Analysis of Test Coverage Data on a Large-Scale Industrial System

by **Erik Sven Vasconcelos Jansson**
<erija578@student.liu.se>

Examinator: Lena Buffoni - **Advisor:** Bernhard Thiele
Department of Computer and Information Science
at Linköping University (LiTH), Sweden

September 15, 2016

Background
Required Theory

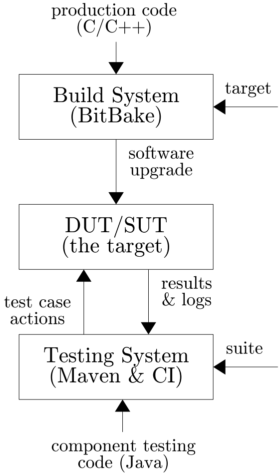
- Importance of *software testing*, within any *mission critical software* products.
- See for e.g. NASA's *Climate Orbiter*...
- The *code/test coverage metric* can be used in determining e.g. *testing holes*.

```c
int factorial(int n) {
    if (n == 0) return 1;
    else if (n < 0) return -1;
    return factorial(n - 1)*n;
}

int main(int, char**) {
    assert(factorial(0) ==  1);
    assert(factorial(4) == 24);
    return 0;
}
```

[1] Climate Orbiter: http://mars.nasa.gov/msp98/

- Project at *Ericsson (R&D) Linköping*, the given *large-scale industrial system*.
- *Production & unit tests*: 960 KSLOC, the *target function tests*: 564 KSLOC.
- *Tasks:* integrate a *coverage gathering*, and *analysis system* for *function tests*; determine *"feasibility"* of the addition.
- *Goal:* develop framework, upon which *test case selection/prioritization* could be derived in future work. PL3 $\approx 18h$.



Motivation

- *Software testing* is *complicated* and *expensive*, research [B+81] has shown 50% project effort.
- *Coverage data* provides *insight* into the *nature* of *tests*; giving useful information about them.
- Research in *large-scale test coverage is scarce*, according to several papers such as [ABR+11]
- *Gathering & analysis* is problematic [ABR+11]
- Articles don't present *development experience*.

Problems

- *Coverage gathering is problematic on large-scale systems:*
  - Existing coverage tools *don't integrate well* on all setups.
  - *Performance* and *resource usage* might be *neg. affected*.
  - *Research question: how feasible is such an extension?*
- *Analyzing raw test coverage data manually isn't feasible:*
  - *Huge amounts* of data produced for *large-scale software*.
  - *Difficult* to extract any *meaningful test case properties*.
  - *Research question: what does the analysis tools find?*

Challenges

- Performance sensitive system.
- Executes a series of *daemons*.
- Coverage for a *remote target*.
- *Huge amount* of *profile data*.
- *Isolate* changes in *behaviour*.

```c
static int bb[4] = {};
static int factorial(int n) {
    ++bb[0];
    if (n == 0) {
        ++bb[1];
        return 1;
    } else if (n < 0) {
        ++bb[2];
        return -1;
    }

    ++bb[3];
    return factorial(n - 1)*n;
}

int main(int, char**) {
    assert(factorial(0) == 1);
    assert(factorial(4) ==24);
    printf("bb 0: %i", bb[0]);
    printf("bb 1: %i", bb[1]);
    printf("bb 2: %i", bb[2]);
    printf("bb 3: %i", bb[3]);
    return 0;
}
```

[2] emacs?: videohive.net/item/burning-notebook/

Proposal

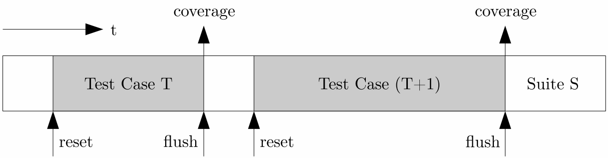
- *Instrument production code* with *coverage capabilities*.
- *Measure* the *performance effects* of these, *statistically*.
- Add *flushing signal*, to deal with the *daemon software*.
- *Extend testing system* to *fetch coverage* on *test's end*.
- *Build/find tool* to *analyze, and locate test similarities*.

