Resumé of a Debate on Tools, Assurance and Ethics in the Development of Software-Based Safety-Critical Systems

held on the safety-critical systems mailing list of the University of York
4-10 September 2009

compiled and edited by Peter Bernard Ladkin
10 September 2009

The debate can be read in full by going to
http://www.cs.york.ac.uk/hise/safety-critical-archive/2009/
sorting the archive by thread, and looking for
«Combining case and default in a switch statement»

Languages and Coding Style

Scott Nowell introduced a question about C style:

> The MISRA C standard does not directly address the issue of combining case: and default: in a switch statement. This use seems to be C99 compliant. Here is a simplified fragment.
>
> ```
> switch (err) {
>   case 0:
>       ...
>       break;
>
>   case 1:
>   case 2:
>   case 3:
>   default:
>       ...
>       break;
> }
> ```
>
> I am seeing code examples like this where multiple cases are being combined with default. Most compiler will simply optimize this out to a simple test for case 0. The programmers are using this as a way to identify the results they might expect. I think having the other cases in a comment would serve this as well. MISRA C does not seem to mention this.
>
> From a test perspective I feel it is necessary to test all declared cases.
>
> This is all new code.
>
> The stated reason for this construct from the developer is that it explicitly states all known cases and keeps them front-and-center for the developer. I contend that the same in a comment does the same, but at the same time, comments are [not] checked by the compiler for invalid names.
>
> I also do not like the single case switch that essentially replaces a simple if statement. I see quite a lot of them as well. ..... I find single case switches with only a break and no other code particularly irksome.
>
> ```
> Switch (err) {
>         case OK: break;
> ```

```
        case OOPS:
        case FAIL:
        case NOT_OK:
        default:
                return (err);
                break;
}
```

> This really seems like a complicated way of saying
> If (err != OK) ...
>
> Unfortunately I think this is a consequence of what I believe to be a push for "pretty" code. Code, like traditional design, should be a case of "Form follows function", not the other way around.

This generated discussion amongst afficionados of C as to appropriate ways to generate the desired effect. This seemed to me reminiscent of discussions which have been going on as long as I have been in informatics about the syntax of programming languages – and their semantics. One of the major problems which I and others see in the use of C for safety-critical systems is that it is a tricky language with constructs which are syntactically valid but which generate «undefined» behavior. Also, not all compilers necessarily generate the behavior one would expect, even from constructs with defined behavior. So it becomes very tricky to determine formally what an arbitrary piece of sophisticated C code actually does in all environments.

Which is why I said:

> You're talking about safety-critical systems. Why don't you write your code, or get it written, in a programming language with a sensible semantics?

As Andy Farnsworth said:

> .... this discussion wouldn't be taking place about Ada or SPARK or B.

Jeff Payne suggested:

> Ada has a strong reputation of being a dead/dying language. ..... It is virtually impossible to persuade [senior management] to return to "old" [technologies, techniques or languages].

Britt Snodgrass responded:

> .... it [is] really a misperception to think of current Ada as "old". Ada 83 was ahead of its time. With the Ada 95 and 2005 changes, much about Ada and associated development tools is rather new. Where I work we are doing some new projects in SPARK (Ada) and using Eclipse/GNATbench as the IDE. While the use of Ada has (irrationally) declined on safety-critical projects, its use on security-critical projects is increasing.

Jeff suggested further:

> If someone were particularly clever, they would follow the the Microsoft CLI model and convert Ada to a C-like syntax. The result being a language that looked much like C, but was a language with "sensible semantics".

To which Britt responded:

> You may be right that more people would adopt "Ada with curly braces" but it would

> lose a lot of the readability of real Ada. One of the benefits of begin ... end keywords instead of { ... } braces is that you can easily name what is ending for nested packages, procedures, loops (i.e. fully named scope), e.g, "procedure Check_Input is ... begin ... end Check_Input;"

And I responded (to Payne):

> We are reduced to playing silly tricks to encourage people to develop safety-critical systems appropriately? Isn't the possibility of killing people enough encouragement?

