2000-

"Testing embedded C programs:
A measurement based approach to balancing static and dynamic testing"

by

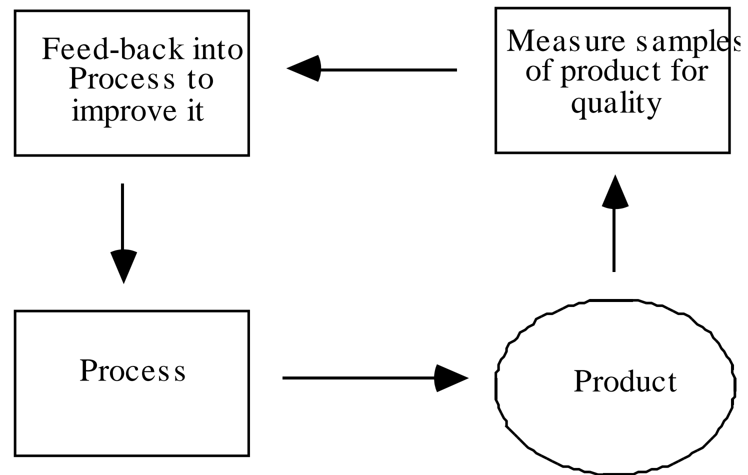
Les Hatton

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-

Control Process feedback - the essence of engineering improvement



If you want to improve reliability, measure and analyse failures.

© L.Hatton, 2000-

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

© L.Hatton, 2000-

Modern embedded systems are characterised by:-

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

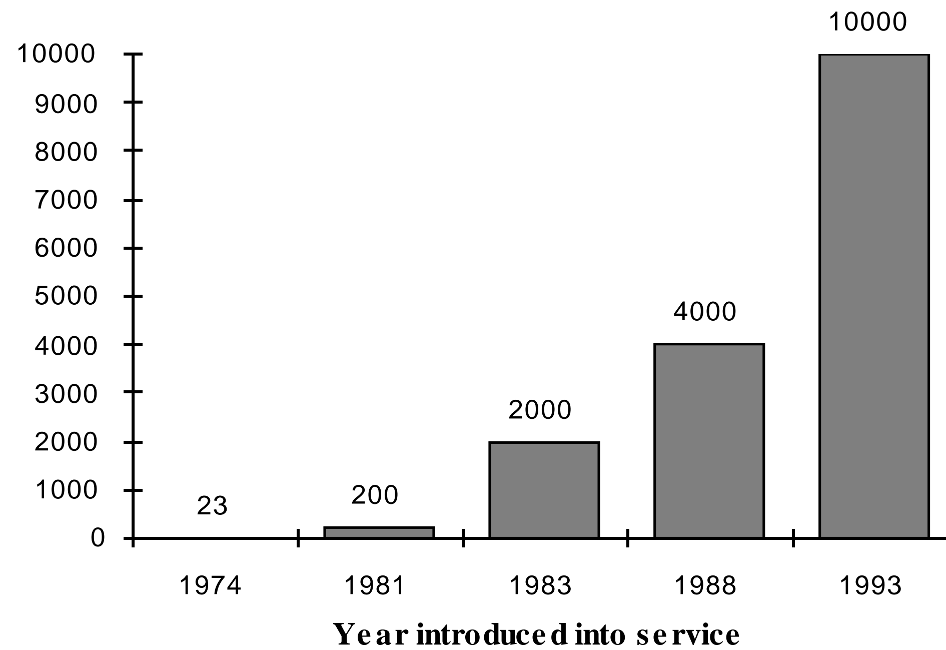
© L.Hatton, 2000-

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.

© L.Hatton, 2000-

Growth of software in Airbus A3XXX



© L.Hatton, 2000-

AT & T Jan, Jan 15, 1990:

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

© L.Hatton, 2000-

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;
...
```

© L.Hatton, 2000-

Most modern systems are implemented as networks

- The Airbus A340 has more than 150 communicating computer systems

© L.Hatton, 2000-

Testing difficulties

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

- Building and executing tests efficiently, particularly with interrupt driven systems
- Diagnosing dynamic failure
- Achieving good test coverage

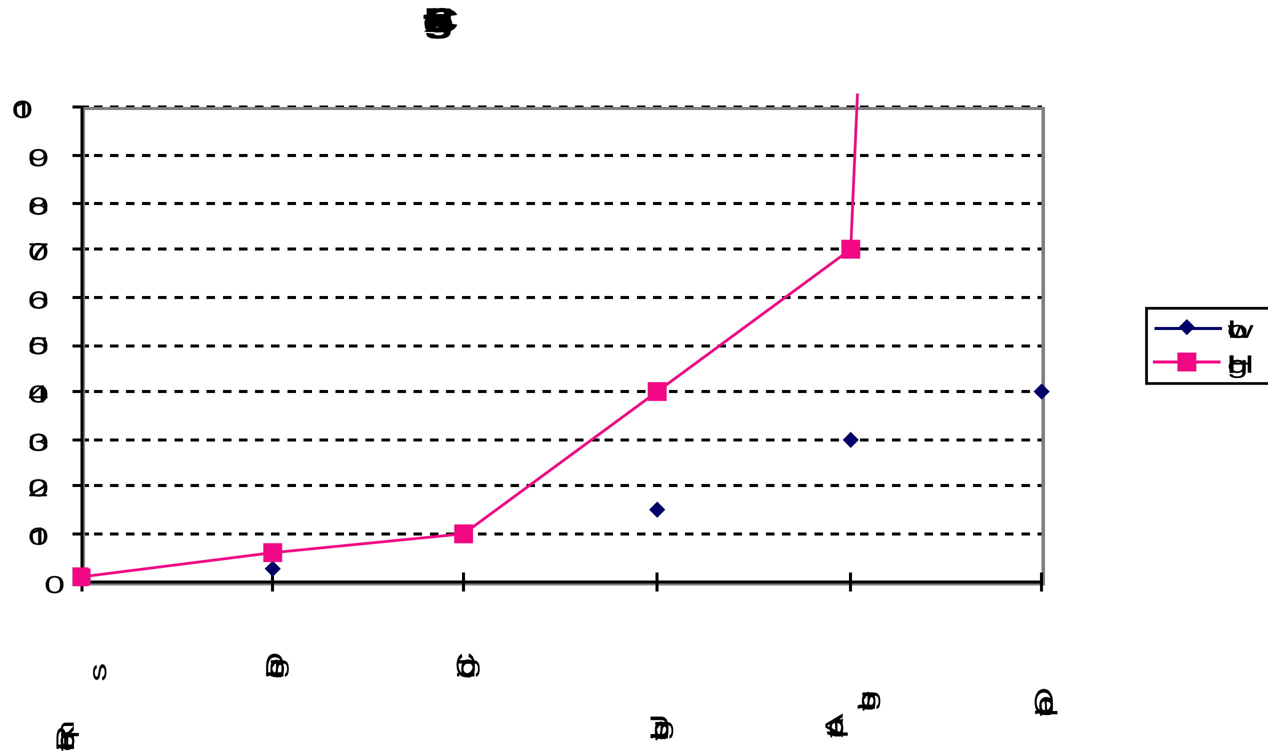
© L.Hatton, 2000-

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).

© L.Hatton, 2000-

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.

© L.Hatton, 2000-

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).

© L.Hatton, 2000-

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

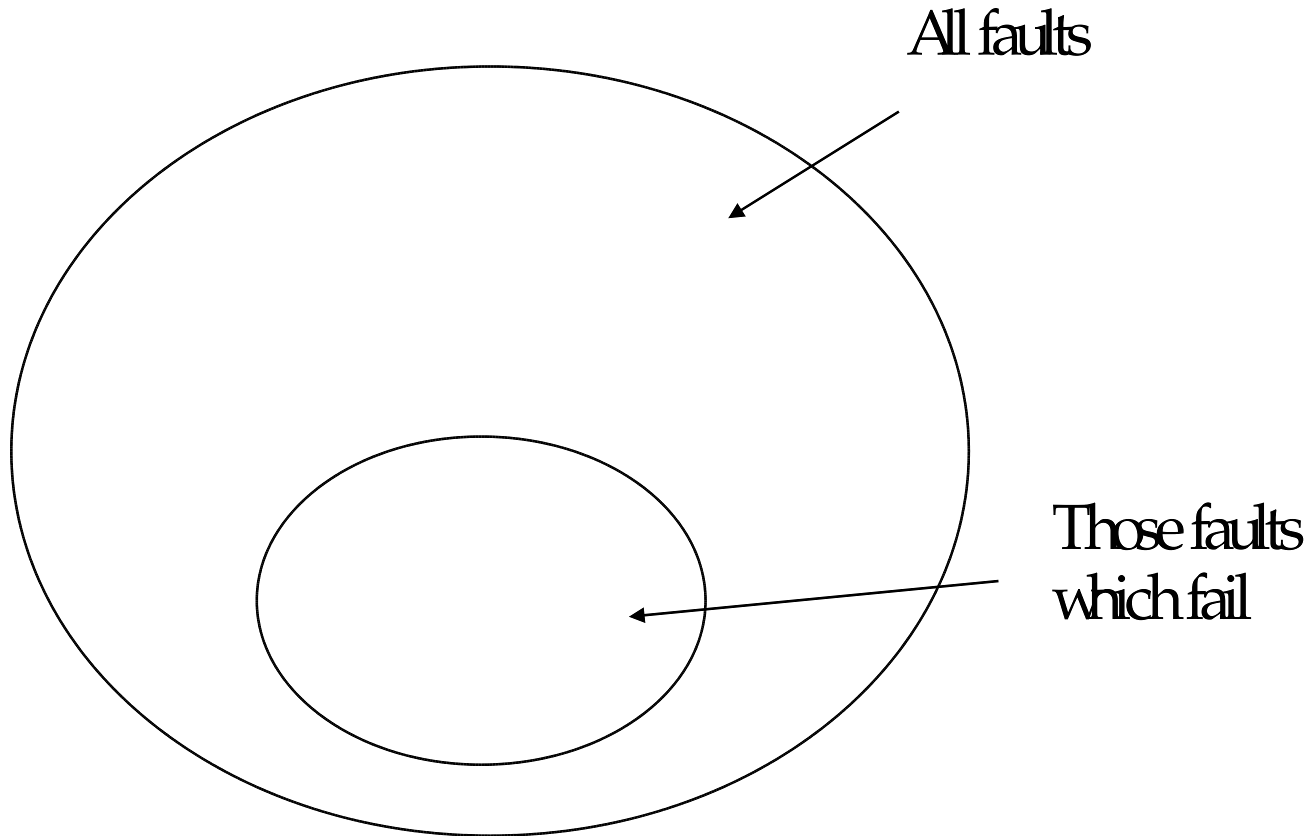
© L.Hatton, 2000-

Fixing the definitions

- A *fault* is a statically detectable property of a piece of code or a design
- A *failure* is a fault or set of faults which together cause the system to show unexpected behaviour at run-time
