Parnas Tables: An Experience with Formal Verification in an Industrial Setting

Parnas Tables
An Experience with Formal Verification in an Industrial Setting

This talk describes the use of a method of formal verification generally called Parnas Tables

Method was developed by D. L.Parnas

Method was extended and made to work by OPGI staff

First use of large scale formal verification in Canada

Background - 1

- 1990 Darlington Nuclear Generating Station brought on-line
  - first time trip-decision logic in reactor safety shutdown system (SDS) entirely software
  - two independent systems (physically and logically)
  - designed differently to reduce common mode errors
    - 7000 lines of FORTRAN
    - 13000 lines of PASCAL

Background - 2

- 1987 regulator (AECB) concerns
  - not properly engineered
  - software functional but not of “high quality”
  - uncertain risk
  - lack of confidence in product, process, and people
- software was already written
  - software had been extensively tested
  - many design documents did not exist
  - those that did, not suited to a formal process

Background - 4

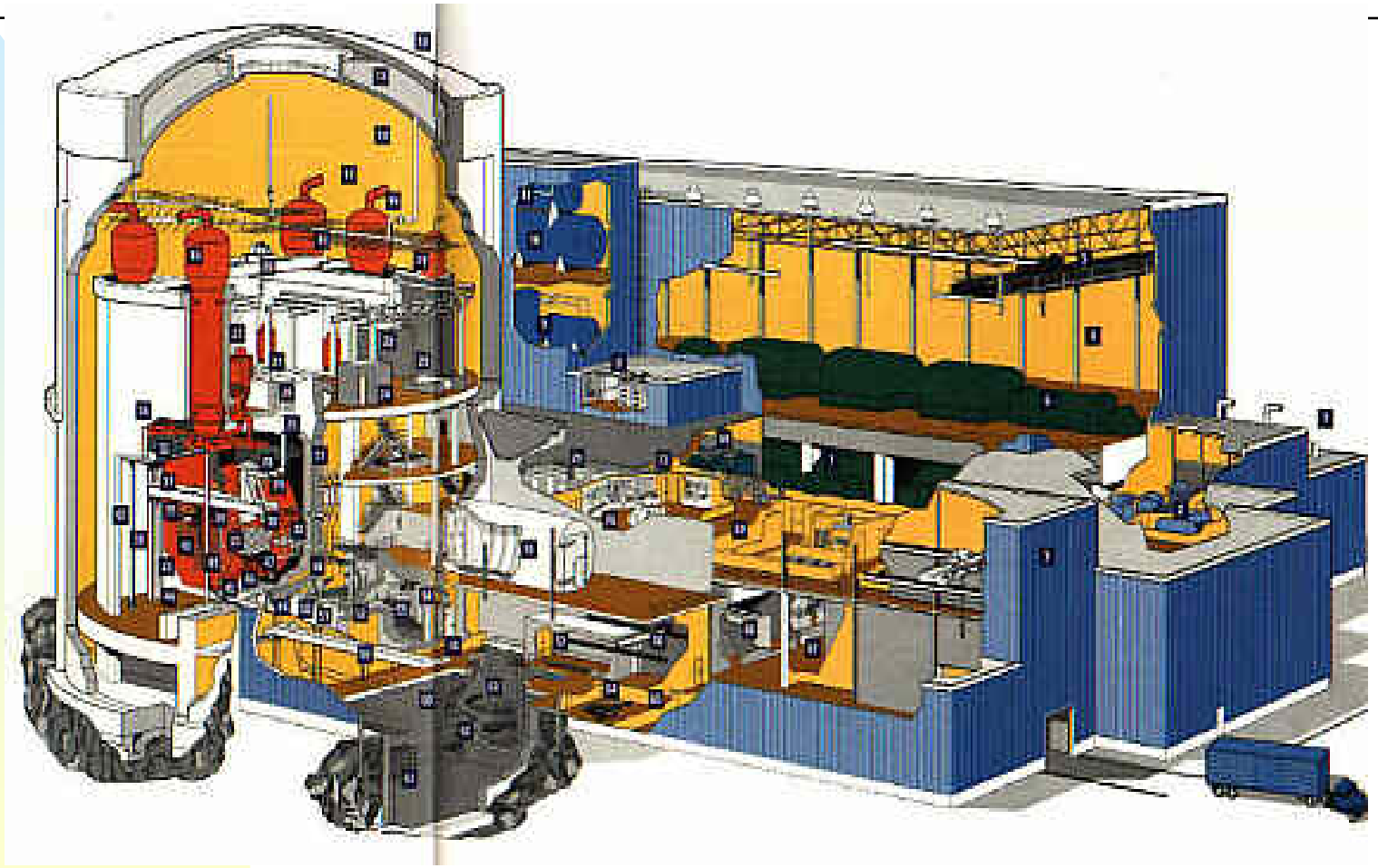
- underlying problems
  - no agreed upon, measurable definition of acceptability for the engineering of safety critical software
  - no widely accepted common practices for specification, design, verification, and testing of safety critical software
  - not possible to quantify the achieved reliability of software component of a safety system

