Safety Critical Systems Development

Prof. Chris Johnson,
Department of Computing Science,
University of Glasgow,
Glasgow,
Scotland.
G12 8QJ.

URL: http://www.dcs.gla.ac.uk/∼johnson
E-mail: johnson@dcs.glasgow.ac.uk
Telephone: +41 330 6053

October 2006 (Part II of the Notes).

Hazard Analysis

- Hazard Analysis.
- FMECA/FMEA.
- Case Study.

Hazard Analysis

- Safety case - why proposed system is safe.

- Must identify potential hazards.

- Assess liklihood and severity.

- Lots of Hazard Analysis techniques:

- fault tress (see later);
- cause consequence analysis;
- HAZOPS;
- FMECA/FHA/FMEA...

FMECA - Failure Modes, Effect and Criticality Analysis

- MIL STD 1629A (1977!).

- Analyse each potential failure.

- Determine impact of system(s).

- Assess its criticality.

FMECA - Failure Modes, Effect and Criticality Analysis

- 1. Construct functional block diagram.
- 2. Use diagram to identify any associated failure modes.
- 3. Identify effects of failure and assess criticality.
- 4. Repeat 2 and 3 for potential consequences.
- 5. Identify causes and occurence rates.
- 6. Determine detection factors.
- 7. Calculate Risk Priority Numbers.
- 8. Finalise hazard assessment.

FMECA - Step 1: Functional Block Diagram

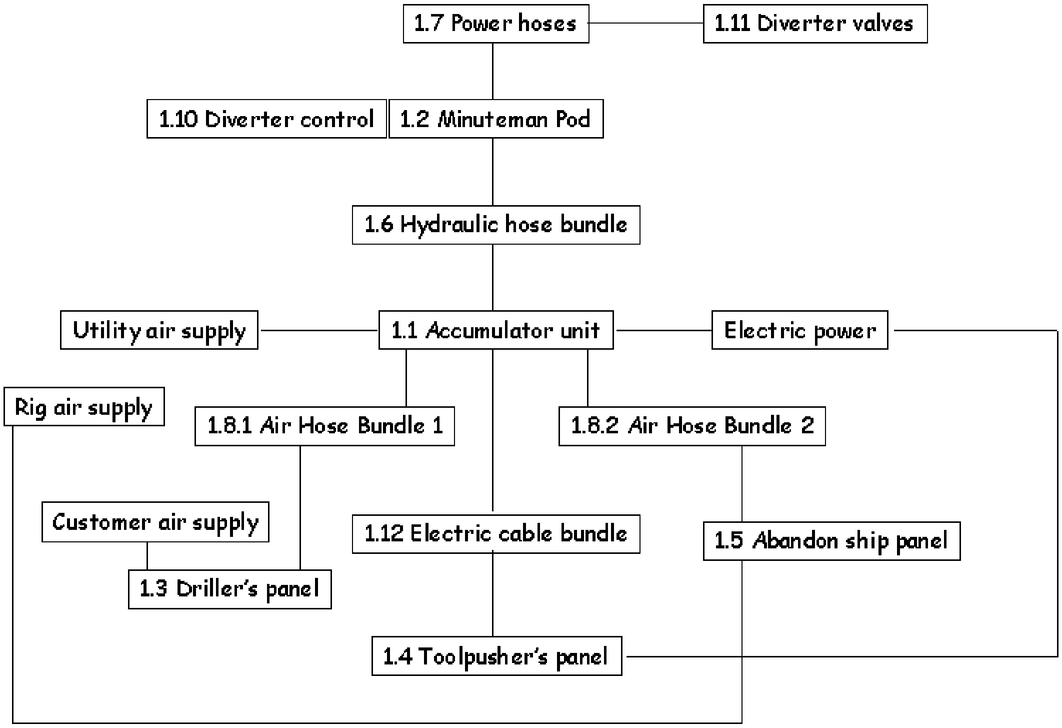
- Establish scope of the analysis.

- Break system into subcomponents.

- Different levels of detail?

- Some unknowns early in design?

FMECA - Step 1: Functional Block Diagram



Acknowledgement: taken from J.D. Andrews and T.R. Moss, Reliability and Risk Assessment, Longman, Harlow, 1993 (ISBN-0-582-09615-4).

