Digital I&C Software Reliability

February 1, 2011

Gerard J. Holzmann
Laboratory for Reliable Software
Jet Propulsion Laboratory /
California Institute of Technology

Background

- **Software controls are ubiquitous and have reached safety-critical systems**
- **Code size & complexity is rapidly growing (often exponentially fast)**
- **Software test and verification methods have not kept pace**
  - **meaning: virtually all software will have latent defects**

Systems

- **Software is a *system* component**
  - **no one system component should be assumed to be perfect**
- **Building reliable systems from unreliable components requires special precautions**
  - **for software this includes self-checking code, strict partitioning, design diversity (*defense-in-depth*), and independent, non-software backup**

Failures : common causes

- **Software failures often follow a common pattern**
  - **many of these common causes can be prevented with the use of risk-based *coding standards* and strong *compliance checkers***

Failures : unintended coupling

- **Software failures in complex systems are often caused by unintended coupling between (assumed to be) independent system components**
  - **many of these causes can be prevented with the use of *model-based design verification* techniques**

Failures : race conditions

- **Software failures are often caused by concurrency (race conditions)**
  - **many of these failures can be prevented with the use of model-based *design verification* techniques**

Evidence for safety

- **Safety claims must include strong *evidence* with all relevant assumptions made explicit**
  - **this includes evidence of *standards* used, *compliance* verification and *design verification* techniques used, use of *source code analysis*, and *formal design and code verification* methods**