Background - 5

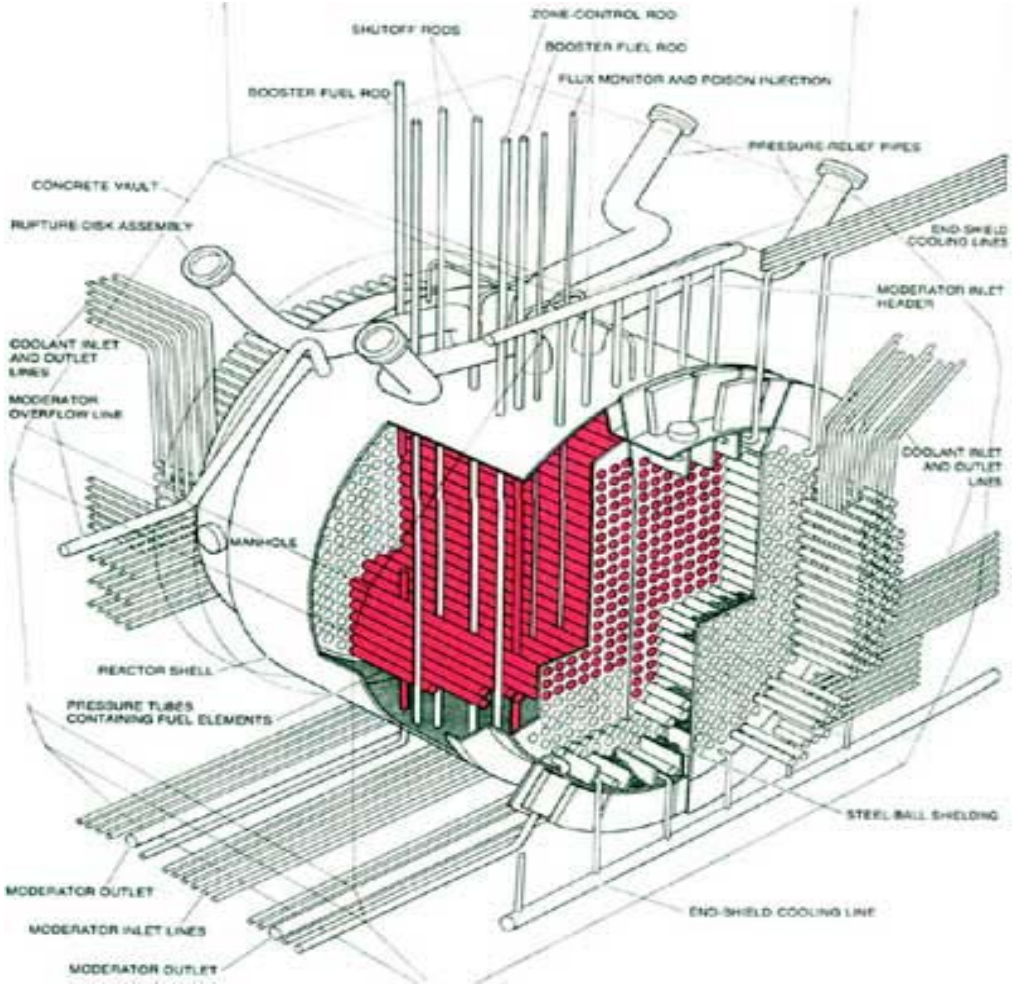
- David Parnas hired by regulator to advise on process
- procedure based on Parnas Tables
  - formal verification
  - rendered code into tabular format
  - rendered requirements into tabular format
  - proofs to show code and requirements the same

