Correctness by Construction: Putting Engineering into Software

Rod Chapman

Praxis High Integrity Systems Limited

Health Warning…

- This is my first Keynote talk…
- Let’s hope it’s not my last…
- Goals:
  - Say something interesting…
  - Say something different from last time…
  - Get conference off to a good start…
  - Don’t be *too* controversial…

Contents

- The problem
- What's CbyC anyway?
- Static Verification and Languages
- The best bits of SPARK are…
- Signs from the outside world (mixed, bad, good)…
- Reflections on failing to sell SPARK…
- Why we still use Ada…

The problem…

- Software plays a critical role in systems all around us…

- For example, in your new car, how much software:
  - Protecting your life?
  - Enables you to drive the car in the first place?
  - Protecting the reputation of the manufacturer?

The problem…

- BUT, size and complexity are growing…
- “Criticality creep” – more and more dependence on the software for the overall system to work at all…

- “Verification by observation” (e.g. testing) is severely limited.
  - “Just test it to death” is not really a rational option…

The problem…

- Add to that:
  - Legal regulation
  - Need to generate a “safety case” or a “security case” for evaluation…
- What can we do?

The problem (counterpoint...)

- When telling people about SPARK and Formal Methods we often hear people say
  - "But there are lots of really reliable, critical systems out there that are written in language <insert language of choice>, so we don't need SPARK..."
- What's going on here?

The problem (counterpoint...)

- Observations:
  - 1. Really talented, motivated people can (and do) produce excellent results with sub-optimal languages and tools. (This approach works, but doesn't scale!)
  - 2. Many systems *evolve* ultra-reliability through *years* of use and correction, plus "patch in the field" distribution approach.
- E.g. Linux kernel – surprisingly *few* people and hundreds of releases.

The problem (counterpoint...)

- Hypothetical question:

1) You are a car manufacturer.

2) The FLASH EEPROM does not exist – you must burn ROMs and solder them into the ECU...

How would you change your software development approach?

What's Correctness by Construction anyway?

- A systems/software engineering approach that emphasizes:

- Don’t introduce defects in the first place.
- If you do introduce defects, detect and remove them as soon as possible.
- Generate evaluation/certification evidence as a natural side-effect of the development process.
- (Easy huh?)

CbyC Principles

- A big emphasis on *Static Verification* (SV) of design artefacts (not just code...)

- Or, put another way...

An independent view

“Some people argue that the easy defects are found by inspections and the difficult ones are left for testing, but I have seen no data to support this. The PSP data show that, for every defect type and for every language measured, defect-repair costs are highest in testing and during customer use. Anyone who seeks to reduce development cost or time must focus on preventing or removing every possible defect before they start testing.”

Watts Humphrey, in “PSP – A Self-Improvement Process for Software Engineers”, Addison Wesley, March 2005, page 141.

The catch with SV…

- Our ability to automatically reason about designs critically depends on the *precision* of the notation under analysis.
- Or…ambiguous languages are a really bad thing!
- Most SV tools are constrained (in efficiency, soundness, completeness, depth…) by the poor definition of the underlying languages…

And so onto programming languages…

- Imperative programming languages
  - Two main groups or “families”
    - In the green corner: Pascal
      - With seconds: Modula-[123], Ada, SPARK, Delphi, Oberon, Eiffel…
    - In the other corner: C
      - With seconds: C++, Java, C#

- (We could go back further – e.g. Algol68 and BCPL, but I’m too young… ☺)

Language design and evolution...

- Languages have evolved, with the main goals seeming to be:
  - Increased expressive power
  - “Backward compatibility”
  - “dynamic features” – e.g. OO, dynamic types, exceptions etc. etc.
- Verifiability has received little attention!
  - (Eiffel and SPARK are the odd ones out perhaps...)

Language design and evolution…

- Pascal family languages – typical properties:
  - Type system focus on problem domain, and largely independent of representation.
  - Separation of specification (“contract”) from body (“implementation”)
  - Nested lexical structure

Language design and evolution…

- C family languages – typical properties:
  - Type system focus on target-domain (e.g. bits, bytes, words…)
  - Exposure of (and implicit dependence on) representation
  - Weak (or non-existant!) separation of specification from body

Language design and evolution…

- So…why is SPARK (still) based on Ada?
  - Originally (1985ish), there was really no contest…

- "Why can't we do SPARK for X" where X = C or C++ or Java or C#?

Language design and evolution…

- The best bit of SPARK95 is…
  - Ada95!

- The best bit of Ada95 is…
  - Ada83!

- It’s actually the really basic stuff from Ada83 that makes SPARK possible at all…

Language design and evolution…

- We couldn’t do SPARK without:
  - Scalar (sub-)types. Just impossible to imagine living without these.
    - (Note John McCormick’s results on students using C and Ada…this really does seem to matter!)
    - Still *horribly* absent from Java and C#.

Language design and evolution…

- We couldn't do SPARK without:
  - Separation of specification from body.
    - Gives us somewhere to put the "contract"
    - Forces you to think in terms of abstractions, not implementation.

Language design and evolution…

- We couldn’t do SPARK without:
  - First-class composite types.
    - Let’s us avoid all explicit use of access types - a *huge* simplification for verification purposes. (Expressive power is still OK once you get used to it!)
    - You can’t really “subset away” pointers from C and its offspring – they’re everywhere!

Signs…

- We've been going to some "Non-Ada" conferences
  - Security – NSA, GCHQ, DHS etc
  - "Grand Challenge" events on programme verification
- Here are some impressions of what's going on…

Some mixed signs…

- Static Verification is undergoing a huge renaissance, mainly owing to concerns of software security and safety.

