Rationale for Formal Methods

by

Ricky W. Butler

NASA Langley Research Center

May 10, 1995

Outline

1. **The Problem**
2. **Why Formal Methods Is Needed**
   (i.e. Why Standard Practice Won't Work)
3. **What is Formal Methods**

A Small Sample of Notorious Design Errors

- **Saturation of AT&T network on 15 January 1990** caused by a timing problem in a fault-recovery mechanism.
- **Failure to launch STS-1** (synchronization of backup computer)
- **Therac 25 medical radiation-treatment machine responsible for the death of two people.**
- **Loss of data from Voyager at Jupiter** (loss of clock sync)
- **Patriot failure at Dharan** (clock drift)
- **In flight tests of the X31, the control system went into a reversionary mode four times in the first nine flights** usually due to a disagreement between the two air data sources.
- **AFTI-F16—Asynchronous operation, skew, and sensor noise led each channel to declare others failed in flight test 44.** Flown home on a single channel. Other potentially disastrous bugs detected in flight tests 15 and 36.
- **X29 bug detected by simulation after 162 "at-risk" flights.** Analysis showed that the bug could have led to instability and consequent loss-of-aircraft.
- **Pentium floating point division bug**

Software Systems Are Unpredictable and Unreliable

Gibbs, W. Wayt: "Software's Chronic Crisis", *Scientific American*, **Sept. 1994, pp. 86–95.**

> **Studies have shown that for every six new large-scale software systems that are put into operation, two others are cancelled. The average software development project overshoots its schedule by half; larger projects generally do worse. And three quarters of all large systems are "operating failures" that either do not function as intended or are not used at all.**
>
> **Despite 50 years of progress, the software industry remains years–perhaps decades–short of the mature engineering discipline needed to meet the demands of an information-age society**

Software Systems Are Unpredictable and Unreliable (cont.)

Wiener, Lauren Ruth *Digital Woes: Why We Should Not Depend Upon Software*

> **Software products—even programs of modest size—are among the most complex artifacts that humans produce, and software development projects are among our most complex undertakings. They soak up however much time or money, however many people we throw at them.**
>
> **The results are only modestly reliable. Even after the most thorough and rigorous testing some bugs remain. We can never test all threads through the system with all possible inputs.**

Hardware Designs Are Unpredictable and Unreliable

Intel's President, Andy Grove, on comp.sys.intel Internet bulletin board:

> **After almost 25 years in the microprocessor business, I have come to the conclusion that no microprocessor is ever perfect; they just come closer to perfection with each stepping. In the life of a typical microprocessor, we go thru [sic] half a dozen or more such steppings....**

Michael Schrage, Washington Post Article (Dec 16, 1994)

> **Pentium type problems will prove to be the rule—rather than the isolated, aberrant exceptions—as new generations of complex hardware and software hit the market. More insidious errors and harmful bugs are inevitable. That is the new reality.**

On the brink?

Commercial aviation has an excellent safety record BUT

- **There are some signs that we are near the threshold where traditional approaches will no longer be able to deliver safe systems.**

- **The industry is rapidly adopting more and more complicated software systems, e.g. Integrated Modular Avionics (SC-182).**

A340 Incident (Near Heathrow on September 19, 1994)

- **The commander's EFIS[1] map display symbology froze and lost all computed data [..].**
- **His MDCU[2] displayed the message 'PLEASE WAIT' and a *preflight* data load page. He was unable to obtain any other display.**
- **At roughly the same time, the copilot's EFIS and MCDU exhibited identical behavior.**
- **They 'dialed in' the ILS[3] using a back-up method,**
- **Then discovered they had some 2000 kgs of fuel less than expected (an erroneous indication).**
- **The autopilot's attempt to follow the glidepath resulted in unusually high pitch rates and so the autopilot was disconnected.**
- **The commander informed the tower they were having problems with the glides-lope and requested an SRA (Surveillance Radar Approach)**
- **They landed without further incident; after taxiing in and shutting down, the fuel indications recovered.**

[1] Electronic Flight Instrument System
[2] Multi-function Control and Display Unit
[3] Instrument Landing System

AAIB[4] diagnosis

The reason for the wrong response of the autopilot and one flight director to the left turn demand was a *software error* ... This error was known to Airbus Industrie and corrective measures for this and several other software deficiencies ... have been issued ...

