

European Software Testing Congress, May 2000

"Embedded software testing"

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

©Copyright, L.Hatton, 2000-

Overview

- v **Overview of embedded systems**
- v **Trends in embedded systems**
- v **Static Fault modes**
- v **Dynamic failure modes**
- v **Embedded systems, finding the balance**



Testing embedded systems in practice

Embedded systems:

The principle test barriers are (in no particular order):-

- u Communicating with the running system on an embedded target
- u Lack of space
- u Lack of reasonable run-time library facilities in compiler
- u Lack of coverage (profiling) information
- u Difficulty of setting up and automating tests
- u Length of time to prepare a test run
- u Setting up the hardware inputs
- u Separating out application defects, compiler defects and hardware defects



Testing embedded systems in practice

Embedded systems:

Run-time analysis can be very difficult for embedded systems. There are basically six options:-

- u Remote debuggers
- u Emulators
- u Simulators

and as last resorts:-

- u Turning LEDs on and off. Too much of this may lead to a short stay in rehabilitation or possibly a change of career.
- u Logic analysers
- u Oscilloscopes



Testing embedded systems in practice

Embedded systems:

Remote debuggers

One of the best examples is the GNU compiler and debugger. This consists of two pieces of software which communicate down a communications link between the host and the target.

- u The frontend. This runs on the host computer with a reasonable human interface at the source code level.
- u The debug monitor. This sits on the target and provides low-level control of the target.

Problem:

- u You have to write your own GDB debug monitor although there is a sample source code file.



Testing embedded systems in practice

Embedded systems:

Emulators

These are an embedded system themselves which emulates the target processor. They are also known as ICE (In-Circuit Emulators). They communicate with a frontend just like the remote debugger (and often the same frontend). They can monitor and control the state of the processor in real-time allowing such things as interrupts to be raised.

Problem

They tend to be expensive but are often the only solution.



Testing embedded systems in practice

Embedded systems:

Simulators

This is a complete host-based program which emulates the entire state of the target whilst running on the host.

Problem

They only emulate the processor and not any of the other peripherals which appear in modern embedded systems. They tend to be useful only in the early stages of a project, perhaps when the hardware is not yet available.



Overview

- v **Overview of embedded systems**
- v **Trends in embedded systems**
- v **Static Fault modes**
- v **Dynamic failure modes**
- v **Embedded systems, finding the balance**



Trends in embedded systems

Modern embedded systems are characterised by:-

- Exponentially increasing complexity
- Chaotic behaviour
- Higher coupling
- Testing difficulties
- Very high cost of failure



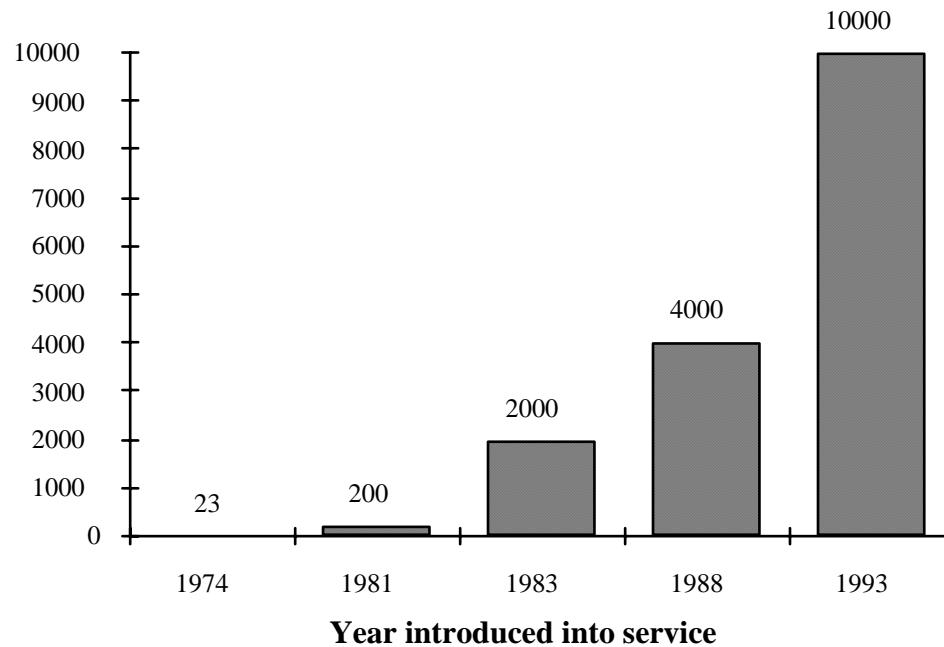
Exponentially increasing complexity

The amount of software in consumer electronic products is currently doubling about every 18 months.

- Line-scan TVs have ~250,000 lines of C.
- There are > 250,000 lines of C in a car. This occurs throughout the car with substantial amounts of critical code in air-bag control, braking systems and engine management.
- Network management systems have multi-million lines of code.
- Aircraft have between 1 and 10 million lines of code.



