MPW PQR4 Appendix B
Release Notes

Table of Contents

Preface

△ **Important** Big Brother can be used only with Macintosh computers whose CPU is an MC68020 or higher. △

Organization of this Manual

The first section of this manual, “An Introduction to Code Coverage and Big Brother,” contains a brief explanation of code coverage theory and describes the benefits of using Big Brother to measure the code coverage of test suites. There are also descriptions of Big Brother’s feature set, hardware and software requirements, and the files that make up the Big Brother product.

“Getting Started” contains brief, step-by-step instructions for getting Big Brother up and running.

“Installing and Removing the Big Brother Parasite,” describes how to use the BigBrother tool to begin and end code coverage sessions. This section is important because it explains how to specify code coverage targets.

“Starting and Stopping Coverage Analysis,” describes what happens after Big Brother is installed (specifically, what events trigger Big Brother into action.)

“Reporting Coverage Results” explains how to use the BBReporter tool and how to interpret the output.

“Technical Information” contains a low level description of how Big Brother works.

“Helpful Hints” contains hints and techniques for using Big Brother more efficiently and effectively.

“Command Reference Pages” contains manual reference pages for the Big Brother tools.

Release Information & Open Bugs

This is an alpha version of Big Brother. **It has not been completely tested and it contains bugs that can cause your Mac to crash.** To save yourself some headaches, please read the following list of open important bugs (as of 4/24/91).

- **Big Brother won't start tracing a targeted MPW tool when the tool is in the root directory and is invoked with a partial pathname.** When in doubt, use full pathnames.
- **The -sys, -aa and -trap options have not been adequately tested.** Use them at your own risk .
- **Big Brother crashes when targeting all 448 CODE resources in 4th Dimension.** Only target a few CODE resources at a time. Your program will execute much faster, and you won't be likely to run into memory limitations.
- **Big Brother is not compatible with Virtual Memory in System 7.0.** BBReporter will usually report 0% coverage when VM is on. Run Big Brother with VM off, if possible.
- **You shouldn't remove the parasite and stop coverage (e.g. BigBrother -r) while the targeted application is still running, or the application will crash when you switch back to it.** If possible, quit the application before removing the parasite. If you can't quit the application, save your important data and reboot before switching back.
- **Specifying -trap sometimes causes the targeted application to hang.**

There are more bugs in Big Brother. Please help me fix them by reporting them.

An Introduction to Code Coverage and Big Brother

What is Code Coverage and Why is it Important?

Code coverage is a measure of the extent to which a set of tests exercises the statements and branches of a program. *Complete* code coverage means that every statement has been exercised at least once, and every conditional branch has been exercised over all possible outcomes (i.e. branched and failed to branch).

If your tests do not provide complete coverage, then there is some section of your code that has not been tested. Without the information that testing provides, you are poorly prepared to make decisions concerning the quality of the code (e.g. Is it ready for release?).

It is important to remember that complete code coverage is not by itself an indication of sufficient testing. In other words, you can execute every line of code and still get incorrect results. Code coverage tools like Big Brother only tell you what code has been executed. It is up to you to decide what should have been executed, and to interpret the results of that execution.

More information about the code coverage and software testing theory can be found in several of the books listed in the bibliography.

What is Big Brother and How do I Use it?

Big Brother is a set of code coverage tools that is part of the Macintosh Programmers Workshop. It consists of the following disk files:

- an MPW Tool, **BigBrother**, that installs and removes the Big Brother parasite. The parasite is the code that runs in the background and records execution information.
- an MPW Tool, **BBReporter**, that interprets the information

collected by the parasite in meaningful ways and generates useful reports,

- an INIT, **BigBroINIT**, that records important system information at startup,
- a text file, **BB.Help**, that may be appended to your MPW.help file to allow you to get information about Big Brother using MPW's Help command,
- this document, the **Big Brother User's Manual**.

To use Big Brother, you install the Big Brother parasite, specifying a section of executable code as a target. As you exercise the targeted code (usually via a test suite), the Big Brother parasite watches, and records the starting and ending addresses of each separate range of executed instructions. Using BBReporter, you take the recorded information and generate reports about what percentages of code were executed and what instructions were executed. BBReporter optionally creates an annotated disassembly of the targeted code with bullets next to each executed instruction.

