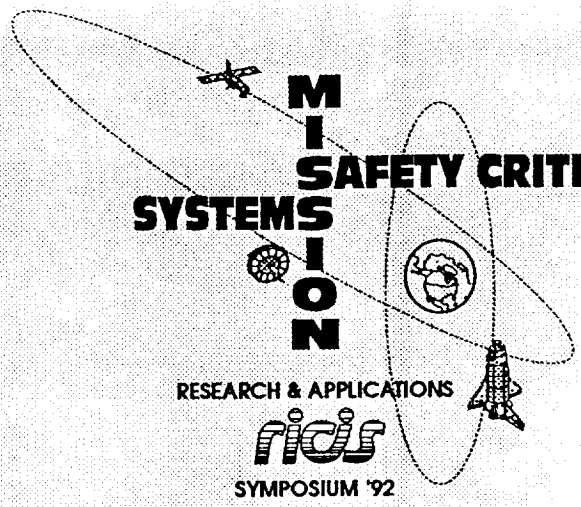



# SOFTWARE DEVELOPMENT FOR SAFETY-CRITICAL MEDICAL APPLICATIONS

*John C. Knight*

## ABSTRACT

There are many computer-based medical applications in which safety and not reliability is the overriding concern. Reduced, altered, or no functionality of such systems is acceptable as long as no harm is done. A precise, formal definition of what software safety means is essential, however, before any attempt can be made to achieve it. Without this definition, it is not possible to determine whether a specific software entity is safe. A set of definitions pertaining to software safety will be presented and a case study involving an experimental medical device will be described. Some new techniques aimed at improving software safety will also be discussed.

## BIOGRAPHY

Dr. Knight is a professor of computer science at the University of Virginia. He joined the university in 1981 after spending seven years at NASA's Langley Research Center. His current research interests are in the development of techniques to support software production for safety-critical computer systems.

# SOFTWARE DEVELOPMENT FOR SAFETY-CRITICAL MEDICAL APPLICATIONS

John C. Knight
Department of Computer Science
University of Virginia
Charlottesville
Virginia 22903

UVA
Department of Computer Science



## OUTLINE

- What?
  - What Is Software Safety Exactly?
  - A Framework Of Definitions
- Why?
  - Why Is Software Safety Important?
  - Who Should Care And Why Should They Care?
- How?
  - How Can Software Safety Be Achieved?
  - What Process, Techniques, And Tools Are Needed?
  - What Questions Remain?
- Case Study:
  - Evaluation Of Proposed Ideas
  - Safety-Critical, Medical Application
- Research Status And Plans

UVA
Department of Computer Science

# SOFTWARE SAFETY

- Public Exposure To Digital Systems Increasing - Serious Problem
- Several Standards Written Or Being Written, E.g. MoD Def. Std 0055
- Lots Of Papers On Software Safety:
  - Extremely Valuable And Important Contributions To The Topic
  - But, They Tend To Stress System Safety
- Some Important Questions:
  - Precisely When Should Software Be Considered Safe?
  - What Is The Role And Responsibility Of The Software Engineer?
  - What Is "Good Engineering Practice" In This Case?
  - Exactly Who Has Legal Responsibility For What?
- Distinguish Between *Safety* And *Reliability*, And Between *Safety* And *Availability*



# WHY IS PRECISION IMPORTANT?

- Concept Is Intuitive And Informal In General - I Know What It Means
- Something Is "Safe" If It Does No Harm - It Had Better Not Harm Me
- Precise Framework Of Definitions Is Important For:
  - Software Engineers
  - Regulatory Agencies
  - Legal System
  - Me
- Software Engineers Need To Know:
  - What Is Required Of Them, Why, And When
  - When Software Is "Good Enough"
- Regulatory Agencies:
  - Responsibility To Protect The Public - FDA, FAA, Etc
- Legal System:
  - Apportioning Blame After An Accident

