Correctness by Construction of High-Integrity Software

Rod Chapman

Praxis Critical Systems Limited

Observation

- Despite good requirements, design, protocols, crypto etc. etc, many software projects "throw it all away" through sloppy implementation practices. For example - the ubiquitous buffer-overflow.

Thesis

- The regulated safety-critical industries (e.g. Mil Aero, Commercial Aero, Rail) have been building very reliable systems for many years. How do they do it?
- The security industry may have something to learn from the safety world.

- This presentation offers a UK-centric view of this situation.

Contents

- Correctness by Construction
- The Catch...
- Languages
- SPARK
  - Design goals and features
  - Security
  - Projects & Theorem proving performance
- What's next
- Conclusions

Correctness by Construction

- See John Rushby's talk from Tuesday!
  - Let's "Narrow the Vee..."

- We can't rely on testing alone as the primary verification activity - much too expensive and risk prone.

- Also, for the most critical systems, testing can **never** generate sufficient evidence.

Correctness by Construction (2)

- A design approach characterized by:
  - Use of static verification to **prevent** defects at all stages.
  - Small, verifiable design steps.
  - Appropriate use of formality.
  - “Right tools and notations for the job” approach.
  - Generation of certification/evaluation evidence as a side-effect of the development process. E.g. for a safety-case.

Correctness by Construction (3)

- Let's focus on what's achievable **now**.
  - Real languages with real tools that are fielded in industry right now.
  - Stuff that we know works at the highest safety-integrity/evaluation levels and is acceptable to the regulatory authorities.
  - Most high-integrity systems are also hard real-time and embedded.
- This may not be "research", but some of this may be new to you - good!

The Catch...

- Our ability to perform static verification critically depends on the language or notation under analysis.
- In particular, **ambiguity** in the definition of the language severely limits what is achievable.
- Ideally, languages and notations should be as unambiguous as possible.

Ambiguity in Computing Languages

- This idea is not new…

"… one could communicate with these machines in any language provided it was an **exact** *language …"*

"… the system should resemble normal mathematical procedure closely, but at the same time should be as **unambiguous** *as possible."*

Ambiguity in Computing Languages

- This idea is not new…

"… one could communicate with these machines in any language provided it was an **exact** *language …"*

"… the system should resemble normal mathematical procedure closely, but at the same time should be as **unambiguous** *as possible."*

Alan Turing (1948)

Ambiguity in Software Engineering

- Unfortunately, ambiguity plagues us at every turn:
  - English requirements
  - UML and other “OO” notations
  - Programming languages
    - Does anyone understand C++ Templates?!?
- Machine code is often the first unambiguous representation we get, which can be tested but not much else…oh dear…

Programming Languages…

- Standard languages? C, C++, Java?
  - All fall down on ambiguity and therefore verifiability.
  - "Modern" language design is going the wrong way! E.g. OO polymorphism, exceptions etc.
- Special purpose languages?
  - Ever heard of "NewSpeak"? Nope…

Programming Languages...

- High-Integrity Language subsets?
  - Potentially combine the best of both worlds: desirable properties for H-I, using standard compilers, tools, staff etc.
  - Integrity achievable critically depends on selection of base language.
  - For the highest integrity levels, subsetting alone may not be enough. Addition of **annotations** to strengthen the language ("design by contract"™) may be required.

So...What is SPARK?

- The “SPADE Ada Kernel”
  - What does the “R” stand for?
- A sub-language of Ada95 with particular properties that make it ideally suited to the most critical of applications:
  - Completely unambiguous
  - All rule violations are detectable
  - Formally defined
  - Tool supported
- SPARK facilitates Correctness by Construction

SPARK Design Goals

- Logical Soundness
- Simplicity of Language Definition
- Expressive Power
- Security and Integrity
- Formal definition
- Verifiability
- Bounded Space and Time
- Verifiability of Compiled Code
- Minimal Runtime Library

SPARK Features

- Base language: ISO-8652:1995 Ada95
- Removes: Tasking, Generics, lots of tricky stuff…
- Limits: Some control flow structures, visibility rules etc.
- Adds: a language of annotations to allow efficient and deep static analysis, including information-flow analysis, and mathematical proof of program properties.
- Tool support: The SPARK Examiner, Simplifier and Checker

SPARK Features (2)

- SPARK is **statically free** from all
  - Aliasing
  - Function side-effects
  - Erroneous behaviour
  - Implementation-dependent behaviour

- These analyses are all decidable in polynomial time. i.e. tool is very fast! This enables **constructive** use.

Static Analysis of SPARK

- The Examiner tool implements a number of analyses, again all in P-Time:
  - Subset checking and static semantics
  - Information flow analysis
  - Verification Condition Generation - allows proof of properties such as exception freedom, partial correctness, and safety properties.
- Theorem prover tool (the Simplifier) does a good job of proving VCs.