Summary of Features

- Any Macintosh resource in any resource file can be specified as a coverage target
- Absolute address ranges (e.g. ROM addresses) can be specified as coverage targets
- No coverage target source code is necessary
- The coverage target code is not modified
- The functionality and interface of a coverage target is not affected during coverage analysis except in time critical applications.
- The scope of coverage analysis is specified by the user
- Cumulative coverage reports, based on any grouping of coverage analyses, that can include:
    - Percentage of target code executed (by resource or module, if Macsbug symbols are embedded in the code)
    - An indication of the code that was executed and the code that was not executed.

System Requirements

- **A Macintosh with a 68020, '030 or '040 processor** - Big Brother makes use of the jump trace exception that is not supported by the 68000 processor in the Mac Plus, SE, Classic, and Portable.
- **The MPW Shell** - Big Brother is currently implemented as a set of MPW tools.
- System Software version 6.0x or 7.0
- **[Optional] Macsbug** (or any debugger that supports DebugStr calls) - For the experimental version of Big Brother, some errors are reported via DebugStr calls to Macsbug.

Getting Started

1) Copy the BigBroINIT to your System Folder
Make sure it is called **BigBroINIT**.

2) Copy the Big Brother tools to your tools folder, or add them to the command search path

3)Reboot your system

4)Start MPW

5)Target the code you wish to investigate using BigBrother -i

6)Exercise the code (by running the test suite, etc)

7)Turn off Big Brother using BigBrother -r

8)Display the results using BBReporter

9)Repeat steps 5-9 as desired

Installing and Removing the Big Brother Parasite

To install the Big Brother parasite, use the BigBrother tool specifying the -i option. Three mutually exclusive command line options: -f, -sys and -aa, tell Big Brother where the targeted code is located. Exactly one of these options must be included in the command line. Use -sys if the target code is located in a resource in the system file (e.g. PACK resources.) Use -f followed by a pathname if the code is in a resource in some other file (e.g. application CODE resources.) The pathname may be full or partial. If it is partial, the location of the file is computed relative to MPW's current default directory. Use -aa followed by a range of absolute addresses if the code is located in some known, absolute address range (ROM routines, for example).

When using -sys or -f, Big Brother defaults to targeting every CODE resource in the specified file (except CODE 0, the jump table). You may override this default by using the -rt option. The syntax for the -rt option is "-rt type=id" where type is the resource type (e.g. CODE or PACK) and id is the resource id number. The resource type is case-sensitive, so "-rt code=1" is not the same as "-rt CODE=1". The resource descriptor is also space sensitive. Do not put spaces before or after the "=", (unless the resource type ends with a space). You may include multiple -rt options in the command line.

Name the output file using the -o option. The pathname following -o may be full, or partial, but it must be a valid filename (i.e. it must not end in a ":").

During program execution, Big Brother normally turns off coverage analysis when an A-trap's code is executing. This is to prevent wasting time computing code coverage for the Macintosh ROMs. If you want to trace A-traps, include -trap on the command line. This will slow down execution significantly.

Finally, there are two restrictions on coverage targets to keep in mind:

- Big Brother cannot trace any software that modifies the trace or A-trap exception handlers in such a way that the exception handlers installed by Big Brother are not executed.
- Big Brother cannot trace exception handlers unless the handler sets the jump trace bit. This is because the 680x0 microprocessors always turn off the tracing bits in the status register when an exception occurs. (The A-trap exception is a special case, and Big Brother can trace it.)

After installing Big Brother, you may begin to exercise the targeted code. The functionality and interface of the code is unaffected by the presence of Big Brother (except that the code runs more slowly.)

When you have finished exercising the code, call BigBrother again, this time specifying -r to write the gathered data to disk and clean up after the parasite. Big Brother automatically names the data file BigBrother.out. You may override this default with the -o option.

You must explicitly remove previous Big Brother installations before re-installing.

Starting and Stopping Coverage Analysis

After the Big Brother parasite has been installed, it waits for the targeted resource file to be opened via _OpenRF . (unless the targeted file is the system file, in which case it is already open). Once the file has been opened, the parasite waits for the specified resource to be loaded via _GetResource (unless the resource is in the system file and is already loaded). After the _GetResource call, coverage tracking begins. Coverage tracking for absolute addresses begins immediately after BigBrother -i, since there are no associated _OpenRF and _GetResource calls.

