SE420
Software Quality Assurance

Lecture 4 – Unit Testing Tools, Part-1

January 29, 2020

Assignment #1

- Avoid Subjective Test Results – Cost, Impact, Probability [Not from this year]

1. **“the code is generally bug free enough to run properly”**
2. **“probably not completely free from minor bugs”**
3. **“Overall the code is robust.”**
4. **“The generator is awful.”**
5. **“I believe that the code is acceptable.”**
6. **“…does not mean the code is bug free since we have found the major flaw already”**
7. **“subcrypt.c seems to work”**
8. **“the randomness appears satisfactory for a pseudo-random number generator.”**
9. **“My testing hasn’t proved it is bug free, as there could be bugs present, just not found in this specific instance.”**
10. **“When ran_seed is set to 0, it screws up the cipher”**

Examples of Non-constructive Code walk-through feedback

- Not actionable
- Appears “mean spirited” or “uncaring”

Issues with feedback:

1) Lack of specificity
2) Localization?
3) “appears”, “probably”, “believe”, … non-definitive assertion
4) Use of non-professional terms
5) Can the author really do anything with the feedback?

Assignment #1

- Provide Constructive, Quantified Well Characterized Results and Clear PASS / FAIL Outcome

1. **"Beyond 120 characters the code will fail in segmentation fault"**
2. **"If it did a good job then the average would have been 7.5000 with a std deviation of about 3."**
3. **"The shifting method took 0.001s to execute." "The 2's compliment standard method took 0.958s."**
4. **"The standard way is .879s to complete the dividing process, I put printf in both to make them equal, since if there was no printf in this one, it also ran at 00s."**
5. **"because of the *Pigeonhole Principle* from mathematics, we would expect each value to be repeated roughly 6-7 (100/16 = 6.25, number of times we would expect any given value to show up for a uniform random variable given the range 0-15 over 100 tests) times for a unbiased random number generator" – "As shown in the table above, only 5 random numbers repeat 6-7 times. This means that 11 random numbers are occurring more or less than we would expect from an unbiased random number generator."**
6. **"… do not have the time or resources to exhaustively test every permutation of a string of characters between 1 and 100 in length to be sure that all strings of those lengths are correctly encrypted."** - *better to point out that in fact this would be number of cases on the order of* $36^{100} + 36^{99} + \ldots + 36^{1}$ *assuming 26 alpha and 0…9 digits could be used in each position*

Constructive!

Actionable, Specific, and suggests improvement

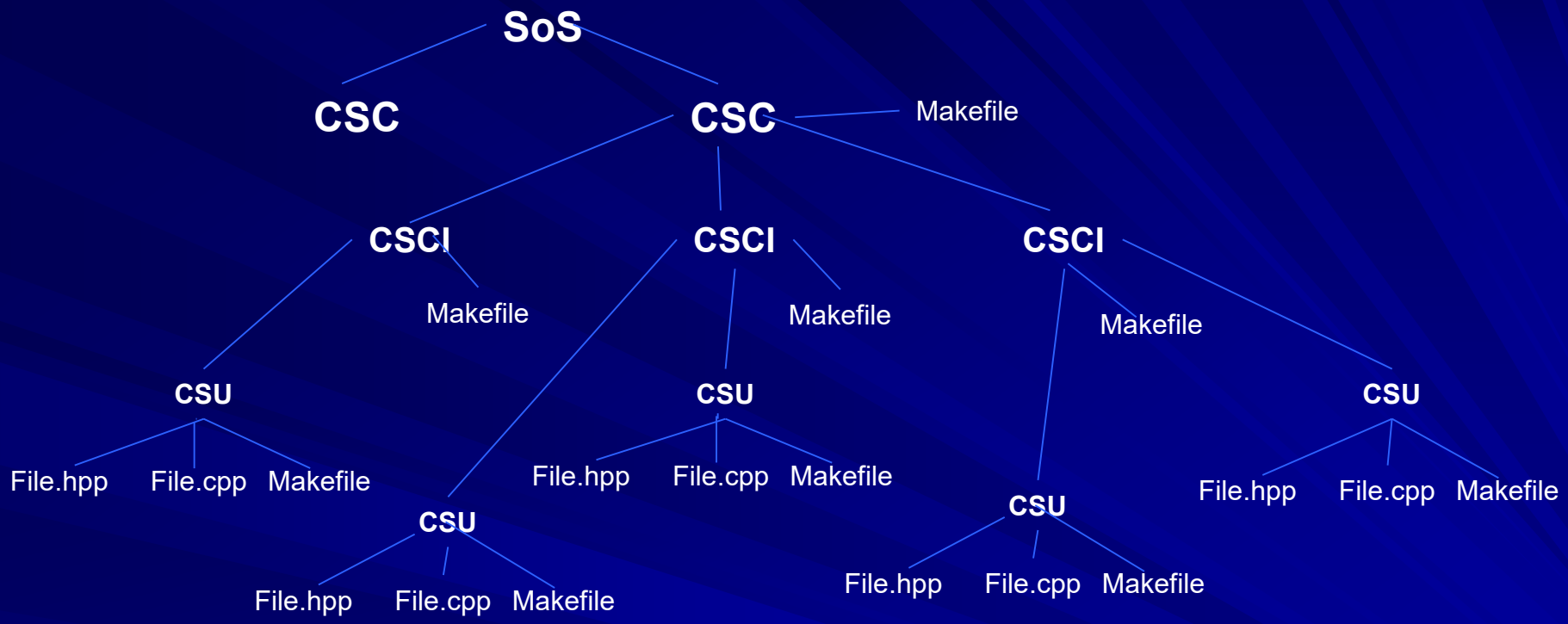
Value of constructive feedback:

1) Actionable
2) Localizes
3) Prevention
4) Objective
5) Professional
6) Improves Quality of Code, Design, Requirements, Documentation, User Guide, …

Design Modular Decomposition

- Enables Abstraction of details
- Test Driven Development – Matches Unit, I&T, System and Acceptance Test
- Enables Team member assignments and ownership
- Allows for and encourages re-use of components, software modules, and libraries
- Can be started in Architecture phase of design
- Parallel CMVC and Software Build Structure

Generic Modular Decomposition of a System or SoS

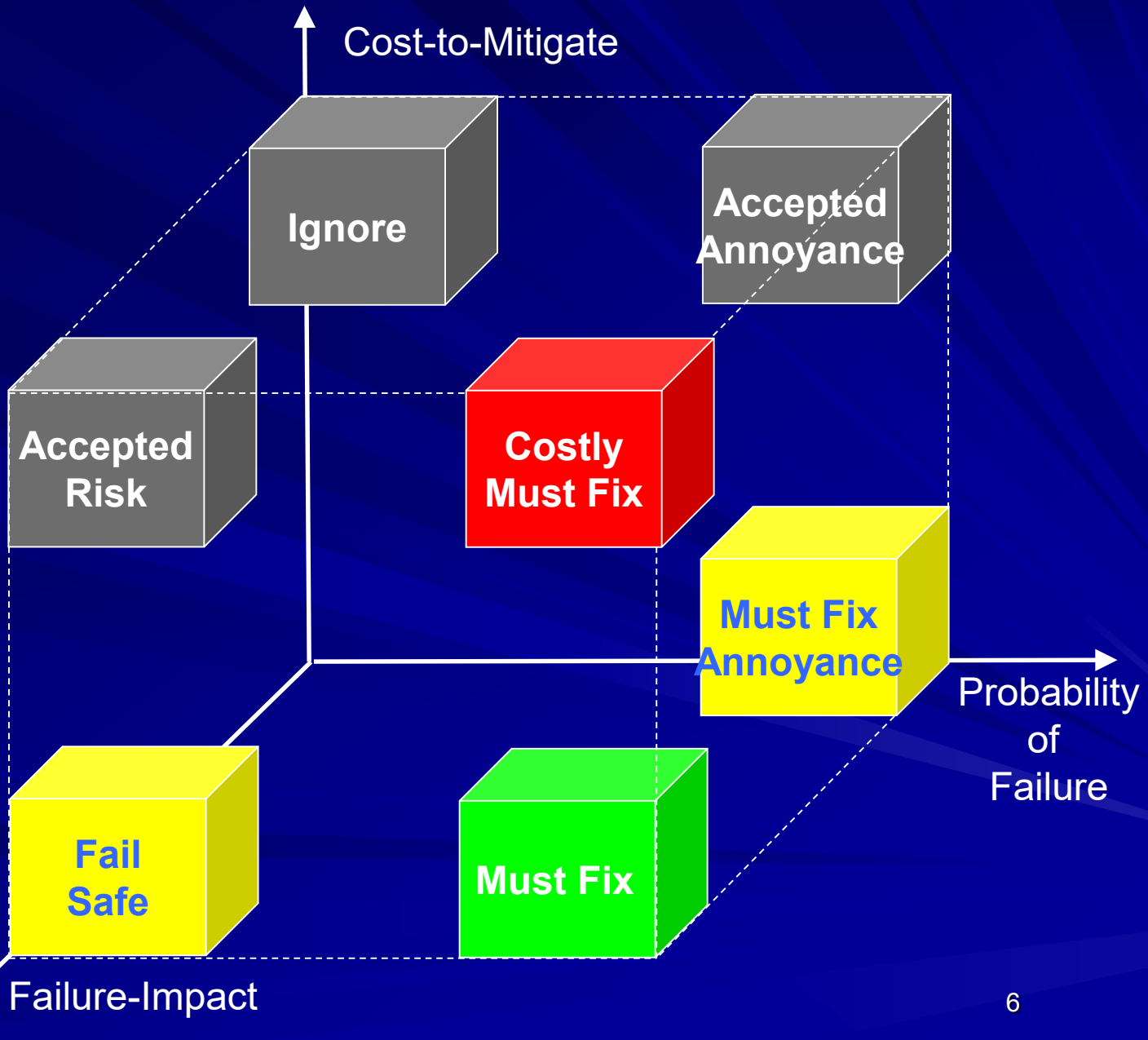


SoS – System of Systems (e.g. Distributed System such as Cloud Services)
CSC (Architecture) – Computer Software Configuration (System, e.g. application or systems software)
CSCI (High Level Design) – Computer Software Configuration Iterm (Sub-system, e.g. libraries of object code)
CSU (Detailed Design) – Computer Software Unit (Module or component, e.g.. Ada package, C header, source)

