Advances In Software Technology Since 1992*

and a modest proposal for dealing with them...

John C. Knight

Department of Computer Science

University of Virginia

August, 2008

***Funded in part by NASA**

About The Title

- **Why “Advances In Software Technology”?**

  Because

  - There have been many
  - These advances are important to aerospace

- **Why 1992?** Because:
  - That was when DO-178B was published, ***16 years ago***
  - Standard reflects the technology of 20 years ago

About The Title

- **Why now?** Because:
  - Software engineering landscape continues to change
  - A lot of effort is being expended on DO-178C
- Term “software engineering” was coined in 1968

40 years ago

DO-178B around roughly ***half that time***

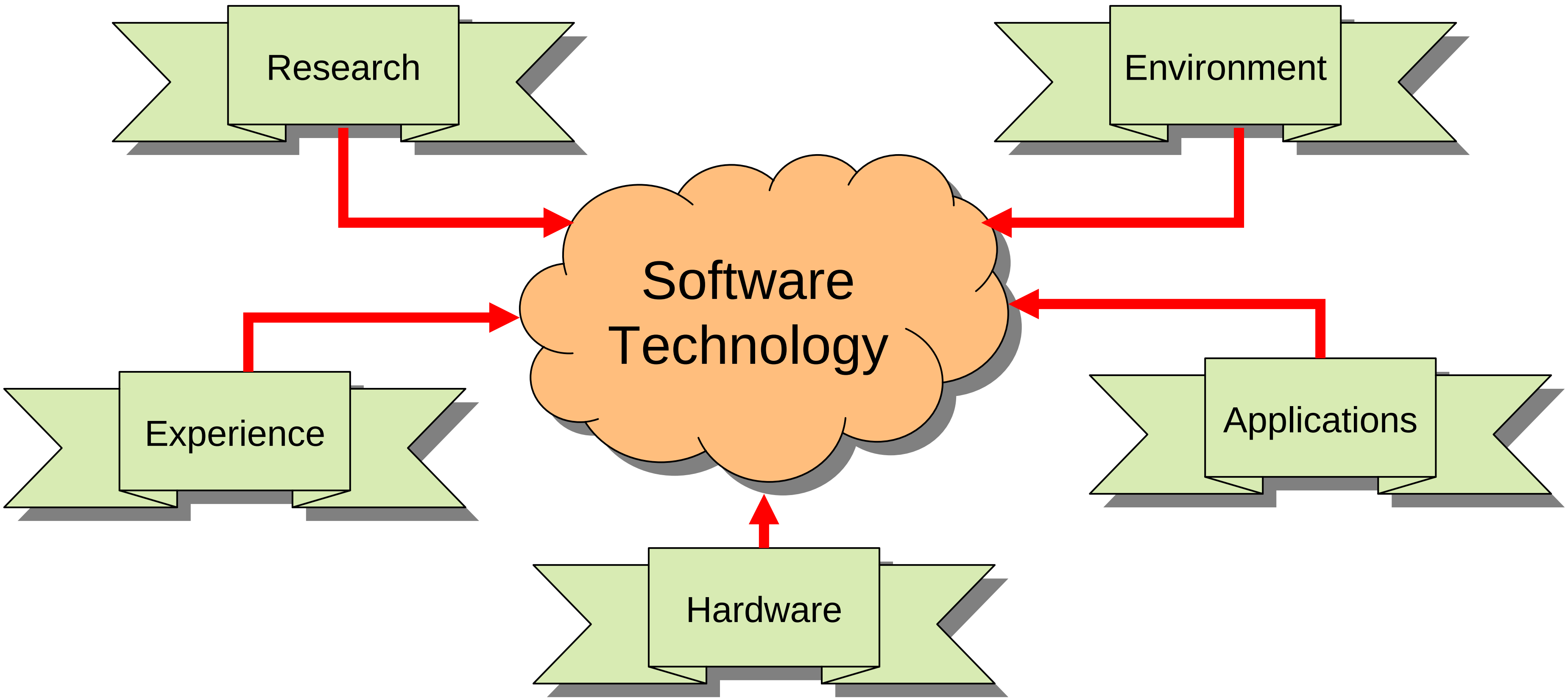
Remember, these are ***strictly*** my views

