2000-

"Software quality and the world automobile industry"

by

Les Hatton

Oakwood Computing, Surrey, U.K. and
the Computing Laboratory, University of Kent
lesh@oakcomp.co.uk

Version 1.1: 24/Mar/2000

Overview

- **<u>Overview</u>**
- **MISRA-C**
- **Future trends**

Trends

Recent trends in the automobile industry include:-

- Very rapidly growing software deployment
- Software deployment in critical areas
- Use of floating point arithmetic
- The use of C as a standard replacement language for assembler
- Recognition of the need for safer language subsets
- Very high cost of failure

Rapidly growing software deployment

It is widely recognised that consumer embedded software systems have been doubling in size every 18 months.

- Cars have gone from around 50,000 lines of assembler to around 250,000 lines of C in around 5 years, a faster rate of growth than the average.

Use in critical areas

As well as 'cosmetic' areas like memory seats and in-car entertainment, software is now widely deployed in critical areas such as:-

- Air-bags, where the complexity has increased by about a factor of 10 in 3 years to address multiple airbags, side as well as front impact, risk to small passengers and other issues.
- Braking systems
- Engine management systems
- Accelerator and other pedal control
- Steering

Use of floating point arithmetic

Driving forces:-

- The demands of modern engine management and emission control and other issues such as navigation require very sophisticated algorithms
- The wide availability of micro-processors with embedded and highly efficient floating point arithmetic

The use of C as a standard language

Driving forces:-

- Need for a high-level language
- Wide availability of compilers for embedded micro-processors
- The most efficient high-level language of all in terms of both space and performance, a critical factor when shipped systems are numbered in the millions.
- Internationally standardised as C90 and now C99 and capable of validation to this standard