Darlington Station

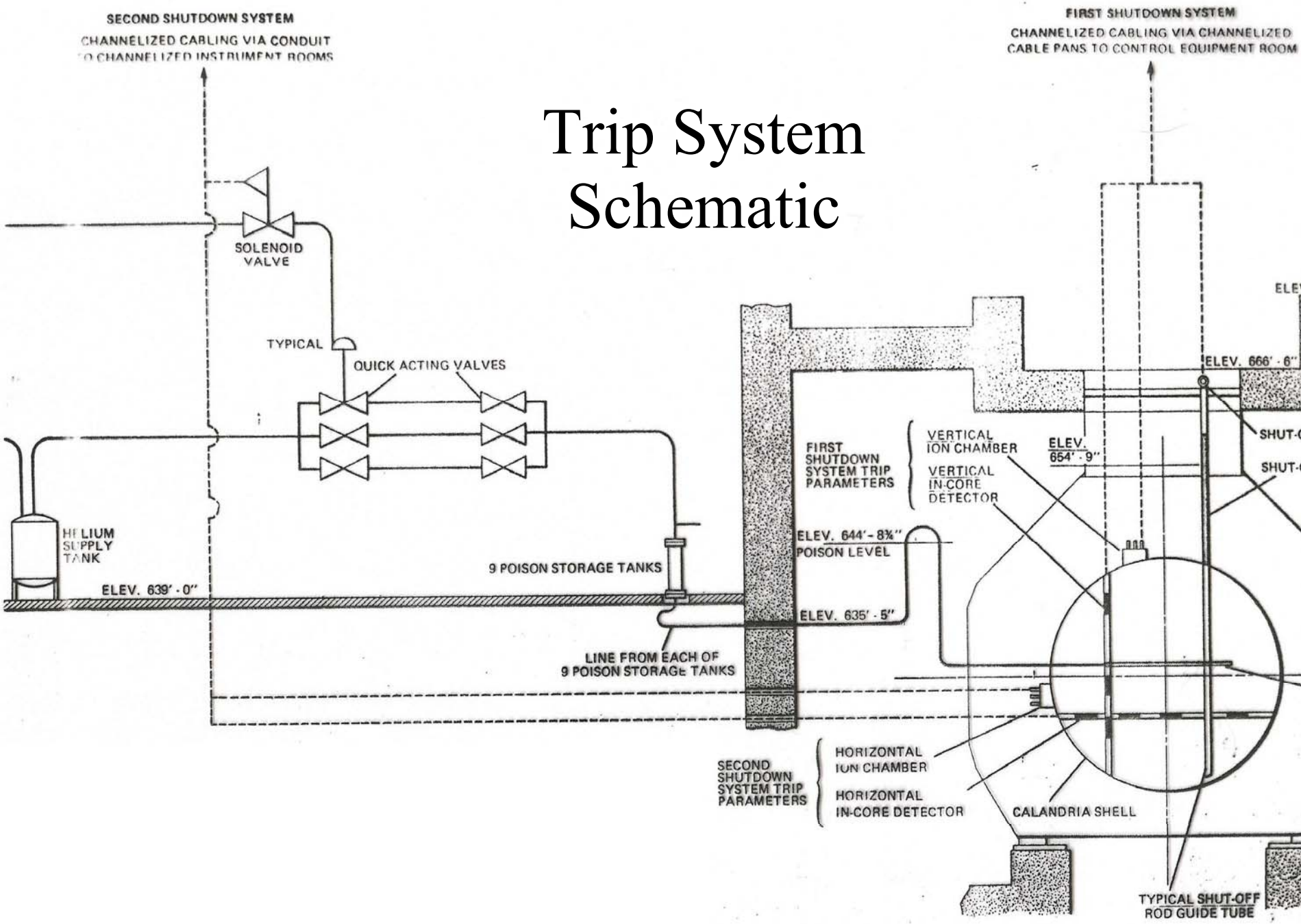
CANDU reactor



Reactor Core



Trip System Schematic



Reasons for Parnas Table Verification

- regulator required that software be verified before put into use
- We had done a verification based on a method by Nancy Levison but it was deemed inadequate
- Hydro agreed “reluctantly” to the formal verification
  - software already written but no formal requirements documents
  - Process unproven
  - Likely to be expensive
  - Likely to be ***lengthy***

Computer Environment

- ***Dedicated computers and operating system***
- ***Two diverse, independent, hardened and obsolete computer systems***
- ***Each system is triply redundant***
- ***Little human interaction during execution***
- ***Receives digital input from measuring devices***
- ***Outputs a go/no-go signal to trip the reactor***

Why they agreed

- If you borrow a lot of money you need to pay it back
- $2000/kWe engphys.mcmaster.ca ($8B)
- $4000/kWe EDA ($14B)
- At 8% this works out to about 1.8M$ per day in interest using the lower figure
- Reactor on (3524Mw) you could earn
  $6,766,080 @ $80 $/Mwhr or
  $3,805,920 @ $45 $/Mwhr
- Formal verification only cost of the order of 60 X 60 X 52 X 100 = 18M$ - ***as long as it didn't hold up the license***

Actual Verification Process

- Team to create PF tables from existing requirements document
