# Taming the Lint Monster

## A personal perspective of the PC-Lint code analysis tool, and how to use it effectively

Anna-Jayne Metcalfe
Riverblade Limited
http://www.riverblade.co.uk

Riverblade

# An All Too Common Story


Riverblade

# What Can We Learn From This?

- Very few teams seem to use code analysis tools effectively
- There is a credibility gap - especially. among “the 80%”:
  - Perceptions such as “too hard to set up”, “too much noise” “bad news” or “telling us how to do our job” can be hard to shift
- Nevertheless, analysis tools **can** uncover real problems in a codebase
- …if you invest the time to learn how to use them *effectively*

# So What is PC-Lint, Anyway?

- C/C++ code analysis tool first introduced in 1985
- Available in both Windows (PC-Lint) and Unix/Linux (Flexelint) variants
- Command line only
- Extremely thorough and very flexible
- Can be difficult to set up correctly
- Analysis runs can be (very) slow on large projects

Riverblade

# Example Analysis Results

```
--- Module: CJFlatHeaderCtrl.cpp
}
CJFlatHeaderCtrl.cpp(160): error 1401:
      (Warning -- member 'CCJFlatHeaderCtrl::m_bSortAsc'
(line 146, file ..\Include\CJFlatHeaderCtrl.h) not initialized by constructor)
}
CJFlatHeaderCtrl.cpp(166): error 1740:
     (Info -- pointer member'CCJFlatHeaderCtrl::m_pParentWnd'
(line 150, file ..\Include\CJFlatHeaderCtrl.h)
     not directly freed or zero'ed by destructor
 -- Effective C++ #6)
```

Riverblade

# Online PC-Lint Demonstrator

http://www.gimpel-online.com/OnlineTesting.html

*and*

http://www.gimpel-online.com/bugsLinkPage.html

Riverblade

# A Few PC-Lint Capabilities

- Detection of dangling/uninitialised pointers
- Variable initialisation/value tracking
- Variable scoping
- Type mismatches and dodgy casts
- Assignment operator/copy constructor checking
- Detection of potential memory leaks
- Analysis of thread behaviour (new to PC-Lint 9.0)
- MISRA C/C++ rule validation

Riverblade

# Other C/C++ Static Analysis Tools

- Splint (C only, but free)
- PreFAST
- QA C++
- Klockwork Insight
- Coverity
- Parasoft
- etc…

# Be Prepared!

- The first time you analyse a codebase, expect both difficulties **and** surprises
  - Configuration issues
  - Lots of unwelcome (at least initially) “noise”
  - Potentially bad news in places you really don’t want to look
- Management and co-workers may not want to know

# A (Very) Simple PC-Lint Command Line

```
lint-nt std.lnt filea.cpp
```

- Single file (“unit checkout”) analysis
- But what is “std.lnt”?

Riverblade

# The Global “Indirect file” std.lnt

- Conventionally holds the “global” PC-Lint configuration
  - High level configuration options
  - Global include folder specification
  - References to other indirect files (compiler options, warning policy etc.)
- Include folder specifications can be machine dependent
- Std.lnt does not usually define the full warning policy

# Warning Policy: options.lnt

- Defines specific messages to globally enable/disable
  - std.lnt actually defines the high level policy, but options.lnt adjusts it to your needs
- Usually comprised of just a list of **-e** directives

# Basic IDE Integration

- Conventionally via custom tools within the IDE
- Analysis results sent to the output window
- May be blocking, so analysis time can be an issue.
  - Especially in whole project analysis (more on that later)
- Detailed analysis configuration can be an issue
- Results are not usually persistent

# A (Not So) Simple PC-Lint Command Line

```
lint-nt.exe -iC:\Lint -background -b --u
SourceVersioner_vs71_Debug_Win32.lnt –u
std_vs71.lnt env-vc7.lnt -t4 +ffb
+linebuf +macrobuf
–iDebug Shared\FileUtils.cpp
```