## SOME TIME-HONORED ANECDOTES

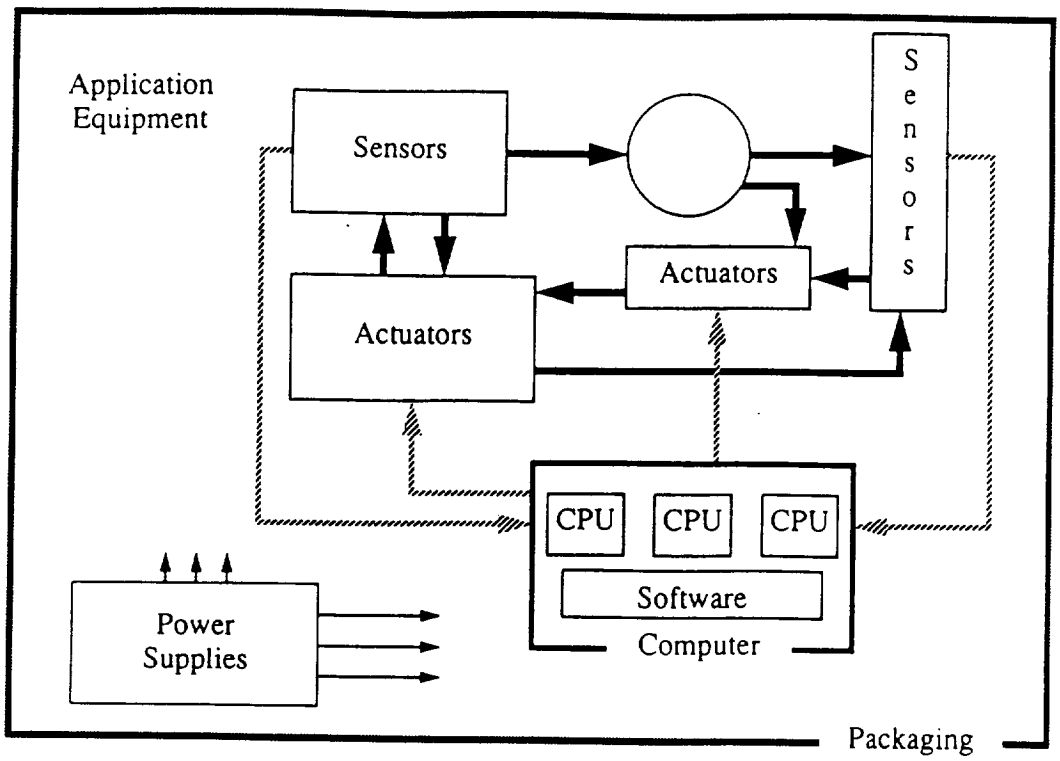
- Aircraft Landing Gear Raised While Aircraft On Ground:
  - Test Pilot Input During Ground Test, Aircraft Damaged
  - "Operational Profile or Specification In Error"
- Computer Controlled Chemical Reactor Seriously Damaged:
  - Mechanical Alarm Signal Generated
  - Computer Kept All Controls Fixed - Reactor Overheated
  - "Systems Engineers Had Not Understood What Went On Inside The Computer"
- F18 Missile Clamped To Wing After Engine Ignition:
  - Aircraft Out Of Control
  - "Erroneous Assumption Made About Time For Engine To Develop Full Thrust"
- All Are Important, Very Serious Incidents - Valuable Insight Gained
- What *Exactly* Is The Safety Issue In Each Case?
- What *Exactly* Is The Responsibility Of The Software Engineer In Each Case?



## *SYSTEM* SAFETY

- Informally, *System Safety* Is Subjective
- *Systems Engineers* Have Formalized The Notion Of Safety:
  - Definitions - Hazard, Risk, Acceptable Level Of Risk, *Safety*
  - View System As Well-Defined Collection Of Components
  - Established Practices And Procedures
- Software Researchers And Engineers Trying To Do The Same For Software
- So Far, Success Is Limited
- Within A System:
  - Software Is Merely Part Just Like Computer Hardware, Sensors, Actuators, Etc.
  - Software Can Cause Failure
  - Software Can Prevent Failure
  - Software Can Stand By And Watch Failure Happen
  - But So Can Any Other Part

# SOFTWARE SAFETY vs. SYSTEM SAFETY








# SYSTEM CONTEXT FOR SOFTWARE

- Common Observation - In Isolation, Software Is Never Unsafe:
  - True, It Cannot Be Executed In Isolation Either
  - Software Is Useless In Isolation
- Most Components In Any System Are Safe In Isolation
- In Isolation, Software Is Removed From The Notion Of Hazard:

  *This Does Not Imply That Software Safety Is Meaningful Only In The Context Of The Entire System*
- Software Engineers Are Not Qualified To Deal With *Systems* Engineering Issues
- Do We Want The Software Engineer:
  - Deciding Action For Unspecified Input?
  - Implementing Functionality That "Seems Right"?
- Hazards, Risks, Etc. Should Not Appear In The Software Specification
- The *Required Treatment* Of Hazard Must Be Present In The Software Specification