- Team to create PF tables from the code
- Team to compare the two documents and *prove* they are equivalent
- All done by hand – Almost no tool support

Code Sample in Pascal

Procedure declaration:

```
procedure Find(e : integer; V : vector; var index : integer;
                    var found : Boolean);
var low, high, med : integer;
begin
{Initialization}
  low := 1; high := n;
  found := false;
  index := 1;
  {Body}
  while not found and (low ≤ high) do
  begin
    med := (low + high) div 2;
    if V[med] < e then low := med + 1 else
      if V[med] > e then high := med – 1 else
      begin
        index := med; found := true
      end {else}
  end {while}
end {Find}
```

Procedure invocation:

```
Find(x, A, j, present)
```

Parameter binding:

('e = val(x)) ∧ ('V = val(A)) ∧ (index ≡ j) ∧ (found ≡ present)

Sample PF Table

Find(x, A, j, present)

$R((\text{'x}, \text{'A}, \text{'j}, \text{'present}), (x', A', j', present')) =$

$((1 \le n) \wedge \forall i\, [\, (1 \le i < n) \rightarrow (\text{'A}[i] \le \text{'A}[i+1])\,]) \rightarrow$

	$\exists i\, [\, (1 \le i \le n) \wedge (\text{'A}[i] = \text{'x})\,] =$	
	true	*false*
$j' =$	$\text{'A}[j] = \text{'x}$	*true*
present' =	true	false

$\wedge\ NC(x, A)$

Fig. 3 Tabular expression of the relational specification of the program shown in Fig. 2.