FMECA - Step 2: Identfy Failure Modes

- Many different failure modes:

- complete failure;
- partial failure;
- intermittant failure;
- gradual failure;
- etc.

- Not all will apply?

FMECA - Step 3: Assess Criticality

10. Hazardous without warning
Very high severity ranking when a potential failure mode affects safe operation or involves non-compliance with a government regulation without warning.

9. Hazardous with warning
Failure affects safe product operation or involves noncompliance with government regulation with warning.

8. Very High
Product is inoperable with loss of primary Function.

7. High
Product is operable, but at reduced level of performance.

6. Moderate
Product is operable, but comfort or convenience item(s) are inoperable.

5. Low
Product is operable, but comfort or convenience item(s) operate at a reduced level of performance.

4. Very Low
Fit & finish or squeak & rattle item does not conform. Most customers notice defect.

3. Minor
Fit & finish or squeak & rattle item does not conform. Average customers notice defect.

2. Very Minor
Fit & finish or squeak & rattle item does not conform. Discriminating customers notice defect.

1. None
No effect

FMECA - Step 4: Repeat for potential consequences

- Can have knock-on effects.

- Additional failure modes.

- Or additional contexts of failure.

- Iterate on the analysis.

FMECA - Step 5: Identify Cause and Occurence Rates

- Modes with most severe effects first.

- What causes the failure mode?

- How likely is that cause?

- risk = frequency x cost

FMECA - Step 5: Identify Cause and Occurence Rates

Very High: Failure is almost inevitable
Rank 10: 1 in 2
Rank 9: 1 in 3

High: Repeated failures
Rank 8: 1 in 8
Rank 7: 1 in 20

Moderate: Occasional failures
Rank 6: 1 in 80
Rank 5: 1 in 400
Rank 4: 1 in 2000

Low: Relatively few failures
Rank 3: 1 in 15,000
Rank 2: 1 in 150,000

Remote: Failure is unlikely
Rank 1: 1 in 1,500,000

FMECA - Step 6: Determine detection factors

Type (1):
These controls prevent the Cause or Failure Mode from occurring, or reduce their rate of occurrence.

Type (2):
These controls detect the Cause of the Failure Mode and lead to corrective action.

Type (3):
These Controls detect the Failure Mode before the product operation, subsequent operations, or the end user.

- Can we detect/control failure mode?

FMECA - Step 6: Determine detection factors

10. Absolute Uncertainty
Design Control does not detect a potential Cause of failure or subsequent Failure Mode; or there is no Design Control

9. Very Remote
Very remote chance the Design Control will detect a potential Cause of failure or subsequent Failure Mode

8. Remote
Remote chance the Design Control will detect a potential Cause of failure or subsequent Failure Mode

7. Very Low
Very low chance the Design Control will detect a potential Cause of failure or subsequent Failure Mode

6. Low
Low chance the Design Control will detect a potential Cause of failure or subsequent Failure Mode

5. Moderate
Moderate chance the Design Control will detect a potential Cause of failure or subsequent Failure Mode

4. Moderately High
Moderately high chance the Design Control will detect a potential Cause of failure or subsequent Failure Mode

3. High
High chance the Design Control will detect a potential Cause of failure or subsequent Failure Mode

2. Very High
Very high chance the Design Control will detect a potential Cause of failure or subsequent Failure Mode

1. Almost Certain
Design Control will almost certainly detect a potential Cause of failure or subsequent Failure Mode

FMECA - Step 7: Calculate Risk Priority Numbers

- Risk Priority Numbers (RPN)

