Noise-Cancelling Embedded C Development

Darren Buttle, Product Manager, Embedded Software Tools

ETAS

Who are we?

Tools, Services & Consultancy for the Automotive Industry

- Measurement & Calibration
- Software Development Tools
  - Code Generation, AUTOSAR Architecture, Virtual ECUs
- Rapid Prototyping Hardware & Software
- Embedded Operating Systems (1,000,000,000+ units on the road)
- Embedded Security

Heresy (or, rather, Here-C)

I am not going to talk about Ada

(well, maybe a little bit)

What can we tell Ada Europe about high integrity software?

Challenges for Automotive Software
Complexity Growth

Complexity issues:
- Number and complexity of functions (30-100 ECUs)
- Interaction between systems
- Many innovations are either **safety-relevant** and/or involve **connectivity** to the outside world.

➔ **Ensuring the safety and security of the E/E systems is growing increasingly more challenging**

Electronic fuel injection
Anti lock brakes

Gearbox control
Traction control
Cruise Control
Electronic fuel injection
Anti-lock brakes

Airbags
Electronic stability control Active body control
Adaptive gearbox control
Adaptive cruise control
Emergency call
Gearbox control
Traction control
Cruise control
Electronic fuel injection
Anti lock brakes

Tele-diagnostics
Internet browser
Apps
Car-2-car communication
Road trains
Software updates
Adaptive headlights
Active steering
Curve warning
Stop and Go
Lane keeping assistance
Automated parking
Collision mitigation
Hybrid powertrain
Emergency call
Airbags
Electronic stability control
Active body control
Adaptive gearbox control
Adaptive cruise control
Gearbox control
Traction control
Cruise control
Electronic fuel injection
Anti lock brakes

1975 1985 1995 2005 2015

Challenges for Automotive Software
Lots of Code

≈ 100,000 SLOC

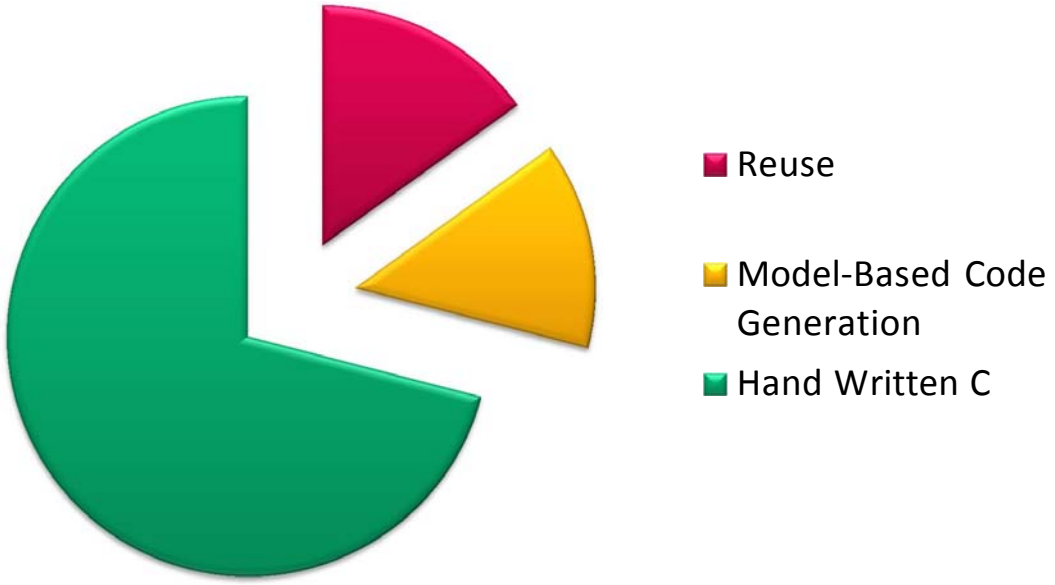
≈ 6,500,000 SLOC

≈ 20,000,000 SLOC

= 250 copies of "The Complete Works of Shakespeare"