Jeff Payne responded to Britt:

> That's the beauty of the CLI - the source language becomes interchangeable. You could literally write your code in one language, translate it to the other, and the CLI guarantees that the two are functionally identical.

Suppose this were true. Suppose you were concerned with translating C into SPARK Ada. SPARK Ada is guaranteed to have a unique meaning (unique defined behavior) for each program. For C this is by no means the case – this is the basis for much criticism of the use of C in safety-critical systems development. The CLI translation from a C program into a SPARK Ada program would necessarily give the C program a unique, defined meaning, because all SPARK Ada programs have such. However, it is by no means guaranteed that a C compiler, even a good C compiler, would compile the original C program into object code which exhibited that behavior. The other way around, that one writes a SPARK Ada program in order to gain the benefits, translates it into C (because, say, a compiler for Ada is not available) and then compiles it with a C compiler, is not at all guaranteed to yield a program which has the same behavior as the original SPARK Ada program, because C programs generally do not have unique meanings in this sense. So it is hard to see what would be gained by source-code translation of this sort, in either direction. (I am informed privately, though, that a SPARK-to-C translator exists which does yield unambiguous programs – that is, programs which have relevantly-identical behavior when compiled with a selection of the most-used C compilers. I suspect that this is not quite subsetting, that it is subsetting+functional constructs. I await more details.)

By the way, the official MISRA-C Committee reply to Scott's query is:

> The code fragment does not violate any MISRA C rules relating to the switch statement.
>
> The number of tests cases suggested by this code fragment will depend on the project testing strategy. This is not something that the MISRA C Guidelines addresses.

If that makes you happy, be my guest, but it doesn't cheer me up much.

Language Use in Safety-Critical Systems Development

The use of imprecise but flexible languages such as C in safety-critical systems development concerns many of us. What are the reasons for its use? In response to my comment

> You're talking about safety-critical systems. Why don't you write your code, or get it written, in a programming language with a sensible semantics?

Jeff Payne replied:

> Because some of us do not have executive authority over the projects we work on.

And Scott Newell, while sympathising with my sentiment about choice of language, clarified about his query:

> As is usually the case though, the customer did not ask what language he should use, he only asked us to help test his code.
>
> .....it is not uncommon for an Ada user to come to us and ask us to convert their "old" Ada code to shiny new C. Go figure.

It is astonishing to imagine that Scott's customers might not be aware that C is forty years old and Ada rather younger, at around twenty-six years. Indeed, he assures me that they are, and this phraseology is tongue-in-cheek.

Chris Hills offered a technical reason:

> .... not all languages with "sensible semantics" are available on all target platforms.

He also justified the use of imprecise and finnicky languages such as C through the following observations:

> .... by far .... most important is the quality of implementation of the compiler/tools. .....
>
> Mathematically correct languages solve nothing if the people using them do not understand the wider concepts and the tools are not of a high standard.
>
> .....I think the same problems of "defensive programming" occur in most languages. It is the principals we are discussing not the minutia of the syntax.

Chris is right to add that having a wonderful language design doesn't help anyone much if the tools available for support of programming in that language are not of very high quality. But I do not agree that the quality of the compiler and tool support should be decisive. Such high-quality compiler and tools cannot help if the language is imprecise: the compiler will compiler any program into specific object code which has specific behavior, but one needs to be able to tell in advance – one needs assurance of – what the compiled program will do, and if its source-code meaning is imprecise then there is no means to do this.

There has been of course considerable work on minimising this imprecision. Les Hatton's book «Safer C» (Addison-Wesley, 1995) has pages and pages of do's and don'ts. His Table 2.1 of «Unspecified Items in Standard C» runs to 22 items and one and a half pages. His Table 2.2 of «Explicitly Undefined Items in Standard C» runs to 97 items and five and a half pages. His Table 2.3 of «Implementation-Defined Features of Standard C» runs to 76 items and five and a half pages.

By contrast, such tables for SPARK Ada are empty.

The Motor Industry Software Reliability Association (MISRA) has attempted to define subsets of C and C++ which it believes are more appropriate to use in reliability-critical and safety-critical projects in the automotive domain. Unlike SPARK Ada, these subsets do not have the property that programs written in them have a unique meaning (in terms of behavior of the program), although this was the goal. MISRA didn't achieve this goal and it looks as though they won't. Their efforts are well more than a decade old.