Verification Process (from a paper by Parnas)

- Inspectors ...need “quiet time to think”
- ..inspections must be interrupted by breaks,evenings and weekends
- ..results of inspections must be scrutinized carefully in open discussions

Reality

- I worked roughly 60-70 hrs. a week for a year and so did a lot of team members
- breaks tended to be trips to Hasty Mart for a bag of cheesies and a coke
- most inspection results were Excel tables passed over a network of Macs
- Discussions were infrequent but there was an open door policy (we also didn’t have doors)

Sample SDS1 code

```
C TITLE:     Average Power Calculation
C
C ======================================================================
C PROJECT 38, Darlington GS
C ======================================================================
C
C SOURCE FILE NAME:    AVPOW
C
C MODULE NAME:         AVPOW (Subroutine)
C
C TARGET MACHINE(S):                        PROGRAM LISTING:
C ==================================     ================
C SDS1 Trip Computer, Channels: D,E,F     38-68258-PLN-057
C
C REV    DATE       AUTHOR       DESCRIPTION
C ===  ========   ========    ========================================
C 00   88.08.30   N.D.Thai     Freeze 2 issue
C 01   89.02.06   N.Thai       Freeze 3 (SCR's 31,61,68,71)
C                 G.Rousseau
C                 P.Rosta
C                 W.Collins
C      89.04.20   D.N.Andrejic Freeze 3 (SCR's 101, 98, 108, 115)
C      89.04.30   D.N.Andrejic Freeze 3 (SCR's 116, 101 correction)
C      89.05.06   D.N.Andrejic Freeze 3 (SCR's 115 - consistency)
C      89.05.11   D.N.Andrejic Freeze 3 (SCR's 82 - unused variables)
C      89.05.24   D.N.Andrejic Freeze 3 (corrections - limit check)
C      89.06.02   D.N.Andrejic Freeze 3 (updated SUMLPV check)
C      89.06.29   D.N.Andrejic Freeze 3 (adjusted LPPC cycling)
C      89.07.07   D.N.Andrejic Freeze 3 (corrected case construct)
C      89.07.07   D.N.Andrejic Issued for Freeze 3 PIT
C 02   89.07.21   D.N.Andrejic Freeze 3 (SCR's 160, 175: comments)
C ======================================================================
C DESCRIPTION:
C
C    There are TSAP (12) NOP detectors selected to
C    produce NAP (4) average powers, ie. each average
C    power is the average of SAP (3) detectors
```

Sample SDS1 code

```
          ENCNT = ENCNT + 1
          IF (CALSEQ(ENCNT  ).NE.LPCLID) CALL SFATAL(ELPSEQ)
          IF (CALSEQ(EXCNT+1).NE.LPCLID) CALL SFATAL(ELPSEQ)
          IF((LPPC.LT.0).OR.(LPPC.GT.LPPCL)) CALL SFATAL(ELPRNG)
          IF (LPPC.EQ.LPPCL) LPPC = 0
          LPPC = LPPC + 1
          IF (LPPC.NE.1) GOTO 299
              DO 255 I=1,NAP
                  SUMLPV(I) = 0
                  ENLPS(I)  = 0
  255         CONTINUE
  299     CONTINUE
          IF(LPPC.GT.TSAP) GO TO 500
          LPSN = LPSID(LPPC)
              LPN  = (LPPC-1)/SAP + 1
          IF(.NOT.(
         +          (LPAI(LPSN).GE.LPAILL).AND.(LPAI(LPSN).LE.LPAIHL)
         +     .AND.(CAMT(LPSN).GE.CAMTLL).AND.(CAMT(LPSN).LE.CAMTHL)
         +       )) GO TO 499
                  ENLPS(LPN)    = ENLPS(LPN) + 1
                  CCLPCV(LPPC) = (LPAI(LPSN)-LPOS+CAMT(LPSN))*CGAIN(LPSN)
                  SUMLPV(LPN)   = SUMLPV(LPN)+CCLPCV(LPPC)
  499         CONTINUE
          GO TO 899
  500        J = LPPC - TSAP
          LPNPFP(J) = DEFAP
          IF(  (ENLPS(J).GE.1).AND.(ENLPS(J).LE.SAP)
         +     .AND.(SUMLPV(J).GE.LPSUML).AND.(SUMLPV(J).LE.LPSUMH) )
         +     LPNPFP(J) = SUMLPV(J)*PFPPMV/(ENLPS(J)*CPPF)
  899     CONTINUE
          EXCNT = EXCNT + 1
          RETURN
```