During coverage tracking, each executed range of memory is compared to the list of targeted ranges, and if there is a match, the range is recorded. The range is merged into a sorted list of ranges to ease memory requirements.

Coverage tracking for a resource ends temporarily when _HUnlock or _DisposHandle is called for that resource. A subsequent _GetResource will resume tracking as part of the same coverage run. Coverage tracking ends permanently (and the data file is written to disk) only when BigBrother -r is executed.

Reporting Coverage Results

Once you have created a Big Brother data file by exercising your code and calling BigBrother -r, you create a coverage report using the BBReporter tool. The default report lists the percent of code executed for each resource (or address range). For example:

```
============================================================
        Coverage Report for Targets in "MyProgram"
============================================================
CODE = 2              STDCLIB         30.25% =    36 /   119
CODE = 3              %A5Init         43.23% =   115 /   266
CODE = 1              Main            42.55% =   220 /   517
============================================================
Total                                 41.13% =   371 /   902
```

This says that the targeted program was “MyProgram”, and that there were three CODE resources targeted, Main, STDCLIB and %A5Init. For the STDCLIB segment, 30.25% of the code was executed. In this case, the number of executable object code instructions in the segment was 119, and the number of executed object code instructions was 36, and 36÷119 = 30.25%.

Specifying the -d option causes BBReporter list a disassembly of the targeted code after the percentage report. The format of the disassembly is very similar to the output of DumpObj. The lines of code that were executed are marked with a bullet character, “•”, as in the following example (which has been compressed to fit into the width of this page):

```
+00000000    •:   48E7 7FF8    'H...'    MOVEM.L       D1-D7/A0-A4,-(A7)
+00000004    •:   49FA 01A6    'I...'    LEA           *+$01A8,A4    ; 000001AC
```



```
+00000008    •:   3014          '0.'     MOVE.W      (A4),D0
+0000000A    •:   5340          'S@'     SUBQ.W      #$1,D0
+0000000C    •:   6704          'g.'     BEQ.S       *+$0006       ; 00000012
+0000000E     :   70FF          'p.'     MOVEQ       #$FF,D0
+00000010     :   6036          '`6'     BRA.S       *+$0038       ; 00000048
+00000012    •:   264D          '&M'     MOVEA.L     A5,A3
+00000014    •:   97EC 0004     '....'   SUBA.L      $0004(A4),A3
+00000018    •:   2F0B          '/.'     MOVE.L      A3,-(A7)
+0000001A    •:   2F2C 0004     '/,..'   MOVE.L      $0004(A4),-(A7)
```


If Macsbug symbols have been embedded in the code by the compiler, BBReporter will recognize them and label the end of the module accordingly (see below) Previous versions of BBReporter allowed you to ignore Macsbug symbols. That is no longer an option.

```
...
+0000001E    •:  7264          'rd'      MOVEQ     #$64,D1           ; 'd'
+00000020    •:  4EBA 0448     'N..H'    JSR       *+$044A           ; 0000046A
+00000024    •:  4E5E          'N^'      UNLK      A6
+00000026    •:  4E75          'Nu'      RTS
•• End of Module "main"

+00000030    •:  2057          '.W'      MOVEA.L   (A7),A0
+00000032    •:  42A7          'B.'      CLR.L     -(A7)
+00000034    •:  486D FE7E     'Hm.~'    PEA       $FE7E(A5)
...
```

Disassembly of Embedded Macsbug symbols

BBReporter -m outputs percentage information *by module*.

```
============================================================
        Coverage Report for Targets in "MyProgram"
============================================================
CODE = 3              STDCLIB          30.25% =    36 /   119
------------------------------------------------------------
_cvt                                    0.00% =     0 /    59
memcpy                                 36.67% =    11 /    30
srand                                 100.00% =     4 /     4
rand                                  100.00% =    13 /    13
strcpy                                  0.00% =     0 /     5
strlen                                100.00% =     8 /     8

CODE = 2              %A5Init          43.23% =   115 /   266
------------------------------------------------------------
_DATAINIT                              92.31% =    24 /    26
uncompress_world                      100.00% =    30 /    30
get_rl                                 56.25% =    18 /    32
relocate_world                         78.12% =    25 /    32
ZEROBUFFER                             78.26% =    18 /    23
N/A                                     0.00% =     0 /   123

CODE = 1              Main             42.55% =   220 /   517
------------------------------------------------------------
main                                  100.00% =     5 /     5
__CplusInit                            38.24% =    39 /   102
dtors()                                 0.00% =     0 /    44
_RTInit                                62.89% =   100 /   159
```



```
exit                                    71.43% =     5 /     7
_RTExit                                 74.19% =    23 /    31
sig_dfl                                 39.34% =    48 /   122
N/A                                      0.00% =     0 /    47
=============================================================
Total                                   41.13% =   371 /   902
```


In this case, every module but “main” was added by the linker. The user written code is all contained in the 5 assembly lines of “main”.