## THE ANECDOTES AGAIN

- Aircraft Landing Gear Raised While Aircraft On Ground:
  - "Operational Profile or Specification In Error"
  - *Systems Engineer's Responsibility*
- Computer Controlled Chemical Reactor Seriously Damaged:
  - "Systems Engineers Had Not Understood What Went On Inside The Computer"
  - *Systems Engineer's Responsibility*
- F18 Missile Clamped To Wing After Engine Ignition:
  - "Erroneous Assumption Made About Time For Engine To Develop Full Thrust"
  - *Probably The Systems Engineer's Responsibility*
- In General, Software Engineer Is Not Trained To Identify Hazards, Consider:

  *Computerized Flight-Control System Commands Aircraft To Flare On Final Approach At Air Speed Of 128 Knots, Height 180 Feet, 15 Knot Headwind, Throttles At 75%, MLS On, Fuel At 14%, 1,027 Feet From Runway Touchdown Point, Undercarriage Down, Flaps At 30%*
- Is This A Hazard?

UVA
Department of Computer Science



## A FRAMEWORK OF DEFINITIONS

**Definition** - Component Intrinsic Functionality Specifications

*The Required Functionality Of The Component Without Regard To Safety*

**Definition** - Component Failure Interface Specifications

*The Required Functionality That Must Be Provided In The Event That The Component Is Unable To Provide Its Intrinsic Functionality*

**Definition** - Component Recovery Functionality Specifications

*The Required Functionality That Must Be Provided In The Event That One Or More Other Component Fail*

**Definition** - Component Safety Specifications

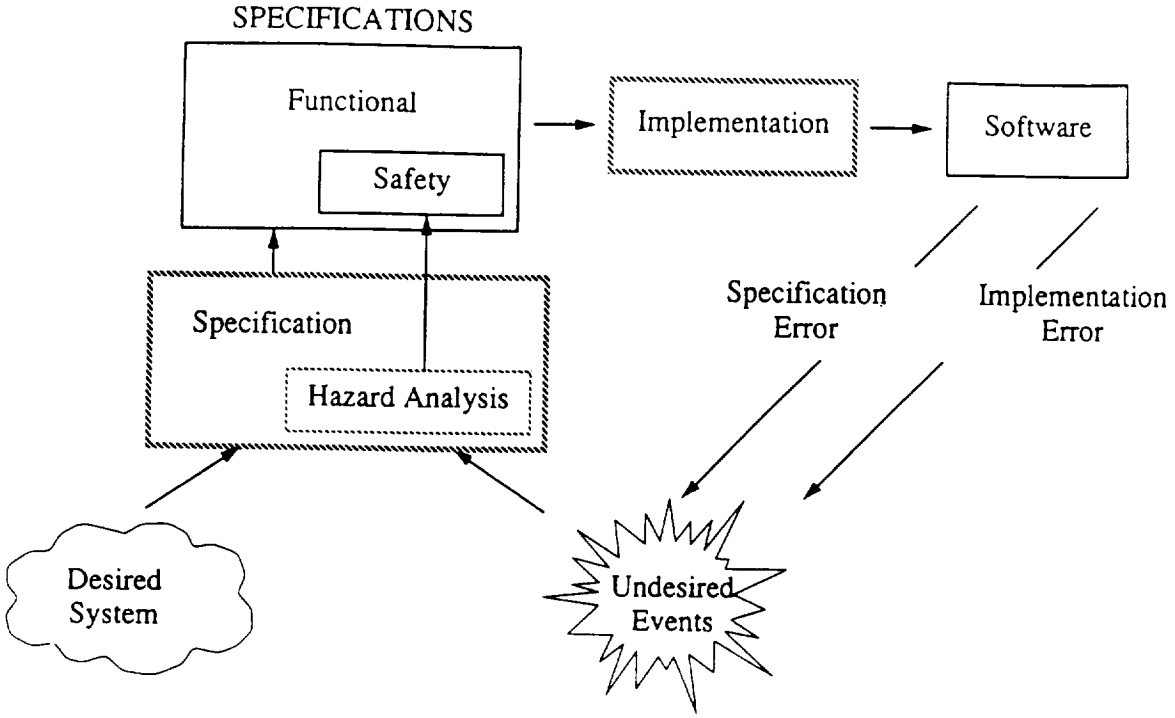
*The Component Failure Interface Specifications Combined With The Component Recovery Functionality Specifications*

