

# Code Coverage and the Definition of Done

Dénes Mátételki

[www.emerson.com](http://www.emerson.com)

October 2, 2012



# Table of contents

- 1 Definitions
  - Legacy code
  - Tests
  - Definition of Done, Continuous Integration
- 2 Code coverage
  - gcov and gprof
  - How gcov works
  - Example
  - lcov
- 3 Final thoughts
  - What have we got actually
  - CC and Dod

# Legacy code, unit test, functional test

## Legacy code

"Code without tests is bad code. It doesn't matter how well written it is; it doesn't matter how pretty or object-oriented or well-encapsulated it is. With tests, we can change the behaviour of our code quickly and verifiably. Without them, we really don't know if our code is getting better or worse." [1]

# Legacy code, unit test, functional test

## Unit Test (UT) - white box testing

"The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. A unit test provides a strict, written **contract** [2] that the piece of code must satisfy." [3]

## Functional Test (FT) - black box testing

"Bases its test cases on the specifications of the software component under test. Functions are tested by feeding them input and examining the output, and internal program structure is rarely considered." [4]

# Definition of Done, Continuous Integration

## Definition of Done - DoD

- "A checklist of valuable activities required to produce software." [5]
- "Different DoD at various levels: feature, sprint, release."
- Each team come up with their own DoD which is reviewed and updated as needed.

## Continuous Integration - CI

"Implements continuous processes of applying quality control - small pieces of effort, applied frequently." [6]

- Automate the build.
- Make the build self-testing.
- ...verifies the DoD.

# gcov and gprof

## gcov

"gcov is a tool you can use in conjunction with GCC to test code coverage in your programs." [7]

- How often each line of code executes.
- Which lines of code are actually executed.

## gprof

The GNU profiler. [8]

- How much computing time each section of code uses.
- Can be used with gcov to locate costly algorithms.

# How gcov works

## gcno

"The .gcno notes file is generated when the source file is compiled with the GCC `-ftest-coverage` option. It contains information to reconstruct the basic block graphs and assign source line numbers to blocks."

## gcda

"The .gcda count data file is generated when a program containing object files built with the GCC `-fprofile-arcs` option is executed. A separate .gcda file is created for each object file compiled with this option. It contains arc transition counts, value profile counts, and some summary information."

# Example

## compile and execute

```

1 gcc -fprofile-arcs -ftest-coverage tmp.c
2 ./a.out
3 gcov tmp.c

```

## tmp.c

```

1  #include <stdio.h>
2
3  int main (void)
4  {
5      int i, total;
6      total = 0;
7      for (i = 0; i < 10; i++)
8          total += i;
9
10     if (total != 45)
11         printf ("Failure\n");
12     else
13         printf ("Success\n");
14     return 0;
15 }

```

## gcov output

```

1  -: 1:#include <stdio.h>
2  -: 2:
3  1: 3:int main (void)
4  -: 4:{
5  -: 5: int i, total;
6  1: 6: total = 0;
7  11: 7: for (i = 0; i < 10; i++)
8  10: 8: total += i;
9  -: 9:
10 1: 10: if (total != 45)
11 #####: 11: printf ("Failure\n");
12 -: 12: else
13 1: 13: printf ("Success\n");
14 1: 14: return 0;
15 -: 15:}

```



# How gcov works

## Note

- You can combine the results of many runs into one gcda data.
- UTs' and FTs' results can be combined too.

## Watch out

- "You should compile your code without optimization."
- "It works best with a programming style that uses only one statement per line."
- "Inlineable functions can create unexpected line counts."