In CODE segments 1 and 2, the last module listed is named “N/A”. This indicates that there was no Macsbug symbol after the last bit of executable code before the end of the segment.

The original BBReporter used resource size as as the denominator in the percentage calculations. This included embedded Macsbug symbols. The new BBReporter uses executable object code statements as the denominator, and executed object code statements in the numerator. Macsbug symbols (and inter-module string constants and other bits of non-executable compiler-generated chaff) are not included in any calculations. The numbers you now see are much more precise.

Technical Information

How Does it Work?

Big Brother uses the jump trace exception available on the MC68020, MC68030 and MC68040 microprocessors. A jump trace exception occurs whenever there is a non-sequential change of program flow. Instructions that can trigger a jump trace exception include branches, jumps, JSRs and returns. When a jump trace exception occurs, the Big Brother parasite gets control and records the range of instructions that have been executed since the last jump trace exception. To get control, Big Brother patches the a-trap exception handler and the jump-trace exception handler. Patching the A-Trap handler is, in general, a very foolish thing to do. Unfortunately, it is not possible to create global patches from within an application because Multifinder swaps out application patches. Since Big Brothers patches need to survive across application swaps, we chose to patch the A-Trap handler.

This implementation yields four principal features:

- The program being tested is not modified in order to enable code coverage checking
- The user of Big Brother does not need the source code to the program being tested

- The program being tested can be implemented in any programming language.
- Coverage analysis can be performed on the Macintosh ROM.

However, this implementation causes several problems:

- Extra code must be run at every change of program flow! This can cause a severe performance hit if program flow changes frequently.
- Bugs in Big Brother are likely to have severe side effects. Since Big Brother works at such a low level, it is very easy to corrupt heaps and damage stacks.
- Big Brother can not tell the difference between executable code and data stored in-line.
- Big Brother is incompatible with other programs that use the jump trace exception.

What does BigBroINIT do?

BigBroINIT records the system values for the A-trap exception vector and the jump trace exception vector. It also provides a handy place to store handles to the locations of the Big Brother Parasite and the Big Brother master data structure in the system heap. The INIT does not install any code or patch any traps. It merely updates one of its resources and quits. Therefore it should not cause incompatibilities or problems with other INITs and programs..

Helpful Hints

- REBOOT AFTER INSTALLING BigBroINIT (or replacing BigBroINIT with a fresh copy.)

  BigBrother will not let you install or remove if the vectors stored in BigBroINIT are incorrect. You must reboot to initialize those values.
- TURN OFF EXTRA BACKGROUND CODE

Big Brother slows down your machine since it must execute additional code every time there is a change in program flow. To minimize this effect, try to have as little extraneous code running as possible. Turn off screen savers, background applications, weird INITs, etc. If you don't, you'll be sorry.

- DONT TARGET EXTRA CODE

  Put the code you will be targeting in a separate segment if possible. This lets you avoid the performance hit that comes from tracing extra code.

- HAVE A DEBUGGER INSTALLED

  It is a good idea to have a debugger installed when running Big Brother. Certain error conditions do not lend themselves to fancy error reporting. (e.g. when the parasite finds an error while running in the background). In these cases, I settled for a quick drop into Macsbug with an error message. Usually, you will be prompted to press “g” to continue. The next version of Big Brother should have better error reporting in these conditions.

- DON’T TRY TO RUN WITH A CORRUPTED SYSTEM HEAP

  Big Brother runs at a very low level. Therefore, Big Brother bugs and incompatibilities can corrupt the system heap as a side effect. If your machine crashes while Big Brother is running, be sure to check for corruption of the System Heap (use HX;HC in Macsbug). If the heap is corrupted you should reboot ASAP. You might be tempted to just exit the current application and continue working without rebooting, but you probably will not get far. In fact, it is often a good idea occasionally to break into Macsbug during long Big Brother runs to see if everything is still ok.