Growth of software in Airbus A3XXX



Chaotic behaviour

AT & T Jan, Jan 15, 1990:

- Single misplaced line of C in 3 million lines bypassed network error-recovery code
- For 9 hours, millions of long-distance callers just heard message “all circuits are busy”
- Reported \$1.1 billion loss



Anatomy of a \$1billion bug

```
...
switch( message )
{
case INCOMING_MESSAGE:
    if ( sending_switch == OUT_OF_SERVICE )
    {
        if ( ring_write_buffer == EMPTY )
            send_in_service_to_smm(3B);
        else
            break;    /* Whoops ! */
    }
    process_incoming_message(); /* skipped */
    break;
...
}
do_optional_database_work();
...

```



Higher coupling

Most modern systems are implemented as networks

- The Airbus A340 has more than 150 communicating computer systems



Testing difficulties

Complex, coupled embedded systems suffer from the following testing difficulties:-

- Building and executing tests efficiently, particularly with interrupt driven systems. (Modern portable scripting languages are reducing this problem)
- Diagnosing dynamic failure
- Achieving good test coverage



High cost of failure

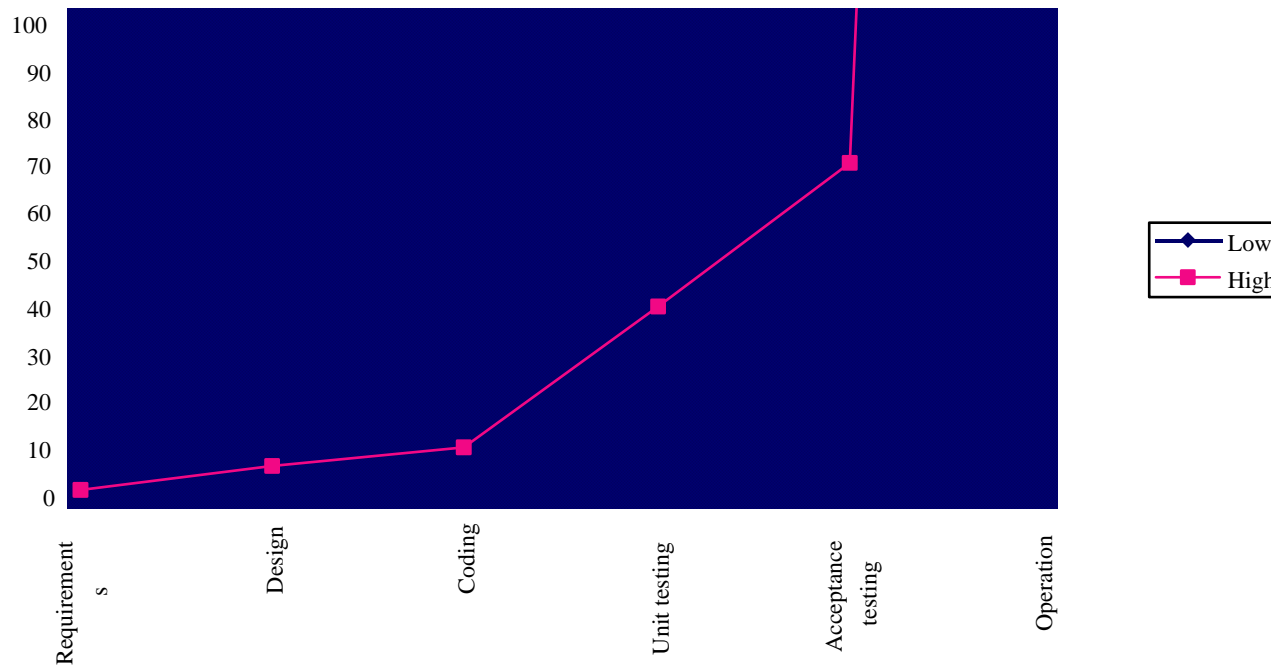
Cars too ...:

- 22/July/1999. General Motors has to recall 3.5 million vehicles because of a software defect. Stopping distances were extended 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

Cost of fixing defects



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



Future trends in embedded systems

The following trends can be identified

- Continued rapid growth
- More powerful processors
- Increasing use of floating point arithmetic
- Use of other languages such as Java although C continues to dominate mainly because of efficiency, (and C is now being targetted on embedded systems by the standards bodies and has recently been re-standardised as C99).



Overview

- v **Overview of embedded systems**
- v **Trends in embedded systems**
- v **Static Fault modes**
- v **Dynamic failure modes**
- v **Embedded systems, finding the balance**



Static fault modes

Statically detectable fault takes two fundamental forms:-

- Linguistic issues
 - Directly detectable fault
- Logic issues
 - These are addressed by inspections and indirect detection techniques.

Because the cost of failure is so high, embedded control systems require better inspection procedures than the average system.



