Fast, Cheap and Under Control: Evaluating Revision Data Reliably

Abram Hindle, Michael W. Godfrey, Richard C. Holt

Software Architecture Group
David R. Cheriton School of Computer Science
University of Waterloo
Canada
http://swag.uwaterloo.ca/

{ahindle,migod,holt}@cs.uwaterloo.ca
Introduction

- Our focus
  - Arguing for Cheap and Fast techniques
    * that can be verified (Under control)
  - Demonstrate 2 of projects
    * Release Mining
    * Indentation Metrics

- Conclusions
Fast, Cheap and Under Control

- Fast - speed, execution time
- Cheap - how much effort to run, to implement
- Under Control - it has been verified or demonstrated that it works.
  - Someone else went through the leg work to provide evidence that it works.
  - Show that it reliably works for many cases
Fast and Cheap methods we have used

- Looked at change history as events and not as diffs
  - Characterize behaviour around events.

- Measured diffs via their indentation to infer complexity
  - If you rank changes by variance or sum of indentation, you’re effectively ranking by complexity!
Figure 1: Partitioned revisions and releases over time in order to focus on behaviour around release time
Get the Diff

```c
void square( int * arr, int n ) {
    int i = 0;
    for ( i = 0 ; i < n ; i++ ) {
        arr[ i ] *= arr[ i ];
    }
}
```

Measure the Indentation

```
Print the Indentation
Raw Indentation
Logical Indentation
```

Produce Summary Statistics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Raw</th>
<th>Logical</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC</td>
<td>6.000</td>
<td>6.000</td>
</tr>
<tr>
<td>AVG</td>
<td>3.330</td>
<td>0.833</td>
</tr>
<tr>
<td>MED</td>
<td>4.000</td>
<td>1.000</td>
</tr>
<tr>
<td>STD</td>
<td>2.750</td>
<td>0.687</td>
</tr>
<tr>
<td>VAR</td>
<td>9.070</td>
<td>0.567</td>
</tr>
<tr>
<td>SUM</td>
<td>20.000</td>
<td>5.000</td>
</tr>
<tr>
<td>MCC</td>
<td>2.000</td>
<td>2.000</td>
</tr>
<tr>
<td>HVOL</td>
<td>152.000</td>
<td>152.000</td>
</tr>
<tr>
<td>HDIFF</td>
<td>15.000</td>
<td>15.000</td>
</tr>
<tr>
<td>HEFFORT</td>
<td>2127.000</td>
<td>2127.000</td>
</tr>
</tbody>
</table>
Challenges

- **Scalability**
  - Software ranges in size

- **Tool Support**
  - How many of you can parse PHP or Perl right now?

- **Flexibility**
  - How much do you have to know or do before hand?

- **Software is complicated and unique**
  - Some analyses will not work
Cheap

• Cost

• Implementation time

• How much prep work do we need to do

• How much data do we need to provide
  – For instance, do we even need the diff or the source code?
  – Do we need language specific tools?
Cheap and Adoptable!

- Even the developers of a project can apply it
- Doesn’t take a lot of work to integrate a technique
- Low cost of adoption and adaption.
- Cheap yet verified!
- End-user doesn’t need to prove the technique works
  - We do
Fast

- Often easy and cheap is fast

- By ignoring certain kinds of information (diffs) we can get results quickly
  - Or at least provide results at an acceptable granularity

- Efficiency: Good run-times over large data sets
Under Control

- The fast part is for the end user
  - We can do the slow parts
    * Verification, corpus, examples
Under Control

- Can we show it works?
- Can we provide a reasonable argument and actual evidence to show our fast, cheap technique works?
  - Sometimes control is slow, but once done we don’t need to do it again
- How flexible is it?
- Can we do more?
Is it Reliable?

- Just because something is easy and fast doesn’t mean it right or doesn’t mean it is accurate

- We have to validate if something is correct and works for many cases
  - With large samples of software we can test reliability, we can have reasonable statistical significance.
  - Test our techniques with large systems or large number of systems
Cheap and Fast Research?