Recognition of the need for subsetting

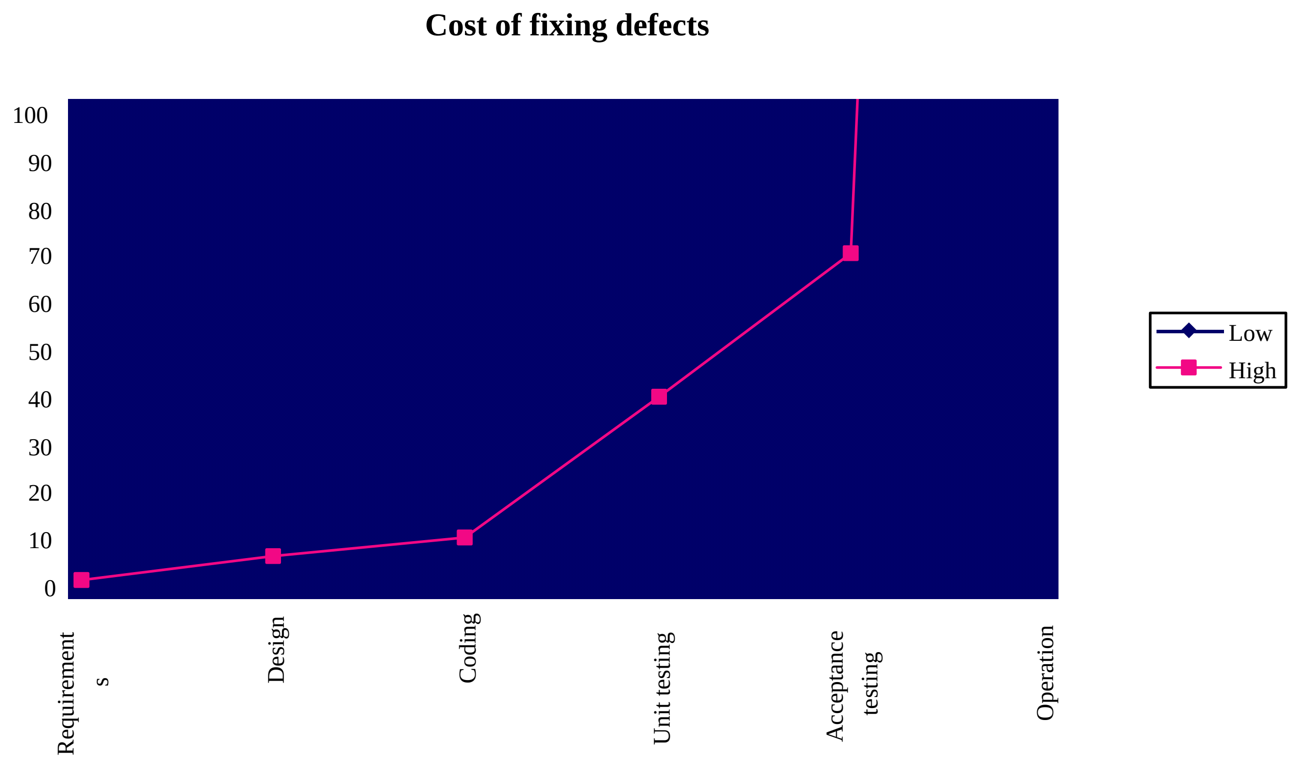
Driving forces:-

- The appearance of C in critical systems
- The cost of failure
- Established published work on the need for subsetting in critical systems which helped to form the basis for the very widely known standard MISRA-C

High cost of failure

- 22/July/1999. General Motors has to recall 3.5 million vehicles because of a software defect. Stopping distances were <u>extended</u> by 15-20 metres.
- Federal investigators received almost 11,000 complaints as well reports of 2,111 crashes and 293 injuries.
- Recall costs ? (An exercise for the reader).

High cost of failure



Embedded systems tend to follow the high curve.
Data from Boehm, (1981) and many others.
Note that curve kicks only around coding stage.

Overview

- Overview
- **<u>MISRA-C</u>**
- Future trends

Some example C standards

- **MISRA™ (April 1998)**
- **NUREG CR-6463 (1996)**

MISRA-C is a trademark of the Motor Industry Research Association

MISRA - a high-quality C standard

- **In April 1998, the Motor Industry Software Research Association (MISRA) published a set of C guidelines for use in vehicle-based software.**
  - 93 rules + 34 guidelines
  - Consistent with development to SIL3
  - Highly enforceable
  - Publicly available
  - Based on reference works such as Koenig (1989) and Hatton (1995)

MISRA - a high-quality C standard

Category	**Rules**	**Guidelines**
Environment	1	3
Character Sets	4	0
Comments	1	1
Identifiers	1	1
Types	3	2
Constants	1	1
Declarations and Definitions	6	4
Initialisation	3	0
Operators	7	3

MISRA - a high-quality C standard

Category	Rules	Guidelines
Conversions	2	1
Expressions	2	4
Control Flow	11	5
Functions	15	4
Pre-processing directives	10	3
Pointers and arrays	5	2
Structures and Unions	6	0
Standard Libraries	14	0

MISRA - a high-quality C standard

- **Around 5-10% NOT automatically enforceable**
  - (Example: Rule 99, All uses of the #pragma directive shall be documented and explained)
- **The standard is cross-referenced against the ISO C 9899 standard for traceability**

MISRA - a high-quality C standard

- **Around 5-10% NOT automatically enforceable**
  - (Example: Rule 99, All uses of the #pragma directive shall be documented and explained)
- **The standard is cross-referenced against the ISO C 9899 standard for traceability**
- **Rule 1 of MISRA C requires ISO C 9899 conformance so any supporting tool should also be checked against FIPS 160, (Official C validation suite)**

MISRA - a high-quality C standard

- **About 5% of the rules are not correct or are redundant as they are already within ISO C 9899**
- **Some of the rules are not statically enforceable. For example, Rule 4 states that there should be provision for run-time checking**
- **It is consistent with C90 but now needs upgrading for C99**

MISRA acceptance

- **MISRA is gaining rapid industry acceptance**
  - It was developed by a consortium of vendors including Ford, Lucas and Rover (now BMW)
  - It is the only standard of its kind in the world
  - It promotes provably good practice
  - It is probably close to achieving ‘critical mass’
  - It is strongly supported by MIRA, (Motor Industry Research Association)

MISRA tool support

- **The standard now has tool support with a number of manufacturers providing checking tools, including**
  - Assent, which only checks for MISRA
  - QAC ™, a C static checker which has a MISRA mode as an optional extra
  - The Safer C ™ Toolset, which includes a MISRA checking mode as standard but also contains a complete MISRA compliance suite and a reference section for engineers.