**Definition** - Software Safety

*Software Is Safe If It Complies With Its Component Safety Specifications*

UVA
Department of Computer Science

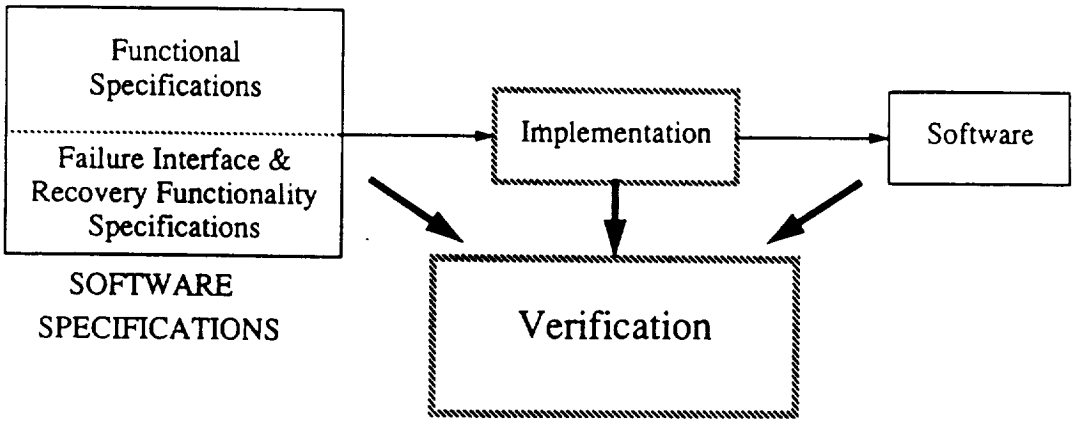
# CONCEPTUAL FRAMEWORK





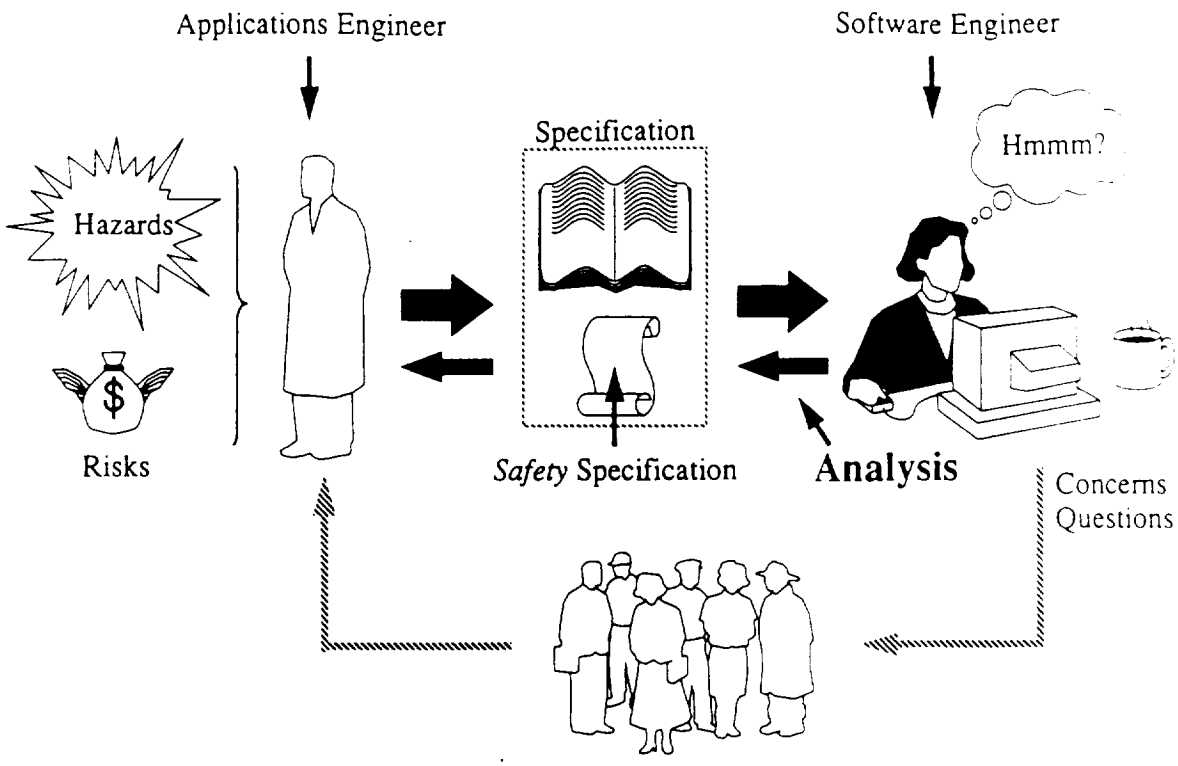



# SOFTWARE SAFETY




- Software Is Safe To The Extent That It Complies With Its Safety Specification
- Safe Software Might Fail - That Is A Subjective Issue, Formally It Was Safe
- Software Engineer's Task Now Clear
- Responsibility For Accidents Can Be Fairly Assigned