- A *defect* or *bug* is a generic term for *either* faults which fail *or* faults which do not.
- *Fault density* is the number of faults divided by the number of lines of code

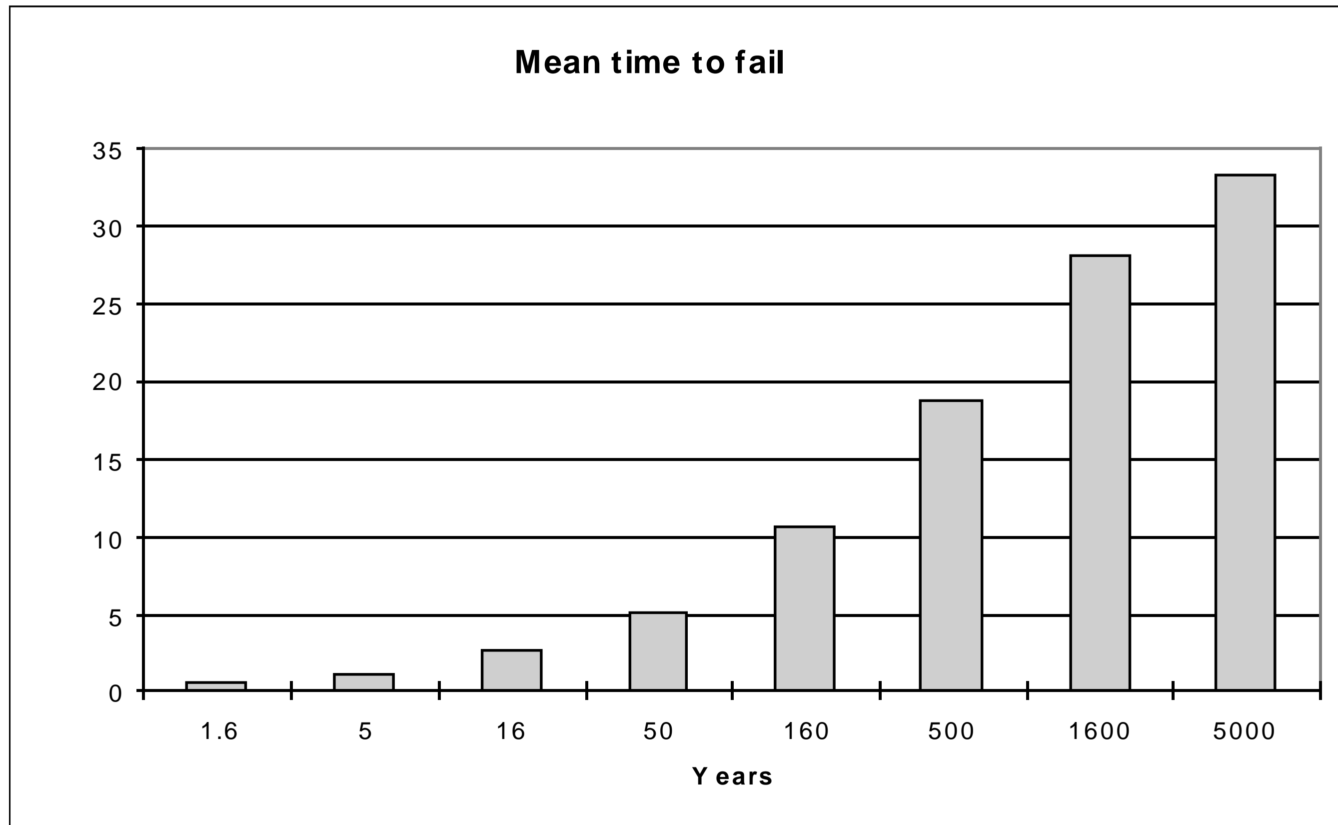
© L.Hatton, 2000-

Where and how do defects occur historically ?



© L.Hatton, 2000-

Mean time to fail in Adams (1984)



© L.Hatton, 2000-

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

© L.Hatton, 2000-

There are two kinds of statically detectable fault mode

- <u>Directly detectable fault</u>
- Indirectly detectable fault

© L.Hatton, 2000-

Directly detectable fault

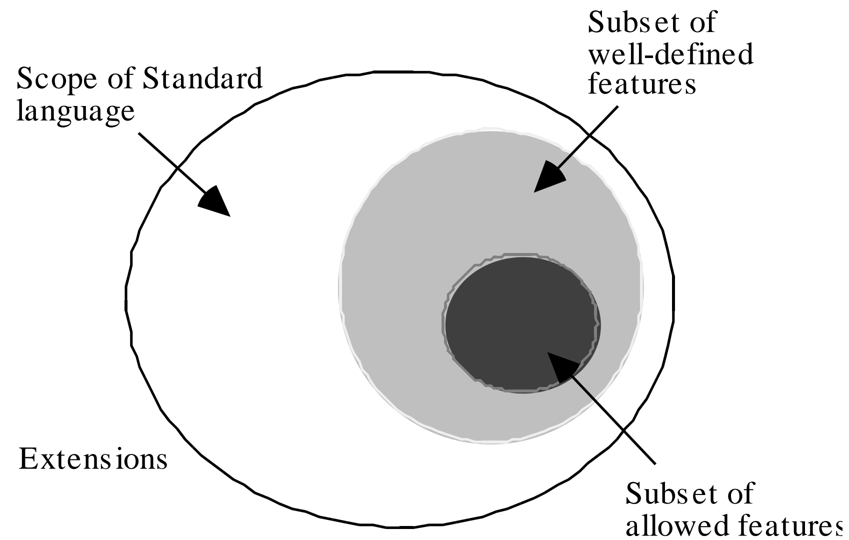
- **We will establish the following chain of reasoning:-**
  - Known fault modes exist in programming languages
  - They appear regularly in user's code
  - These faults fail with a certain frequency

© L.Hatton, 2000-

- **Sources of information on problematic behaviour in languages 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.

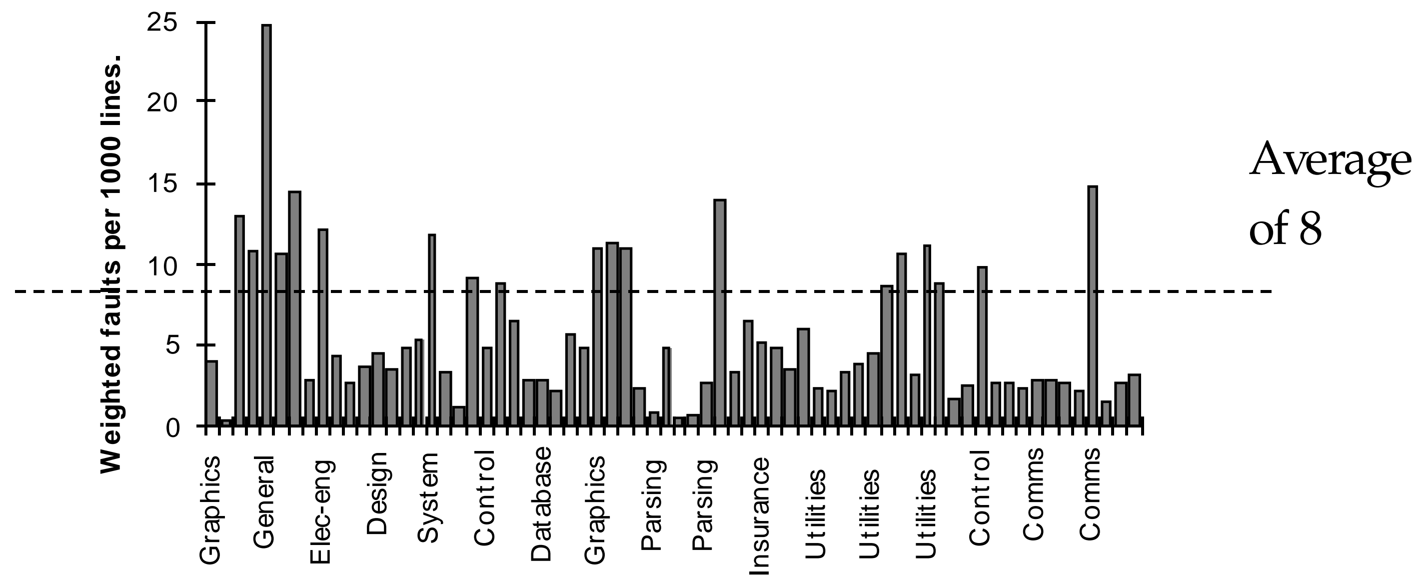
© L.Hatton, 2000-

The need for subsetting programming languages



© L.Hatton, 2000-

Fault frequencies in C applications

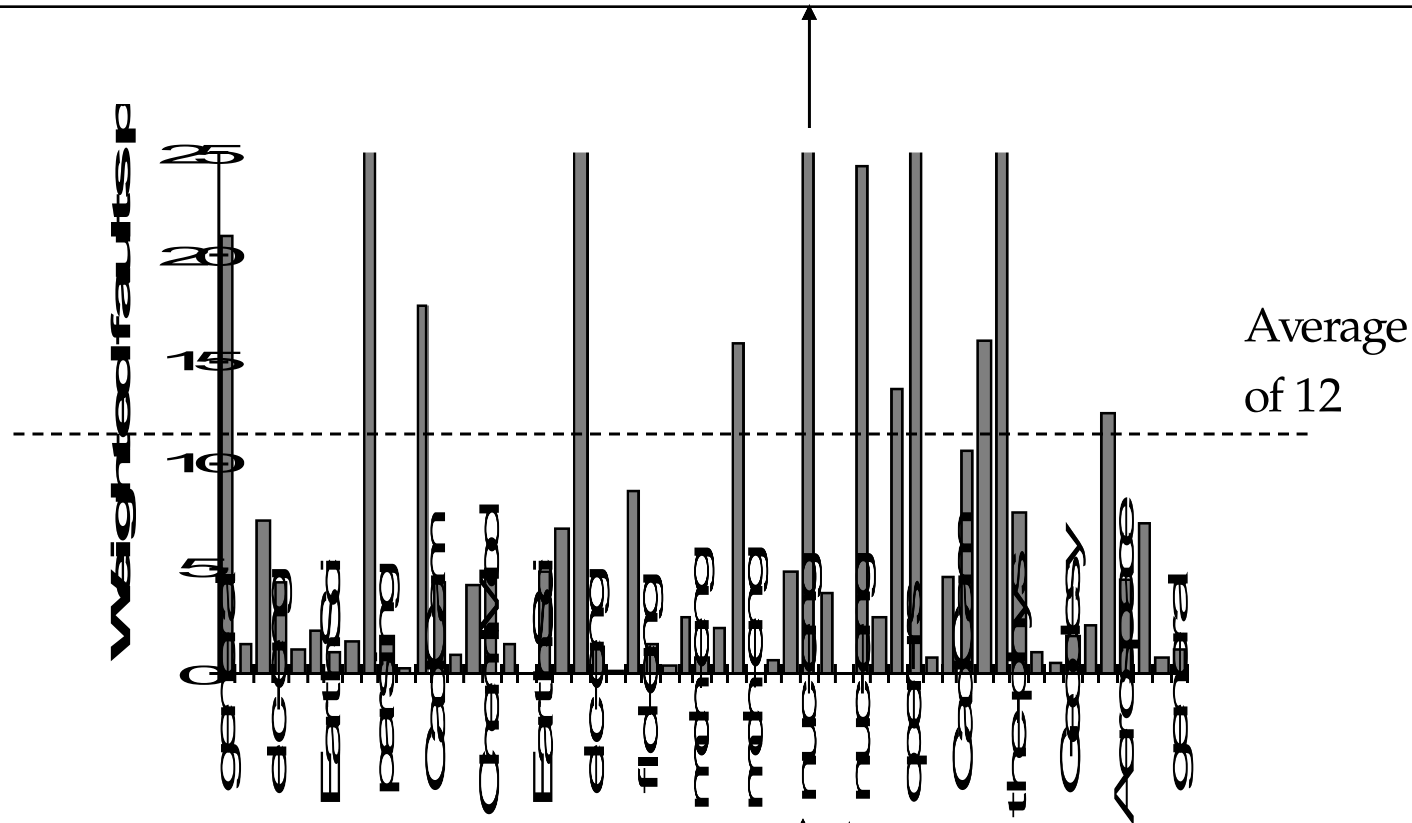