• Cheap and Fast doesn’t necessarily make for fast research
  – You have to verify it works
  – Since we deal with software you should try it on real software or good representatives
  – If we can show it works, we stay in control
Cheap Example: Release Patterns

- Characterize a project’s behaviour around release time before and after release
- Break a stream of revisions down by file type around release time and analyze that stream
- Cheap: Only needs file names and revision times
- Verification: Some by hand investigation
  - Ask the developers if this sounds right
Verification Example

- Indentation study
  - Is indentation regular?
    ∗ Parse through 300 popular projects on source
      forge and find out!
  - For revisions, is indentation correlated with
    complexity?
    ∗ Run complexity metrics on all source code
      revisions of 300 projects
Get the Diff

```c
void square( int * arr, int n ) {
    int i = 0;
    for ( i = 0 ; i < n ; i++ ) {
        arr[ i ] *= arr[ i ];
    }
}
```

Measure the Indentation

<table>
<thead>
<tr>
<th>Metric</th>
<th>Raw</th>
<th>Logical</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC</td>
<td>6.000</td>
<td>6.000</td>
</tr>
<tr>
<td>AVG</td>
<td>3.330</td>
<td>0.833</td>
</tr>
<tr>
<td>MED</td>
<td>4.000</td>
<td>1.000</td>
</tr>
<tr>
<td>STD</td>
<td>2.750</td>
<td>0.687</td>
</tr>
<tr>
<td>VAR</td>
<td>9.070</td>
<td>0.567</td>
</tr>
<tr>
<td>SUM</td>
<td>20.000</td>
<td>5.000</td>
</tr>
<tr>
<td>MCC</td>
<td>2.000</td>
<td>2.000</td>
</tr>
<tr>
<td>HVOL</td>
<td>152.000</td>
<td>152.000</td>
</tr>
<tr>
<td>HDIFF</td>
<td>15.000</td>
<td>15.000</td>
</tr>
<tr>
<td>HEFFORT</td>
<td>2127.000</td>
<td>2127.000</td>
</tr>
</tbody>
</table>
Verification Example

- Large Data-sets push statistical packages to their limits.
  - Don’t ask R to give you a correlation coefficient of 1 million or more elements.

- To verify, we spent 8 CPU years.
  - Ran thousands of jobs of on the Ontario Sharcnet cluster (Whale has 3000 CPUs)
  - Luckily correlation is associative (parallelizable)
Conclusions

- Fast and Cheap is easier for everyone (except us)
- We stand a better chance to see Fast and Cheap techniques adopted
  - SourceForge now has CVS/SVN viewers which graph repository activity
- Verification of claims can be done on large data-sets
  - Parallelizable tests allow years of CPU time to be folded into days.
Final

• Any Questions?

This work was partially funded by an NSERC PGS-D scholarship.

We thank Ontario Sharcnet for the computer equipment provided.
Figure 2: Frequency of Physical Indentation of various languages (Log Scale)
Figure 3: Frequency of Physical Logical Indentation of various languages (Log Scale)
Figure 4: Frequency of Gap Indentation (non-logical units of indentation) of various languages (Log Scale)
Figure 5: Similarity of Indentation between Languages
Figure 6: Revisions and releases over time. Extract the revisions.
Figure 7: Partitioned revisions and releases over time
Figure 8: Partitioned revisions and releases over time, separated
Figure 9: Partitioned revisions aggregated per day
Figure 10: Partitioned revisions aggregated per day and smoothed
Figure 11: Select the revisions around release times
Figure 12: Aligned revisions aggregated
Figure 13: Align and aggregate revisions of each class
Figure 14: Analysis: averages and linear regressions
Distribution of logical indentation for C files

Number of cases

Indentation in spaces
Distribution of indentation for C files

Number of cases

Indentation in spaces

Indentation Lines

Abram Hindle
Distribution of logical indentation for C/C++ Header Files

Number of cases vs. Indentation in spaces

- Number of cases: 0, 500000, 1e+06, 1.5e+06, 2e+06, 2.5e+06, 3e+06
- Indentation in spaces: 0, 5, 10, 15, 20, 25, 30

Abram Hindle
Distribution of indentation for C/C++ Header Files

Number of cases vs. Indentation in spaces
Distribution of logical indentation for PHP Files

Number of cases

Indentation in spaces

Indentation Lines

Abram Hindle
Distribution of logical indentation for Perl Files

Number of cases

Indentation in spaces

Indentation Lines

Abram Hindle
Distribution of logical indentation for Python Files

Number of cases

Indentation in spaces

0 2 4 6 8 10 12 14 16 18 20

0 50000 100000 150000 200000 250000 300000 350000 400000 450000 500000

Abram Hindle
Distribution of indentation for Python Files

Number of cases

Indentation in spaces

Abram Hindle