# FORMAL PLACEMENT OF RESPONSIBILITY






UVA
Department of Computer Science

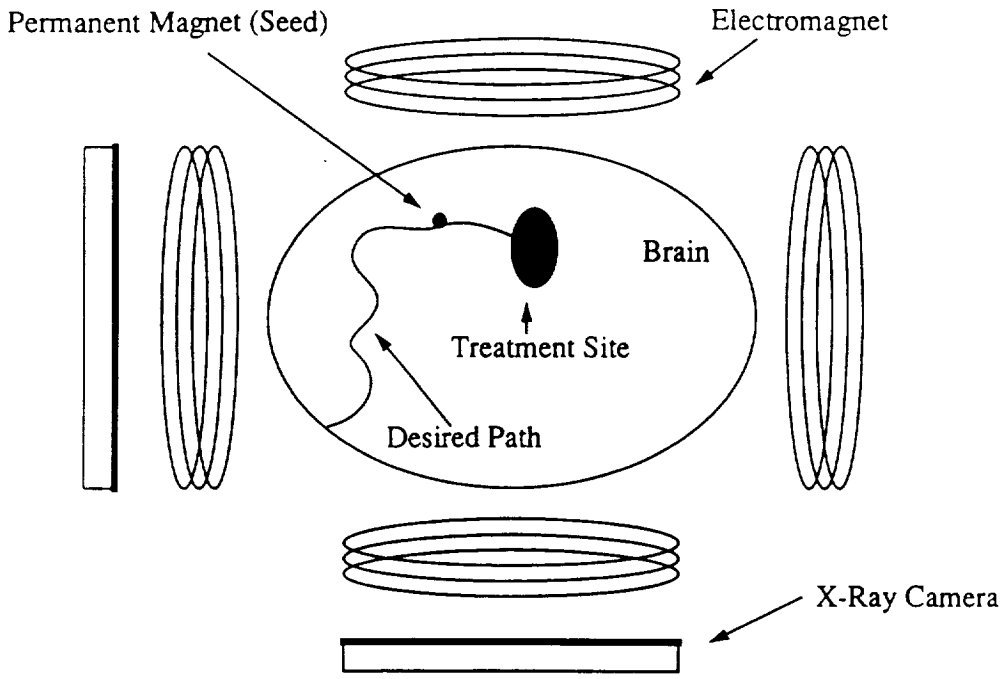
# A CASE STUDY

- Is This Conceptual Framework Useful? If Not Why Not?
- If Useful, How Can Safe Software Be Built And Demonstrated?
- Approach:
  - Case Study Based On Safety-Critical Application
  - "Gloves Off", No Assumptions, Not An Academic Study, Do It Right Or Else
- *Magnetic Stereotaxis System (Video Tumour Fighter)*:
  - Experimental Device For Human Neurosurgery
  - Complex Physical System, Clearly Safety-Critical
  - Stringent Safety Requirements
  - Minimal Reliability And Availability Requirements
- Primary Safety Issues:
  - Patient Safety
  - Equipment Safety