# lcov

## lcov

"LCOV is a graphical front-end for GCC's coverage testing tool gcov. It collects gcov data for multiple source files and creates HTML pages containing the source code annotated with coverage information. It also adds overview pages for easy navigation within the file structure." [9]

## compile and execute

```
1 gcc -fprofile-arcs -ftest-coverage tmp.c
2 ./a.out
3 lcov --directory . --capture -o lcov.info
4 mkdir cov
5 genhtml lcov.info --frames -o ./cov
```

# lcov picture - overview

## LCOV - code coverage report

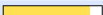
Current view: [top level](#)

Test: [lcov.info](#)

Date: **2012-10-01**

Legend: Rating: low: < 75 % medium: >= 75 % high: >= 90 %

	Hit	Total	Coverage
<b>Lines:</b>	7	8	<b>87.5 %</b>
<b>Functions:</b>	1	1	<b>100.0 %</b>
<b>Branches:</b>	3	4	<b>75.0 %</b>

Directory	Line Coverage ↕	Functions ↕	Branches ↕
<a href="#">cpp_play/lcov_prez</a>	 <b>87.5 %</b> 7 / 8	<b>100.0 %</b> 1 / 1	<b>75.0 %</b> 3 / 4

Generated by: [LCOV version 1.9](#)

# lcof picture - source browser

[Top](#)

## LCOV - code coverage report

Current view: [top level](#) - [cpp\\_play/lcof\\_prez](#) - [tmp.c](#) (source / functions)

Test: [lcof.info](#)

Date: 2012-10-01

Legend: Lines: hit not hit | Branches: + taken - not taken # not executed

Hit Total Coverage

Lines: 7 8 87.5 %

Functions: 1 1 100.0 %

Branches: 3 4 75.0 %

Branch data	Line data	Source code
		1 : #include <stdio.h>
		2 :
		3 : 1 : int main (void)
		4 : {
		5 : int i, total;
		6 :
		7 : 1 : total = 0;
		8 :
[ + + ]	11 :	11 : for (i = 0; i < 10; i++)
	10 :	10 : total += i;
		11 :
[ - + ]	1 :	1 : if (total != 45)
	0 :	0 : printf ("Failure\n");
		14 : else
	1 :	1 : printf ("Success\n");
	1 :	1 : return 0;
		17 : }

Generated by: [LCOV version 1.9](#)

# Final thoughts

## What have we got actually

- How often each line of code executes.
- What are the untested parts.
- Dead code detection.

## Line testing $\neq$ functionality testing

- UTs report code correctness.
- CC reports...UT correctness?

It's easy to write meaningless UTs to raise CC...it is not enough by itself.

# Final thoughts

## Testing

- UTs can be written with Test Driven Development - TDD [10]
- Test and code writer better be a different person otherwise if the code writer misunderstood the requirement, he tests his false model.
- Review tests, not just code. Test code quality should meet the same level as code quality.
- Test plan and test documentation.

## CC as a part of the DoD

- The team agrees on testing principles.
- CC should be  $>X\%$
- This rule can be enforced by CI: build fails if  $CC < X\%$ .

# Links



Michael C. Feathers, *Working Effectively with Legacy Code* Prentice Hall, 2004 ISBN 0-13-117705-2



[http://en.wikipedia.org/wiki/Design\\_by\\_contract](http://en.wikipedia.org/wiki/Design_by_contract)



Kolawa, Adam; Huizinga, Dorota (2007). *Automated Defect Prevention: Best Practices in Software Management* Wiley-IEEE Computer Society Press. p. 426. ISBN 0-470-04212-5.



Kaner, Falk, Nguyen. *Testing Computer Software* Wiley Computer Publishing, 1999, p. 42. ISBN 0-471-35846-0.



<http://www.scrumalliance.org/articles/105-what-is-definition-of-done-dod>



[http://en.wikipedia.org/wiki/Continuous\\_integration](http://en.wikipedia.org/wiki/Continuous_integration)



<http://gcc.gnu.org/onlinedocs/gcc/Gcov.html>



[http://www.cs.utah.edu/dept/old/texinfo/as/gprof\\_toc.html](http://www.cs.utah.edu/dept/old/texinfo/as/gprof_toc.html)



<http://ltp.sourceforge.net/coverage/lcov.php>



[http://en.wikipedia.org/wiki/Test-driven\\_development](http://en.wikipedia.org/wiki/Test-driven_development)

# Thank you for your attention!

This presentation can be found at:

[http://github.com/cs0rbagomba/cc\\_and\\_dod/cc\\_and\\_dod.pdf](http://github.com/cs0rbagomba/cc_and_dod/cc_and_dod.pdf)

or



Any questions?