- LOW PERCENTAGES DON’T ALWAYS MEAN POOR COVERAGE

  There are certain situations in which the reported coverage percentages will be lower than you might first expect:

  - The MPW Linker puts a lot of extra code in the “Main” segment, and Big Brother will include that code when figuring coverage percentages.

- Sometimes, compilers store data inline. These memory locations are part of the code, but they are never executed. As a result, Big Brother will report lower execution percentages. For example, switch statements with many case labels cause the MPW C compiler to generate a table of addresses of case labels. The case selector is used to look up an address in that table. The table itself (which can be arbitrarily large) is never executed, but it is part of the code.

- COVERAGE PERCENTAGES ARE NOT A SUFFICIENT MEASURE OF TESTING ADEQUACY

  It is important to interpret Big Brother reports correctly. High percentages do not necessarily indicate sufficient testing, and low percentages do not necessarily indicate insufficient testing. Big Brother reports what code HAS been executed. It is up to you to determine what code SHOULD have been executed.

- BIG BROTHER IS NOT REENTRANT

  This means that Big Brother should not be used to trace interrupt routines, VBL tasks, multiple processes or Big Brother itself.

- TO TARGET CODE IN THE MPW SHELL (E.G. PROJECTOR) QUIT THE SHELL, AND REENTER.

After you run BigBrother -i, Big Brother waits for a GetResource call for the CODE resources you have targeted. (e.g. the Shell calls GetResource on tool CODE resources when invoking tools). This means that if you target code in the MPW Shell (such as the Projector code), YOU MUST QUIT THE SHELL and re-enter so that Big Brother gets its GetResource call. (or, write a tool which calls GetResource for each CODE resource in the MPW shell.)

- WHEN USING -AA BE SURE YOUR TARGET WON'T MOVE OUT FROM UNDER YOU.

  Memory resident copies of Macintosh resources can move. If you target some code that is sitting in memory and then the code moves, Big Brother will continue blindly to target the obsolete locations. Any data you get back would be suspect.

  Even if the code doesn't move before you call "BigBrother -r" it will likely have moved by the time you invoke BBReporter, making the generated report meaningless. In other words, -aa is really useful only for things that don't move ever, like ROM routines.

  One final note: if you target the ROM addresses of a routine that has been patched, you will get 0% coverage, because the targeted locations will never be executed.

- OUTPUT FILES CONTAIN ABSOLUTE REFERENCES TO THE TARGETED CODE, SO IF YOU MOVE THINGS, BBREPORTER WILL COMPLAIN AND FAIL.

  The target files name, volume and directory (among other things) are stored in the output file created by BigBrother -r. If you move that output file to another machine, rename the targeted file, or move directories around, those references will no longer be valid and BBReporter will get confused.

- ALWAYS USE THE -TRAP OPTION WHEN TARGETING “PACK” RESOURCES.

  PACK resources (like the Standard File Package) are collections of routines accessed through a single A-Trap. If you have not specified -trap, coverage will be turned off as soon as the PACKs code begins executing. This is exactly the opposite of the desired behavior.

Command Reference Pages

Command Reference Pages

BigBrother - Install/Remove Big Brother Parasite

ßyntax

BigBrother [-p] [-i [nstall] [*install options*]] | [-r [emove] [*remove options*]]

install options: [-trap] -f *pathname* [-rt *type=id*] | -sys [-rt *type=id*] | -aa *start-stop*

remove options: [-o]

Description

BigBrother installs or removes the BigBrother code coverage parasite depending on which of the **-i** or **-r** options is specified. If neither **-i** or **-r** is specified, then BigBrother returns version and build date information. Once the parasite has been installed, coverage tracking begins.

BigBrother can target three types of program code: resources in a resource file (using **-f** and **-rt**), resources in the system file (using **-sys** and **-rt**), and absolute address ranges in memory (using **-aa**).