Exception freedom

- Exception freedom proof - why is it important?
  - Can be attempted without a formal spec., or explicit pre- and post-conditions, so is approachable.
  - Provides evidence that compiler-generated checks can be turned off with justification, or left on for "belt and braces."
  - Forces you to really think about your code. Correctness emerges.
- You mainly need CPU cycles for theorem proving - and these are cheap.

SPARK and Secure Systems

- SPARK has many properties that make it ideal for the implementation of secure, embedded systems:
  - No data-flow errors. A subtle and possibly covert source of information flow.
  - Verification of required information flow. Very useful to support system and software partitioning.
  - Proof of the absence of exceptions. Virtually free given theorem proving, and very worthwhile.
  - SPARK can be compiled with absolutely no COTS run-time library or operating system. No acquisition or evaluation problem!

SPARK and Secure Systems (2)

- Ironically, SPARK was pretty-much invented by the security community:
  - 1977 Denning/Denning paper on information flow analysis.
  - Later work at UK DERA Malvern and CESG.
- SPARK "diverted" into the safety world in about 1990 - it's about time it came home!

SPARK Projects

- Military Aerospace:
  - EuroFighter Typhoon - nearly all critical systems are SPARK - about 5 Million lines of code.
  - Harrier II SMS. Partly specified in Z and 100% implemented in SPARK. Approx 5000 VCs discharged in proof work.
  - SHOLIS - First Def Stan 00-55 SIL4 project. 9000 VCs proved, including top-level safety-properties, partial correctness, and exception freedom. 200 pages Z spec.

SPARK Projects (2)

- Commercial Aerospace: LM C130J Mission Computers and Bus-Interface units.
  - Dual cert to DO-178B Level A and 00-55.
  - Latent defect rate of SPARK code found to be >10 times better than any other software on the aircraft.
  - Proof of partial correctness (against Parnas tables) and exception freedom for core functions - about 40 kloc.

SPARK Projects (3)

- Security:
  - The MULTOS CA. (See last year's HCSS…)
  - All Praxis-generated deliverables to ITSEC E6.
  - Formal Security Policy in Z
  - Functional spec in Z (500 pages)
  - Concurrency design in CSP + Model Checking
  - 100,000 lines of code (mixed-language), 3500 person-days, 27 loc per day.
  - Only 4 defects 1 year after delivery, corrected under our warranty of course!

Some performance data for the theorem prover

- These figures are for discharging the VCs for exception freedom for 3 programs:
  - The SPARK Examiner
  - SHOLIS
  - "Project R" - a SIL3 stores management system

Performance data (December 2002)

Examiner 6.1, Simplifier 2.07, running on 1.3GHz Athlon, Windows 2000. All runtime-check VCs generated (including Overflow_Check).

Test Set	Examiner	SHOLIS	Project R
Executable loc	56760	16388	22968
Analysis & VCG time	4 mins 58 secs	4 mins 34 secs	2 mins 2 secs
Simp. time	5 hours 19 mins	8 hours 14 mins	1 hours 48 mins
Total RTC VCs	20833	6741	10963
RTC VCs proven by Simplifier 2.07	19127	6088	10017
Hit rate	91.8%	90.3%	91.4%

What's next

- Distributed theorem-proving.
  - All VCs are independent, so why not use a network of N PCs?
- Tasking! SPARK now includes a deterministic, predictable tasking subset - the "Ravenscar Profile".
  - Amenable to static schedulability analysis.
- Model Checking (much further off, but looks interesting...)

So What's Wrong with SPARK?

- It's unfashionable, and British...
- "But we can't hire Ada programmers..."
- Selling an approach that slows coding is very hard.
- Fear of formality. (Don't mention the "P" word!)
- Adopting SPARK is seen as difficult.

Conclusions

- C-by-C works - we have projects and data to prove it, meeting the most demanding levels of all the toughest standards.
- Having done DO-178B level A, 00-55 SIL4, ITSEC E6 etc., we feel that CC EAL5 is well within reach.

Conclusions (2)

- Design-by-Contract in software is a good thing. Simply writing the contracts forces you to think more.
- So write stronger contracts elsewhere - in specifications, in designs, in requirements and in procurement.

Final Quote

"There is still no silver bullet, but dramatic improvements in software quality can be achieved through the rigorous and systematic application of *what we already know…"*

Martyn Thomas - the founder of Praxis.

Resources

- Book: “High-Integrity Software: The SPARK Approach to Safety and Security” by John Barnes.
  ISBN 0-321-13616-0
- www.sparkada.com
  - Information
  - White papers and publications

Praxis Critical Systems Limited
20 Manvers Street
Bath BA1 1PX
United Kingdom
Telephone: +44 (0) 1225 466991
Facsimilie: +44 (0) 1225 469006

Website: www.praxis-cs.co.uk, www.sparkada.com

Email: rod.chapman@praxis-cs.co.uk