- Some major research efforts and tools:
  - Microsoft Research
  - Patrick Cousot's team at ENS Paris
  - Stanford (and now Coverity Inc…)
  - Plus many more: PolySpace, Klocwork, Fortify, SofCheck etc. etc.

Some mixed signs…

- “Annotations” (aka “design by contract”) is suddenly fashionable! For example:
  - JML for Java
  - Microsoft PreFast and SDV for C
  - Microsoft Spec# for C#
  - Splint for C

- BUT…almost everyone is “stuck with” the unsuitability of the “popular” base languages, and many wheels are being re-invented…

Some mixed signs...

- Wheels (re-invention thereof...)
- Microsoft's PreFast allows annotations to strengthen C's function prototypes:

```
void *memset (__out_bcount(s) char *p,
              __in            int v,
              __in            size_t s);
```

- Look familiar? ☺

Some mixed signs…

- There are some *very* advanced verification techniques being developed for OO languages – for example, verification of class-invariants in Spec#.

- BUT…Spec# fails to fix the lack of scalar subtypes in C#…

Some bad signs…

- There is also a tendency to attempt to re-apply “popular” language to totally inappropriate application domains…
  - e.g. “Real-Time Visual Basic”
    - OK – that’s a joke…
    - (but you all think Real-Time Java is a great idea, right?!?!?)

What about new languages?

- Almost no-one has had the nerve to try to design a programming languages *from scratch* for verification *and* bring it to industrial use:
  - Eiffel
  - SPARK
  - BitC – new language from Coyotos operating system research group at John Hopkins – watch out for this.

What about new languages?

- Some researchers advocate the dropping of imperative languages altogether, in favour of functional languages – e.g. Haskell, Standard-ML etc.
  - Basically, a good idea, but try convincing the FAA to let you put a Haskell program on an aeroplane! ☺

Failing to sell SPARK…

- Convincing people to use SPARK is *much* harder than convincing them o use Ada.
- Even if they already use Ada, it's still hard!
- How come?
- Why aren't we rich yet?
  - The technical "win" is easy…
  - We mostly lose for non-technical reasons…

(Not) selling SPARK – the top 5 excuses

- Process-ism
- Change, disruption, inertia
- Magics, wizards, snake-oil…
- Procurement/funding
- The A word

Process-ism

- “We’re CMM Level 5, so all our stuff in great.”
- “Programming languages don’t matter because our process is so good.”
- Trying to speed up code/test/debug is still pervasive.

Change, disruption, inertia

- SPARK is disruptive – it means changing many aspects of development process to be used effectively.
- This *scares* project managers.
- Doing nothing is seen as lower risk than changing your ways.
- Larger organisations exhibit massive political and process inertia.

Magics, Wizards, Snake-oil…

- A market worth several billion dollars a year…
- Most products don't deliver what they say on the tin…
- To make a lasting difference, a real change of lifestyle is needed.

- Is this software tools or dieting?

Magics, Wizards, Snake-oil…

- It's hard to differentiate oneself from the Wizards.

- "SPARK is like jazz – hard but worth it in the long run!" (Peter Amey…)
  - Telling people we *won't* instantly solve all their problems.

Procurement/funding

- In some industries, there is (currently) little pressure to do any better.
  - We have *zero* SPARK customers in medical, automotive, telecoms etc.
- Procurers write contracts that allow suppliers to deliver a defective product.
- The FAA *laughed at us* when we suggested asking for a warranty.
- "If all software is junk, we might as well buy cheap junk…"

The A word…is “Ada”

- “We don’t do Ada…”
- “We can’t hire Ada programmers…”
- “No university in Texas teaches Ada…”
  - (honestly…guess which project!)
- Recruitment focus remains on tools/technologies/languages rather than skill and domain knowledge.

Some lessons

- Mere technical strength is not enough to get beyond the early adopters.

- Packaging and presentation are really important
  - e.g. making the maths "disappear"

- Success is not the same as dominance.

Some good signs…

- But enough gloom and doom…there are some significant lights at the ends of various tunnels…

1. Customers are “coming back to SPARK and/or Ada…”
    - Some who have “flirted” with other approaches and had a bad experience.

Some good signs…

2. Some customers *really* understand and appreciate SPARK, regardless of the "A word" connection – the government security community for example. "Security has changed everything" (Watts Humphrey again).
3. SPARK is growing, *in its niche,* and the niche itself appears to be growing.
4. Lots of academics getting back into SPARK and Ada – how many GAP members now?
   - (You can teach SPARK without telling your students, fellow faculty or funding agency that you're teaching Ada! ☺)

Some good signs…

5. New projects
   - Praxis just won a major new development. We bid full-blown correctness-by-construction, formal methods, SPARK etc.
   - We won, against very stiff and entrenched competition.
   - Probably with biggest new Ada projects in the UK for many years…
   - Watch out for more news soon…

Some good signs…

- An appeal….
- The Ada community has much to be proud of…
- Let’s publish our successes more widely (i.e. not just SIGAda and Ada Europe!)
- Let’s go to SIGPLAN PLDI, POPL, SIGCSE etc. Ada2005 gives us a catalyst to do this…

Why we still use Ada…

- Because…
  - It's the right technical choice for high-integrity systems.
  - It's the right commercial choice for our business.
  - Customers (eventually) come to see the strengths of sound engineering, embodied in CbyC, Ada, and SPARK. They rarely turn back.
  - Because SPARK *is* Ada, and we wouldn't want it any other way…

The end...

Questions?