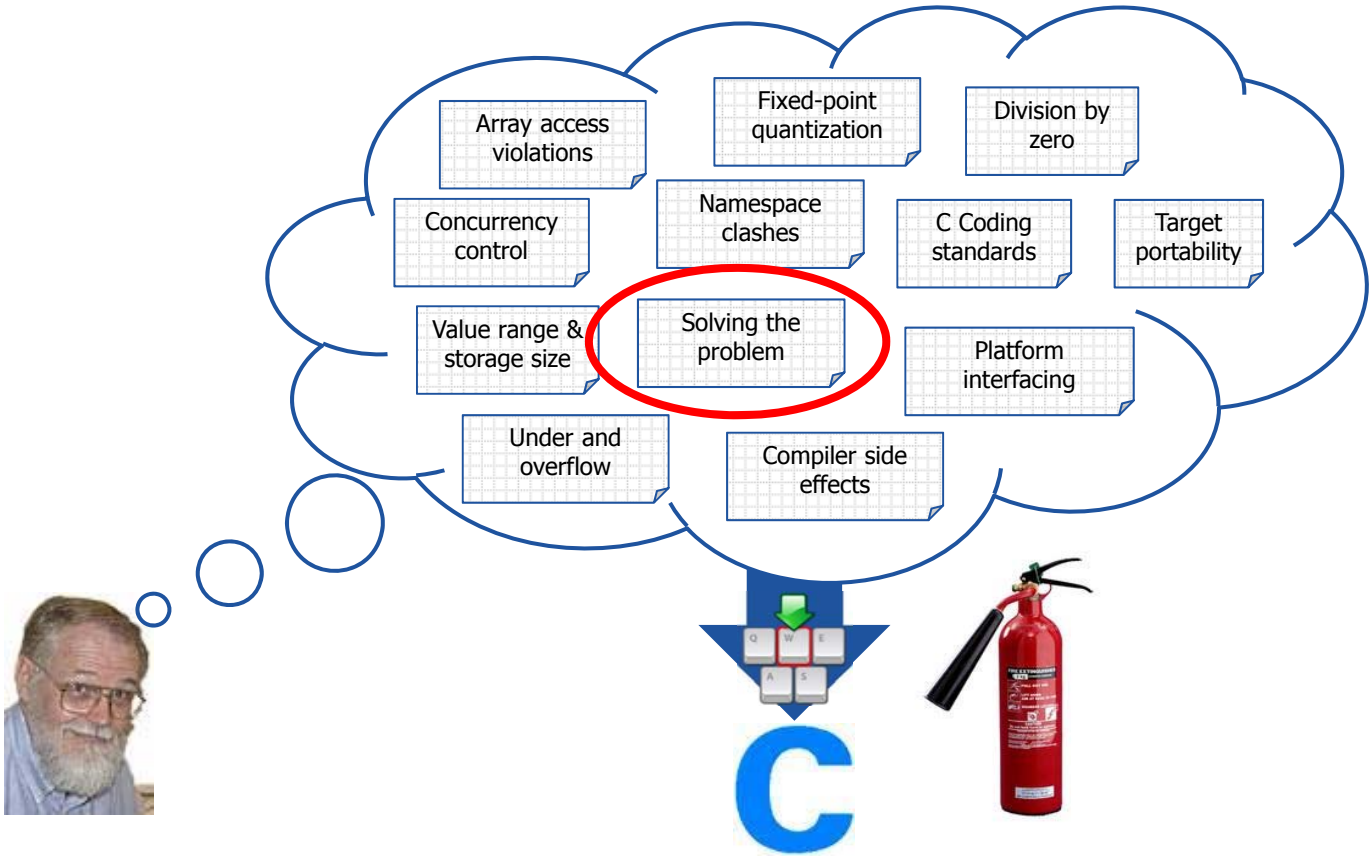
Sources
Pavey & Winsborrow, "Demonstrating Equivalence of Source Code and PROM Contents", Computer Journal Vol 36, No 7, 1993
Charette, "This car runs on code", IEEE Spectrum, Feb 2009

Sources of Source



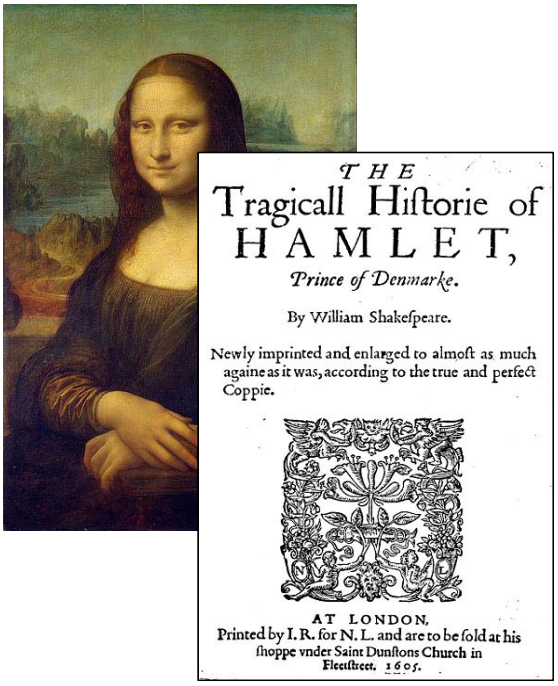
The Challenge of Embedded C

Why is it hard? Lots of "noise"