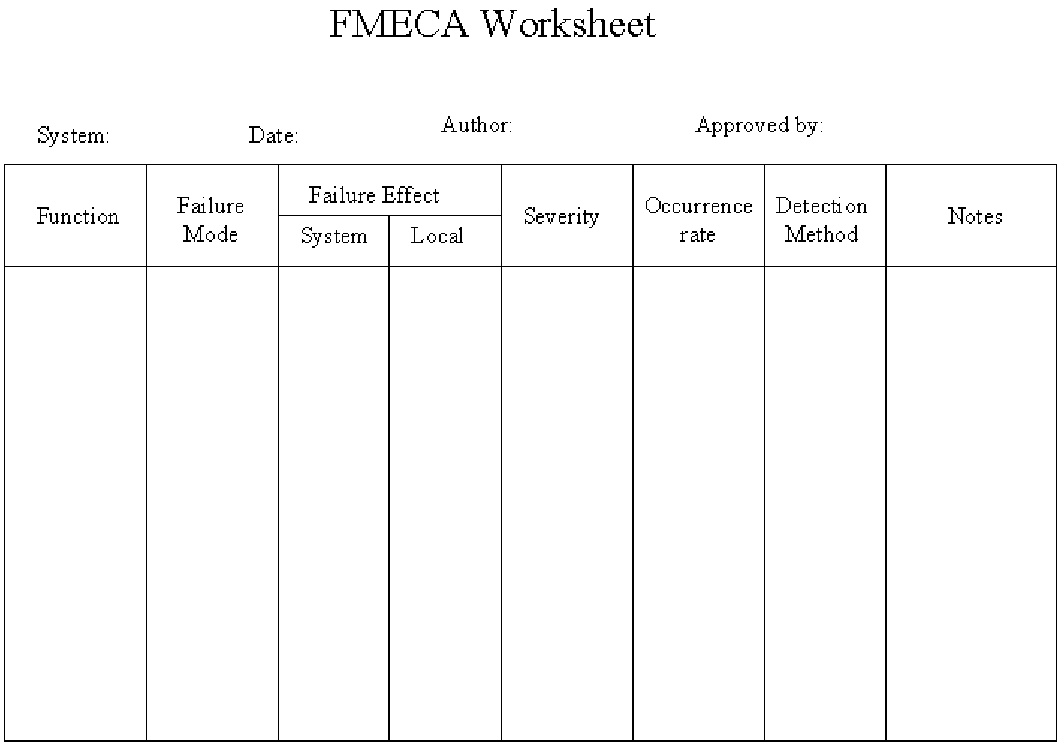
- RPN = S x O x D, where:

- S - severity index;
- O - occurence index;
- D - detection index.

FMECA: Step 8 - Finalise Hazard Analysis

- Must document the analysis...
- ...and response to analysis.
- Use FMECA forms.
- Several formats and tools.

FMECA: Step 8 - Finalise Hazard Analysis

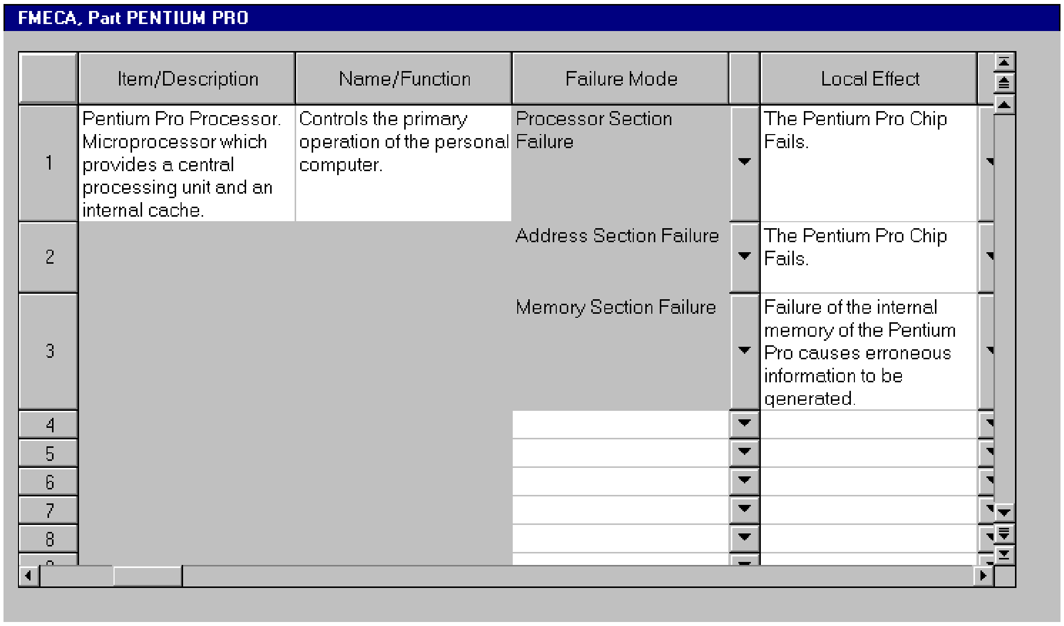


FMECA Worksheet

System: Date: Author: Approved by:

Function	Failure Mode	Failure Effect		Severity	Occurrence rate	Detection Method	Notes
		System	Local				

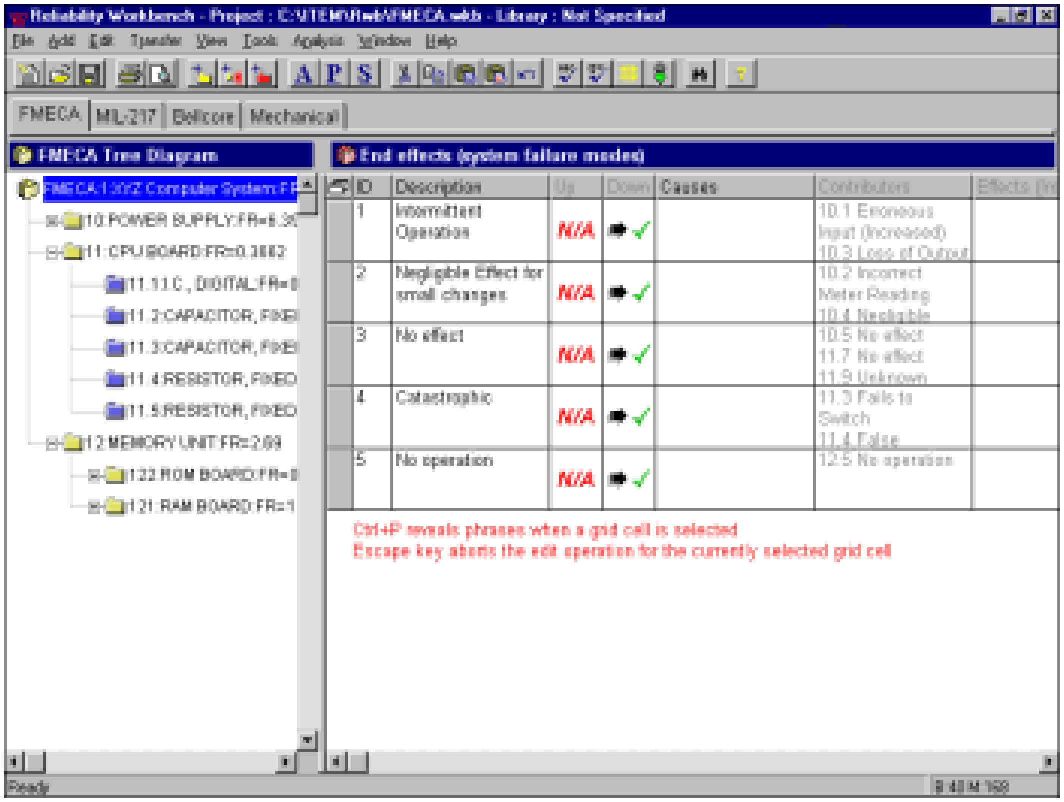
FMECA: Tools



FMECA, Part PENTIUM PRO

	Item/Description	Name/Function	Failure Mode		Local Effect
1	Pentium Pro Processor. Microprocessor which provides a central processing unit and an internal cache.	Controls the primary operation of the personal computer.	Processor Section Failure		The Pentium Pro Chip Fails.
2			Address Section Failure		The Pentium Pro Chip Fails.
3			Memory Section Failure		Failure of the internal memory of the Pentium Pro causes erroneous information to be generated.
4					
5					
6					
7					
8					

FMECA: Tools



Reliability Workbench - Project : C:\TEMP\Rwb\FMECA.wkb - Library : Not Specified
File Add Edit Transfer View Tools Analysis Window Help
A P S
FMECA
MIL-217
Bellcore
Mechanical
FMECA Tree Diagram
End effects (system failure modes)
Ready
8:40 M 168