Solution
Instrumentation

- Production code is built with GCC $\rightarrow$ GCov is built-in, & proven to have minimal performance impact, $\approx$ 3%.
- e.g.: `-fprofile-arcs -ftest-coverage -O0`.
- However, remote target $\rightarrow$ coverage is dumped wrong, solved with `GCOV_PREFIX` & `GCOV_PREFIX_STRIP`.
- Enabled in the *build system* with: `--coverage` flag. ($\leftarrow$)

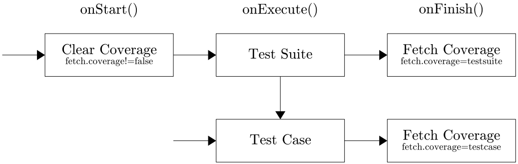
Solution
Daemon Flushing

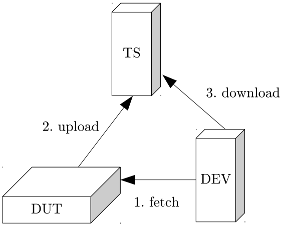
- Several instrumented daemons on the target device.
- Coverage information is dumped when **terminated**, however, device *reboots* if this is done... Not good.
- By calling `__gcov_flush()`, processes can dump, without being terminated explicitly (no rebooting).
- Synchronize flush: `kill -sUSR1 $(pidof *)`, handler enabled on all daemons by: `--coverage`. (←)



Solution
Automatic Fetching

- Extends the Maven/Jcat testing flow.
- Automatically *flushes coverage* down, splitting the *individual test case* data.
- Switched with `fetch.coverage = (testcase | testsuite)`. (←)





Solution
Analysis Methods

- How to measure *test case similarities*? see *Jaccard index*, *Hamming distance*.
- Initial solution modified `gcov-tool`.
- Current implementation `scovat.py`, gives *set operations* on *coverage data*, required by *Hamming* & *Jaccard*. ($\leftarrow$)

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|},\ d_J = 1 - J(A,B)$$

A	B	
0	1	+1
1	1	
1	1	
1	0	+1
0	0	
1	1	
0	1	+1

Solution
Usage Workflow

1. Build software with `bb --coverage <recipe>`, enabling coverage instrumentations and daemon signal handlers setup.
2. Upload software packages with `stp_set_up <deviceid>`.
3. Enable the `fetch.coverage=testcase` testing property, enabling the Maven/JCat testing framework fetch coverages.
4. Execute the desired test case/suite, e.g. `PL2` on the devices.
5. Test coverage data is continuously fetched to the developer.
6. `scovat.py -gb $BUILDLOC -o cache` generates the intermediate coverage format, used later for modifying data.
7. `scovat.py -ao analysis cache/<A> cache/<B>` gives *Jaccard coefficient*, and *Hamming distance* for (A, B).

- Software *instrumentation* haven't made any tests unstable.
- Fetching coverage from target $\rightarrow$ developer adds 20s time, largely caused by bugs in the *Trilead SSH* implementation.
- Consumes a *fixed disk space* of 8.3 MiB for each test case, which is transferred & removed continuously when fetched.
- The *suite execution time*, *processor*, and *memory usage* are measured statistically after a *t-distribution*, $\alpha = 5\%, n = 4$.

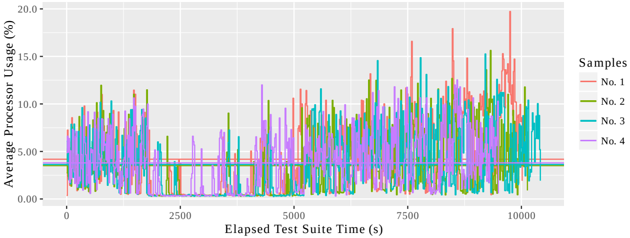
Sampled Dataset	Lower Time (h)	Upper Time (h)
Instrumented	2.656	2.911
Non-Instrumented	2.075	2.260