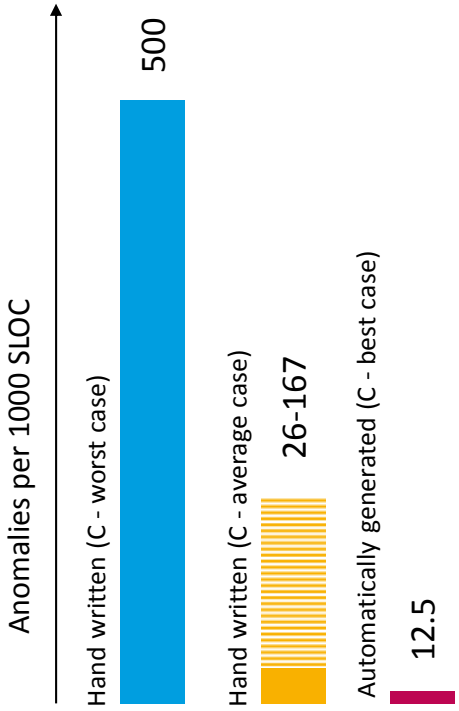


Other Common Languages? Modelling?

- C++, C#, Java, etc.?
  - ☹ Runtimes are too memory hungry
  - ☹ Not designed for predictability
  - ☹ Limited compiler support for automotive microcontrollers
- Model-driven development?
  - You're do it already
    - Code is a model ;-)
  - Graphical modeling == Drawing
    - Not suitable for all problems
- What else?
  - …assuming we're stuck with a C compiler



- Takes the "noise" out of C by
  - Removing dangerous features
  - Adding "good old fashioned" software engineering features
- Generate the C code that is hard to write (or hard to get right)
  - Evidence suggests this is significantly better than writing C by hand
- With the goal to
  - Make it easier to prevent bug introduction
  - Make implicit assumptions explicit
  - Reduce scope for getting it wrong



Source: "Static code analysis lessons learned", Andy German, Crosstalk 2003

(C – Bad) + Good = ESDL

- Syntactic style
- Statements
- Expressions

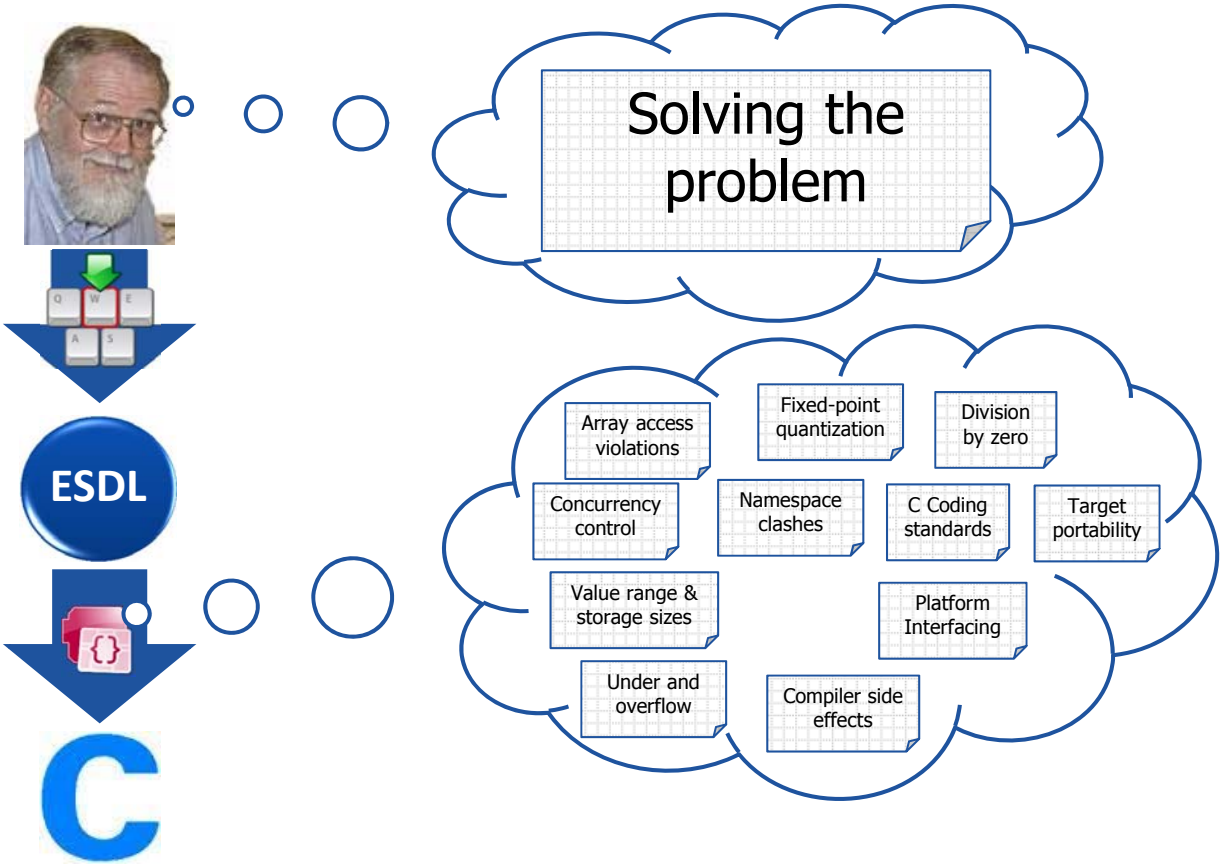
- Assignment in expressions
- Optional braces
- Assignment to loop guards
- Automatic case fall through
- Pointer arithmetic
- Goto
- Macros
- …