NUREG CR-6463

- Sponsored by the US Nuclear Regulatory Commission
- Guidelines for Ada, C/C++, PLC Ladder Logic, IEC 1131 sequential function charts, Pascal, PL/M
- C discussed with C++ pages 4-1 to 4-64
- Written in the form of an essay with examples so quite difficult to enforce.
- Rules and guidelines not clearly distinguished.

Useful links

- **On MISRA:-**
  - http://www.misra.org.uk/
  - http://www.oakcomp.co.uk/, (MISRA compliance validation)

Overview

- **Overview**
- **MISRA-C**
- **<u>Future trends</u>**

Possible future directions

The following have been discussed in general embedded systems work:-

- Higher-level design systems generating C
- Use of Java
- Use of C++, (and EC++)
- Use of C99

Higher level design systems

- **Advantages**
  - Closer to the design process
- **Disadvantages**
  - Code generation is not very good
  - There is a tendency to modify the generated code, making things worse not better

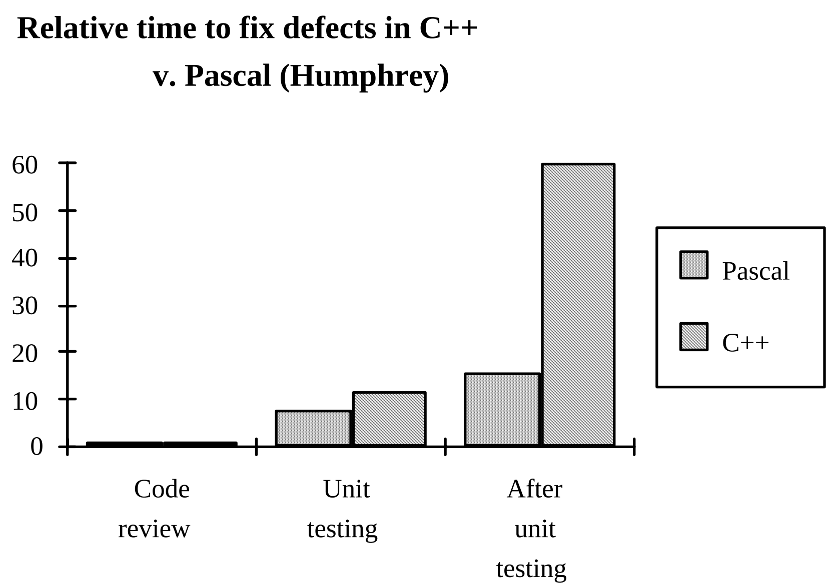
Use of Java

- **Advantages**
  - Simple and fashionable
  - Allows use of OO directly within language
  - Trys to control some of the worst features of C and C++
- **Disadvantages**
  - Inherently very inefficient compared with either C or C++ even when compiled
  - New failure modes as yet unknown
  - Not internationally standardised so its use is a risk in critical systems