Average of 8

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

© L.Hatton, 2000-

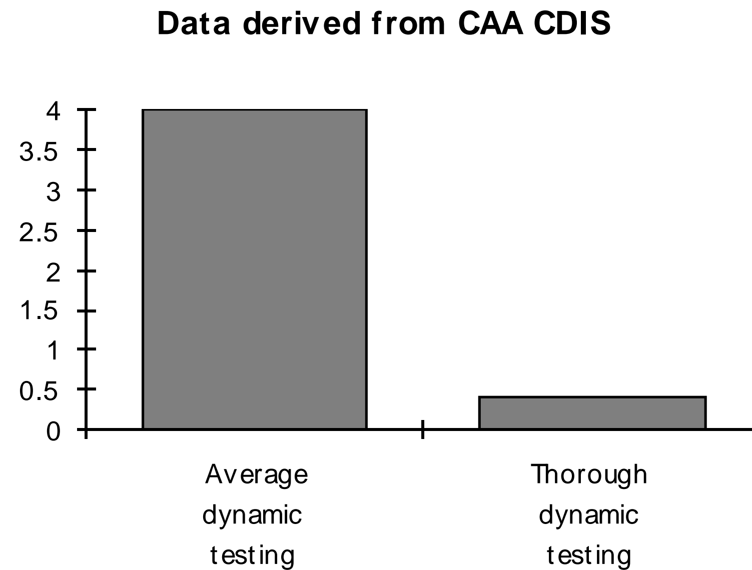
Fault frequencies in Fortran 77 applications



Same application area
one at 140 /KLOC and one
at 0 /KLOC

© L.Hatton, 2000-

Where and how do defects occur historically ?



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

© L.Hatton, 2000-

Conclusions on safer subsetting:

– We can prove the following:
  - There is a class of defect in programming languages which to a significant extent is statically detectable, widely reported and entirely avoidable
  - This class of defect evades conventional testing to the extent of around 8 residual defects per 1000 lines of code
  - 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 and tool support is crucial to the control of this class of defect. Tools *which only detect but do not educate* do not have the desired effect - the static noise problem.

© L.Hatton, 2000-

Statically detectable fault

Static analysis suffers from a noise problem

- When sometimes its a fault and sometimes not, for example:-

  if (a = b)

  instead of

  if (a == b)

