Safety Assurance of Commercial-Off-The-Shelf (COTS) Software

Dr Peter Lindsay & Dr Graeme Smith

COTS = standard commercial software developed without any particular application in mind

but much of talk also applies to any “Non Development Item” e.g. reused components, GOTS, SOUP,… (+ hardware)
Why use COTS?

- **Cost**
  - cheaper because of economies of scale

- **Functionality**

- **Useability**
  - e.g. user familiarity

- **Tested**

- **Support**
  - e.g. training

- **“Future proof”**
  - through upgrades
What’s needed for safety assurance?

- Use of a recognised safety standard
  - e.g. Def(Aust) 5679, IEC 61508
- Safety Case: documented evidence that system is safe to operate
  - analysis and testing
- High quality development & assurance processes
  - rigour commensurate with criticality of component
3 Technical Assurance Criteria

Need to be able to:

◆ verify specified behaviour
  – plus show elimination of unspecified behaviour
◆ validate specification in the component’s operational context
  – specified behaviour is safe & appropriately robust
◆ ensure safety under change
COTS vs Safety Assurance?

- Future proof: upgrades are a problem
- Tested: unfortunately, not usually in the same operational environment (& same version)
  - anyway, tested /= correct
- Useability: ok, but operator may assume too much
- Functionality: but what about elimination of unspecified behaviour?
- Cost: when problems found, may be difficult to get them fixed
Strategies for assuring COTS software

4 broad strategies:

- Transferring assurance from another assessment
  - e.g. from one standard to another
  - requires access to design info & development history
  - may be difficult to overcome differences in approach

- Argument based on operational experience

- Protection through design
  - isolation of COTS from safety-critical functions

- Reverse-engineer a safety case
Argument based on operational experience

- Need to justify that component will be used in the same way - and in “same environment” - as that for which evidence was collected
  - usually not an option where upgrades likely, or where operational profile unstable
  - probably only applies to e.g. nuclear reactors, or operating systems

- In general, need $10^{n+1}$ hours of testing to assure $10^{-n}$ probability of failure per hour
  - accelerated life testing might apply
Protection through design

Wrappers and other encapsulation mechanisms:
- requires well understood (stable?) interface
- can use fault-injection testing on wrappers
- **BUT...** can become complex, & may not catch unintended functionality

Redundant architecture: e.g. replication & majority voting
- will not usually safeguard against design errors
- effectiveness of N-version programming debatable

Partitioning of performance & integrity: e.g.
- simplex architecture used in process control industries
- safety watchdog advocated in STANAG 4404
Reverse-engineered safety case

- The most common approach
- Requires access to design information
- Can be much more difficult than for bespoke software
  - original typically not designed with assurance in mind
    » e.g. separation of critical functions, to avoid common-mode failure
- Black-box testing with system-level fault injection and operational system testing
  - limits to what Safety Integrity Level (SIL) can be achieved this way
- What if a problem is found?
Conclusions

In summary:

- must be able to produce a safety case
  – may be “product-based” rather than “process-based”

- whole lifecycle costs
  of developing and maintaining safety cases
  for systems with COTS components,
  may outweigh any purported savings

See also John McDermid article “The cost of COTS”, IEEE Computer, June 1998