Linguistic issues

- v **We will establish the following chain of reasoning:-**
 - Known fault modes exist in programming languages
 - They appear regularly in user's code
 - These faults fail with a certain frequency



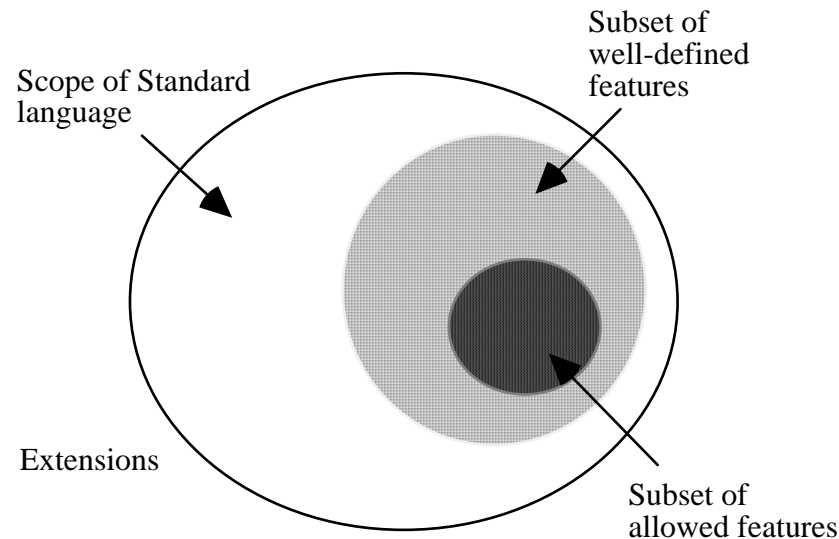
Sources of information

- ∇ **Sources of information on problematic behaviour in C come from two sources:-**
 - The committee's work, (formally identified problem areas). Approximately 300 items.
 - Experience in the world at large through news groups, comp.lang.c, the Obfuscated C competition and so on, (informally identified problem areas). Approximately 400 items.