About Me *Please Forgive My Saying This*

- Why qualifies me to speak about this?
- Professor of Computer Science at the University of Virginia
  - Teaching & research on software engr. for safety critical systems
- Editor in Chief, IEEE Transactions on Sw. Engr, 2002-2005
- General chair of:
  - 2000 International Symposium on Foundations of Sw Engr (FSE)
  - 2007 International Conference on Software Engineering (ICSE)
- IEEE CS Harlan Mills Award, 2006
- ACM SIGSOFT Distinguished Service Award, 2008

Software Technology

What Affects Software Technology?



Going to look at a few sample topics

Please Note...

Talking about technology that has been ***developed***

NOT

Technology that has necessarily been widely ***adopted***

What Is The Major Challenge?

- 1992:

  Implementation defects dominated

- 2008:

  Requirements defects dominate

--- This is a *huge* difference ---

Why Has This Occurred?

Better implementation techniques

Larger and more complex applications

Don’t worry, the gene pool has not changed.

Implementation Technologies

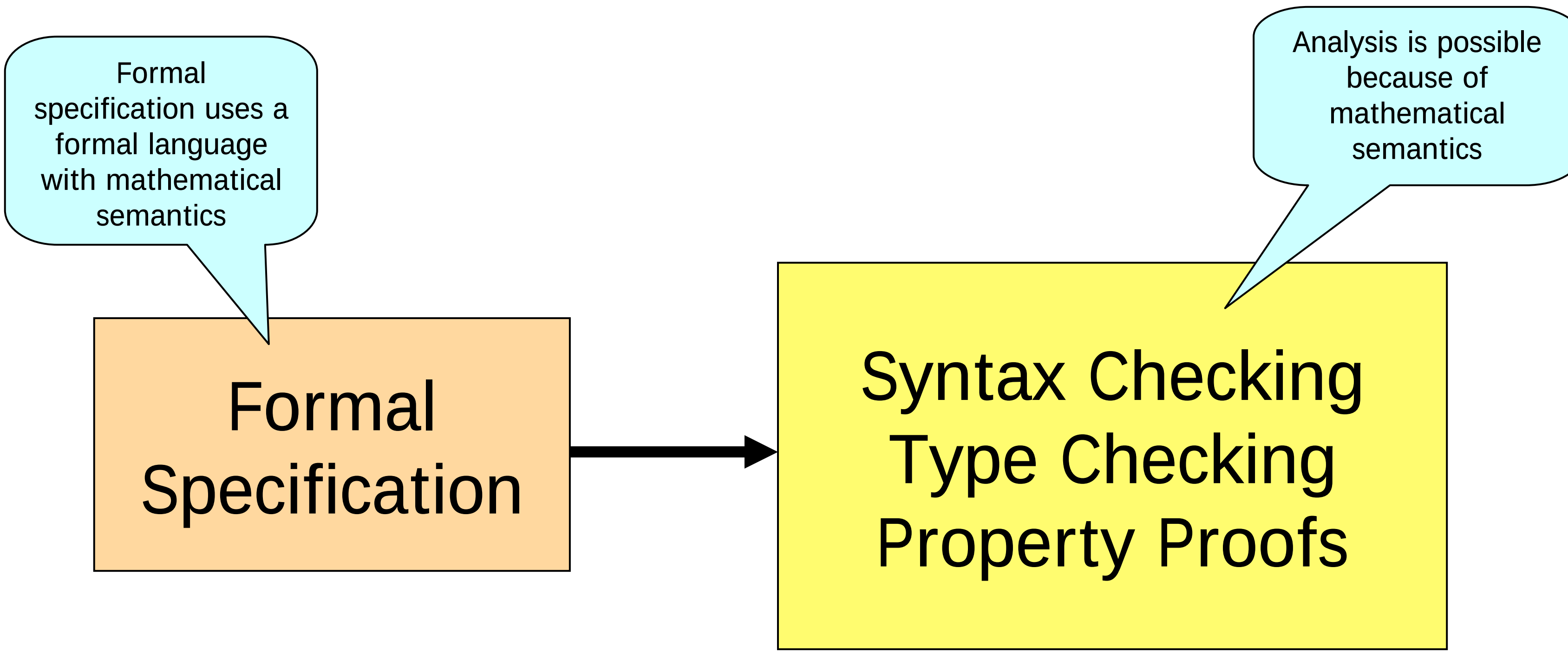
Implementation Technologies

- Practical formal specification languages, tools & techniques
- Effective software reuse
- Model-based development
- Better high-level languages
- Practical formal verification
- Model checking
- Powerful static analysis
- Better inspections and reviews
- Better software assessment techniques
- Managed development processes
- High quality COTS components

Formal Specification

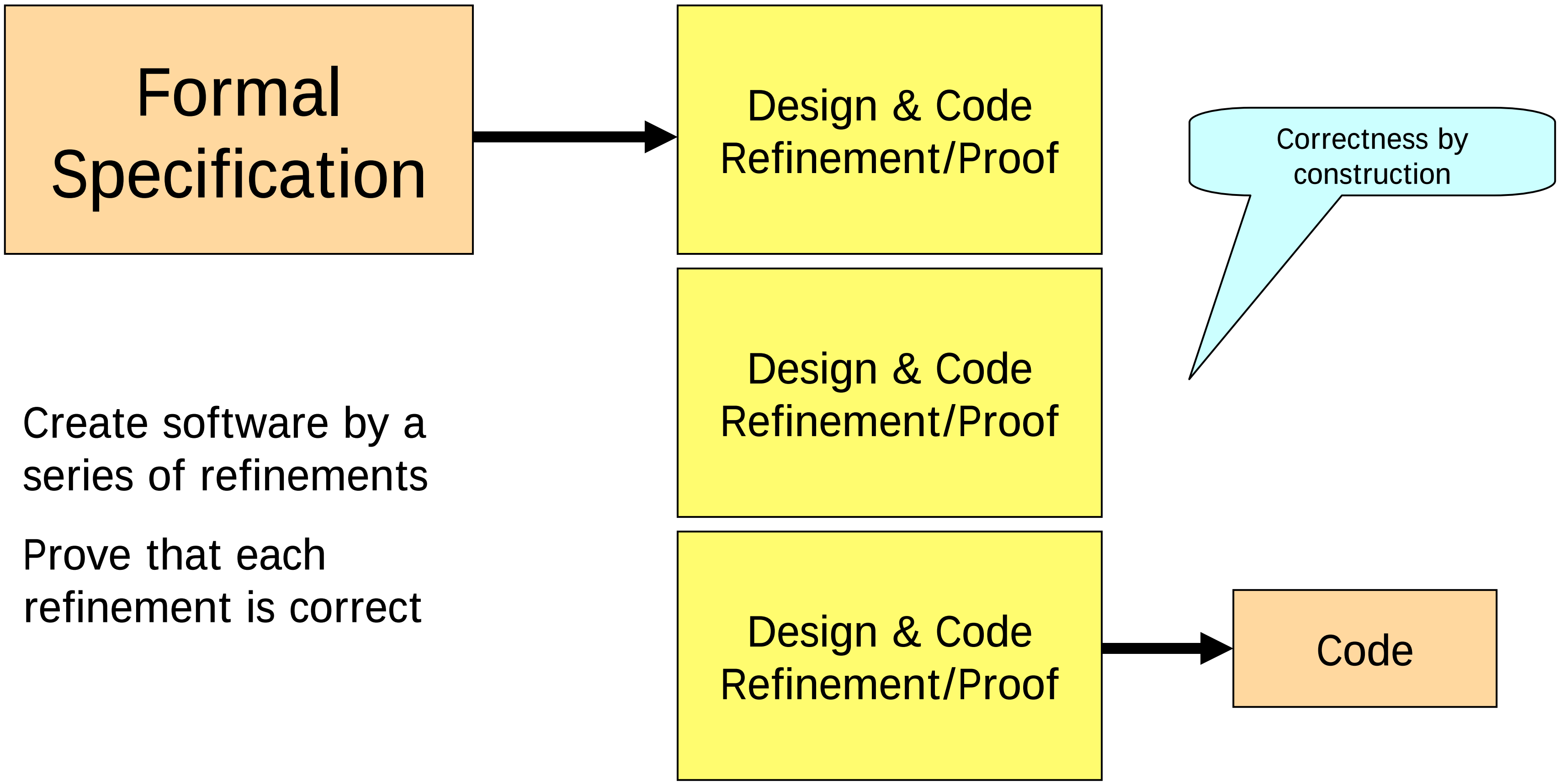
- 1992:
  - Few formal languages, mostly laboratory curiosities
  - Natural language dominated
- 2008:
  - Many formal languages
    - Z, VDM, RSML, Statecharts, PVS
  - And some narrow-domain, semi-formal languages:
    - SCADE, Simulink
  - Permit analysis and much better communication
  - Demonstrated value
  - Substantial tool support
- Many reasons to use them, ***especially in safety-critical systems***