UVA
Department of Computer Science

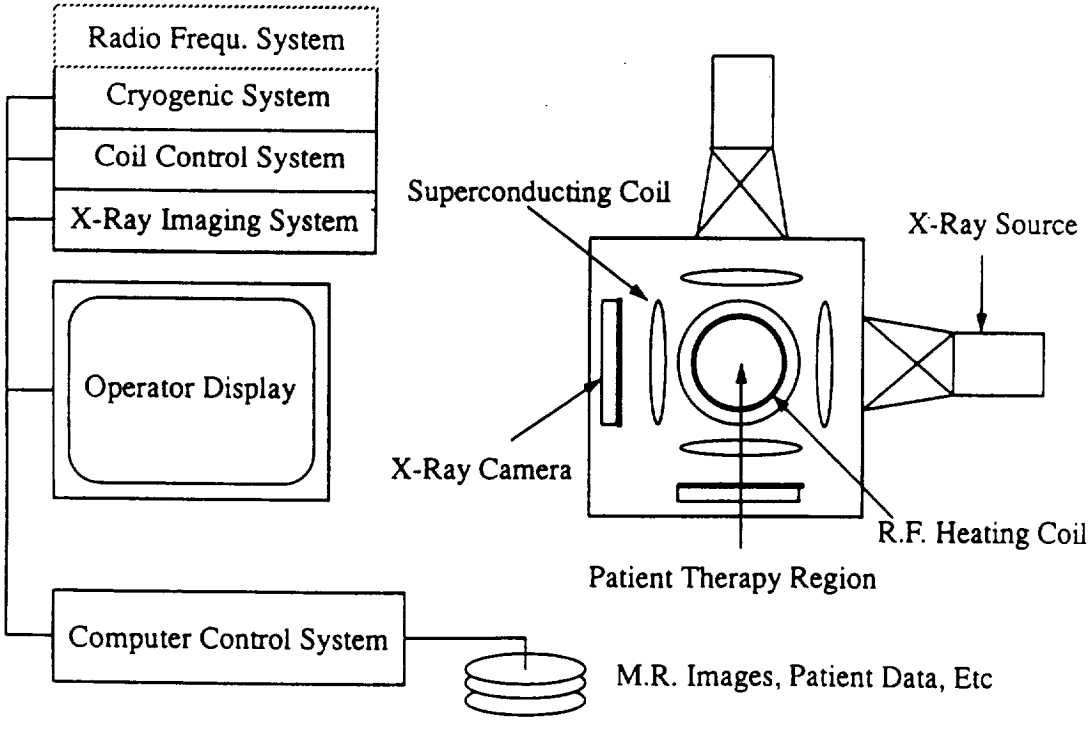
# MAGNETIC STEREOTAXIS SYSTEM - CONCEPT








# MAGNETIC STEREOTAXIS SYSTEM



## SOME OF THE MSS/VTF SAFETY ISSUES

- External Superconducting Coils:
  - Incorrect Current Calculated By Software And Applied
  - Coil(s) Fail, Incorrect Coil Shutdown Effected
  - Coil Controller(s) Fail, Incorrect Coil Shutdown Effected
  - Signals Scrambled Between Computer And Coil Controller(s)
- X-Ray Subsystem:
  - Hardware Fails On When Supposed To Be Off Or Vice Versa
  - Software Commands On Incorrectly
  - Image Defects - Ghost, Incorrect, Or ''Old'' Image Used
  - Incorrect Target Identification - Marker Rather Than Seed
- Radio Frequency System:
  - Hardware Fails On When Supposed To Be Off Or Vice Versa
  - Software Commands On Incorrectly
  - Wrong Power Level Administered



## MORE OF THE MSS/VTF SAFETY ISSUES

- Display System:
  - Wrong Seed Location Shown On Magnetic-Resonance Image
  - Wrong Magnetic-Resonance Image Displayed
  - Other Incorrect Data Displayed
- Operator Error:
  - Commands Erroneous Movement
  - Fails To Observe Error Message
- Software System:
  - File System Supplied Erroneous Information
  - Interference From Non-Safety-Critical Elements