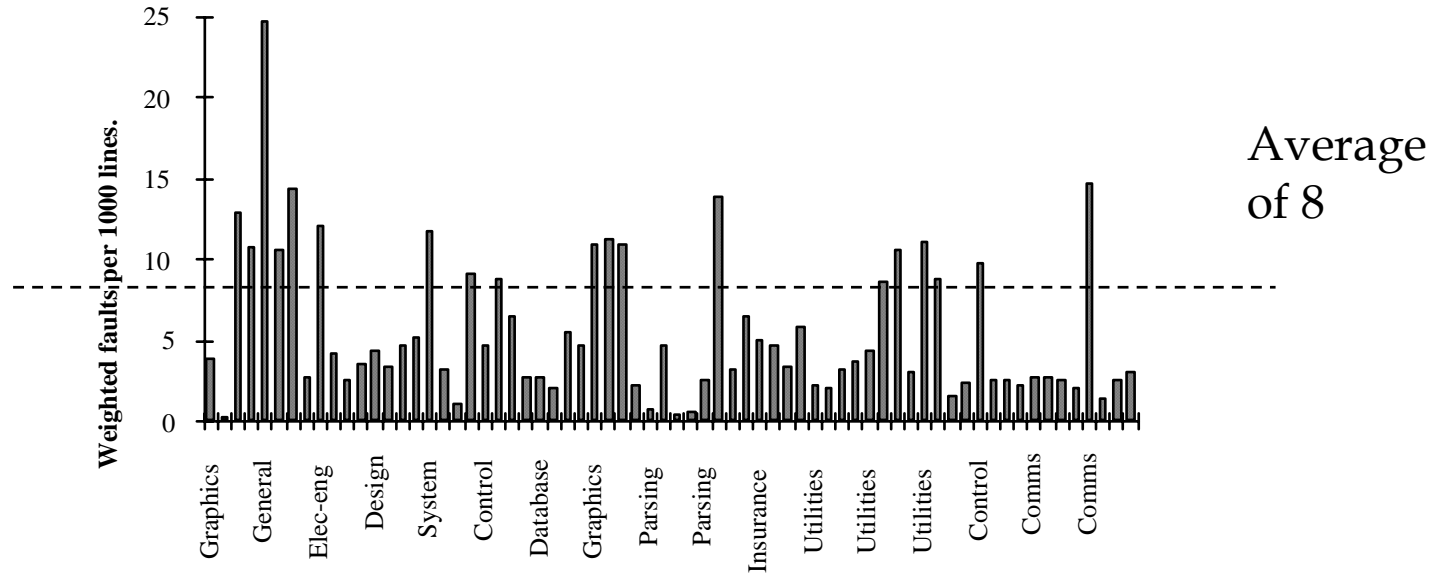


Problems with programming languages

The need for subsetting programming languages



Fault frequencies in C applications

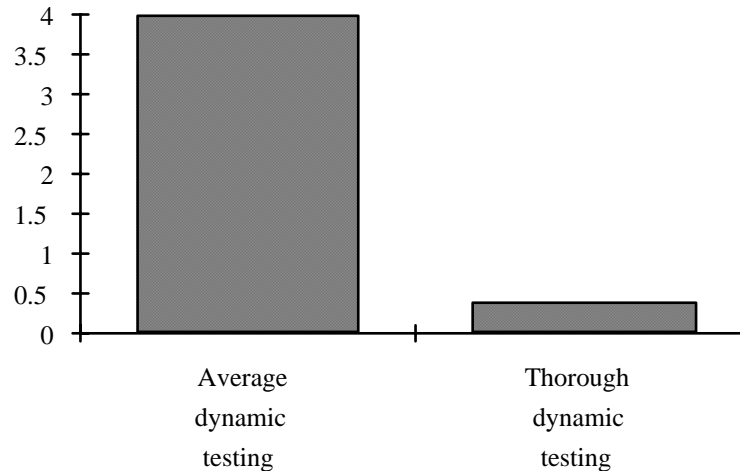


Data like this is extractable using tools such as the *Safer C Toolset*, (<http://www.oakcomp.co.uk>)



Where and how do defects occur historically ?

Data derived from CAA CDIS



This study shows that statically detectable faults do in fact fail during the life-cycle of the software.



Where and how do defects occur historically ?

Conclusions on safer subsetting:

- We can prove the following:
 - u There is a class of defect in programming languages which to a significant extent is statically detectable, widely reported and entirely avoidable
 - u This class of defect evades conventional testing to the extent of around 8 residual defects per 1000 lines of code
 - u A significant percentage of this class of defect fails during the life-cycle of the code but we are not able to predict which faults fail, so we must remove them all.
- Engineer education with tool support is crucial to the control of this class of defect.

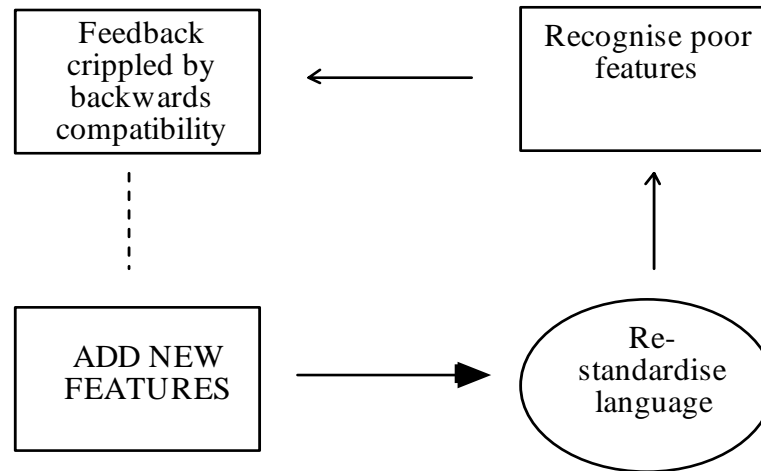


Do languages improve with time ?

- v **Things get worse with time. The following areas of C are problematic because the committee could not agree:**
 - At standardisation in 1990 (197 items)
 - At re-standardisation in 1999 (366 items)
- v **By comparison, C++99 contains the words:-**
 - Undefined, 1825 times
 - Unspecified, 1259 times.



Why languages can't improve



Using the model of control process feedback, we see that the feedback stage is crippled by the “shall not break old code” rule or “backwards compatibility” as it is more commonly known.



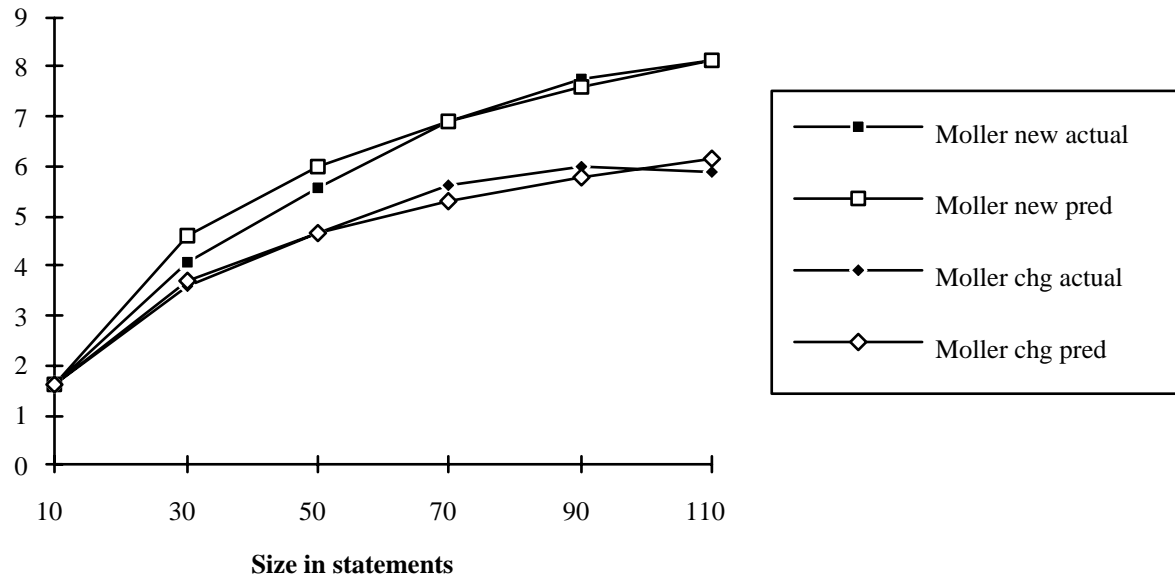
Logic issues

We have two possibilities:-

- Inspections. These are well documented and there is nothing particularly special about the procedures for embedded systems
- Indirect detection techniques. These exploit *defect clustering* and are less well-documented so we will describe them here