EMBRY-RIDDLE
ARIZONA

Quality, Safety, Cost, Effort - FMEA

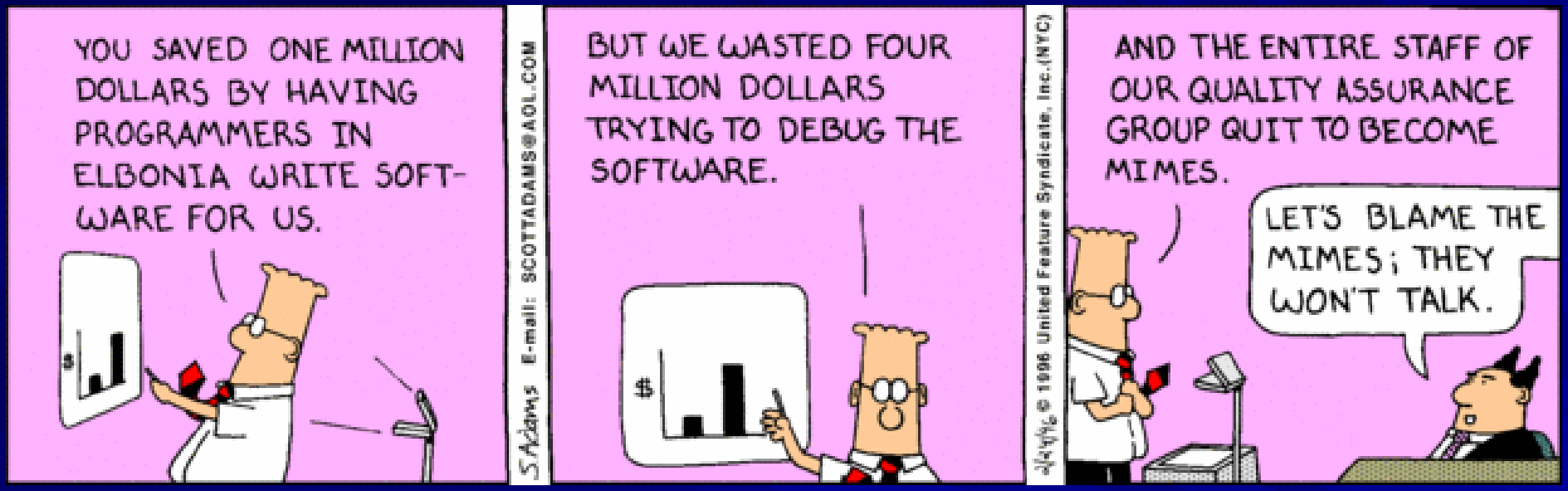
1. Failure Probability
2. Impact of Failure
3. Cost to Mitigate



Reminders

- Assignment #2
  - Questions?
  - Assignment #1 grading
- Remaining Assignments [Top Down]
  - #3 – Specification and Acceptance Test
  - #4 – System Design and System Integrated Test
  - #5 – Design, Module Unit Tests and Regression Suite
  - #6 – Complete Code, Refine and Run all V&V Tests and Deliver

White-Box & Black-Box Testing Methods



http://dilbert.com/strips/comic/1996-02-24/

Tools and Methods

White-Box (Inside Code under Test)

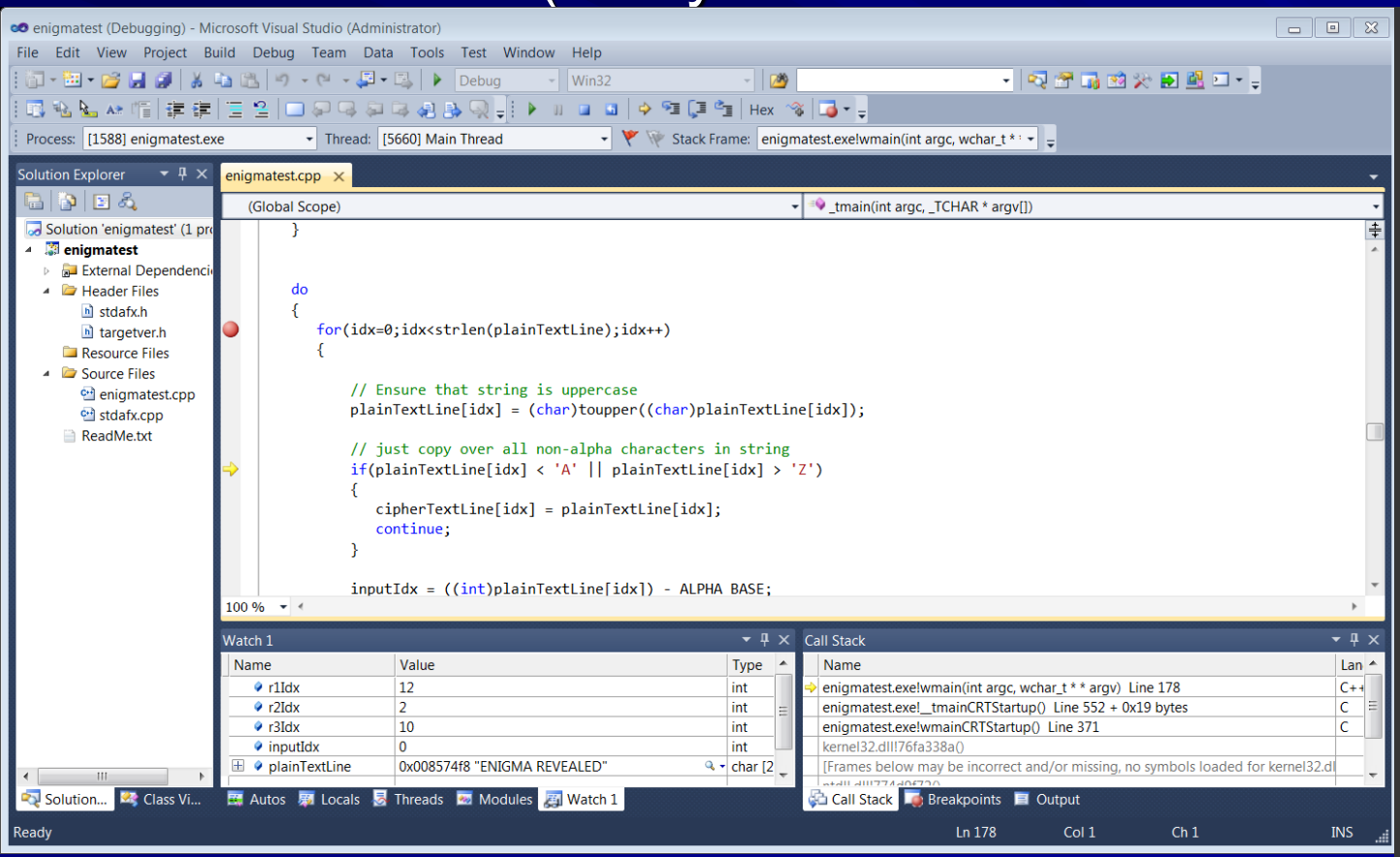
- Single Step Debug and Examine State – DDD, Visual Studio 20XX, IDE of your choice