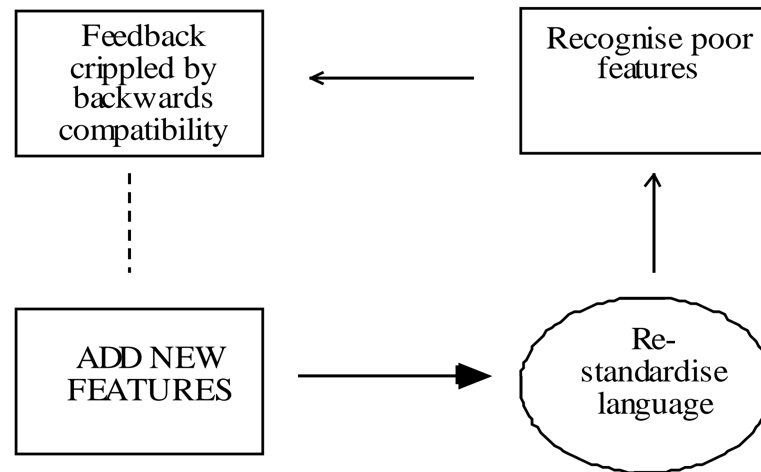
- In this case, if we warn of all transgressions those statements which are OK will tend to hide those which are not from the programmer. The ‘ signal’ is hidden by the noise.
- Some form of filtering is necessary, to maximise the likelihood of positive detection, for example a safer subset standard.

© L.Hatton, 2000-

- **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)
- **By comparison, C++99 contains the words:-**
  - Undefined, 1825 times
  - Unspecified, 1259 times.

© L.Hatton, 2000-

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.

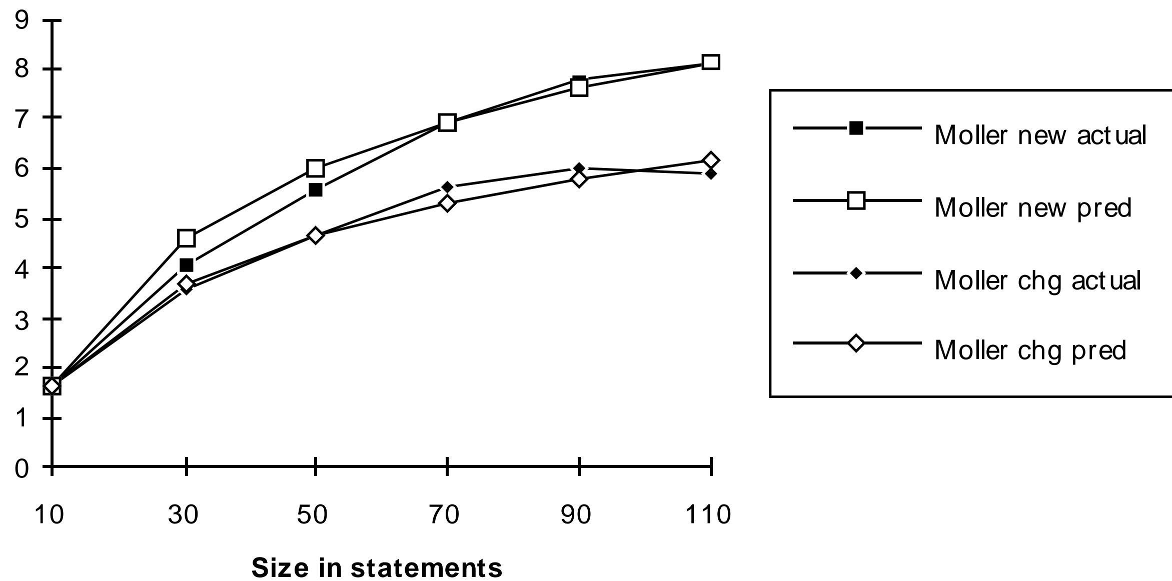
© L.Hatton, 2000-

There are two kinds of static fault mode

- Directly detectable fault
- Indirectly detectable fault

© L.Hatton, 2000-

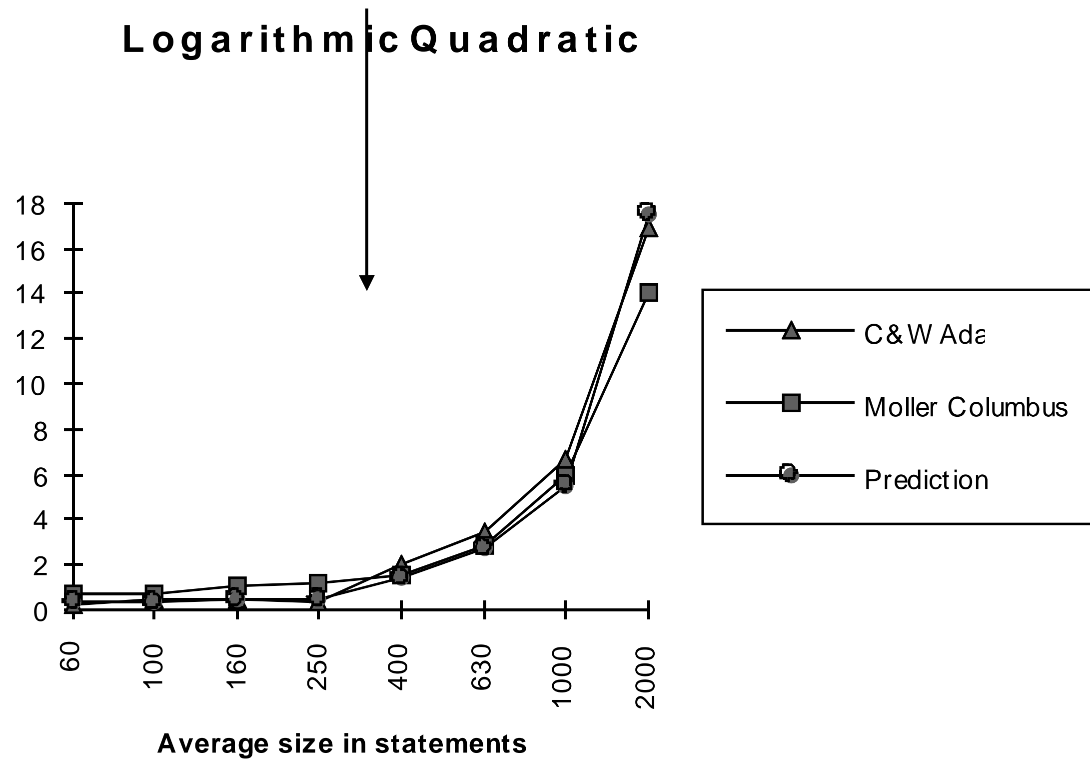
Failures and component size, (new and changed)



Data from an OS study at Siemens (1993)

© L.Hatton, 2000-

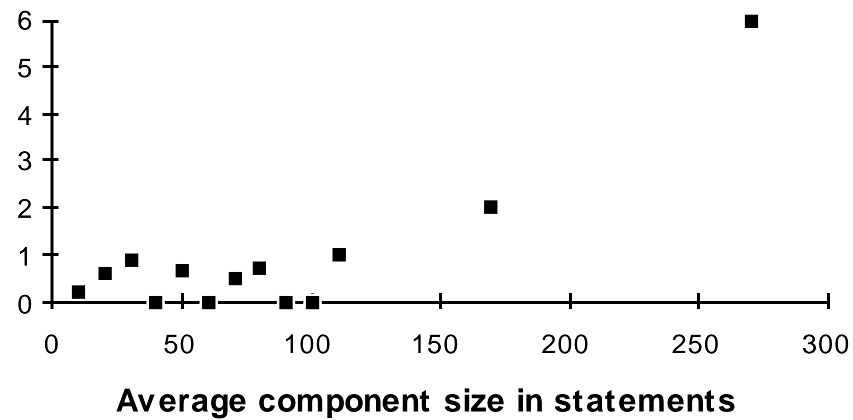
What happens for big components ?



Model due to Hatton (1997)

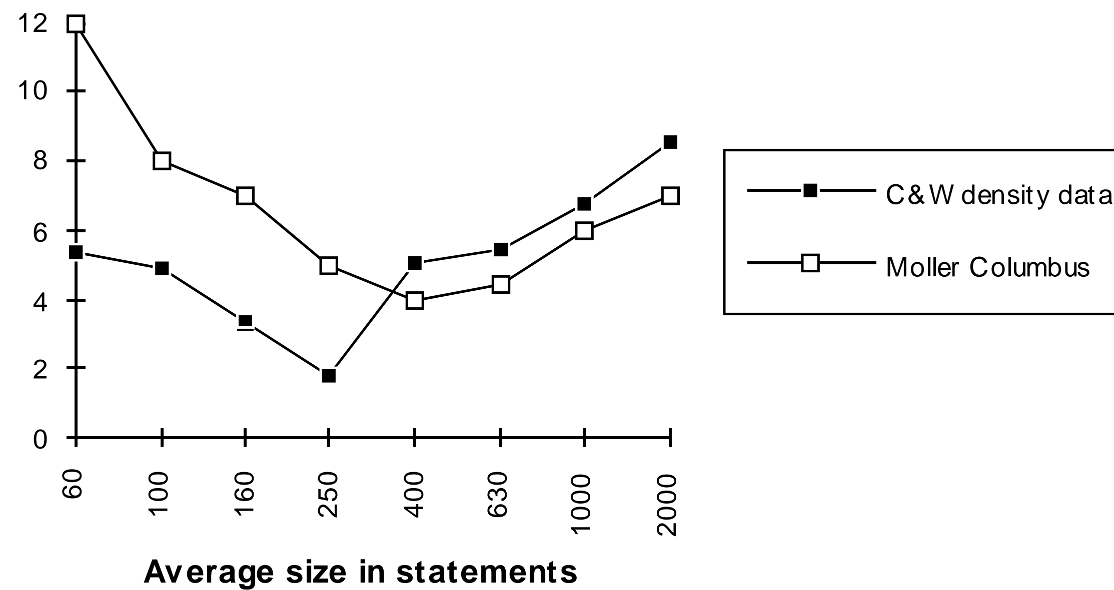
© L.Hatton, 2000-

Another example



© L.Hatton, 2000-

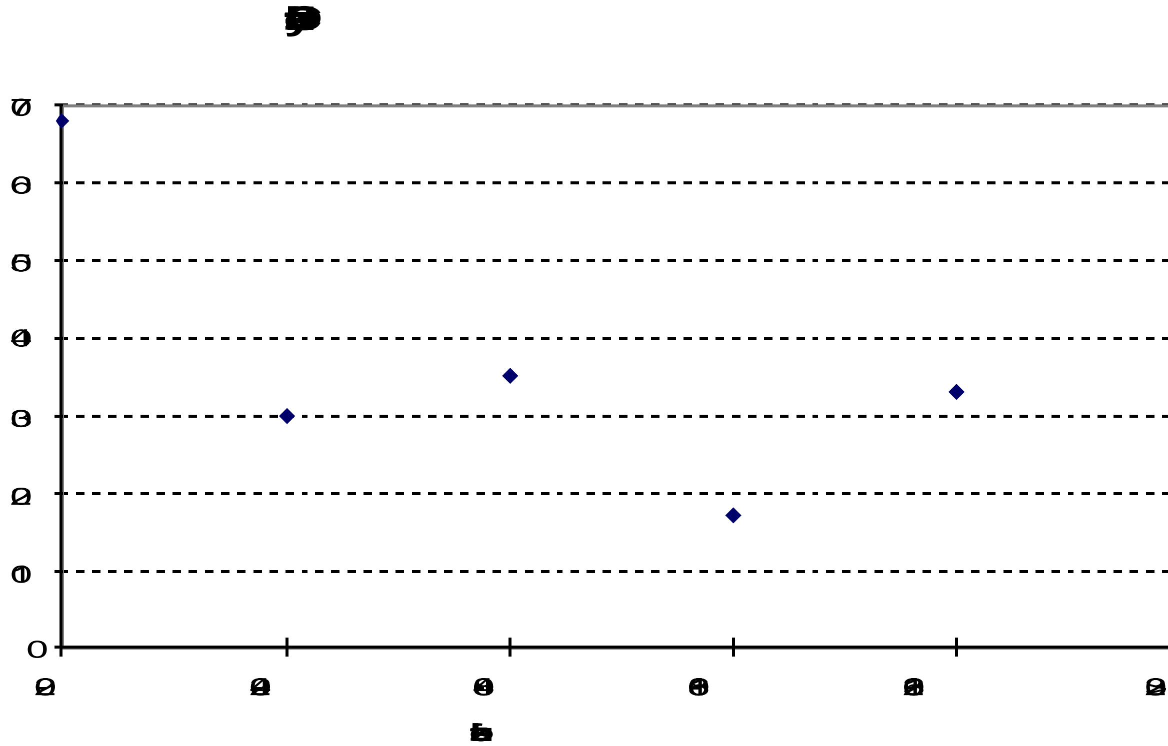