Sample code statistics

- Sample is 433 lines
- 328 lines are comment
- 68 lines are declaration (one variable per line
- 34 lines are executable (6K$/line)?
- This would be considered reasonably complex
- The corresponding PF tables would be about 21 pages.The complete set was twenty four 2” binders

Code Features

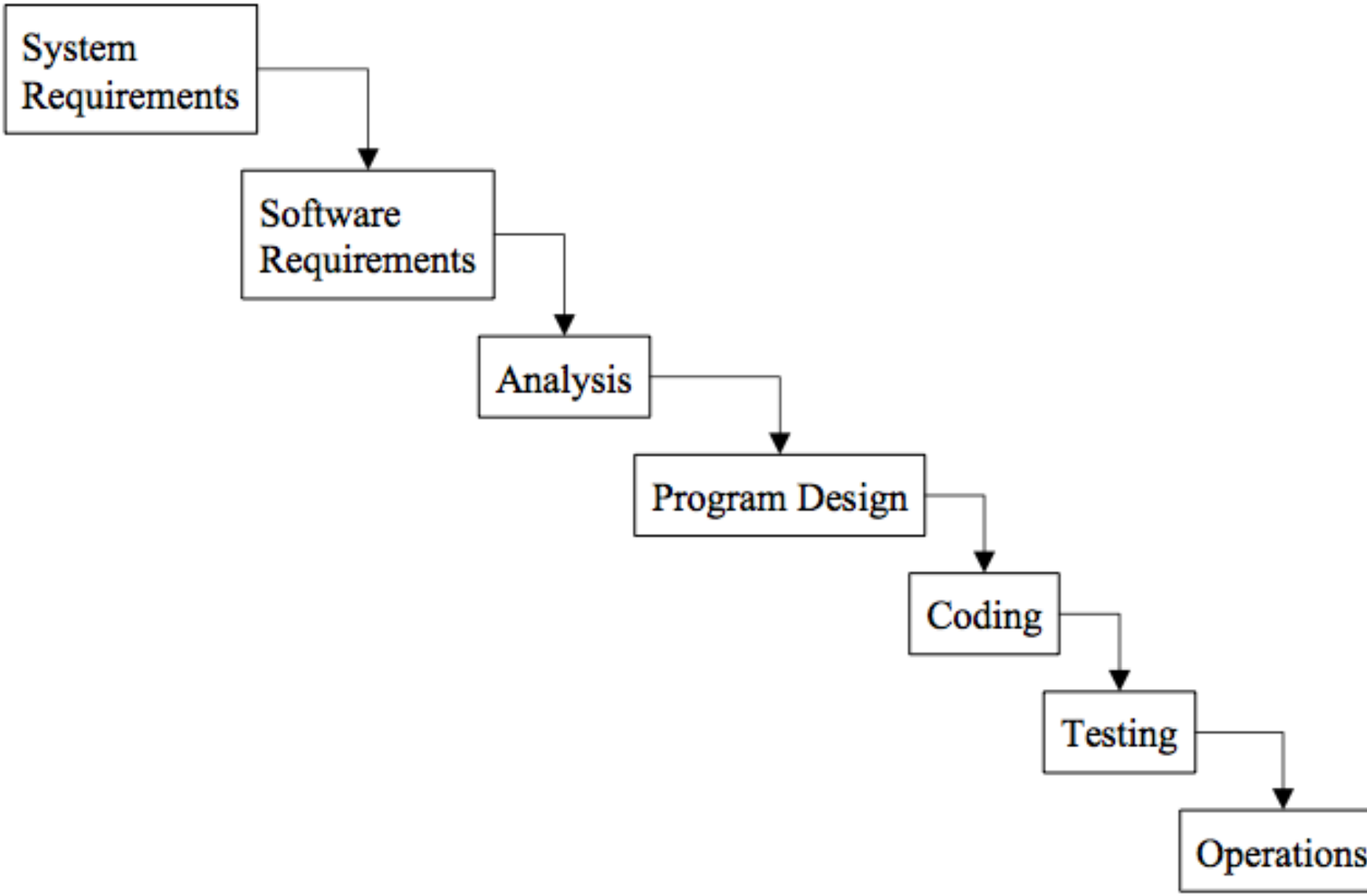
- Baton Passing
- Guarding
- Convoluted execution
- CRC checks
- Simple calculations
  $L1_i < \overline{W_i S_i} < L2_i$