- Basic Block – Function Body or Entry to Exit Point
  - Instruction Path Length
  - Clock / Time Path Length
  - Does the Block Return or Terminate?

- Coverage Criteria
  - Function (method)
  - Path
  - Statement
  - Instruction (Instructions Can Be Conditional)
  - Short-Circuit Logic Coverage

Tool-chain / IDE - DDD, Eclipse CDT, VSC++

- Great Start, but Hard to Automate White-Box Tests
- “do { <code_block>} while (condition);” is complex
- Software Breakpoint – Debug EXC instruction inserted into Machine Code (Every CPU has EXC Instruction)



Tool-chain

1. IDE - Edit, debug, build, run
2. Platform - Linux VM
3. Build - Makefile
4. Linux Top Errors
5. Vbox shared files
6. CMVC - GitHub

https://github.com/siewertserau

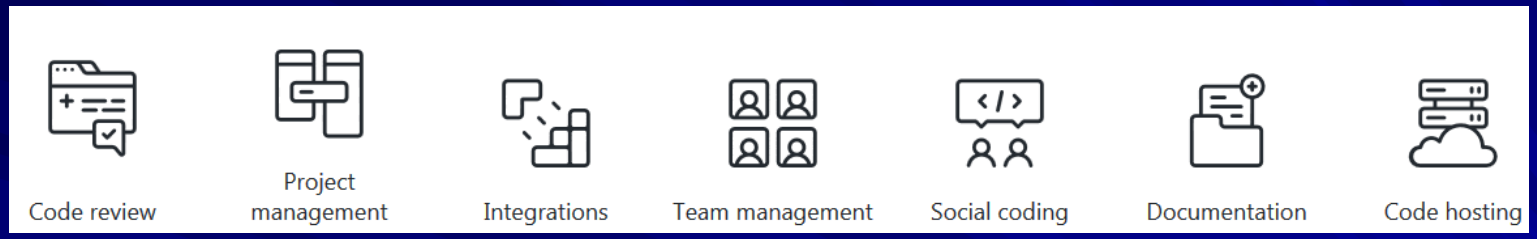
Alternatives to Linux VM, SE Workstation, or PRCLab1 - DevOps

- GitLab! - https://about.gitlab.com/ , Community Edition
  - Git CMVC and Tools to go with it (support for DevOps)
  - DevOps Tools for each phase of incremental/evolutionary lifecycle (coverage, project management, backlog, bug tracking, etc.)

  - Very well adapted for **Web and Mobile App** development process model (***Continuous Integration, Delivery, and Deployment***)
  - Useful for phases of Spiral / V, Agile Scrum Sprint, RUP as well
  - One stop, Cloud-based Incremental/Evolutionary Development support - Containers (e.g. Docker)
  - ERAU Prescott Academic request submitted (Gold), 30-day trial
  - Design support?
  - Iteration cycle length?
  - Suited for Mission Critical and Enterprise?

Cloud-Based Software Development

- GitHub, GitHub Tools, SQA - Standard Method
  - Free, Use with Kanboard
  - Simple to Learn, Use, Sufficient
  - Most any Incremental, Evolutionary Process



- GitLab - Better, but ***NOT Required***, Use instead if you wish
  - Integrated DevOps lifecycle tools
  - Continuous Integration, Delivery, Deployment

- Atlassian - Agile Scrum process, Use instead if you wish
  - Free for CMVC, other features vary
  - Bug Tracker,
  - Bitbucket CMVC (https://bitbucket.org/),
  - Jira Project Management (https://www.atlassian.com/software/jira)

Difference Between Breakpoints?