Riverblade

# Coping with Project Configurations

- For analysis to work effectively, the PC-Lint configuration must match that of the compiler
- Any mismatches will lead to analysis errors
- C++ project configurations can be **very** complex
- PC-Lint can write suitable configuration (“project.lnt”) files for most Visual C++ projects directly
- For other platforms, you are (unfortunately) on your own

Riverblade

# What is in a project.lnt file?

- Basically a subset of the compiler configuration for lint purposes
  - Preprocessor definitions
  - Additional include folder paths
  - A list of files in the project, relative to the project folder

Riverblade

# Whole Project Analysis

```
lint-nt.exe -iC:\Lint -background -b
std_vs71.lnt env-vc7.lnt -t4 +ffb
+linebuf –iDebug
SourceVersioner_vs71_Debug_Win32.lnt
```

- Can identify functions, enums etc. which are not used in that project
- Single threaded, and can be slow on large projects

Riverblade

# PC-Lint Message Categories

- Five categories, of varying severity:
  - Elective Notes
  - Informational
  - Warnings
  - Errors
  - Fatal Errors
- Individual categories and messages can be selectively enabled via –w and +e/-e options

# Common Analysis Failures

- Fatal Error 314: Previously used .lnt file
- Fatal Error 307: Can't open indirect file
- Fatal Error 322/Error 7: Unable to open include file
- Error 91: Line exceeds Integer characters (use +linebuf)
- Error 303: String too long (try +macros)

Riverblade

# Analysis Speed

- Influenced by CPU/disk speed and project structure
  - Include dependencies can be **very** significant
- PC-Lint 9.0 adds precompiled and bypass headers
  - Can potentially cut analysis time by 3-4 times
- PC-Lint is currently single threaded
  - Adding more cores won't help unless you run multiple analysis tasks simultaneously
  - Single file analysis is amenable to parallelisation

Riverblade

# Some issues to look out for

- 429 (Custodial pointer not freed or returned)
- 578 (Declaration of symbol hides another)
- 716 (while(1))
- 717 (do…while(0))
- 777 (Testing floats for equality)
- 795 (Conceivable division by zero)

Riverblade

# Some issues to look out for (cont.)

- 801 (Use of goto is deprecated)
- 825 (Control flows into case/default)
- 1506 (Call to virtual function in constructor or destructor)
- 1725 (Class member is a reference)
- 1735 (Virtual function has default parameter)
- 1773 (Attempt to cast away const or volatile)

# Tuning Out Issues in Libraries

- Issues in library header files can cause “noise” elseware in a project
- These can be dealt with in several ways:
  - Reduce the warning level while including library headers
  - Modify the library to fix it or add lint directives
  - Create an indirect file containing “tuning” directives (e.g. `-etype(1746, boost::shared_ptr<*>)`

# Turning Down the Volume

- How can I cope with this deluge of analysis results?
  - (the “noise” issue again)
- Define your initial warning policy carefully
  - Either start with a reasonably relaxed warning policy and gradually make it more aggressive, *or:*
  - Start with an aggressive policy and carefully analyse the results to determine which ones you don’t care about

Riverblade

# Tools/Techniques Which May Help

- Aloa
- LintProject
- Grep
- XSLT transformations highlighting issues you care about (and potentially ignoring the ones you don't)
- SourceMonitor (or other complexity measuring tools)
  - If you code is "noisy", you probably have architectural issues too

# Summary

- Analysis tools such as PC-Lint can uncover real problems in your codebase
- There is no “Quick Fix” for poor code quality
- Be prepared to invest significant time (at least at first) in:
  - Configuring the tool to work well with your codebase
  - Developing your warning policy
  - Interpreting analysis results
- Consider also analysing complexity and design

Riverblade

# Any (more) questions?

# Taming the Lint Monster

## A personal perspective of the PC-Lint code analysis tool, and how to use it effectively

Anna-Jayne Metcalfe
Riverblade Limited
http://www.riverblade.co.uk

Riverblade