Formal Specification



Establish useful properties of the specification

Refinement



Create software by a series of refinements

Prove that each refinement is correct

Software Reuse

- Three approaches to reuse:
  - Very high level languages
  - Application generators
  - Component libraries and canonical designs
- 1987:
  - Software Productivity Consortium
  - Reuse was viewed as a panacea
  - Still an embryonic technology in 1992
- 2008:
  - Mature technology
  - Reuse is being applied to all software artifacts
  - Important technology for cost control and quality improvement

Programming

- 1992:
  - Ad hoc, procedural languages
  - FORTRAN, C, Pascal
  - Ada '83
- 2008:
  - Pascal derivatives:
    - Modula
    - SPARK Ada
    - Ada 2007
  - es:
    - C#
    - Java
- How different are they?

Designed For Scientific

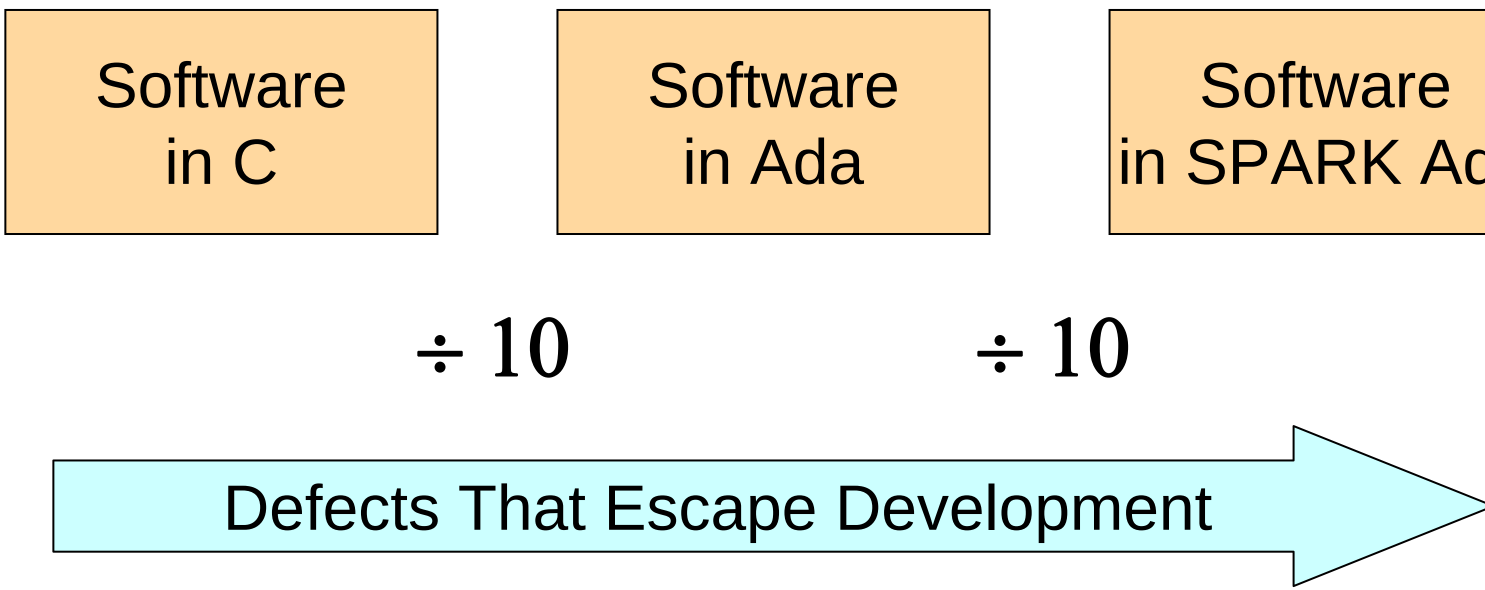
Designed For

Pro

Designed For Teaching

Designed For Embedded, Real-time, Safety-Critical Systems

Benefits Of Types & Static Analysis



Model Checking

- 1992:
  - Only just invented
- 2008:
  - In common use

- Manual model building
- Manual property development
- Automatic analysis

Model Development

Property Development

Check Model

Model Checking

- A model is:
  - A program in a modeling language
  - Describes some of the computation, typically:
    - Concurrency
    - Synchronization
    - Communication
  - A “model” of the concurrent part of the program
- Desired temporal conditions are checked, e.g.:
  - This never happens
  - This happens at some point
- Allows things like deadlock to be specified
- Defined in a temporal logic

Implementation Technologies

- Practical formal specification languages
- Effective software reuse
- Model-based development
- Better high-level languages
- Practical formal verification
- Model checking
- Powerful static analysis
- Better inspections and reviews
- Better software assessment techniques
- Managed development processes
- High quality COTS components

Requirements Technologies

Community Response

- International Conf. on Requirements Engineering:
  - Started 1993