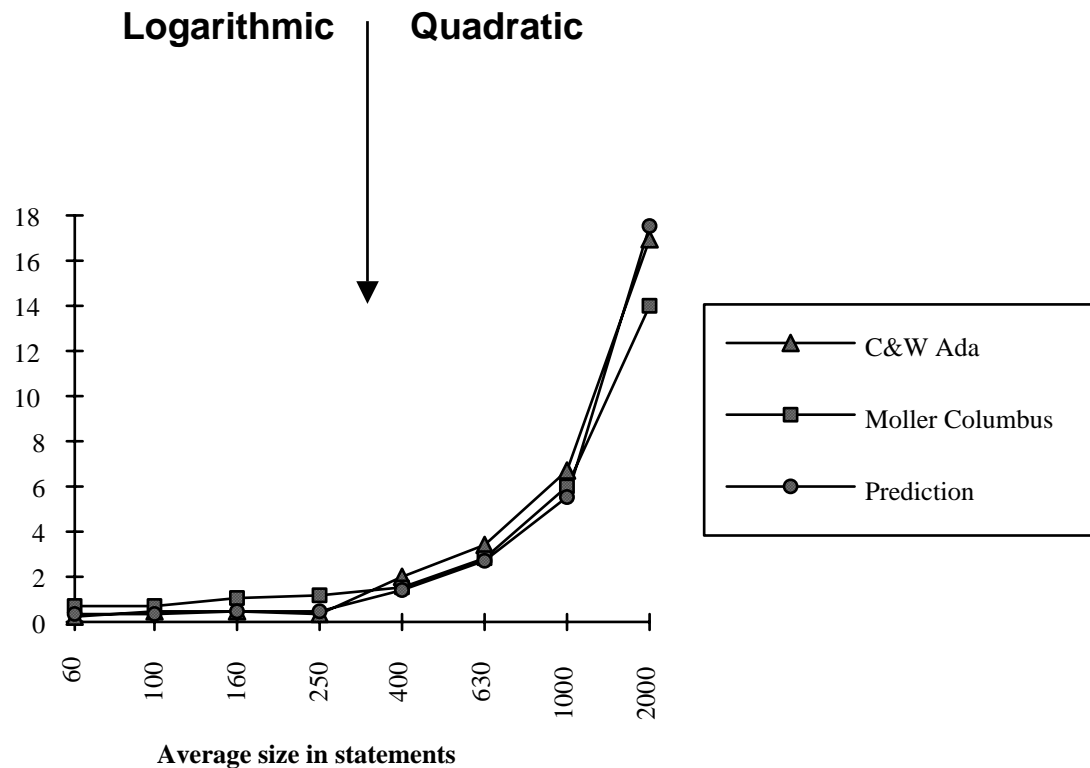
Failures and component size, (new and changed)



Data from an OS study at Siemens (1993)



What happens for big components ?

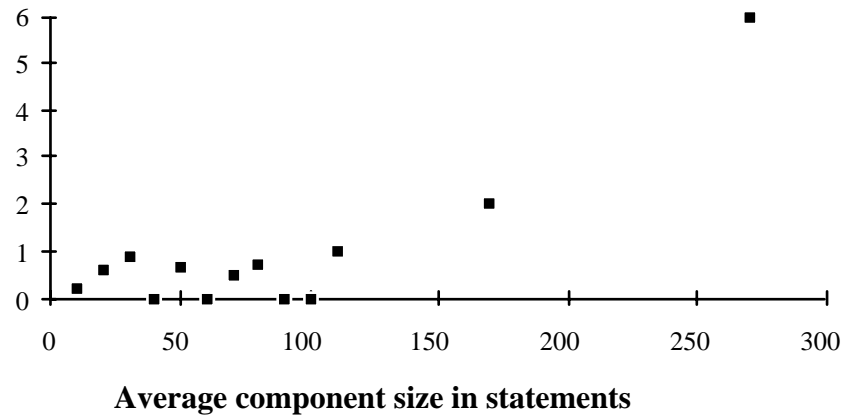


Model due to Hatton (1997)