- Object-based encapsulation
- Strong Typing
  - Explicit (enforced) ranges
  - Native fixed-point
  - Units (e.g. mph)
- Define behaviour in presence of runtime errors
  - No over/underflow
  - No div by zero

- C-like language
  - Easy to learn
  - Hard(er) to make mistakes
- ASCII on-disk format
  - Easy to version control, diff etc.
- Easy to review
  - No implicit data assumptions

The Challenge of Embedded C

Working more effectively: Concentrate on "what" not "how"



Making it easier to express intent

- Modularity & Encapsulation
  - A "package" notion
  - Object-based encapsulation
    - With tight visibility control
    - No dynamic creation

```
package My_Package;
class My_Class
{
    // Hidden: no "public" variables
    Some_Type Some_Variable = 42;

    // Methods are private by default
    void Private_Method(integer in X, …) {…};
    // …made public explicitly
    public void Public_Method(…){…};
}
```

- Stronger Typing
  - Named numeric types with ranges
  - Fixed point types
  - Enumerations are proper types
  - Arrays are not pointers

```
type Hours is real 0.0 .. 24.0 delta 0.25;
type Day is {Mon,Tue,Wed.Thur,Fri,Sat,Sun};

type TimeSheet is [Day] of Hours;

TimeSheet[Day.Tue] = 8.75;s
```


Language Tour
Something Borrowed

Generate code that is hard to get right (and maintain) code

```
sint32 t1sint32;
t1sint32 = (x >> 1) + (y >> 1);
add = ((t1sint32 >= -1073741824) ?
            ((t1sint32 <= 1073741823L) ? t1sint32 << 1 :   2147483647L) : (sint32)SINT32_MIN);

t1sint32 = (x >> 1) - (y >> 1);
sub = ((t1sint32 >= -1073741824) ?
           ((t1sint32 <= 1073741823L) ? t1sint32 << 1 :    2147483647L) : (sint32)SINT32_MIN);

t1sint32 = (x >> 16) * (y >> 16);
mul = ((t1sint32 >= 0) ? ((t1sint32 <= 0) ?
           (t1sint8 = (sint8)t1sint32 << 32 , t1sint8) : 2147483647L) : (sint32)SINT32_MIN);

div = ((y == 0) ? x :
           ((x == (sint32)SINT32_MIN && y == -1) ?
            2147483647L : x / y));
```