- Requirements Engineering Journal (Springer):
  - Started 1996
- Numerous web sites started:
  - See http://www.systemsguild.com/GuildSite/Guild/resources.html
- Many tools created
  - See http://www.volere.co.uk/tools.htm
- Many important techniques developed:
  - E.g., Use cases

Formal Specification

- As noted earlier:
  - 1992: laboratory curiosity (except for CICS)
  - 2008: practical technology, fully supported
- What change has this brought?
- Analysis:
  - Syntax—we are all talking the same language
  - Types—we don’t mix apples and oranges
  - Properties—things like:
    - Input coverage completeness
    - Freedom from transitions to undesired states
- Vastly better communication and understanding

Rapid Prototyping

- Major practical advances since 1992
- Attacks uncertainty in requirements
- A prototype can be used to answer a wide range of questions, e.g.:
  - Important aspects of functionality
  - Determination of performance adequacy
  - Whether systems are acceptable to users
- Incomplete or defective requirements are not an excuse
- You can’t build if you don’t know what to build

Executable Specifications

- Literally formal specifications that can be executed
- 1992:
  - Embryonic technology
  - Laboratory curiosity
- 2008:
  - Serious capabilities with serious tools
  - Examples in narrow domains:
    - SCADE, Simulink
  - Examples in broad domains:
    - NRL’s SCR system
    - Statecharts and Statemate

Computer System Architecture

Distributed Systems

- 1992:
  - A few specialized systems
  - 1553 bus dominated
- 2008:
  - Local and wide-area networks, including real-time buses
  - Multiple advantages from both
  - Many technical issues solved

But

  - Some solutions absolutely ***require*** proof, e.g.:
    - Distributed agreement
    - Clock synchronization

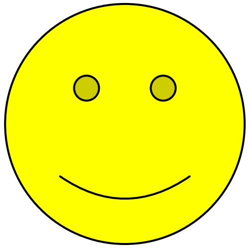
Software Architecture

- 1992:
  - Term had not been coined
- 2008:
  - Major field of practice and study
  - Powerful concepts and associated results
  - Standard patterns with important properties
  - Middleware
  - Objects at the system level:
    - .Net
    - Corba
    - Etc.