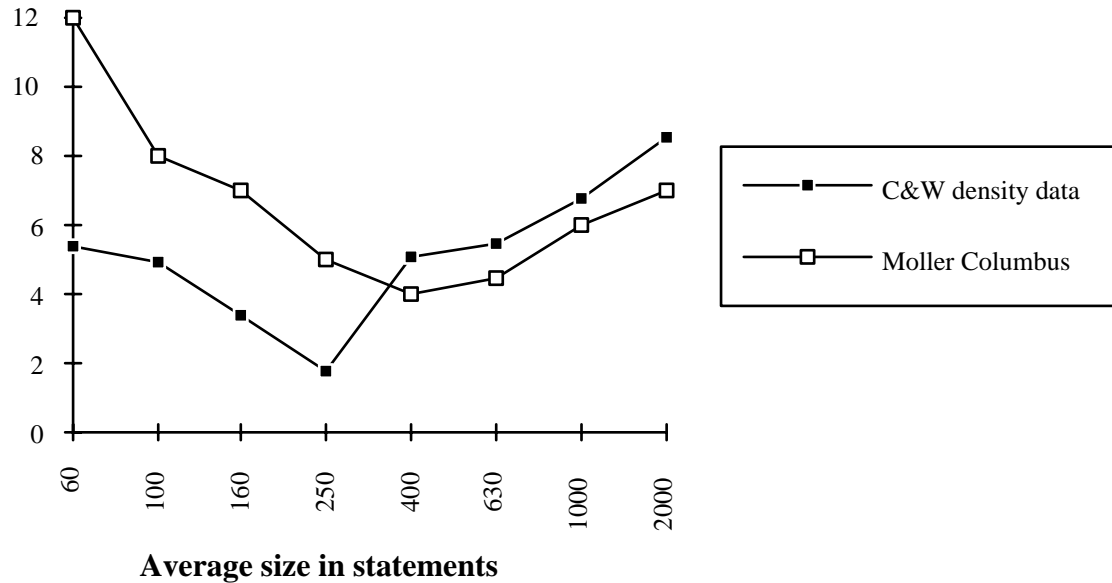


Another example

**Pascal Data supplied by Shepperd
(1995)**



Failure density and component size

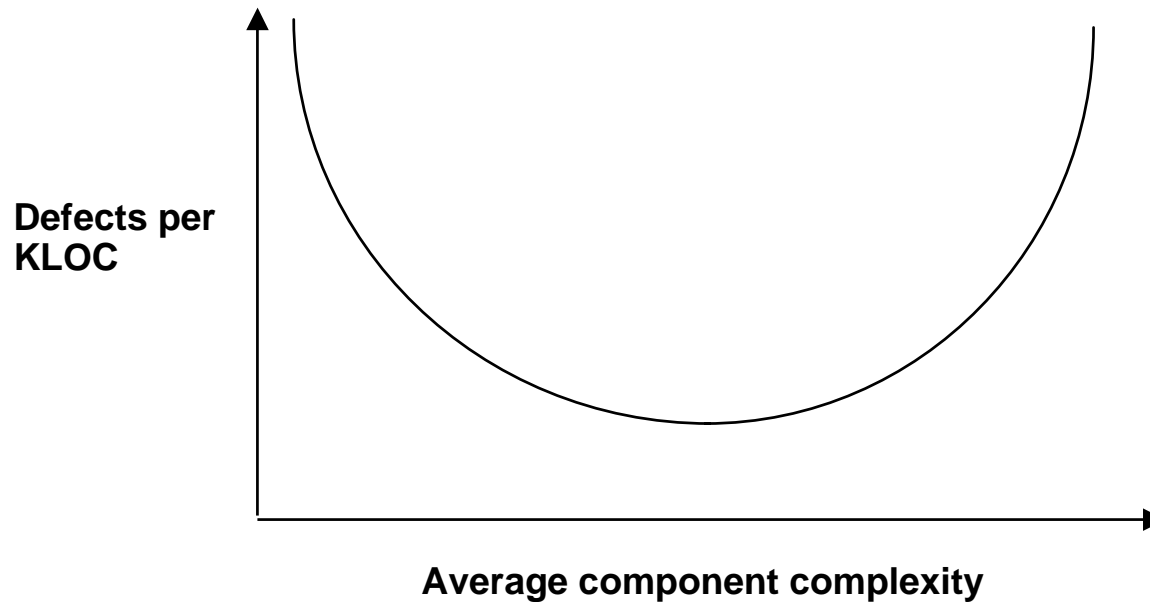


Comparison of Ada and assembler,
Hatton (1997)