Options

-p	This option will display messages that reflect the progress of the Big Brother tool.
-install	This option tells BigBrother to install the parasite and begin code coverage on the targets specified in the remainder of the command line (using -f, -sys or -aa). -i and -r are mutually exclusive.
-remove	This option tells BigBrother to stop code coverage, remove the parasite, and write the collected data to disk.
-f *pathname*	This option specifies a file as the targeted resource file. This file contains the resources specified by the **-rt** option. BigBrother will quit with an error if this file does not exist or does not have a resource fork. BigBrother will quit with an error if this option is used in conjunction with another **-f** option or the **-sys** option.
-sys	This option specifies the current system file as the targeted resource file. The system file contains the resources specified by the **-rt** option. BigBrother will quit with an error if this option is used with the **-f** option.
-rt *type=id*	This option specifies the resource identified by the type and id as a targeted resource. BigBrother will quit with an error if this resource does not exist within the file identified by the **-f** or **-sys** options. This option is used once for each targeted resource. If the **-rt** option is not used and the **-f** or the **-sys** option are used, then all of the resources of type 'CODE' within the targeted resource file become targeted resources.

-aa *start -stop*	This option specifies the given range of addresses as a coverage target. This option can be used any number of times. The given addresses are required to be in hexadecimal and can be from 1 to 8 hex-digits. BigBrother will quit with an error if the starting address of a range is not less than the stopping address or two different ranges overlap.
-trap	If this option is specified, BigBrother will trace the execution of all a-traps. This is useful if the code targeted for coverage analysis will be executed during an a-trap execution. By default Big Brother does not trace the a-trap calls. Big Brother has a much greater impact on speed when a-traps are traced.
-o *pathname*	This option specifies the name of the coverage data file. **Note: If the file already exists, it is deleted and a new file is created.** By default, Big Brother names its output file “BigBrother.out".

Notes

1) When installing, you must specify exactly one of -f, -sys or -aa.

2) You may specify multiple -rt options.

3) Invoking BigBrother with no arguments returns version and build information.

BBReporter - Generate Coverage Report

Syntax

BBReporter *pathname*... [-p] [-d] [-m] > reportFile

Description

BBReporter displays the report files created during coverage analysis in a useful format. All output from BBReporter goes to standard output.

When BBReporter is called for one coverage report file percentages corresponding to statement coverage and branch coverage are displayed for each resource and address range targeted in that report file.

Following these percentages, a coverage percentage for the combined resources and address ranges is displayed.

If BBReporter is called for more than one coverage report file and the coverage reports target the same resource file, the generated report shows the cumulative coverage represented by all of the coverage report files. BBReporter quits with an error if the coverage reports do not target the same resource files.

Options

-d This option will cause BBReporter to disassemble the coverage targets and display the disassembled listings with the "•" character next to instructions that were executed.

```
…
…
+00000000 •:    48E7 7FF8        'H...'    MOVEM.L     D1-D7/A0-A4,-(A7)   +00000004 •:   49FA 01A6
        'I...'              LEA           *+$01A8,A4     ; 000001AC      +00000008 •:        3014
        '0.'                MOVE.W              (A4),D0
+0000000A •:    5340             'S@'         SUBQ.W      #$1,D0
```

```
+0000000C •:    6704          'g.'     BEQ.S     *+$0006 ; 00000012   +0000000E :   70FF
                'p.'          MOVEQ    #$FF,D0
+00000010 :     6036          '`6'     BRA.S     *+$0038 ; 00000048   +00000012 •:  264D
                '&M'          MOVEA.L  A5,A3
+00000014 •:    97EC 0004     '....'   SUBA.L    $0004(A4),A3
+00000018 •:    2F0B          '/.'     MOVE.L    A3,-(A7)
+0000001A •:    2F2C 0004     '/,..'   MOVE.L    $0004(A4),-(A7)
…
…
```


Copyright Apple Computer, Inc.
1990-1991. All rights reserved.

-m	This option will mark Macsbug symbols in the code as executed, and include Macsbug symbol bytes as executed in the coverage percentages.
-p	This option will display messages that reflect the progress of the BBReporter.

Notes

1) -m is only useful if there are Macsbug symbols embedded in the code.

Bibliography

68030 User's Manual [1989], Motorola, Inc.

Beizer, Boris [1984]. *Software System Testing and Quality Assurance*, Van Nostrand Reinhold Company, New York.

Hetzel, Bill [1988]. *The Complete Guide to Software Testing*, QED Information Sciences, Inc., Wellesley, MA.

McEntee, Kevin [1989], *Big Brother, A Code Coverage Tool For Macintosh: External Requirement Specification*, Apple Computer, Inc.