FMECA:1:XYZ Computer System:FF
10:POWER SUPPLY:FR=6.30
11:CPU BOARD:FR=0.3662
11.1:I.C., DIGITAL:FR=0
11.2:CAPACITOR, FIXEI
11.3:CAPACITOR, FIXEI
11.4:RESISTOR, FIXED
11.5:RESISTOR, FIXED
12:MEMORY UNIT:FR=2.09
12.2:ROM BOARD:FR=0
12.1:RAM BOARD:FR=1

ID	Description	Up	Down	Causes	Contributors	Effects (In
1	Intermittent Operation	N/A	➡ ✓		10.1 Erroneous Input (Increased) 10.3 Loss of Output	
2	Negligible Effect for small changes	N/A	➡ ✓		10.2 Incorrect Meter Reading 10.4 Negligible	
3	No effect	N/A	➡ ✓		10.5 No effect 11.7 No effect 11.9 Unknown	
4	Catastrophic	N/A	➡ ✓		11.3 Fails to Switch 11.4 False	
5	No operation	N/A	➡ ✓		12.5 No operation	

Ctrl+P reveals phrases when a grid cell is selected
Escape key aborts the edit operation for the currently selected grid cell

Conclusions

- Hazard analysis.
- FMECA/FMEA.
- Qualitative → quantitative approaches.

PRA Introduction

The use of PRA technology should be increased in all regulatory matters to the extent supported by the state of the art in PRA methods and data and in a manner that complements the NRC's deterministic approach and supports the NRC's traditional defense-in-depth philosophy.

PRA and associated analyses (e.g., sensitivity studies, uncertainty analyses, and importance measures) should be used in regulatory matters, where practical within the bounds of the state of the art, to reduce unnecessary conservatism associated with current regulatory requirements, regulatory guides, license commitments, and staff practices.

NRC - REGULATORY GUIDE 1.177
An Approach for Plant-Apecific, Risk-Informed Decisionmaking: Technical Specifications

Hazard Analysis vs PRA

- FMECA - hazard analysis.

- PRA part of hazard analysis.

- Wider links to decision theory.

Decision Theory

- Risk = frequency x cost.

- Which risk do we guard against?

$$\forall x \in X : F_u(x) = P([0, x]) \tag{1}$$

Risk is modelled through the set $D(X)$ where the probability, P, of obtaining a reward in the range $[0, x]$ is determined by a cumulative distribution functi on $F_u \in D(X)$, the number $F_u(x)$ is interpreted as the probability of receiving an amount less than or equal to x.

$$\forall F_u \in D(X) : V(F_u) = \sum_{i=1}^{n} u(x^i)p^i \tag{2}$$

$$\forall F_u, G_u \in D(X) : F_u \succ G_u \leftrightarrow V(F_u) > V(G_u) \tag{3}$$

- Are decision makers rational?

- Can you trust the numbers?

PRA - Meta-Issues

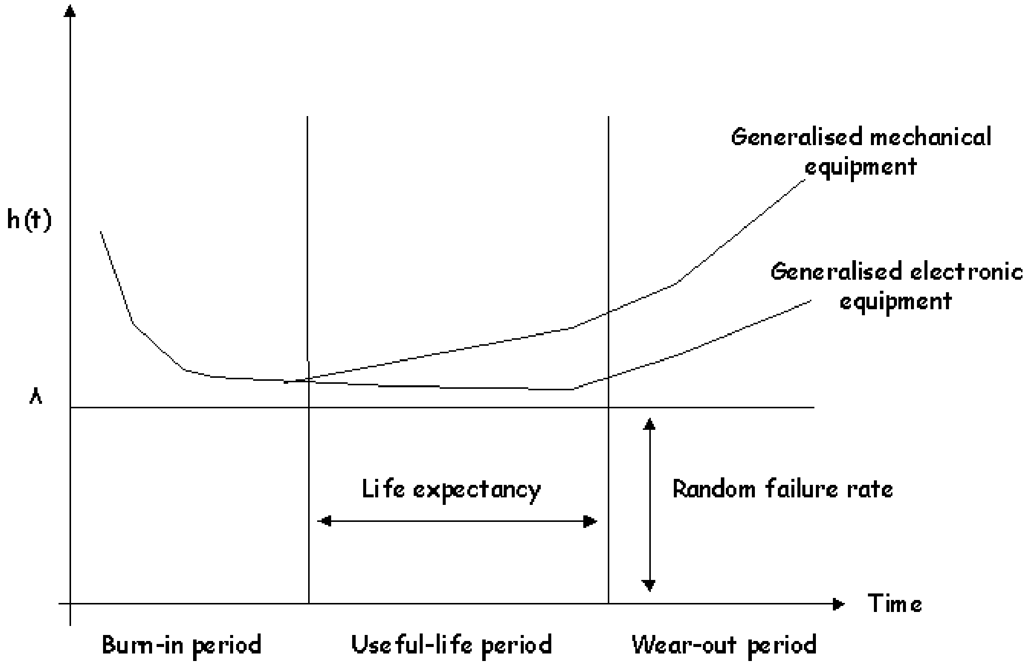
- Decision theory counter intuitive?