Use of C++

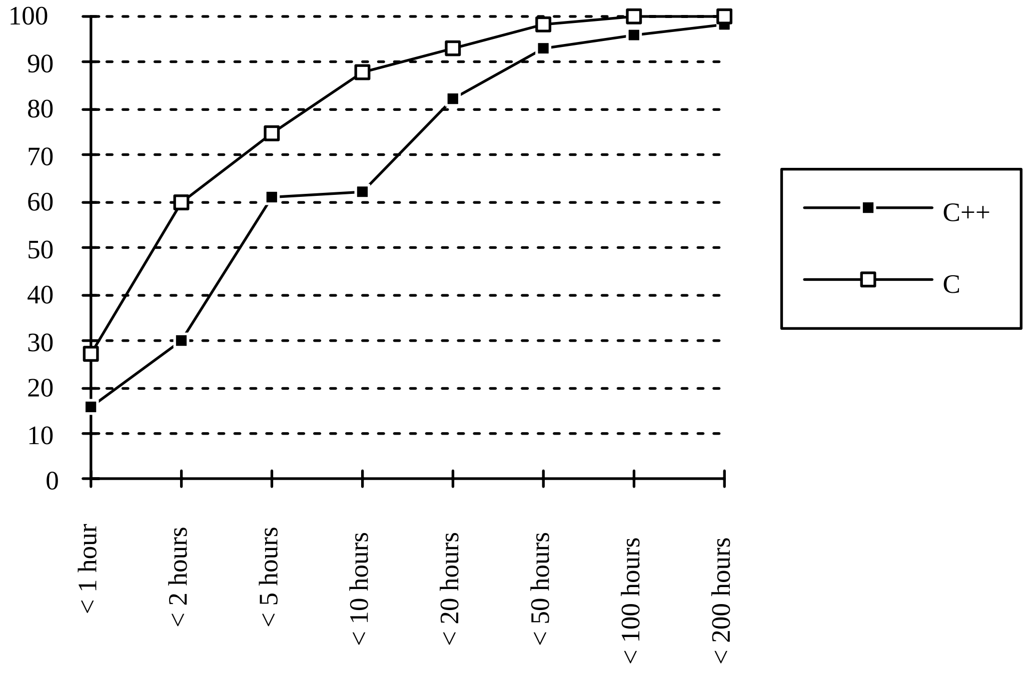
- **Advantages**
  - Allows object orientation to be directly used within the language
  - EC++ subset exists
- **Disadvantages**
  - Inefficient compared with compiled C both in terms of space and performance
  - Failure modes as yet unknown
  - OO systems in C++ have some disturbing characteristics
  - Very large amount of undefined behaviour in ISO C++99, (the word ‘undefined’ appears 1825 times for example)

Measurement feedback on object-orientation



This data is due to Humphrey, (1995)

Measurement feedback on object-orientation



This data is due to Hatton, (1998)

Measurement feedback on object-orientation

Summary of known measurements

- C++ OO systems have comparable defect densities to conventional C or Pascal systems
- Each defect in a C++ OO system takes about *twice as long to fix as a conventional system*. This is true for both simple defects AND difficult ones. The whole distribution is right shifted
- Components using inheritance have been observed to have 6 times the defect density

How much of this is attributable to C++ is unknown.

Use of C99

- **Advantages**
  - C90 no longer officially exists
  - The C committee now has a special group targetted at standardising C extensions for embedded systems
- **Disadvantages**
  - Twice as many undefined and unspecified items in C99 (366) as with C90 (197)
  - New failure modes still unknown

Which direction ?

- **Summary:-**
  - As of ISO/IEC JTC 1/SC22 meeting, 24-27 Aug, 1998.
    - “recognising increasing divergence of C and C++ user communities, WG14 (C) and WG21 (C++) no longer have to remain ‘compatible’ although are urged to cooperate where possible”.
    - C90 -> C99 is being targetted on embedded systems and C++ on general OO systems
  - OO systems in C++ are fine unless you make a mistake and then it is more expensive to fix
  - Java seems doomed to remain inefficient and its arithmetic is highly criticised by Kahan and others.
- **C90 -> C99 seems to be the dominant trend**

Conclusions

- The auto industry will continue to use software in growing quantities with a million lines in a car likely in the next 3-5 years putting very big demands on software quality
- The demand for more sophisticated algorithms will lead to much greater use of floating point arithmetic
- Most systems will be continue to be produced in C although with a greater percentage automatically generated by tools
- Networking both in cars and amongst cars will grow dramatically
- The cost of failure will remain very high