Hardware Technology

Integration Levels

- 1992:
  - Intel 80486
  - *1.2M* transistors *50* MHz clock
- 2008:
  - Intel Core 2 Extreme QX6700
  - *582M* transistors *2,930* MHz clock



- DRAM/SRAM memories ~100 times larger
- Non-volatile CF memory Not available in 1992
- Entire range of data communications equipment

Microprocessor Architecture

- Very large address spaces
- Sophisticated virtual memory structures
- On-chip large caches
- Out-of-order execution
- Sophisticated pipelines
- Multi-threaded hardware
- Multiple cores

And

- Variety of architectures and instruction sets

Hardware Dependability

- Fundamental characteristics of hardware failure have changed
- 1992:

  Degradation faults dominated

- 2008:

  Design faults dominate

  SEUs significant

  Byzantine faults significant

Impact Of Hardware On Software

- **Much** more software:
  - Many more critical applications possible
  - Introduction of non-critical applications
  - Advent of data-intensive applications
- Vastly more **complex** software:
  - Distributed systems
  - Highly concurrent systems
- Software support for hardware:
  - Management of resources
  - Dealing with hardware faults
  - Unpredictable hardware performance, esp. timing

And Finally...

Security

- 1992:
  - Security? What's that?
- 2008:
  - Security:
    - Authentication, tamper-proofing
    - Confidentiality, integrity
  - Important for airborne and ground systems
  - Going to get a lot worse:
    - Data links from everywhere to everywhere
    - Mobile devices
- Security is not an "add on", it ***has*** to be built in

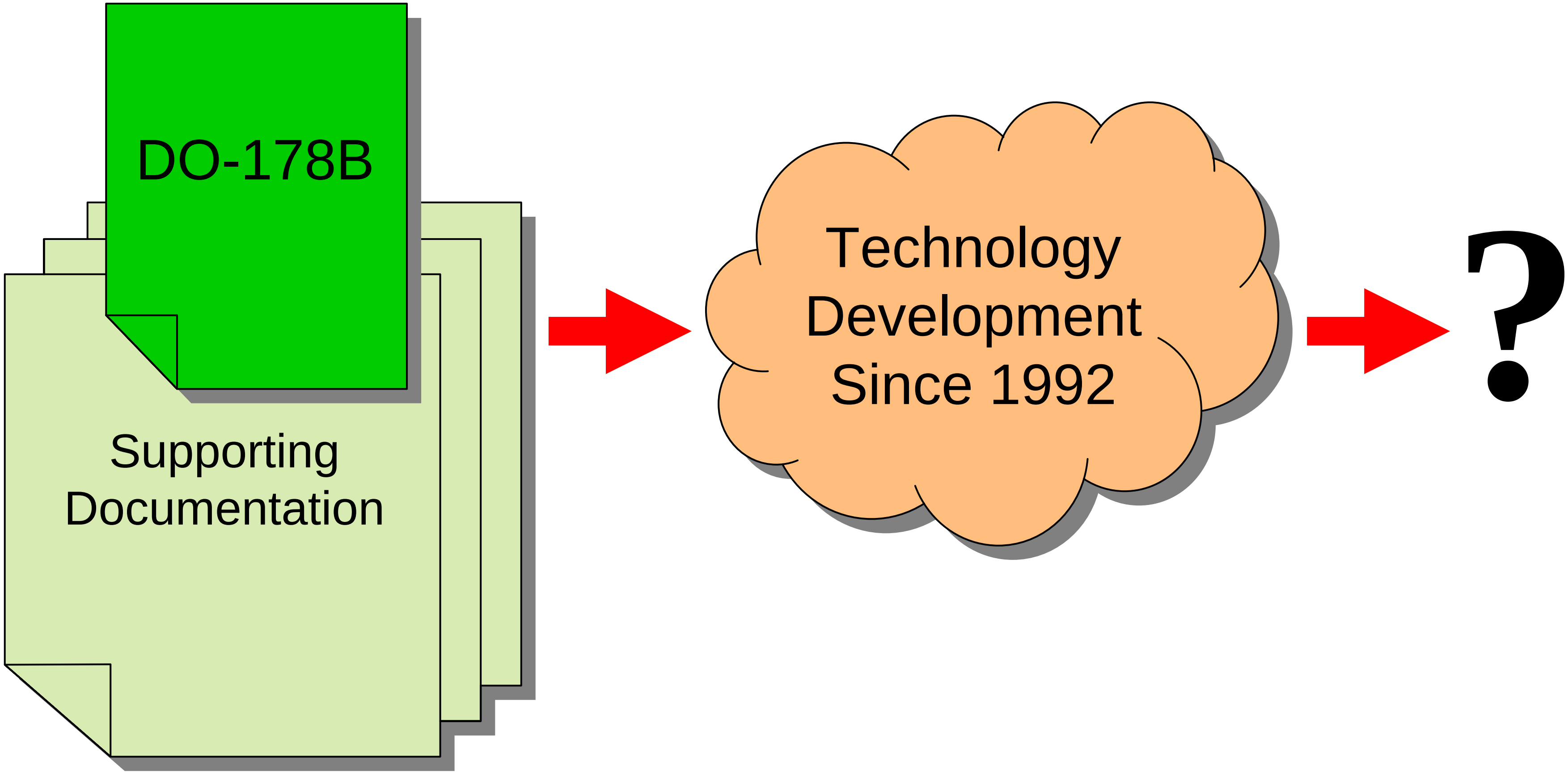
Oh No, One More Thing...