- But just a formalisation of FMECA?

- What is the scope of this approach?

- hardware failure rates (here)?
- human error rates (here)?
- software failure rates?

PRA



Acknowledgement: J. D. Andrews and T.R. Moss, Reliability and Risk Assessment, Longman, New York, 1993.

PRA

- Failure rate assumed to be constant:

- electronic systems approximate this.

- Mechanical systems:

- bed-down failure rates;
- degrade failure rates;

- MTTF:

reciprocal of constant failure rate;
MTTF = 1 / λ. λ - base failure rate.

- 0.2 failures per hour: MTTF = 1 / 0.2 = 5 hrs.

- See Andrews and Moss for proof.

PRA - Or Put Another Way...

Probability that product will work for T without failure:
R(T) = exp(-T/MTTF)

- If MTTF = 250,000 hours.
- Over life of 10 years (87,600 hours).
- R = exp(-87,600/250000) = 0.70441
- 70.4% prob of no failure in 10 years.
- 70.4% of systems working in 10 year.

PRA

- For each failure mode.

Criticality_m = a x b x λ_p x time

λ_p - base failure rate with environmental/stress data

a - proportion of total failures in specified failure mode m

b - conditional prob. that expected failure effect will result

- If no failure data use occurence rate estimates.

PRA - Sources of Data

- MIL-HDBK-217:

Reliability Prediction of Electronic Equipment

- Failure rate models for:

- ICs, transistors, diodes, resistors,
- relays, switches, connectors etc.

- Field data + simplifying assumptions.

- Latest version F being revised.

PRA

- 217 too pessimistic for companies...

- Bellcore (Telcordia):

- reliability prediction procedure..

During 1997, AT&T's Defects-Per-Million performance was 173, which means that of every one million calls placed on the to a network failure. That equals a network reliability rate of 99.98 percent for 1997.

www.att.com

- Business critical not safety critical.

PRA

- Commercial reliability databases.

- But MTTF doesnt consider repair!

- MTTR considers observations.

- MIL-HDBK-338B (1,000+ pages!)