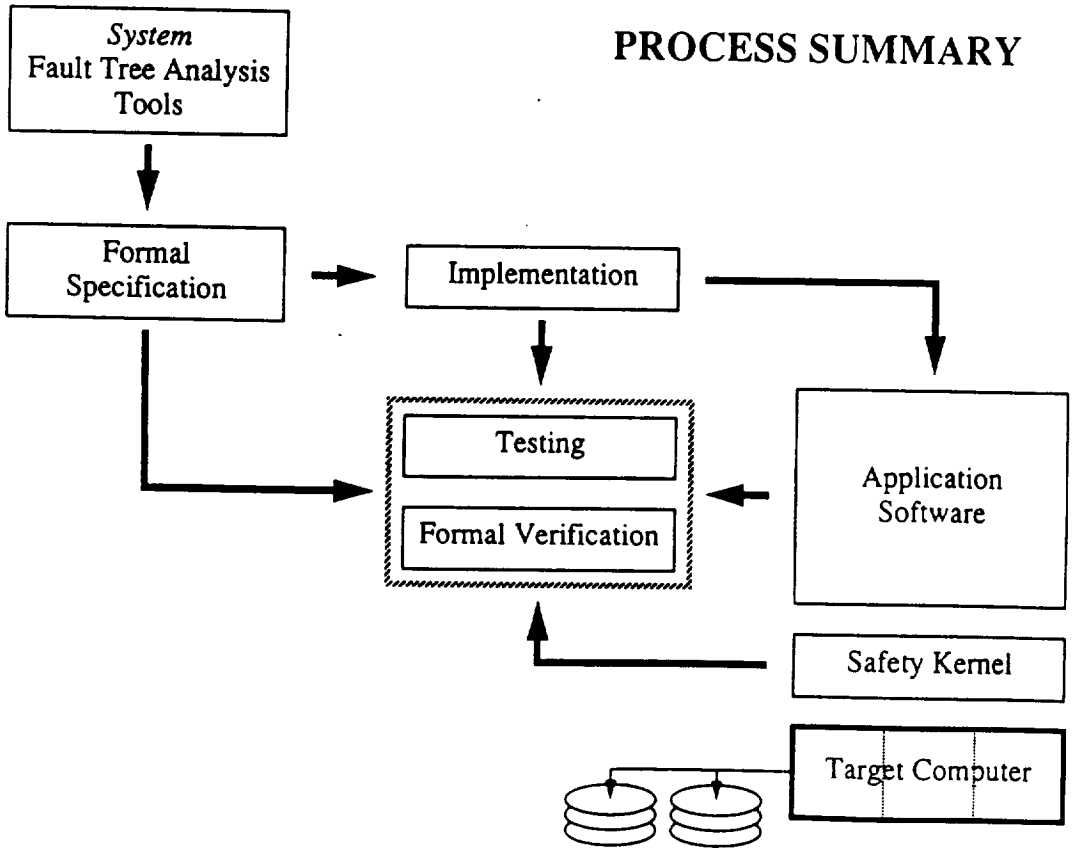
# DEVELOPING SAFE SOFTWARE

- Framework Of Definitions Is First Step:
  - Now We Know What We Have To Achieve
  - Safe Software Is Well-Defined Target
  - Also Know Who Is *Formally* Responsible For What
- How Is Safe Software Developed?
- There Is No Magic Bullet Is Specified Verification Level Is Very High
- For Safety-Critical Software:
  - No Dependable Technology Exists
  - Many Open Research Areas
  - Safety Is ''Simpler'' Than Reliability, In Many Cases More Important