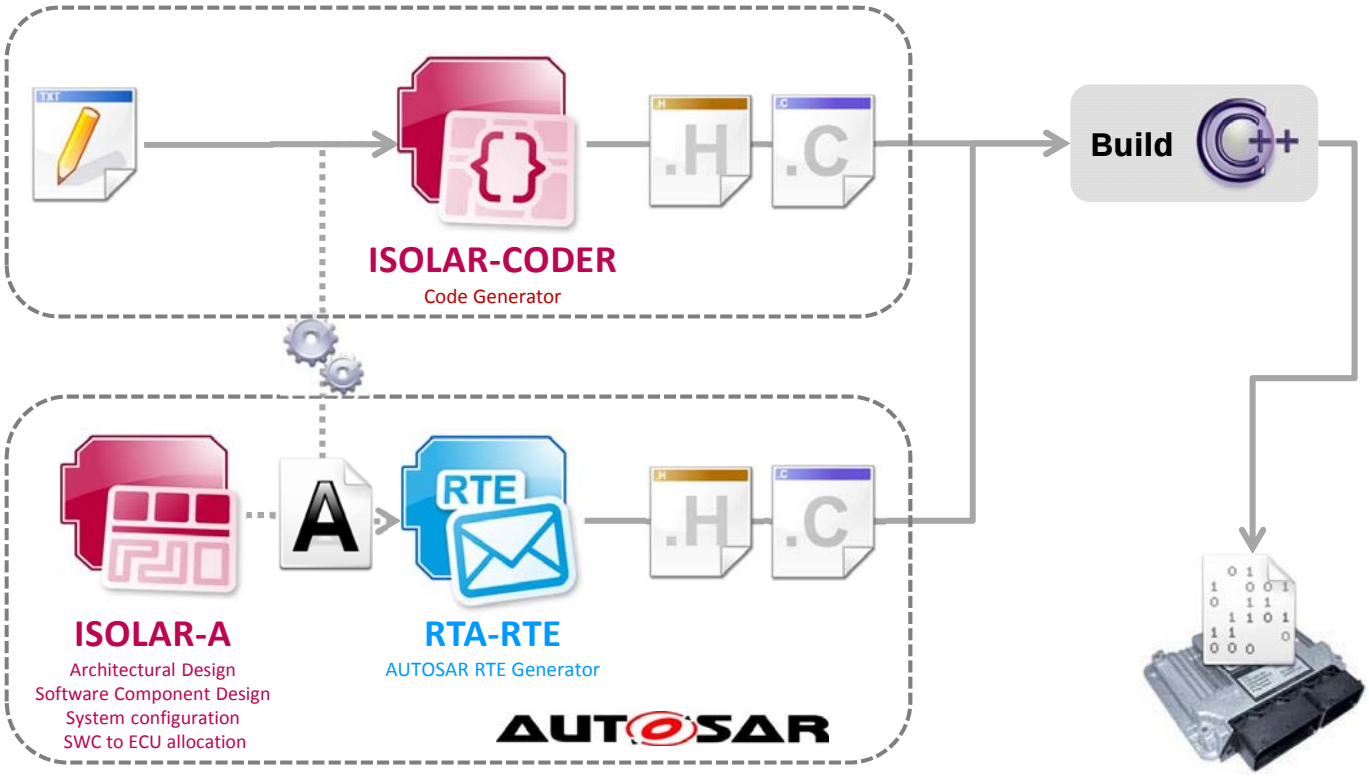
```
t1sint32 = (sint32)x * y;
mul = (sint16)(((t1sint32 >= -123) ? ((t1sint32 <= 456) ? t1sint32 : 456) : -123));

t1sint16 = ((y == 0) ? x : x / y);
div = ((t1sint16 > -123) ? t1sint16 : -123);
```

Abstracting to the real world

- Support for physical units
  - And conversion relationships between them
- Automatic conversions between units of the same dimension
- Automatic conversions & checking for calculations
  - E.g. distance/time is a speed

```
unit meters;  // Create meters "ex nihilo"
unit seconds; // Create seconds "ex nihilo"

unit centimeters = 0.01 * meters;
unit kilometers = 1000.0 * meters;
unit minutes = 60.0 * seconds;
unit hours = 60.0 * minutes;

unit kph = kilometers / hours;
```

```
unit kilograms;
unit pounds = 0.453* kilograms;

kilograms My_Weight;

if (My_Weight > 180 [pounds])
{
  // Got to burn some calories!
  Bicycling_Speed += 2.0 [kph];
}
```

More discipline = Fewer bugs

- ESDL aims to make it easier to be disciplined
- Stops the worst horrors of C
- Capture assumptions explicitly
  - Check them statically
- Generate code that is
  - **hard to write** or
  - **hard to get right**

Noise-Cancelling Embedded C Development