Post-flight analysis:

- **The BITE data dump showed that the No 2 FMGEC had detected a CLASS 1 HARD failure within itself and a simultaneous fault within FMGEC No 1, but no hardware fault could be found.**
- **Airbus Industrie was aware of the double FMGS failure mode which had first emerged on the A320 series aircraft.**
- **According to Airbus Industrie, there are several ways in which the exchange of data and/or a problem in one computer can affect the other computer.**
- **Airbus Industrie has succeeded in radically reducing the frequency of double FMGS failures on the A320 series aircraft; they are also addressing the problem on the A330 and A340 series. [...]**

---

[4] Air Accidents Investigation Branch of the UK Ministry of Transportation

Radical Change In Digital Avionics for Commerical Aircraft

- **Digital systems for commerical aircraft about to undergo a radical change**

- **The fundamental architecture of the avionics systems will move from "federated" to "integrated":**

  - **<u>federated:</u> Each aircraft function resides on its own computer. Aircraft functions are certified.**

  - **<u>integrated:</u> There is a shared computing resource on which multiple functions execute.**

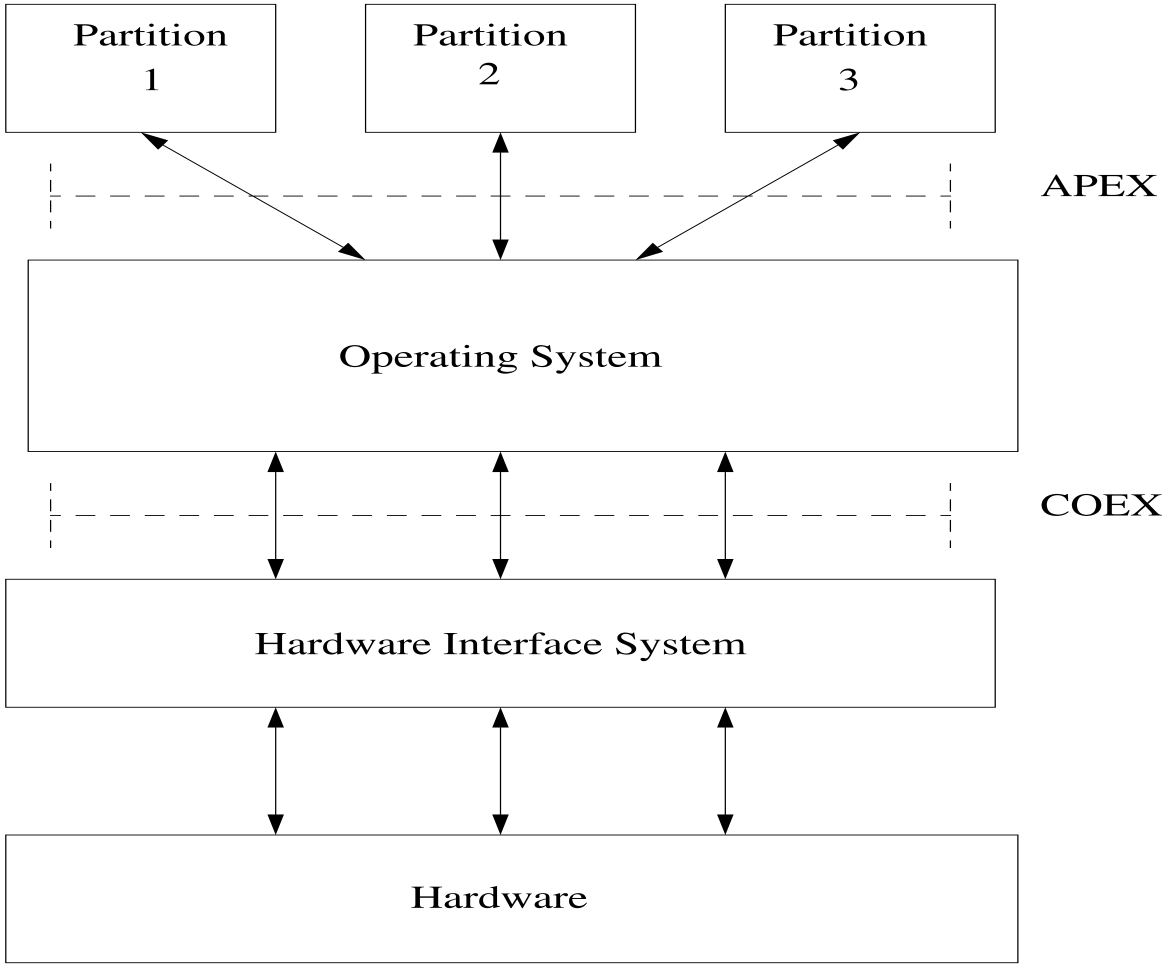
The Motivation for this Change

According to ARINC 651, this approach [i.e. Integrated Modular Avionics (IMA)] will:

- **Reduce life-cycle cost through**
  - **Reduced weight, power, and EE bay volume**
  - **Reduced unscheduled maintenance, reduced spares and increase parts commonality**

- **Reduce first cost and cost of service life support through**
  - **Reduced development, certification and production costs**
  - **Reduced avionics weight and increased payload volume**
  - **Flexibility to efficiently meet customer requirements and implement improvements**

IMA Picture


Partition
1
Partition
2
Partition
3
APEX
Operating System
COEX
Hardware Interface System
Hardware

IMA Issues

- **IMA will introduce complex operating systems into avionics.**
- **Critical tasks and non-critical tasks will reside on same hardware.**
- **Task partitioning becomes** ***life-critical*****.**

IMA needs formal methods technology

FAA Interest

Mike DeWalt, FAA National Resource Specialist For Aircraft Computer Software, encourgaged NASA Langley to pursue a research project aimed at applying Formal Methods to the partitioning problem:

> **RTCA DO-178B recognizes partitioning as a crucial element to manage complexity.... However, the FAA has not developed any supportable criteria for establishing a partition. Although partitioning has been part of the certifications of the Gulfstream 4, Airbus A320, Boeing 747-400, Douglas MD-11 and is being used extensively in the certification of the Boeing 777 and the ongoing certification of the 412CF helicopter,** *the assurance of these approvals has been non standardized and was largely based on individual engineering judgement. There is a critical need to establish supportable criteria for approving partitioning in avionics applications.*

Why is Digital Design (SW and HW) so Hard to get Right?

- **There's nothing to go wrong but the *design***
- **Our intuition and experience is with continuous systems—but in digital system's the behavior is *discontinuous***
- **Have to separately reason about or test millions of sequences of discrete state transitions**
- **Complexity exceeds our (unaided) intellectual grasp**

Classical vs. Computer Systems

Classical Systems	Computer Systems
continuous state space	discrete state space
smooth transitions	abrupt transitions
finite testing and interpolation OK	finite testing inadequate, interpolation unsound
build to withstand additional stress	build to specific assumptions
mathematical modeling	prototyping and testing
predictable	surprising

Why Formal Methods is Needed:

Three Basic Approaches to Overcoming the Design Error Problem

- **Testing (Lots of it)**
- **Design Diversity (i.e. Software Fault-Tolerance: N-version programming, recovery blocks, etc.)**
- **Fault Avoidance:**
  - **Formal Specification/Verification**
  - **Automatic Program Synthesis**
  - **Reusable Modules**

Life Testing

Basic Observation:

10^{-9} **probability of failure estimate for a 1 hour mission**

Requires

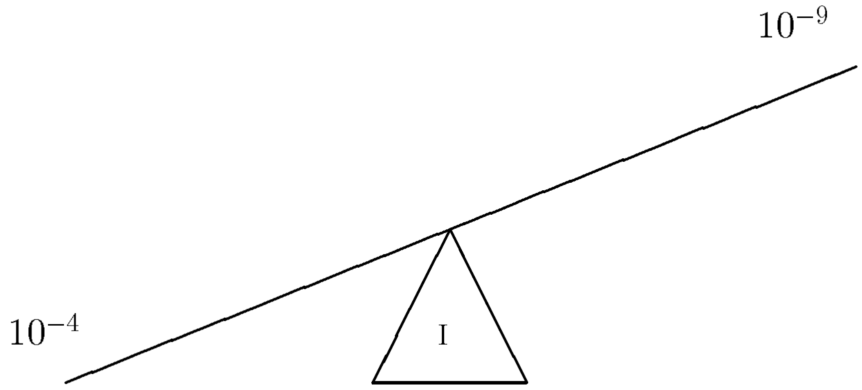
$> 10^9$ **hours of testing**

(10^9 **hours = 114,000 years)**

Design Diversity (i.e. Software Fault Tolerance)

- **Use different versions of a program developed by different teams.**
- **Vote the outputs of the different versions.**
- **Hope that the redundant programs will fail <u>independently</u> or nearly so.**

What Enables Ultra-Reliability Quantification For Physical Failure



- **The only thing that enables quantification of ultra-reliability for hardware systems with respect to physical failure is the:**

  INDEPENDENCE ASSUMPTION.