Professional Ethics

The use of an imprecise language for programming the source code of a critical system, along with a compiler which may choose any of various ways of implementing a source-code construct, as is the case with C, means that it is not effectively possible to predict how, in detail, a program will behave. If you cannot predict how a program will behave in all relevant aspects, then assurance of this behavior becomes much more difficult.

Assurance of the behavior of critical systems has come to the fore in the last decade, in particular through the influence of the international standard IEC 61508 on Functional Safety of Electrical/Electronic/Programmable Electronic (E/E/PE) Systems. IEC 61508 requires a document be prepared which contains reasoned justification that the system as built satisfies certain behavioral standards in terms of the allowable rate(s) of dangerous failures per hour of operation of the system. Such a document is generally known as a «Safety Case».

Other standards require safety cases, indeed I understand the word comes from a prior British Ministry of Defence standard which required one. Up until IEC 61508 and its adoption by such agencies as the U.K. Health and Safety Executive, safety cases were not generally required for the deployment of safety-critical systems involving software.

There is a current wave of opinion, led by prominent standards designers and senior technical authorities on safety-critical software-based systems, which focusses on assurance, and by implication the safety case, as the critical item in development and deployment of critical systems. Roughly speaking, you can develop such a system anyway you think fit, providing that you can provide a safety case: a reasoned, correct, argument that the system satisfies the conditions determined for its deployment, say in terms of rates of dangerous failures – in short, that the system is «fit for purpose».

To my mind, such considerations yield guidance on professional behavior. So, for example, I responded to Chris Hills's comment:

> .... not all languages with "sensible semantics" are available on all target platforms.

with the question:

> So it's OK to use a language with indeterminate and very variable semantics, and thus behavior, because your client said "here's the hardware, now [write] the program for me". [?]
>
> Shouldn't a professional rather say: "I'm sorry, then I cannot guarantee the appropriate behavior of programs for this hardware, and assurance requirements necessitate that I do so." [?]

The questions are rhetorical: I don't think it is OK, and I do think a professional should decline to work on a software project for which heshe cannot provide appropriate assurance. Indeed, I had responded to Jeff Payne's comment:

> Because some of us do not have executive authority over the projects we work on.

with the observation that:

> that is a question of professional integrity which I would refer to colleagues [Felix] Redmill, [Martyn] Thomas and [John] McDermid [who have been organising workshops in Britain on professionalism in software-based safety-critical system

> engineering, in the aim of eventually establishing a professional engineering organisation with this focus]. Maybe professionals should not work on projects for which they are not provided the means for appropriate assurance?

Martyn Thomas amplified:

> I suggest that the ethical principles laid down by professional institutes is a good starting place to answer that question.
>
> Here's where the Royal Academy of Engineering's **Statement of Ethical Principles** can be found: http://www.raeng.org.uk/societygov/engineeringethics/principles.htm
>
> There's also a survey there, and here: http://tinyurl.com/lz432j about ethics in industry, which I would encourage people on this list to complete. I doesn't take more than a minute or three.

The introduction of ethics, professionalism, professional behavior, or whatever one wants to call it into the discussion led to two divergent themes. One theme concerned the centrality of assurance to professional behavior. The other theme concerned whether there was consensus over the use of formal methods.

Adoption of So-Called «Formal Methods»

Jeff Payne responded to my suggestion that a professional should not work on projects for which heshe can not guarantee appropriate assurance:

> To claim that not using formal methods ..... is unethical implies that there is universal - or at least near universal - acceptance of the effectiveness of t[hose] method[s]. With none of the predominant standards requir[ing] formal methods for most levels of integrity, [it] seems that there is significant disagreement on this point among the domain experts
> participating on those committees.

And he amplified:

> The authors of IEC 61508 define "appropriate assurance" for SIL3 in terms that do not require (but recommend) formal methods.
>
> The authors of DO 178-B define "appropriate assurance" for Level A in terms that do not require (but recommend) formal methods.