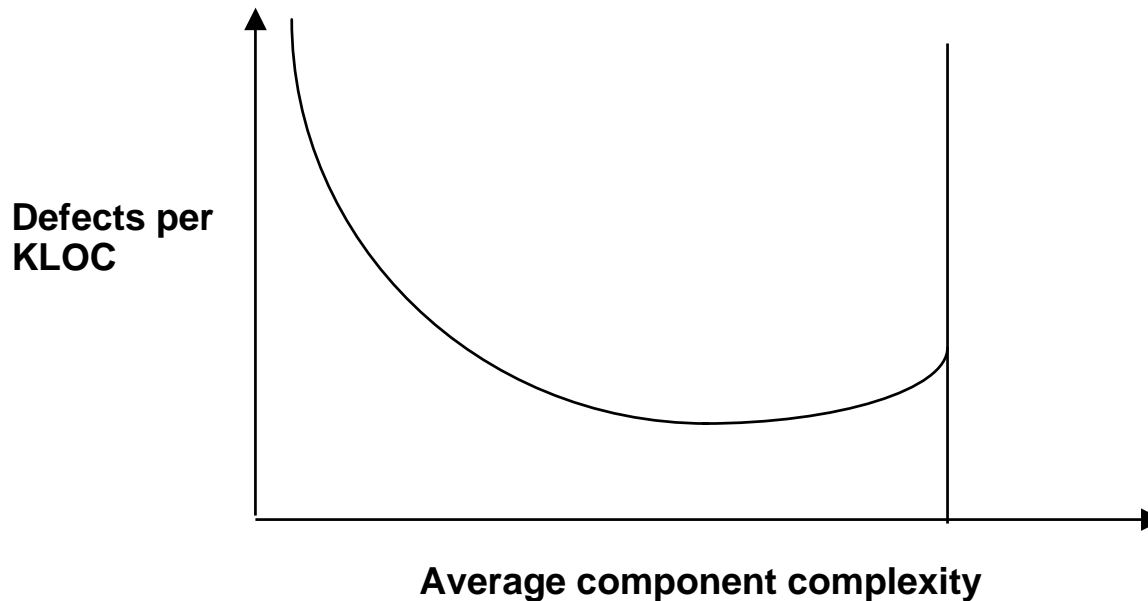
The defect density U curve

For Ada, various assembler, C, C++, Fortran, Pascal and PL/M systems:



The defect density U curve - invasive truncation

In those systems where excessive complexity has been restricted:-



The importance of test planning

By far the most effective way of truncating the top end and avoiding the production of untestable programs is to teach developers how to test.

This has the following effects:

- It very often simplifies the code significantly, (a factor of 3 has been observed)
- It facilitates important test targets like 100% statement coverage to be achieved, (for example, Heathrow air-traffic control system, Pfleeger & Hatton (1997))



The importance of test planning

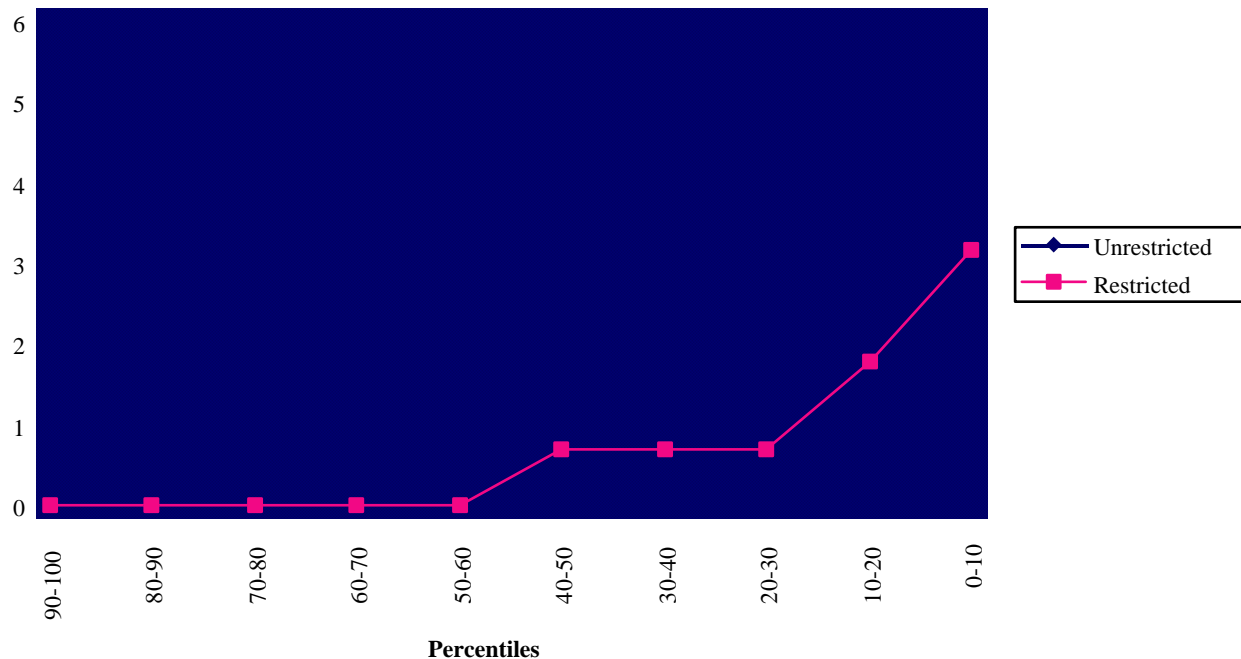
Notes:

- This does not undermine the existence of independent test groups. It simply makes the work easier for the independent test groups
- It encourages the developers to think about the possibility of failure which improves diagnostic procedures



Complexity limiting, Hatton (1995)

Path complexity distributions



The effects of test planning during development on system complexity is very clear here.