Unmanned Air Systems

An unmanned aircraft is ***not*** just a manned aircraft without a pilot.

A Modest Proposal

Enhancing DO-178B?



Challenges

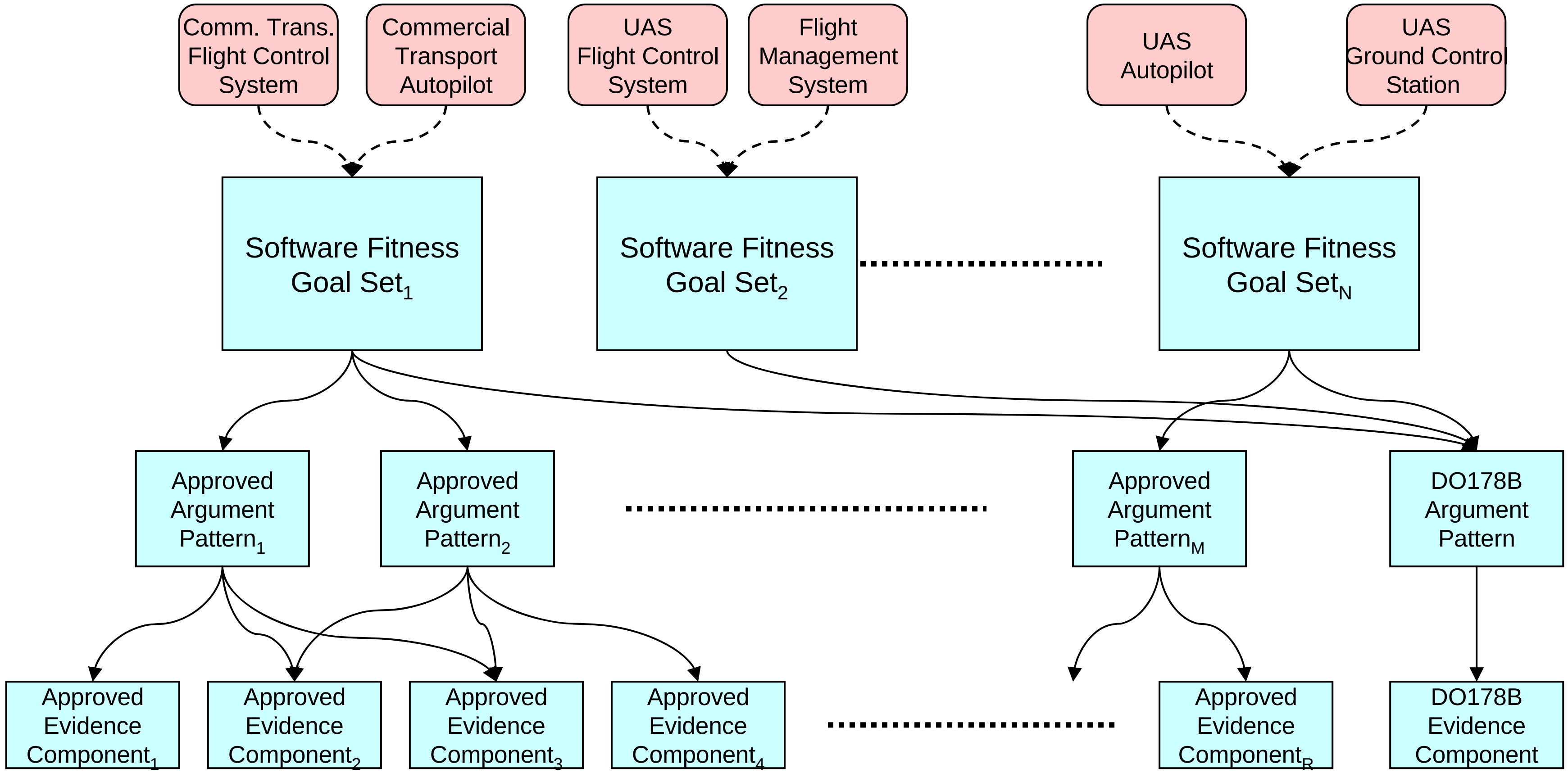
- Wide variety of systems:
  - Commercial transports
  - Unmanned air systems
  - Ground systems
- Wide variety of technologies
- Wide variety of assurance requirements
- Backward compatibility with DO-178B
- Switch in basic certification approach to rigorous argument
- Addressing the NRC report:
  "*Software for Dependable Systems: Sufficient Evidence?*"