Ross Hannan clarified:

> This is not correct. The authors of DO-178B neither required formal methods nor recommended them. Keep in mind those authors wrote DO-178B in the late 1980s when they, perhaps, did not fully understand formal methods - and were bold enough to admit it. What those gallant folk did was leave a little hook to formal methods - nothing else.

as did Brian Reynolds:

> Formal Methods are address[ed] in paragraph 12.3.1 of DO-178B as [one of] "Alternative Methods" and are otherwise unaddressed and so are neither required or explicitly recommended (this was due to time constraints in the committee working groups so anything which could not be worked on but seemed like it might be appropriate got dumped under "alternative methods").

to which Jeff Payne replied:

> this confirms my key point: there is still not a clear consensus that FM are fundamental to "appropriate assurance"

and Ross Hannan agreed that :

> .... there is no clear consensus that FM are fundamental to "appropriate assurance" (whatever that means)

Ross amplified on DO-178B that:

> It would generally fair to say that the authors did not have the time to bottom out particular issues. However, it should be remembered that (a) DO-178B is, with some tiny exceptions, an objective based document that avoids (in nearly all cases) requiring the application of specific techniques ..... , and (b) Formal methods ..... were reasonably immature in the late 1980s and were very much still in the academic domain.

Brian Wichmann pointed out, in contrast to (a), that:

> DO-178B requires rather aggressive testing at Level A.

but Ross did say «in nearly all cases», so presumably would accept the MC/DC testing requirement at Level A as one of those exceptions.

Jeff Payne said what these observations meant for his argument:

> ....if it were so blindingly obvious that formal methods are fundamental to safety critical development that to not use them would be ethically questionable, then they would not be relegated to "alternative methods". This seems to be confirmed by the fact that the technique remains in the "alternative methods" section of the current draft of the standard.

Martyn Thomas replied:

> We are part of a community that has not yet built a consensus position on many aspects of the science and engineering that underpins our work. That should be seen as a symptom of the immaturity of software and systems engineering, not as a justification for maintaining a status quo that ignores both the science and the evidence. It would be a mistake to assume that standards (especially those whose committees are populated mainly by people from large companies) reflect the best available scientific, engineering and practical knowledge in the world.

Jeff Payne replied:

> I am not arguing to maintain the status quo, I am simply arguing that some have to work within the existing framework until consensus is reached.

Martyn, in reply:

> Mostly, yes - for practical reasons. But senior engineers need to be robust in saying that there are better methods and that safety is likely to be compromised if they are not used where they are practical. At some point, the risk to safety becomes so great that it is unethical simply to follow the standard - which is Peter's point, I believe. Where should we draw that line? I agree with Peter that it is unethical to use a language such as C rather than SPARK where it is practical to use SPARK,

> because similar levels of programming skill will lead to many more errors in the delivered C code that in the equivalent SPARK code, simply because proper use of SPARK will eliminate whole classes of possible error that may exist in the C code.
>
> That's one example of an ethical decision relating to a specific project. More widely, I believe that it is unethical for professionals to fail to add their weight to initiatives to improve the scientific and engineering strength of the languages, methods and tools that are available and practical on our projects. As you say you agree - the status quo is unacceptable.

This firmly identifies the topic of which technologies one uses as an «ethical» issue. Brian Reynolds addressed that:

> [one could derive from this discussion a view that] anyone who uses C .... on any program which has any aspect of hazard to it is now considered to be unethical..... [it follows that] a call to immediately ground all aircraft, both civil and military should be next.....
>
> My response..... :a poor craftsman blames their tools. .... my experience [shows me that it] is not how we create the software, it is what requirements we implement that is the real issue. Pointing at C and saying "Bad, Bad, Bad!" is, as would be said in the Navy - trying to push a wet rope. It's here, it's in wide use, it's not going to go away anytime soon....
>
> A more appropriate discussion is simply to ask "How were the system requirements allocated to software and hardware validated, and how will each be verified?" .....