Sample SDS1 code

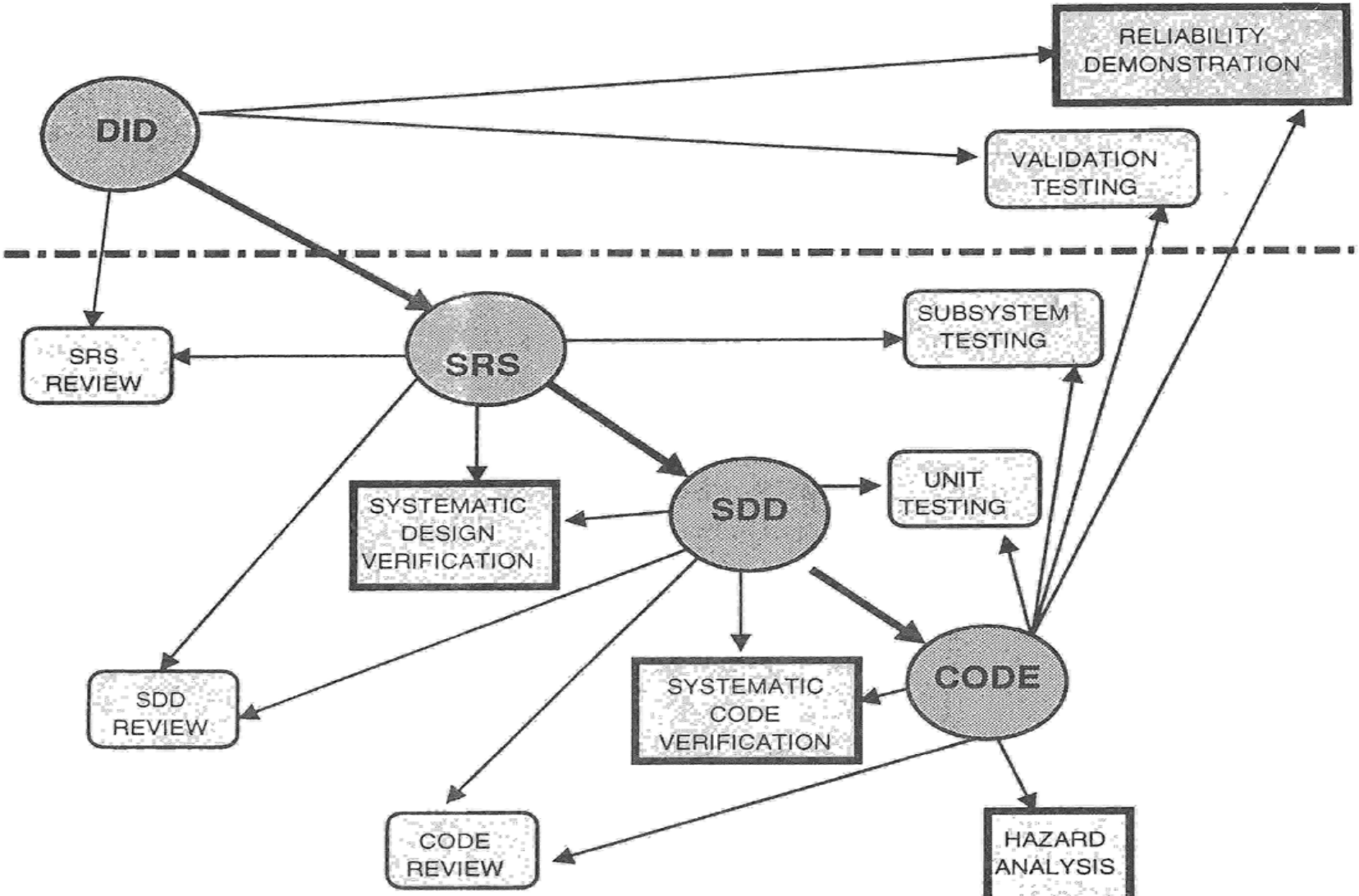
```
          ENCNT = ENCNT + 1
          IF (CALSEQ(ENCNT  ).NE.LPCLID) CALL SFATAL(ELPSEQ)
          IF (CALSEQ(EXCNT+1).NE.LPCLID) CALL SFATAL(ELPSEQ)
          IF((LPPC.LT.0).OR.(LPPC.GT.LPPCL)) CALL SFATAL(ELPRNG)
          IF (LPPC.EQ.LPPCL) LPPC = 0
          LPPC = LPPC + 1
          IF (LPPC.NE.1) GOTO 299
              DO 255 I=1,NAP
                  SUMLPV(I) = 0
                  ENLPS(I)  = 0
  255         CONTINUE
  299     CONTINUE
          IF(LPPC.GT.TSAP) GO TO 500
          LPSN = LPSID(LPPC)
              LPN  = (LPPC-1)/SAP + 1
          IF(.NOT.(
         +          (LPAI(LPSN).GE.LPAILL).AND.(LPAI(LPSN).LE.LPAIHL)
         +     .AND.(CAMT(LPSN).GE.CAMTLL).AND.(CAMT(LPSN).LE.CAMTHL)
         +       )) GO TO 499
                  ENLPS(LPN)   = ENLPS(LPN) + 1
                  CCLPCV(LPPC) = (LPAI(LPSN)-LPOS+CAMT(LPSN))*CGAIN(LPSN)
                  SUMLPV(LPN)  = SUMLPV(LPN)+CCLPCV(LPPC)
  499         CONTINUE
          GO TO 899
  500     J = LPPC - TSAP
          LPNPFP(J) = DEFAP
          IF(  (ENLPS(J).GE.1).AND.(ENLPS(J).LE.SAP)
         +     .AND.(SUMLPV(J).GE.LPSUML).AND.(SUMLPV(J).LE.LPSUMH) )
         +     LPNPFP(J) = SUMLPV(J)*PFPPMV/(ENLPS(J)*CPPF)
  899     CONTINUE
          EXCNT = EXCNT + 1
          RETURN
```