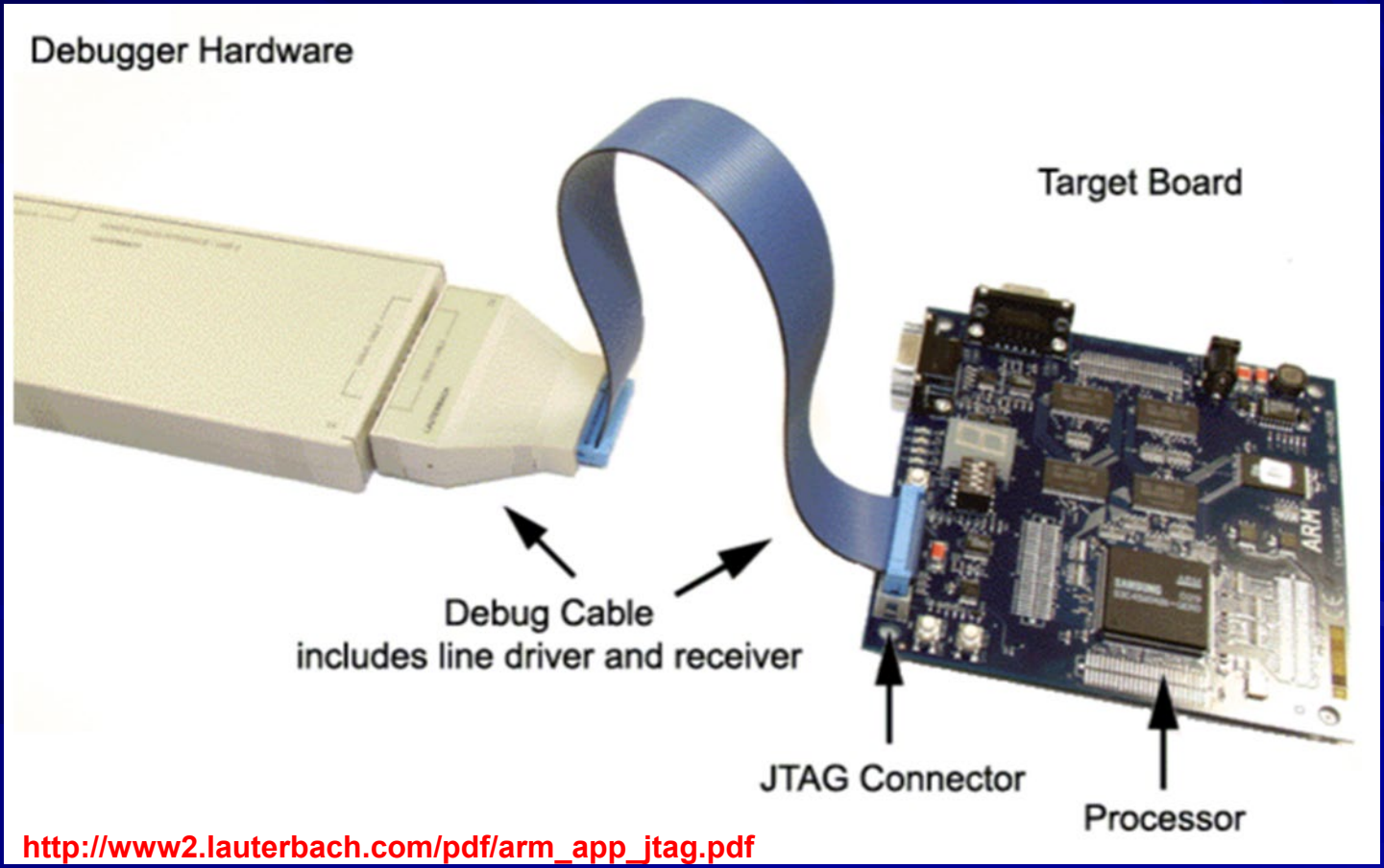
- Software Breakpoint (Debug EXC insertion)
  - How Implemented?
  - How Many Can You Have?
  - How Accurate and Can One Ever be Missed?
- Hardware Breakpoint (Address Comparator Hardware)
  - How Implemented?
  - How Many Can You Have?
  - How Accurate and Can One Ever be Missed?
- When should Each be Used?

White-Box Tools – Basic Block

- Function and Path Coverage – Gcov (GCC Coverage) tool
  - Lcov for Examples-Crypto - http://mercury.pr.erau.edu/~siewerts/se420/code/Enigma-LCOV-results/
  - Generated browse files from Gcov, Using Lcov
- Instruction Path Length and Clocks Per Instruction –
  - Vtune for Intel Architecture (Performance Tuning, PMAPI)
  - Count Instructions in Debugger or via Assembly code generation and Time Blocks with Timestamps
- Block Termination – Set Breakpoint Beyond Block, Run to BP
- Advanced Coverage Criteria – Instruction and MCDC (Multiple Condition Decision Coverage)
  - JTAG, Hardware Debugger
  - ICE – In Circuit Emulator
  - Address and Data bus Monitoring with Logic Analyzer

JTAG – Joint Test Applications Group

- IEEE Standard
- Designed for Hardware Scans, Useful for Low-Level Software Debug using TAP (Test Access Port)
- External Clocking of CPU to Step Through Instructions and to Run to a Hardware Breakpoint (Supported by CPU)