Martyn points out in reply:

> I'm really not saying anything so extreme, merely using C vs SPARK to illustrate the more general issue that there are ethical decisions in the decisions that we take about our methods and tools. I agree, wholeheartedly, that the question in your final paragraph is far more important than any decision about programming languages.
>
> But let's try to fix each of the deficiencies in our methods and tools, whenever we get the chance to move forward.

I responded to Brian's suggestion that discussing requirements analysis is «more appropriate» than discussing programming tools:

> I think it is completely appropriate to ask whether the use of certain technologies enables or hinders the development of appropriate assurance that the systems built using those technologies are fit for purpose.
>
> A point which people such as I and Martyn wish to make is that, with certain technologies, with the tools for those technologies as they currently exist, it does not appear to be possible to provide appropriate assurance.

So let's look at what I take to be the main issue.

Appropriate Assurance

I proposed a principle:

> [Appropriate Assurance Principle (AAP)]: Safety-critical system engineers should not work on projects for which they are not provided the means for appropriate assurance.

> Suppose one declines to agree with the [AAP]. It follows that one thinks it is OK for SCS-engineers to work on a project for which they do not have the means for appropriate assurance.
>
> .....
>
> Let's consider the proposition "not using formal methods is unethical" [introduced by Payne in the discussion on consensus]
>
> I do believe that SCS-engineers should provide appropriate assurance of fitness for purpose of the systems they build.
>
> If delivering appropriate assurance of fitness for purpose of system component S cannot be accomplished without using mathematical assurance techniques, then it follows that anyone who does not use formal methods while developing S would be violating [AAP].
>
> If appropriate assurance of fitness for purpose of S can be accomplished without using mathematical assurance techniques, then more power to the developers of S.

I asked Jeff Payne whether he agree with the AAP. His reply:

> I don't disagree with the [AAP]

which didn't sound to me too enthusiastic. (Although Jeff has subsequently indicated he does agree with the AAP.) It seems as if there is a real issue here. Andy Shore explicitly disagreed:

> I disagree with your proposed Principle.
>
> To justify my disagreement, I propose the alternative [Weak Assurance Principle (WAP)]: Safety-critical system engineers should not ASSURE projects for which they are not provided the means for appropriate assurance. ..... an engineer should work towards such assurance, but not complete it (eg by signature).

and thereby raises a point about assurance which I had missed. I had offered a definition of what I take to be appropriate assurance, namely all that is required to write a valid safety case:

> "Appropriate assurance" for a proposition P is a demonstration D following the usual technical tenets of correct reasoning that P is true.
>
> If a demonstration D has been proposed, and there are good reasons for thinking that P might nevertheless not be true, then D does not count as "appropriate assurance".

The propositions P to which I am referring here are statements that a particular system (or subsystem) fulfils its safety requirements as set out by an appropriate standard, nowadays usually given in terms of acceptable rates of dangerous failure per operating hour, under some conception of «dangerous». This is a much stronger condition than «appropriate assurance» being given by, say, an individual engineer looking at a system, determining that it is «OK», and signing a document to that effect. This is the method current used by civil aerospace for certification, with a «Designated Engineering Representative» at a company being responsible for assessing whether the component built by that company fulfils the certification criteria, and signing a document to that effect. The DER process is judged to work well for mechanical-engineering components, but I would question whether it can work at present for software, of which the assessment is generally far more complex than for mechanical components, involving a significant amount of logic.

Andy Shore has indicated:

> I don't think the emphasis of what I wrote was upon the signature..... My point was about not completing the assurance, about only going as far as means allow.
>
> However, since part of this discussion regards ethical considerations, I feel the matter of signature is not really as trivial as you seem to be suggesting.

Fair enough, but I wasn't suggesting the matter of assurance through signature, which is the prevailing model of assurance, is trivial:

> [Andy's use of the word «signature»] reminded me that what is current in "assurance" in practice is not what I have been advocating on this thread. What is current is assurance by authority (signature). What I have been advocating is assurance by judged-correctness-with-evidence.

I am advocating a paradigm shift in assurance. And this resonates with some. A private correspondent wrote to me:

> I think your .... comment ".....advocating .... assurance by judged-correctness-with-evidence" is the key.