PRA and FMECA Mode Probability

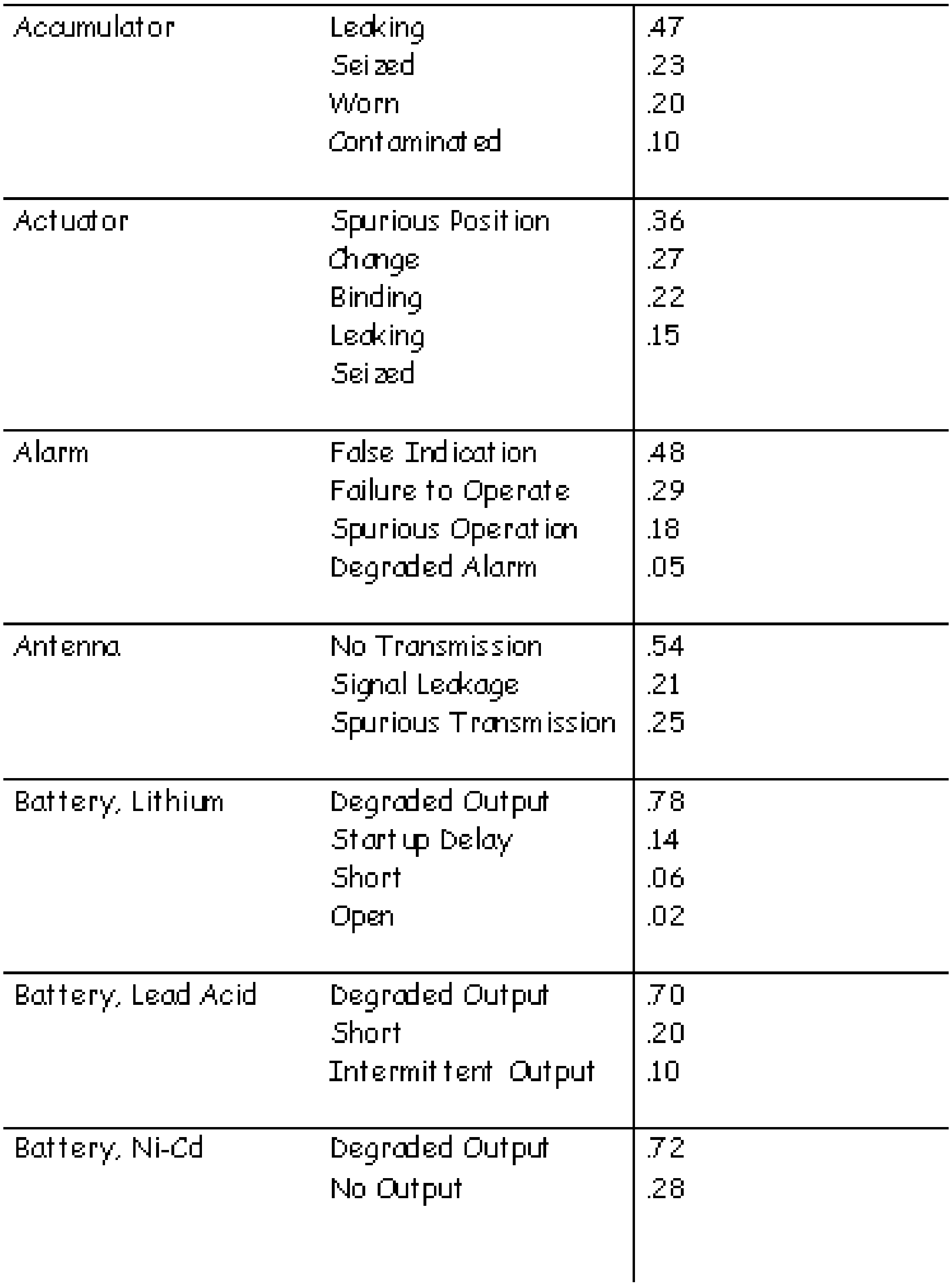
- FMECA:

- we used subjective criticality;
- however, MIL-338B calculates it;
- no. of failures per hour per mode.

- CR $= \alpha x \beta x \lambda$:

CR - criticality level,
α - failure mode frequency ratio,
β - loss prob. of item from mode
λ - base failure rate for item.

PRA and FMECA Mode Probability



Accumulator	Leaking	.47
	Seized	.23
	Worn	.20
	Contaminated	.10
Actuator	Spurious Position	.36
	Change	.27
	Binding	.22
	Leaking	.15
	Seized	
Alarm	False Indication	.48
	Failure to Operate	.29
	Spurious Operation	.18
	Degraded Alarm	.05
Antenna	No Transmission	.54
	Signal Leakage	.21
	Spurious Transmission	.25
Battery, Lithium	Degraded Output	.78
	Startup Delay	.14
	Short	.06
	Open	.02
Battery, Lead Acid	Degraded Output	.70
	Short	.20
	Intermittent Output	.10
Battery, Ni-Cd	Degraded Output	.72
	No Output	.28

PRA

- We focussed on hardware devices.

- PRA for human reliability?

- Probably not a good idea.