http://www2.lauterbach.com/pdf/arm_app_jtag.pdf

Gcov – Criteria for Black or White-box Testing

- https://gcc.gnu.org/onlinedocs/gcc/Gcov-Intro.html#Gcov-Intro
  - gcc -MD -Wall -O0 -fprofile-arcs -ftest-coverage -g -c enigma3.c
  - gcc -Wall -O0 -fprofile-arcs -ftest-coverage -g -o enigma3 enigma3.o –lgcov
  - ./enigma3
  - gcov -b -c enigma3.c
- How often each line of code executes
- What lines are actually executed
- How much computing time each section uses

```
siewerts@prc-ul8se-rac:~/se420/Examples-Crypto-Gcov$ make
cc -MD -Wall -O0 -fprofile-arcs -ftest-coverage -g   -c enigma3.c
cc  -Wall -O0 -fprofile-arcs -ftest-coverage -g   -o enigma3 enigma3.o -lgcov
siewerts@prc-ul8se-rac:~/se420/Examples-Crypto-Gcov$ ./enigma3
Will run simple demo
Input = ENIGMA REVEALED
Output = QMJIDO MZWZJFJR
siewerts@prc-ul8se-rac:~/se420/Examples-Crypto-Gcov$ ls
enigma3    enigma3.d     enigma3.gcno  enigma3.o  plaintext.in  result
enigma3.c  enigma3.gcda  enigma3.info  Makefile   README
siewerts@prc-ul8se-rac:~/se420/Examples-Crypto-Gcov$ gcov -b -c enigma3.c
File 'enigma3.c'
Lines executed:73.20% of 97
Branches executed:76.32% of 76
Taken at least once:44.74% of 76
Calls executed:29.27% of 41
Creating 'enigma3.c.gcov'

siewerts@prc-ul8se-rac:~/se420/Examples-Crypto-Gcov$ ls
enigma3    enigma3.c.gcov  enigma3.gcda  enigma3.info  Makefile      README
enigma3.c  enigma3.d       enigma3.gcno  enigma3.o     plaintext.in  result
siewerts@prc-ul8se-rac:~/se420/Examples-Crypto-Gcov$
```

Gcov report

- What was executed, what was not
- Lcov is easier to read and to produce as regression report – e.g. http://mercury.pr.erau.edu/~siewerts/se420/code/Enigma-LCOV-results/

```
        -:  112:   //int notch1Idx=16;
        1:  113:   int notch2Idx=4, notch3Idx=21;
        -:  114:
        -:  115:
        1:  116:   if(argc == 1 || argc == 2)
branch  0 taken 0 (fallthrough)
branch  1 taken 1
branch  2 never executed
branch  3 never executed
        -:  117:   {
        1:  118:      demoRun=1;
        1:  119:      printf("Will run simple demo\n");
call    0 returned 1
        2:  120:      if(argc == 2 && (strncmp(argv[1],"-D", 2) == 0))
branch  0 taken 0 (fallthrough)
branch  1 taken 1
branch  2 never executed
branch  3 never executed
        -:  121:      {
    #####:  122:         dbgOn=1;
        -:  123:      }
        -:  124:   }
    #####:  125:   else if(argc >= 3)
branch  0 never executed
branch  1 never executed
        -:  126:   {
    #####:  127:      strcpy(inputFile, argv[1]);
    #####:  128:      strcpy(outFile, argv[2]);
    #####:  129:      printf("Will run Enigma3 encode on input file=%s and produce file=%s\n",
call    0 never executed
        -:  130:             inputFile, outFile);
        -:  131:
```

Simple.c Example - Here

```
#include "stdio.h"
#include "stdlib.h"

int main(void)
{
    int x;

    if(x > 0)
    {
        printf("TRUE\n");
    }
    else
    {
        printf("FALSE\n");
    }

    exit(1);
}
```

```
%make
cc -MD -Wall -O0 -fprofile-arcs -ftest-coverage -g    -c simple.c
simple.c: In function 'main':
simple.c:8: warning: 'x' is used uninitialized in this function
cc -Wall -O0 -fprofile-arcs -ftest-coverage -g    -o simple simple.o
%ls
Makefile  simple  simple.c  simple.d  simple.gcda  simple.gcno  simple.o
%./simple
FALSE
%gcov simple
File 'simple.c'
Lines executed:80.00% of 5
simple.c:creating 'simple.c.gcov'
```

```
        -:    0:Source:simple.c
        -:    0:Graph:simple.gcno
        -:    0:Data:simple.gcda
        -:    0:Runs:1
        -:    0:Programs:1
        -:    1:#include "stdio.h"
        -:    2:#include "stdlib.h"
        -:    3:
        1:    4:int main(void)
        -:    5:{
        -:    6:    int x;
        -:    7:
        1:    8:    if(x > 0)
        -:    9:    {
    #####:   10:        printf("TRUE\n");
        -:   11:    }
        -:   12:    else
        -:   13:    {
        1:   14:        printf("FALSE\n");
        -:   15:    }
        -:   16:
        1:   17:    exit(1);
        -:   18:}
```

Enigma3 Example

- More Complex, but Same Idea
- Note that My Test Driver Executes Only 73.47% of Code

```
siewerts@prc-u18se-rac:~/se420/Examples-Crypto-Gcov$ make
cc -MD -Wall -O0 -fprofile-arcs -ftest-coverage -g   -c enigma3.c
cc  -Wall -O0 -fprofile-arcs -ftest-coverage -g   -o enigma3 enigma3.o -lgcov
siewerts@prc-u18se-rac:~/se420/Examples-Crypto-Gcov$ ./enigma3
Will run simple demo
Input = ENIGMA REVEALED
Output = QMJIDO MZWZJFJR
siewerts@prc-u18se-rac:~/se420/Examples-Crypto-Gcov$ ls
enigma3    enigma3.d     enigma3.gcno  enigma3.o  plaintext.in  result
enigma3.c  enigma3.gcda  enigma3.info  Makefile   README
siewerts@prc-u18se-rac:~/se420/Examples-Crypto-Gcov$ gcov -b -c enigma3.c
File 'enigma3.c'
Lines executed:73.20% of 97
Branches executed:76.32% of 76
Taken at least once:44.74% of 76
Calls executed:29.27% of 41
Creating 'enigma3.c.gcov'

siewerts@prc-u18se-rac:~/se420/Examples-Crypto-Gcov$ ls
enigma3    enigma3.c.gcov  enigma3.gcda  enigma3.info  Makefile      README
enigma3.c  enigma3.d       enigma3.gcno  enigma3.o     plaintext.in  result
siewerts@prc-u18se-rac:~/se420/Examples-Crypto-Gcov$
```

