
</tr>
<tr>
<td>BitMap</td>
<td>21</td>
<td>-4</td>
<td>+8</td>
<td></td>
</tr>
<tr>
<td>BoundedFloatingPoint</td>
<td>1034</td>
<td>-3</td>
<td>+22</td>
<td></td>
</tr>
<tr>
<td>BoundedInteger</td>
<td>684</td>
<td>-1</td>
<td>+9</td>
<td></td>
</tr>
<tr>
<td>BtreeNode1</td>
<td>200</td>
<td>-3</td>
<td>+3</td>
<td></td>
</tr>
<tr>
<td>BulkData</td>
<td>25</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ButtonGadget</td>
<td>34</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>ButtonStack</td>
<td>12</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ByteArray</td>
<td>156</td>
<td>-4</td>
<td>+12</td>
<td></td>
</tr>
</tbody>
</table>

[819200] [8388608] [13252]
Exploring an existing code base: measurements and instrumentation

19 Gerrit Muller

version: 0.4
March 6, 2013
EBMImemoryInstrumentation

Memory Instrumentation

200 MB

measured

bulk data

data

code

OS

unaccounted

big lump

unaccounted

leftover

accountable

by OS services

and OIT

budget

manually

instrumented

fragmentation

# Overview of Benchmarks and Other Measurement Tools

<table>
<thead>
<tr>
<th>test / benchmark</th>
<th>what, why</th>
<th>accuracy</th>
<th>when</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>public</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SpecInt (by suppliers)</td>
<td>CPU integer</td>
<td>coarse</td>
<td>new hardware</td>
</tr>
<tr>
<td>Byte benchmark</td>
<td>computer platform performance OS, shell, file I/O</td>
<td>coarse</td>
<td>new hardware</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>new OS release</td>
</tr>
<tr>
<td>file I/O</td>
<td>file I/O throughput</td>
<td>medium</td>
<td>new hardware</td>
</tr>
<tr>
<td>image processing</td>
<td>CPU, cache, memory</td>
<td>accurate</td>
<td>new hardware</td>
</tr>
<tr>
<td>as function of image, pixel size</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>self made</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Objective-C overhead</td>
<td>method call overhead</td>
<td>accurate</td>
<td>initial</td>
</tr>
<tr>
<td></td>
<td>memory overhead</td>
<td></td>
<td></td>
</tr>
<tr>
<td>socket, network</td>
<td>throughput</td>
<td>accurate</td>
<td>ad hoc</td>
</tr>
<tr>
<td></td>
<td>CPU overhead</td>
<td></td>
<td></td>
</tr>
<tr>
<td>data base</td>
<td>transaction overhead</td>
<td>accurate</td>
<td>ad hoc</td>
</tr>
<tr>
<td></td>
<td>query behaviour</td>
<td></td>
<td></td>
</tr>
<tr>
<td>load test</td>
<td>throughput, CPU, memory</td>
<td>accurate</td>
<td>regression</td>
</tr>
</tbody>
</table>
**Tools and Instruments Positioned in the Stack**

**typical small testprogram**

create steady state
\[ t_s = \text{timestamp()} \]
for(i=0;i<1M;i++) do something
\[ t_e = \text{timestamp()} \]
duration = \[ t_s - t_e \]

**Exploring an existing code base: measurements and instrumentation**

version: 0.4
March 6, 2013
EBMibenchmarkPositions
Case 2: ARM9 Cache Performance

- CPU
- Instruction cache
- Data cache
- Memory
- Cache line size: 8 32-bit words

200 MHz

100 MHz

PCB
Example Hardware Performance

memory request

22 cycles

memory response

word 1
word 2
word 3
word 4
word 5
word 6
word 7
word 8

data

38 cycles

memory access time in case of a cache miss

200 Mhz, 5 ns cycle: 190 ns
### Actual ARM Figures

**ARM9 200 MHz**  \( t_{\text{context switch}} \)

as function of cache use

<table>
<thead>
<tr>
<th>cache setting</th>
<th>( t_{\text{context switch}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>From cache</td>
<td>2 µs</td>
</tr>
<tr>
<td>After cache flush</td>
<td>10 µs</td>
</tr>
<tr>
<td>Cache disabled</td>
<td>50 µs</td>
</tr>
</tbody>
</table>
## Context Switch Overhead

The overhead time for context switching can be calculated using the formula:

\[ t_{\text{overhead}} = n_{\text{context switch}} \times t_{\text{context switch}} \]

### Table

<table>
<thead>
<tr>
<th>( n_{\text{context switch}} ) (s(^{-1}))</th>
<th>( t_{\text{context switch}} )</th>
<th>CPU load overhead</th>
<th>( t_{\text{context switch}} )</th>
<th>CPU load overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>5ms</td>
<td>0.5%</td>
<td>1ms</td>
<td>0.1%</td>
</tr>
<tr>
<td>5000</td>
<td>50ms</td>
<td>5%</td>
<td>10ms</td>
<td>1%</td>
</tr>
<tr>
<td>50000</td>
<td>500ms</td>
<td>50%</td>
<td>100ms</td>
<td>10%</td>
</tr>
</tbody>
</table>
Performance as Function of all Layers

system performance = f(
, applications
, services
, operating system
, hardware
, tools
)

what is used?
how often?
how much
does it cost?
system performance = f(
applications
 hit-rate, miss-rate,
 #transactions
 interrupt-rate, task switch rate
 CPU-load
,
services
 transaction overhead: 25 ms
,
operating system
 interrupt latency: 10 us
 task-switch: 10 us
 (with cache flush)
,
hardware
 cache miss: 190ns
,
tools
)
Keep iterating!

- New measurements and experiments
- Zoom in on suspect parts
- Code reading
- Problematic dynamic properties
- Static structure

Create (recovered) insight in complex system
0. many design teams have lost the overview of the system

1. a good (sw) architect has a quantified understanding of system context, system and software

2. a good design facilitates measurements of critical aspects for a small realization effort