This Is A Very Hard Problem

- Can an enhanced standard deal with these challenges?
- Some, but not all
- Cannot get a quart into a pint pot
- Any comprehensive solution faces the prospect of evolving into a “Swiss Army Knife”
- **Trying to do so, puts tremendous pressure on DO-178C**
- So, I propose DO-1743

I have a bottle of wine for the first person to figure out why it’s 1743 without a hint

DO-1743



Comm. Trans. Flight Control System
Commercial Transport Autopilot
UAS Flight Control System
Flight Management System
UAS Autopilot
UAS Ground Control Station
Software Fitness Goal Set$_1$
Software Fitness Goal Set$_2$
Software Fitness Goal Set$_N$
Approved Argument Pattern$_1$
Approved Argument Pattern$_2$
Approved Argument Pattern$_M$
DO178B Argument Pattern
Approved Evidence Component$_1$
Approved Evidence Component$_2$
Approved Evidence Component$_3$
Approved Evidence Component$_4$
Approved Evidence Component$_R$
DO178B Evidence Component

Advantages Of DO-1743

- Can accommodate all advances in software technology
- **Includes DO-178B yet compliance will be for DO-1743**
- Provision for inclusion of DO-178C once it is complete
- **Removes pressure from DO-178C to be comprehensive**
- Provides a mechanism for FAA to require certain combinations of technology for certain purposes
- Applicant can choose technology and processes suitable for the system the applicant is building

Advantages Of DO-1743

- Incorporates the modern notion of safety cases
- **Addresses the issues raised in NRC Committee Report**
- Can be applied to ground systems immediately and without modification
- Can be applied to unmanned air systems immediately and without modification
- Alignment with:
  - British MoD Defence Standard 00-56
  - U.S. FDA planned replacement for 510K

Conclusion

- The software world has changed dramatically
- Arguably:
  - The challenges cannot be met **fully** by an enhanced DO-178B
  - Many can be, so DO-178C will provide a lot of value
  - Comprehensive approach requires a new paradigm
- New paradigm is carefully managed safety-case structure
- DO-1743 is a start at the necessary framework

Contact

- E-mail address:

  knight@cs.virginia.edu

- For more information see:

  http://www.cs.virginia.edu/knight/

  http://dependability.cs.virginia.edu/