Failure density and component size



Comparison of Ada and assembler, Hatton (1997)

© L.Hatton, 2000-

Failure density and component size

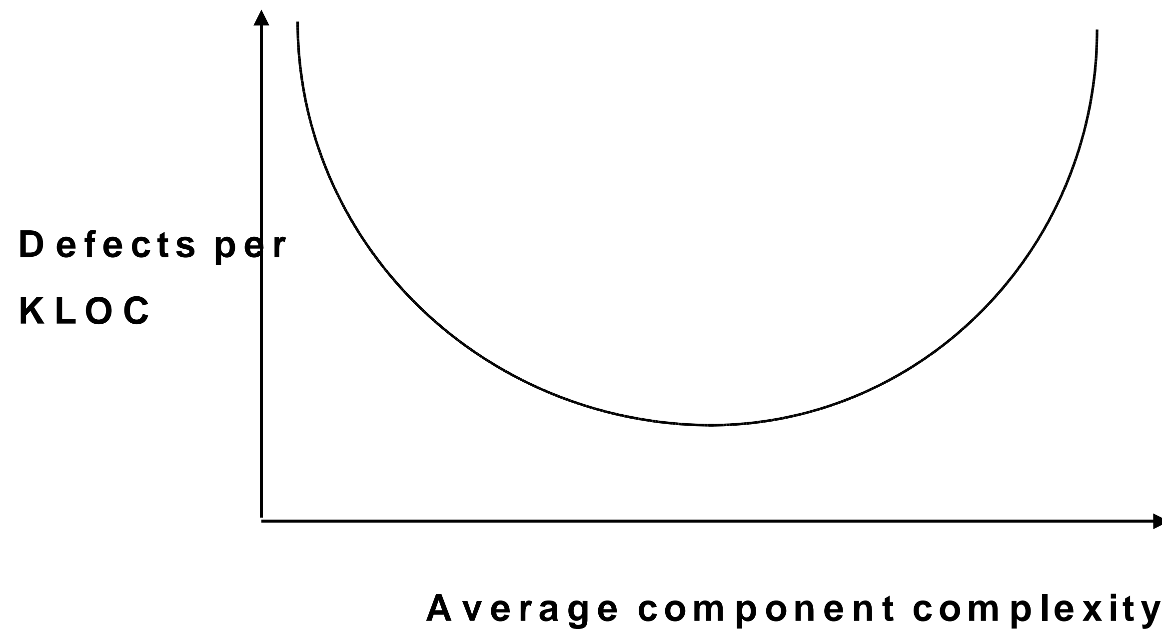


Data from the GNU indent program, Swanton (1996)

© L.Hatton, 2000-

The defect density U curve

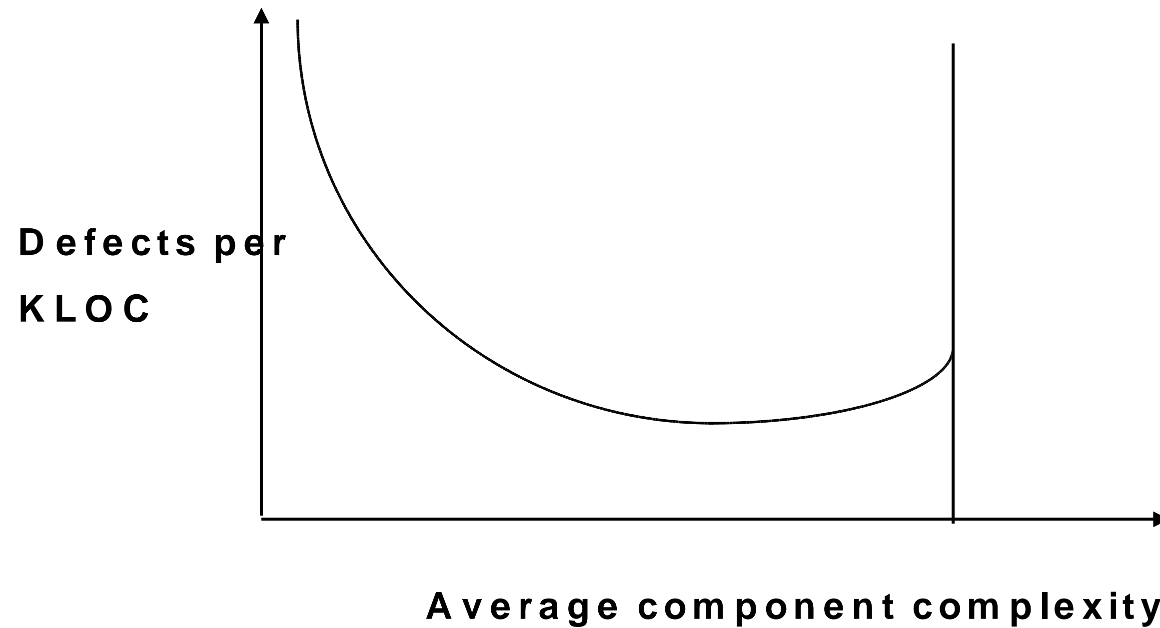
For Ada, various assembler, C, C++, Fortran, Pascal and PL/M systems:



© L.Hatton, 2000-

The defect density U curve - invasive truncation

In those systems where excessive complexity has been restricted:-



© L.Hatton, 2000-

By far the most effective way of avoiding the production of untestable programs is to <u>teach developers how to test</u>.

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))

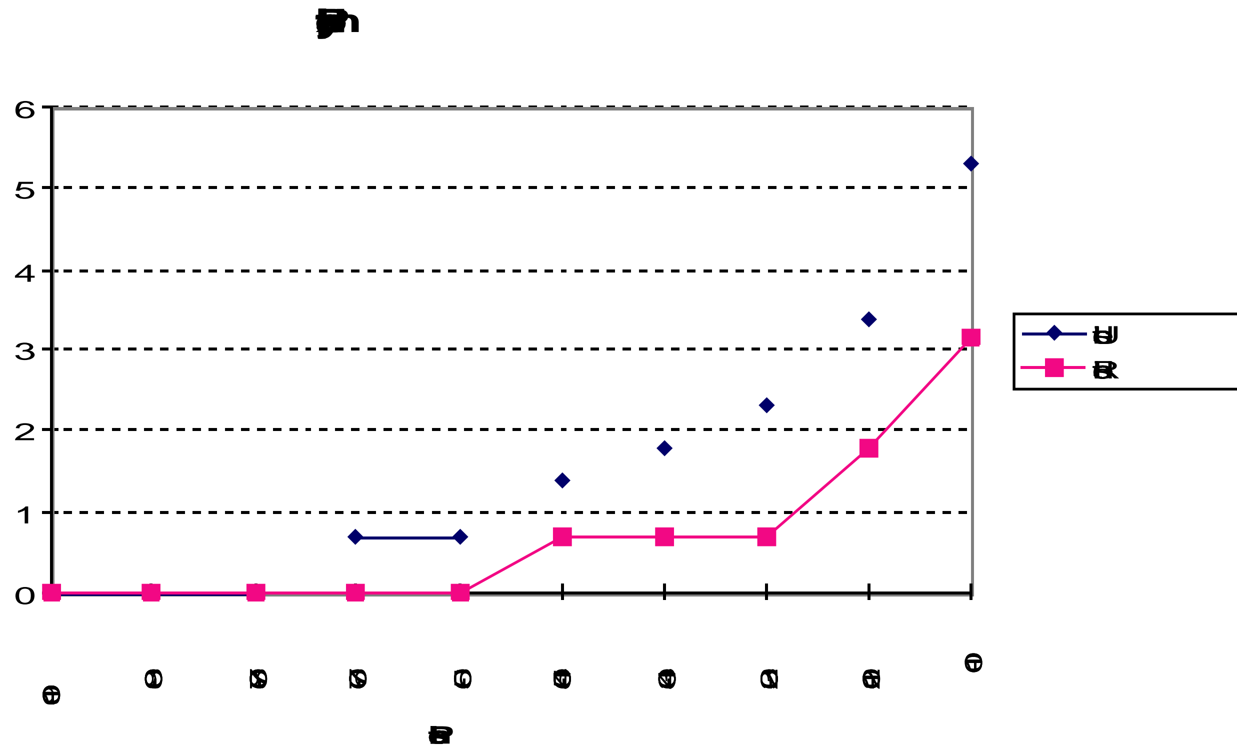
© L.Hatton, 2000-

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 design issues.

© L.Hatton, 2000-

Complexity limiting, Hatton (1995)



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

© L.Hatton, 2000-

Statically detectable fault

We do not know in advance which statically detectable faults will fail, but on average a significant percentage will

Conclusions:

- Source code should not be released with any statically detectable fault
- Learn about the fault modes of your language
- Beware of the static noise problem
- Train developers in testing so they do not produce untestable components

© L.Hatton, 2000-

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

© L.Hatton, 2000-

Dynamic failure modes

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

The rest must be found dynamically. This raises the connected problem of test coverage.

© L.Hatton, 2000-

Dynamic failure modes

C (and C++) suffer from the following dynamic failure modes:-

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

© L.Hatton, 2000-

Expression based failures

These include:-

- Arithmetic overflow, (a+b too big)
- Arithmetic underflow, (a-b too small) (floating point only)
- Division by zero, (z / 0.)
- Integer division of negative, (3/-2)
- Right shift by or of negative, (a >> -1, -1 >> 3)
- Loss of precision during conversion
- Use of uninitialised variables
- Array bound violation
- Dereferencing NULL

© L.Hatton, 2000-

These include:-

- Unchecked return codes
- Illegal parameter values
- Illegal overwrites
- Illegal file operations
- Unchecked 'errno' values

© L.Hatton, 2000-

These include:-

- Illegal free
- Multiple free
- Memory leaks
- Illegal overwrites
- Illegal reads
- Freeing NULL, (not an error but unusual)

© L.Hatton, 2000-

These include:-

- Loss of significance due to arithmetic cancellation
- Default float computation rather than double, (a problem shared with C++ and Java).

© L.Hatton, 2000-

Coverage

Satisfactory dynamic testing requires an estimate of test coverage using at least one of the following criteria:-

- Block coverage
- Exit point coverage
- Decision coverage in various forms.

© L.Hatton, 2000-

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

© L.Hatton, 2000-

Getting the balance right

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

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

After this, dynamic testing ideally should achieve:-

- No dynamic failure modes for some acceptable level of test coverage, for example, 100% of all executable blocks.

© L.Hatton, 2000-

More information ...

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

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

© L.Hatton, 2000-