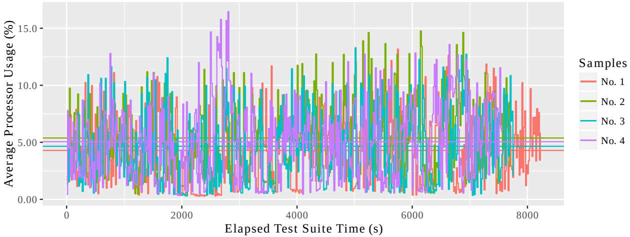
Results
Memory Usage

Results
Memory Usage

Sampled Dataset	Lower Usage (%)	Upper Usage (%)
Instrumented	16.7663	18.8446
Non-Instrumented	17.8428	18.0185

Results
Processor Usage





Sampled Dataset	Lower Usage (%)	Upper Usage (%)
Instrumented	3.43483	4.12369
Non-Instrumented	4.28527	5.40044

- Retrieved from the primary development test suite.
- Upon producing *lcov report* with fetched coverage:
    - *Statement coverage: 59.3% out of 96 158* (lines).
    - *Function coverage: 70.7%, of 23 870* (functions).
    - *Branch coverage: 24.6%*, (retrieved from: *Gcov*).

- Demonstration of *test similarity analysis*, with 3 tests:
  - **A:** IPForwarding#testCliRejectsInvalidAddressOnDstMo,
  - **B:** IPForwarding#testCliRejectsInvalidAddressOnNexthopMo,
  - **C:** PL1#testPL1TestSuite, *all three from Ericsson's tests*.
- Both **A** and **B** should exercise very similar code locations.
- While **C** exercises more varied locations, different from **A**.
- *Similarity* leads to *potential for test redundancy* [CNM07].
- *Note!: exercising similar locations* $\nRightarrow$ *exactly same tests!*

Results

Interpretation

Criterion	$d_H(A, B)$	$A \cap B$	$A \cup B$	$J(A, B)$
Statement	0	400	400	1.00000
Function	0	90	90	1.00000
Branch	0	132	132	1.00000

Criterion	$d_H(A, C)$	$A \cap C$	$A \cup C$	$J(A, C)$
Statement	21 691	398	22 089	0.01801
Function	5 369	90	5 459	0.01648
Branch	21 960	123	22 083	0.00557

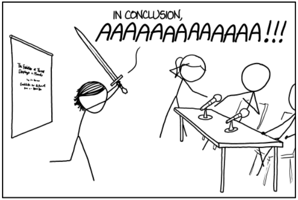
Conclusions

- *Feasibility: deemed possible, since tests aren't unstable; however, it increases softw. execution time significantly.*
- *Measurements: project has similar coverage to Google's average project (C) coverage (statement) measurement.*
- *Interpretation: analysis tool/method, show locations of: test case similarity, and pot. test redundancy, [CNM07].*
- *Limitation: still requires engineers to verify redundancy.*
- *Future Work: clustering, test selection & prioritization.*

Summary

In a nutshell…

- *Software testing* is *hard*; *test coverage* give us *valuable metadata* about *tests*.
- *Gathering & analyzing code coverages* on *large-scale systems* proves difficult.
- Integration of the system was *feasible*, but *prog. execution time* was *affected*.
- *Metrics* of a *real, large-scale test suite* were given since such *data* was *scarce*.
- Finally, *analysis tool* shows *similarities* between *test cases*, which sifts results.



THE BEST THESIS DEFENSE IS A GOOD THESIS OFFENSE.

[3] Thesis Defense: https://www.xkcd.com/1403/

Questions?

Bibliography

- Yoram Adler, Noam Behar, Orna Raz, Onn Shehory, Nadav Steindler, Shmuel Ur, and Aviad Zlotnick.
  Code coverage analysis in practice for large systems.
  In *Proceedings of the 33rd International Conference on Software Engineering*, pages 736–745. ACM, 2011.

- Barry W Boehm et al.
  Software engineering economics, volume 197.
  Prentice-hall Englewood Cliffs (NJ), 1981.

- Emanuela G Cartaxo, Francisco G Oliveira Neto, and Patrícia DL Machado.
  Automated test case selection based on a similarity function.
  GI Jahrestagung (2), 7:399–404, 2007.