Browsing Coverage for Module

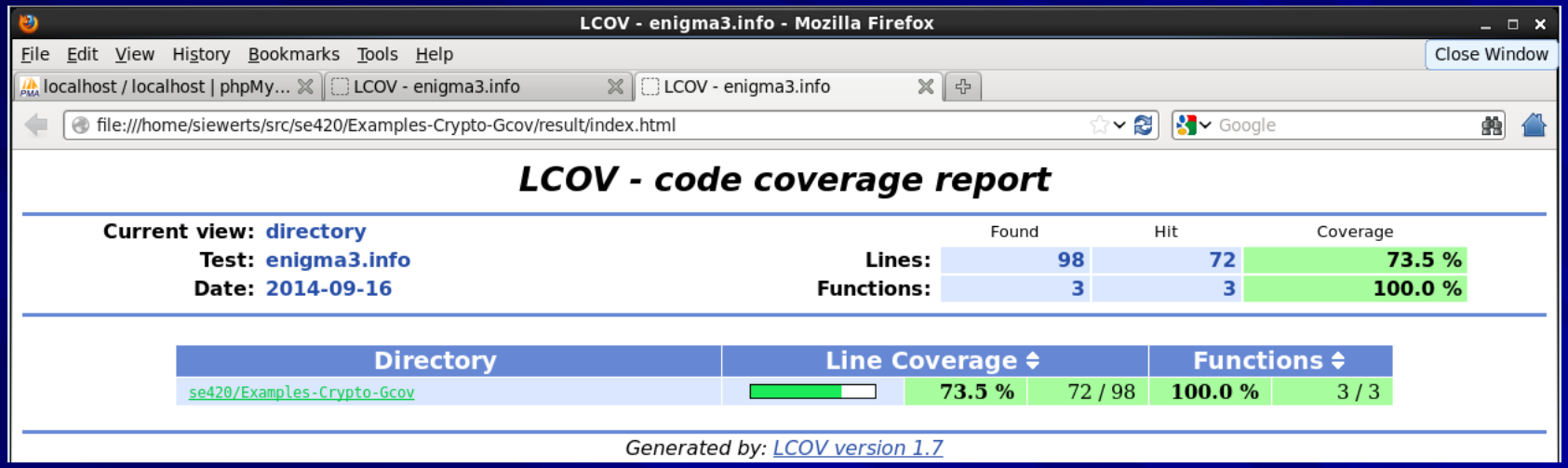
- Generate LCOV HTML Result Directory to Browse Coverage

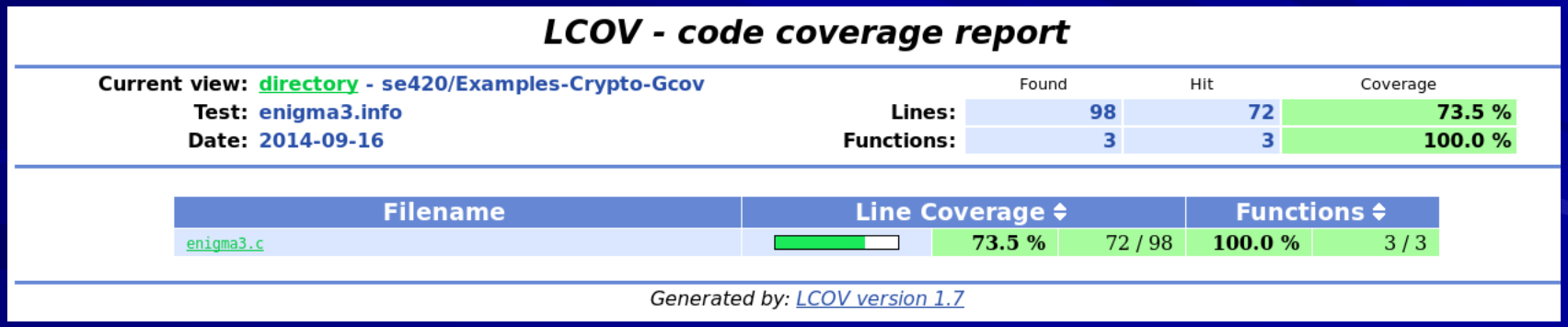
```
[siewerts@localhost Examples-Crypto-Gcov]$ lcov -t 'Enigma report' -o enigma3.info -c -d .
Capturing coverage data from .
geninfo: WARNING: invalid characters removed from testname!
Found gcov version: 4.4.7
Scanning . for .gcda files ...
Found 1 data files in .
Processing ./enigma3.gcda
Finished .info-file creation

[siewerts@localhost Examples-Crypto-Gcov]$ genhtml -o result enigma3.info
Reading data file enigma3.info
Found 1 entries.
Found common filename prefix "/home/siewerts/src"
Writing .css and .png files.
Generating output.
Processing file se420/Examples-Crypto-Gcov/enigma3.c
Writing directory view page.
Overall coverage rate:
  lines......: 73.5% (72 of 98 lines)
  functions..: 100.0% (3 of 3 functions)
```

Coverage Report Browsing

- Top Level Shows Coverage for Each White-Box Tested Module (Enigma-LCOV-results/)