Let me follow this further. The difference between a designated engineer signing off on a development, and a designated engineer saying that heshe judges a safety case valid, is not a significant difference, as far as I see. The paradigm shift comes when many eyes, as many eyes as will, look at the safety case. One might well consider that safety cases for software-based systems should be public documents, open to all to inspect and, if necessary, criticise.

Whatever one thinks about the future of safety cases, I don't find the present so satisfying:

> I don't think anybody should be accepting the lousy safety cases that get written at the moment. I think they should be sent back to the authors, and sent back, and sent back, until either they are valid or the authors give up.

I made some predictions about what will happen in the near future with assurance:

[One prediction] The argument that "we have X years of experience producing programs for safety-critical purposes and so this system we have built will work like the others" will no longer be accepted when safety-cases become the norm. The argument will progress to one of the form "we have X years of experience producing programs for safety-critical purposes. The quality we have achieved in our various projects is shown in the accompanying table. We predict on the basis of this table, using appropriate statistical methods of analysis of our previous projects, along with the following mitigating factors for the differences, that this SW will demonstrate the following quality....."

[A second prediction] Some usable subset of C will get a formal semantics which ensures that programs written in that usable subset will exhibit completely predictable behavior in their relevant aspects. (This may well occur through [some commercial tool providers] compiling their high-level specifications to some subset of C and then needing to ensure that these are compiled to a specific unique meaning, because that is what their clients will need in order to provide assurance.)

[A third prediction] The standards of reasoning used in providing assurance will improve, at least to the point at which those of us who read such assurance documents will not be able to trash them, almost all of them, as easily as we can at the moment. (The reasoning I see at present is mostly abominable. Both deductive and inductive reasoning.)

The focus on safety cases has a specific point, namely that with safety cases «what you see is what you get»:

> whatever one may think about programs written in C which one may claim, but cannot prove, are well fit for purpose, one cannot think it about safety cases, which cannot be fit for purpose unless they are demonstrably valid.

Some Issues with the AAP

Safety-critical systems have an obviously moral aspect to them, in that if they are not built properly then people may get hurt or die. It seems to me therefore that there is a certain interest in moral philosophy which comes in to play when discussing the professional ethics of building safety-critical systems.

In the context of his disagreement with the AAP, Andy Shore raised some ethical questions worth considering:

> If a project is being conducted dangerously, for example by inadequate provision of the means for assurance, is it more ethical to walk away and let it continue to be conducted dangerously, or to participate so as to effect improvement?
>
> If most of the means of assurance are provided, but a small proportion aren't, is it ethically justifiable to refuse to assure those aspects that can be whilst reporting those that cannot be?
>
> Might it sometimes (even usually) be easier to improve the assurance of a project if you are participating in it than if you aren't?
>
> I know there can be times when one has to walk away in despair, but shouldn't that be after one has tried one's hardest to persuade them to do it safely enough?

I added to his list of questions:

> [S]uppose you, as an engineer, are asked to work on a project, but you cannot see your way towards demonstrating, while following the usual tenets of correct reasoning, that the functional part of your deliverable is fit for purpose.
>
> The difference here is: I say you should decline to work on the project. [Andy Shore says] (I take it) that it is OK to work on the project.
>
> Let me refine the example. Suppose you know there is no way that your part can be demonstrated fit for purpose by anyone, anyhow. Would you still think it OK to work on the project?
>
> What I am getting at is that work can be divided. Your company may have employed brilliant applied logicians whose job is to produce watertight safety cases for the products your company produces. And you may well realise that you cannot do as well yourself on the safety case as they can. But what if you know that, no matter hard they try, the brilliant applied logicians cannot devise a valid safety case involving your bit?
>
> What do you do then?

I think such questions, amongst others, must be considered by engineers working on safety-critical systems. Along with the appropriate choice of development methods and tools, or must also consider an appropriate choice of behavior during development. I believe this behavior should be guided by principles developed and promoted by a professional organisation, and these principles

can only effectively be formulated if safety-critical systems professionals are willing to engage in, and eventually gain expertise in, those aspects and skills of moral philosophy which are relevant to the saving or spending of lives as indirect consequences of other activity.