Can We Assume Independence for Multi-version Software?

- **Unlike physical failures governed by the laws of physics, software errors are the products of human reasoning.**
- **Subjective arguments have been offered for and against the independence assumption.**
- **Subjective arguments for independence are not compelling enough to assume it as an axiom.**

Independence Model

Experiments on Low Reliability Software (Knight and Leveson)

- **27 versions of a program subjected to 1,000,000 input cases.**

- **Observed average failure rate per input was 0.0007.**

- **Independence model was rejected at 99% confidence level.**

- **Other researchers have confirmed Knight-Leveson result.**

A More General Coincident Error Model?

Validating a general coincident error model would require more time than life testing a single system:

- **Coincident errors must be observed**
- **Coincident errors be shown to correspond to the general model**
- **Extremely rare coincident errors must be observed to show model applies in ultra-reliable region**

CONCLUSION:

For ultrareliable software, experiments cannot demonstrate that a general coincident error model is accurate.

FM Versus Traditional Engineering of Digital Systems

Method	**Approach**	**Confidence**
traditional	**Build according to engineering judgement, test and measure failure rate**	**Empirical**
formal methods	**Design according to fundamental principles (*axioms*) and use *deduction* for refinement**	**Rational**

Although testing methods will be used in conjunction with formal methods the *primary confidence* in this system is *not* derived from the empirical data.

What is Formal Methods

Digital System Modeling and Prediction

- **One way to predict behavior of a system is to construct a mathematical model and calculate it.**

- **For this to be effective, the model must be reasonably accurate and the calculations must be performed without error.**

- **For many continuous systems of traditional engineering, well-developed mathematical theories (e.g., Navier-Stokes equations for aerodynamics) are available.**

- **For computer systems, we must use discrete mathematics (logic, set theory) and build our own theories. Proofs of theorems take the place of numerical calculation.**

- <u>This is the essence of formal methods.</u>

Formal Methods

Formal Specification: Use of notations derived from formal logic to describe

- *assumptions* about the world in which a system will operate
- *requirements* that the system is to achieve
- a *design* to accomplish those requirements

Formal Verification: Use of methods from formal logic to

- *analyze* specifications for certain forms of consistency, completeness
- *prove* that the design will satisfy the requirements, given the assumptions
- *prove* that a more detailed design *implements* a more abstract one

- **Instead of differential equations, the properties and behaviors are described using concepts from discrete mathematics: sets, graphs, partial orders, finite-state machines, and so on.**
- **The specification may be in traditional mathematical notation or in a computer-processable language:**
  - **enables typechecking**
  - **some automatic consistency checks**
- **Provides a clear, unambiguous specification.**

Formal Methods: Verification

- **Formal Verification is more akin to proving theorems in geometry than to ordinary numerical calculation,**
  - **but like calculation it is performed according to strict rules**
  - **so one person can check the work of another**
  - **and computers can be used to automate some of the steps.**
- **The particular importance of formal methods to safety-critical systems is that they uniquely permit analysis of all the logical behaviors of a digital system [subject to some caveats].**
- **Total exploration is the only way to provide assurance that catastrophic failure does not lie hidden among the vast number of possible behaviors.**

How Is Total Exploration Possible?

Mathematical logic provides ways to reason about the properties of large or infinite collections of related things in a finite manner:

- *skolemization* **allows us to draw conclusions for all values of a given variable by reasoning about a single representative symbolic constant**

- *mathematical induction* **allows us to deduce properties for all values of some ordered domain (e.g., the natural numbers, 0, 1, 2,) by showing**

  (a) that the property is true for the least element (e.g., 0)

  (b) that when it is true for all members up to some point (e.g., n), then it is also true for the next point (e.g., n + 1)

The Advantages of Formal Methods

- **Enable faults in assumptions, requirements, design to be detected earlier than otherwise due to greater precision and explicitness early in the lifecycle.**

- **Enable faults to be detected with greater certainty than otherwise, by augmenting reviews with analysis.**

- **Can provide total coverage of selected, modeled properties.**

- **Guarantee absence of specified faults (subject to accuracy of modeling employed), because the calculations (proofs) can be checked mechanically (by a theorem prover)**

Conclusions

1. **Formal methods offers the most intellectually defensible means of producing life-critical systems**

2. **Formal methods is the application of mathematical logic to digital system design and verification**