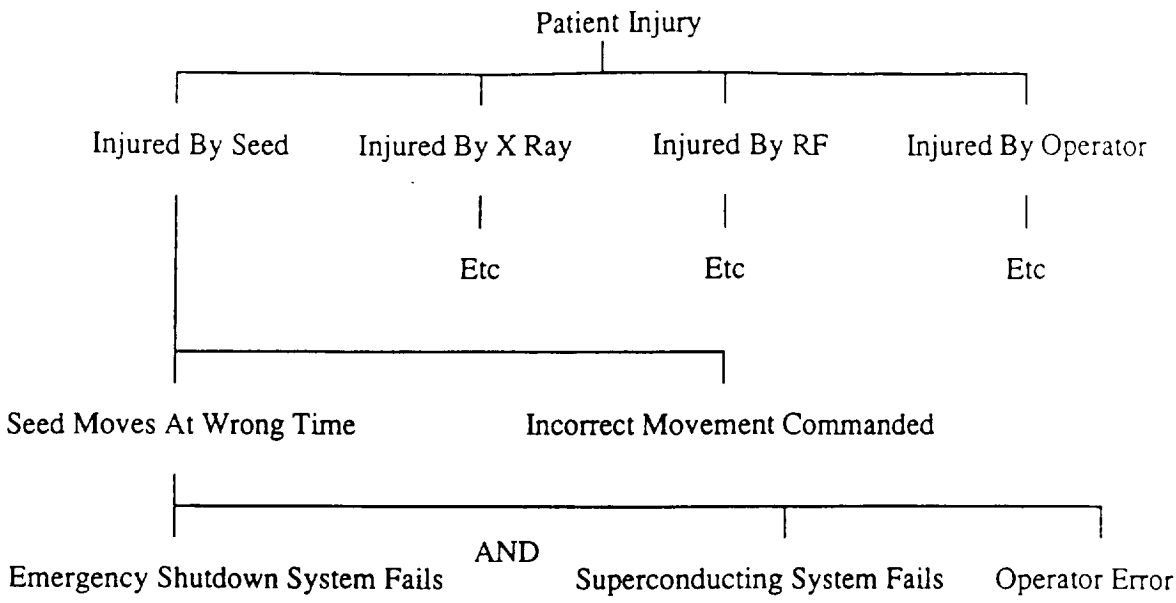


# PROCESS SUMMARY






# SYSTEM FAULT TREES





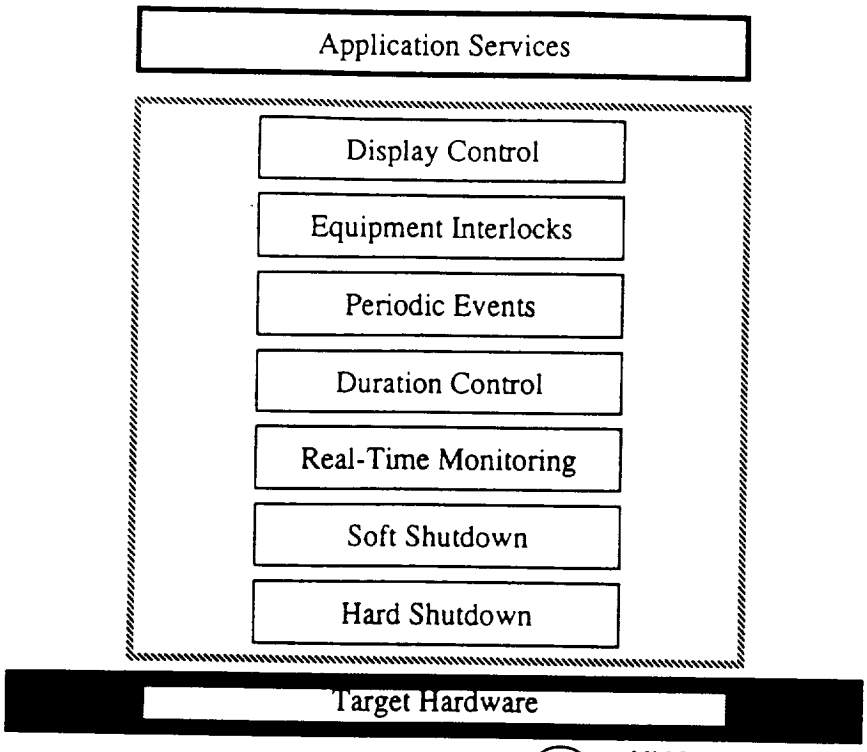



# SYSTEM FAULT TREE ANALYSIS

- System Fault Trees Are:
  - Large And Complex
  - Very Hard To Get Right
  - Involve Software In Two Distinct Ways - Software Failure And Software Response
- Tools Needed To:
  - Manipulate Fault Trees To Facilitate Software Analysis
  - Permit Rigorous Software Safety Requirements Process
  - Enable Assurance Of Adequate Coverage By Software
  - Permit Formal Software Safety Specifications To Be Derived
- Tools Concepts:
  - Build Hardware-Only Fault Trees
  - Display, Inspect, Analyze Probabilities, Etc
  - Add Functional Software Nodes
  - Mechanically Derive Software Failure Cases And Required Software Responses
  - Assist With Derivation Of Specifications And Various Property Proofs

# SAFETY KERNEL CONCEPTS

- Verifying Safety Properties Is The Single Design Constraint








# CASE STUDY EXPERIMENTAL APPROACH

- Develop System Fault Tree And Software Specifications (Drafts Completed)
  - Specifications Presently Written In 'Z' (Draft Completed)
  - Safety Specification Delimited
- Implement Complete, Non-Safe Prototype System Based On UNIX And X
- Add Facilities And Transition To:
  - Safety Kernel On Bare Hardware
  - Progressively "Safer" System
- Verify Safety Properties:
  - Exhaustive Testing - Carefully Avoiding Butler & Finelli's Result
  - Formal Proofs Where Possible
  - Rigorous Argument, Static Analysis, Inspection
- Goal - Repeatable, Dependable Process Providing Assured Software Safety
- Also, A Process That Has Been Evaluated

# SUMMARY

- "Invasive" Computer-Controlled Medical Devices Becoming More Common
- Serious Safety Requirements, Often Very Limited Reliability And Availability Needed
- Technology To Deliver Software Safety Is Elusive
- Formalization Of The Meaning Of Terms And The Role Of The Software Engineer
- Software Engineer Is *Not Qualified* For Anything But Software Engineering
- Case Study Being Undertaken To:
  - Evaluate Definitions, Process Concepts, Tools, Techniques
  - Demonstrate Workable Process With Realistic Example
  - Support The MSS Project