Coverage Inside Module

- Code Not Covered by Tests Shown in Red
- Browse - http://mercury.pr.erau.edu/~siewerts/se420/code/Enigma3-gcov-results/

```
100              :
101            1 : int main( int argc, char *argv[] )
102              : {
103            1 :    int demoRun=0;
104            1 :    int dbgOn=0, foundEnd=0;
105              :    FILE *fpIn, *fpOut;
106            1 :    int lineSz=128;
107            1 :    int bytesRd=0;
108              :
109              :    // Initial conditions for machine
110            1 :    int r1Idx=12, r2Idx=2, r3Idx=10, inputIdx=0, idx, rIdx, rDiff;
111            1 :    int notch1Idx=16, notch2Idx=4, notch3Idx=21;
112              :
113              :
114            2 :    if(argc == 1 || argc == 2)
115              :    {
116            1 :       demoRun=1;
117            1 :       printf("Will run simple demo\n");
118            1 :       if(argc == 2 && (strncmp(argv[1],"-D", 2) == 0))
119              :       {
120            0 :          dbgOn=1;
121              :       }
122              :    }
123            0 :    else if(argc >= 3)
124              :    {
125            0 :       strcpy(inputFile, argv[1]);
126            0 :       strcpy(outFile, argv[2]);
127            0 :       printf("Will run Enigma3 encode on input file=%s and produce file=%s\n",
128              :              inputFile, outFile);
129              :
130            0 :       if(argc == 4 && (strncmp(argv[3],"-D", 2) == 0))
131              :       {
132            0 :          dbgOn=1;
```

Test case did not turn on debug feature in main test driver code

Remove dbgOn code block or drive it with a test case and Leave it in – but do not ship untested!

Gcov

- Drive Module Testing with main() programs in each directory
- Build and execute main() drivers with gcov
  - Driver Test Cases from White-Box Design (Knowledge of Source Code at Debugger Level)
  - Driver Test Cases from Black-Box function call perspective
- Generate Coverage reports
- Provides Clear Exit Criteria (Coverage) for Both Methods!

Enigma3.c Validation

- Enigma is a Well Documented Machine
- I wrote the C code Based on Paper Enigma(s)
- Consider the Paper Enigma and Documents a Specification
- Consider Enigma Emulator (Executable Design Spec)
- Consider the C code enigma3.c a Unit to Verify and Validate
- Question for Next Time – How Would We Compose and Acceptance Test?

Summary of Gcov Commands

```
%make clean
rm -f *.o *.d *.exe sclogic *.gcov *.gcno *.gcda *.info
%ls
Makefile  SC-Logic-2-LCOV-results  SC-Logic-2-LCOV-
results.zip  sclogic.c

<< ALWAYS MAKE CLEAN FIRST>>

%make
cc -Wall -O0 -fprofile-arcs -ftest-coverage -
g      sclogic.c   -o sclogic

<< DO NEW BUILD ON YOUR TEST MACHINE AND NOT PROFILE-
ARCS and TEST-COVERAGE GCOV INSTRUMENTATION DIRECTIVES
>>

%./sclogic
function_A
function_B
do function_C
do function_D
function_A
do function_D
…
do function_C
function_A
do function_D
function_A
function_B
do function_D

<<RUN OF CODE TO BE TESTED WITH SOME PRINTF DEBUG OUTPUT
HERE>>
```

```
%gcov sclogic.c
File 'sclogic.c'
Lines executed:100.00% of 29
sclogic.c:creating 'sclogic.c.gcov'

<<RUN POST RUN COVERAGE ANALYSIS ON SOURCE FOR LAST
RUN OF INSTRUMENTED CODE>>

%gcov sclogic
File 'sclogic.c'
Lines executed:100.00% of 29
sclogic.c:creating 'sclogic.c.gcov'

<<RUN POST RUN COVERAGE ANNOTATED TEXT
GENERATION>>

%cat sclogic.c.gcov
      -:    0:Source:sclogic.c
      -:    0:Graph:sclogic.gcno
      -:    0:Data:sclogic.gcda
      -:    0:Runs:1
      -:    0:Programs:1
      -:    1:#include <stdio.h>
…
      1:   41:int main(void)
      -:   42:{
…
      -:   52:    // Test Case #2, Test use in logic
     11:   53:    for(testIdx=0; testIdx < 10; testIdx++)
      -:   54:    {
     10:   55:        if((rc=(function_A() && function_B())))
      2:   56:            function_C();
      -:   57:        else
      8:   58:            function_D();
      -:   59:
      -:   60:    }
      -:   61:
      1:   62:    return(1);
      -:   63:}
```

Summary of Lcov Commands

```
%lcov -t 'SC LOGIC 2 REPORT' -o sclogic.info -c -d .
Capturing coverage data from .
Found gcov version: 4.4.7
geninfo: WARNING: invalid characters removed from testname!
Scanning . for .gcda files ...
Found 1 data files in .
Processing sclogic.gcda
Finished .info-file creation

<<RUN LCOV TOOL ON GCOV RESULTS>>

%genhtml -o result2 sclogic.info
Reading data file sclogic.info
Found 1 entries.
Found common filename prefix "/home/facstaff/siewerts/se420/src"
Writing .css and .png files.
Generating output.
Processing file MCDC2/sclogic.c
Writing directory view page.
Overall coverage rate:
  lines......: 100.0% (29 of 29 lines)
  functions..: 100.0% (5 of 5 functions)
%

<<GENERATE THE WEB PAGES FROM LCOV RESULTS FOR BROWSING>>
```