Overview

- v **Overview of embedded systems**
- v **Trends in embedded systems**
- v **Static Fault modes**
- v **Dynamic failure modes**
- v **Embedded systems, finding the balance**



Dynamic failure modes

Only around a half of all the static fault modes are detectable statically with tools.

If your other static techniques miss them, they must be found dynamically. This raises the connected problem of test coverage.



Coverage

Satisfactory dynamic testing requires a high coverage using at least one of the following criteria:-

- Function coverage
- Block coverage
- Exit point coverage
- Decision coverage in various forms
- Higher level coverage (but small steps first)

There is nothing new here other than this information is generally very difficult to get in an embedded system



Dynamic failure modes

We have two possibilities:-

- Failures caused by logic problems. In essence, the system does something unexpected because the logic is wrong. Back to the drawing board
- Failures caused by exceptional conditions. These result from a wide variety of issues and again are unusually difficult to detect on embedded systems



Exceptional condition failures

C (and C++ and to a certain extent Java) suffer from the following:-

- Expression based failures
- Library based failures
- Memory based failures
- Non-robust arithmetic



Overview

- v **Overview**
- v **Faults v. Failure**
- v **Static Fault modes**
- v **Dynamic failure modes**
- v **Embedded systems, finding the balance**



Getting the balance right

The general difficulties of dynamic testing in embedded systems along with their growing complexity suggest as a minimum:-

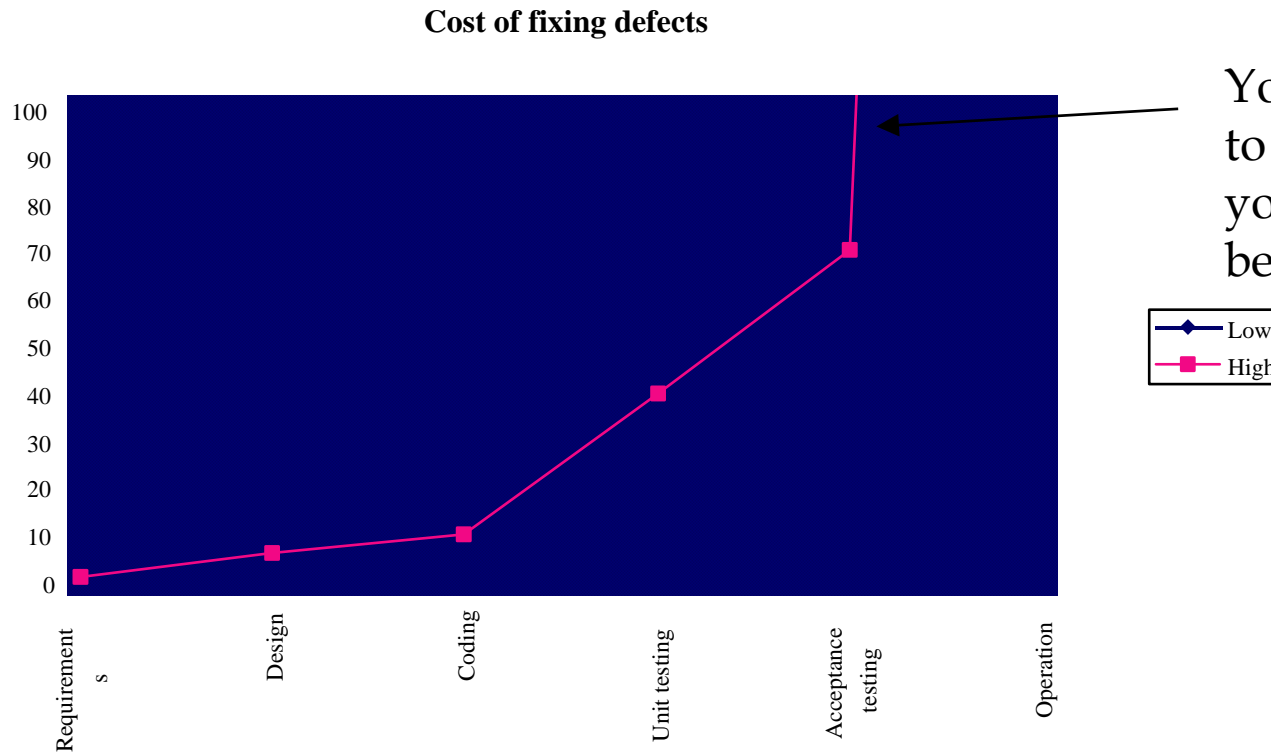
- There should be no residual statically detectable fault.
- Code inspections should be used extensively

In addition, dynamic testing ideally should reveal:-

- No exceptional condition failures for some acceptable level of test coverage, for example, 100% of all executable blocks.



High cost of failure



You don't want to be here but you wouldn't be alone.

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



More information ...

For more information on safer subsets, static and dynamic testing, downloadable technical publications and other links, you are invited to browse:-

<http://www.oakcomp.co.uk/>