Bibliography

- Bach, R. (1997) “Test automation snake oil”, 14th annual conference on Testing Computer Software, Washington, USA
- Beizer, B. (1990). Software Testing Techniques. Van Nostrand Reinhold.
- Brettschneider, (1989) “Is your software ready for release ?”, IEEE Software, July, p. 100-108
- Fagan, M.E. (1976) “Design and code inspections to reduce errors in program development”, IBM Systems Journal, 15(3), p. 182-211.
- Fenton, N. E. (1991). Software Metrics: A Rigorous Approach. Chapman and Hall.
- Genuchten, M. v. (1991). Towards a Software Factory. Eindhoven.
- Gilb, T. & Graham D. (1993) Software Inspection, Addison-Wesley
- Grady, R. B. and D. L. Caswell (1987). Software Metrics: Establishing a Company-Wide Program. Englewood Cliffs, N.J., Prentice-Hall.
- Graham, D. (1995) “A software inspection (failure) story”, EuroStar’95, London, November
- Hatton, L. et. al. (1988). “SKS: an exercise in large-scale Fortran portability”, Software Practice and Experience.
- Hatton, L. (1995) “Safer C: Developing for High-Integrity and Safety-Critical Systems. McGraw-Hill, ISBN 0-07-707640-0.
- Hatton, L. (1997) Re-examining the fault density - component size connection, IEEE Software, March-April 1997.
- Hatton, L. (1997) The T experiments: errors in scientific software, IEEE Computational Science & Engineering, vol 4, 2
- Hatton, L. (1998) Does OO sync with the way we think ?, IEEE Software, May/June 1997
- Hatton, L. (2000) “Software failure: avoiding the avoidable and living with the rest”, Addison-Wesley, to appear in 2000.
- Humphreys, W. (1995) “A discipline of software engineering”, Addison-Wesley, ISBN 0-201-54610-8

Bibliography

- IEC 61508 (1991). Software for computers in the application of industrial safety-related systems. International Electrotechnical Commission: Drafts only - cannot yet be referenced.
- Kahan, W., Darcy, J.D.(1998) “How Java’s Floating-Point Hurts Everyone Everywhere”, ACM 1998 workshop on Java, Stanford California
- Knight, J. C., A. G. Cass, et al. (1994). Testing a safety-critical application. International Symposium on Software Testing and Analysis (ISSTA'94), Seattle, ACM.
- Kolawa, A. (1999) “Mutation Testing: a new approach to automatic error detection”, StarEast ‘99, Orlando, May 1999
- Liedtke, C, and Ebert, H. (1995), “On the benefits of reinforcing code inspection activities”, EuroStar’95, London
- Leveson, N. (1995). “Safeware: System Safety and Computers.” Addison-Wesley, ISBN 0-201-11972-2.
- Littlewood, B. and L. Strigini (1992). “Validation of Ultra-High Dependability for Software-based Systems.” Comm ACM to be published:
- McCabe, T. A. (1976). “A complexity measure.” IEEE Trans Soft. Eng. SE-2(4): 308-320.
- Mills, H.D. (1972) “On the statistical validation of computer programs”, IBM Federal Systems Division. Gaithersburg, MD, Red. 72-6015, 1972
- Myers, G. J. (1979). The Art of Software Testing. New York, John Wiley & Sons.
- Nejmeh, B. A. (1988). “NPATH: A measure of execution path complexity and its applications.” Comm ACM 31(2): 188-200.
- Parnas, D. L., J. v. Schouwen, et al. (1990). “Evaluation of Safety-Critical Software.” Comm ACM 33(6): 636-648.

Bibliography

- Pfleeger, S and Hatton L. (1997) “How well do Formal Methods work ?”, IEEE Computer, Ian 1997.
- Pfleeger, S. (1998) “Measurement and testing: doing more with less”, ICTCS’98, Washington.
- Porter, A.A., Siy, H.P., Toman, C.A., Votta, L.G. (1997) “An experiment to assess the cost-benefits of code inspections in large scale software development”, IEEE Transactions, 23(6), p. 329-345
- Roper, M. (1999) “Problems, Pitfalls and Prospects for OO Code Review”, EuroStar’ 99, Barcelona, November
- Veevers, A. and A. C. Marshall (1994). “A relationship between software coverage metrics and reliability.” Software Testing, Verification and Reliability 4(1): 3-8.
- Vinter, O. and Poulsen, P-M (1996) “Improving the software process and test efficiency”, ESSI Project 10438, http://www.esi.es/ESSI/Reports/All/10438
- Warnier, J. D. (1974). Precis de logique informatique: les procedures de traitement et leurs donnees. H.E. Stenfert Kroesse.
- Woodward, M. R., D. Hedley, et al. (1980). “Experience with path analysis and testing of programs.” IEEE Transactions 6(3): 278-286.