- **The Good News**
  - The AECB allowed Darlington to go into production
- **The *Not-So-Good-News***
  - Having spent $18M+ proving the software was correct the AECB now required the software to be rewritten following a prescribed process

Waterfall Phases



Safety Critical Software Process



Software Engineering Lifecycle

Misconceptions - Issues

- The SDS software initiates the shutdown
- An increase in reliability results in an increase in safety
- “..safety requires correctness..”
- “The programmers had added something extra thinking that they were improving things”
- “There was a coding error but it would not adversely affect safety...”
- “There was a coding error that did affect safety...”
- “..hazard analysis should not have been performed on the code”

My Observations on the exercise

- The formal process grinds incredibly fine
- Upwards of 30M$ was spent with no increase in safety
- Extensive focus on meeting requirements but not on meeting objectives.
- The degree of rigor applied to the symbolic was disproportionate
- The Formal process ignored real issues: kernel,compiler,timing,optimal arithmetic
- Formal methods require tool support to be economically feasible
- Correctness is not enough. You must demonstrate correctness. This means ***process***

References

1. David Parnas, “Inspection of Safety-Critical Software using Program Function tables”, Chapter 19, Software Fundamentals, Collected Papers, Addison-Wesley, 2001
2. Ontario Hydro’s Experience with New Methods for Engineering Safety Critical Software; M.Viola, Proceedings Safecomp 1995, Italy
3. Reactor Physics Aspects of CANDU Safety Analysis; G.M. Frescura