- But for completeness...

Technique for Human Error Rate Prediction (THERP)

“The THERP approach uses conventional reliability technology modified to account for greater variability and independence of human performance as compared with that of equipment performance... The procedures of THERP are similar to those employed in conventional reliability analysis, except that human task activities are substituted for equipment outputs.”
(Miller and Swain, 1987 - cited by Hollnagel, 1998).

A.D. Swain and H.E. Guttman, Handbook of Human Reliability with Emphasis on Nuclear Power Plant Applications NUREG-CR-1278, 1985.

Technique for Human Error Rate Prediction (THERP)

- $Pe = \Sigma_{k=1}^{n}\ Psf\_k * W\_k + C$

- Where:

Pe - probability of error;
He - raw human error probability;
C - numerical constant;
Psf_k - performance shaping factor;
W_k - weight associated with PSF_k;
n - total number of PSFs.

Technique for Human Error Rate Prediction (THERP)

- “Psychological vaccuous” (Hollnagel).

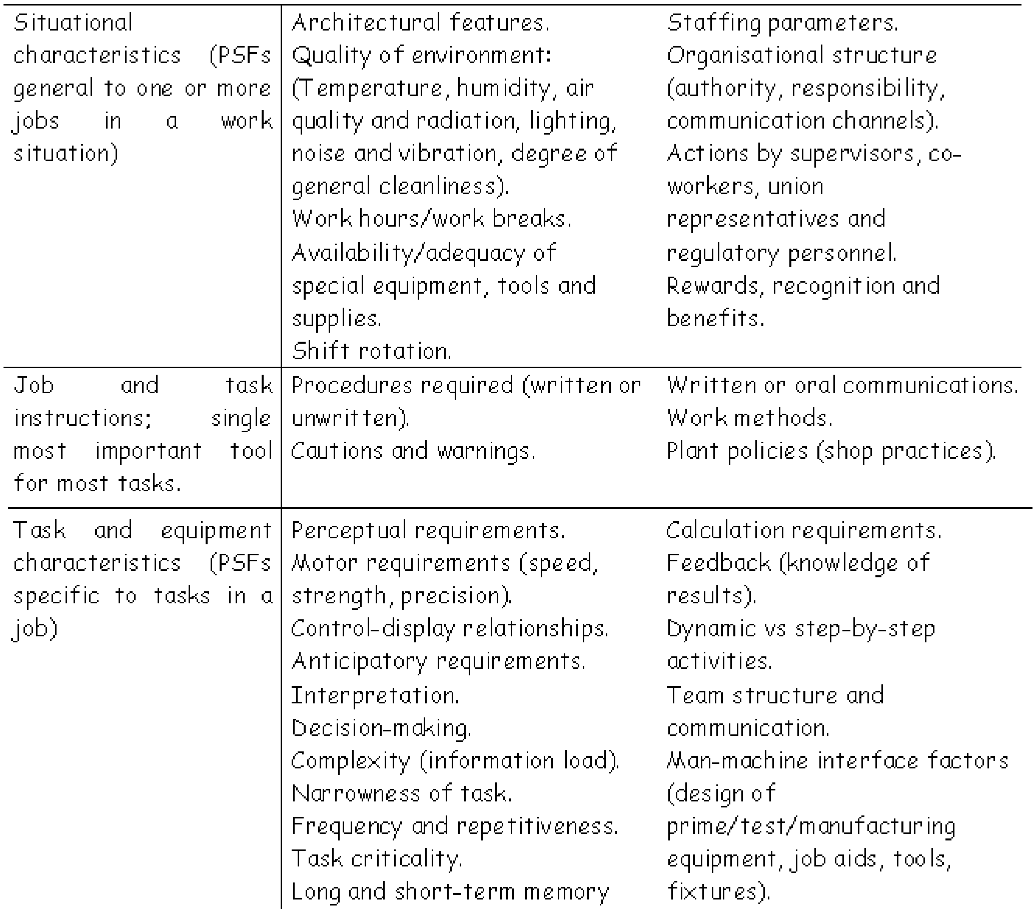
- No model of cognition etc.

- Calculate effect of PSF on HEP

- ignores WHY they affect performance.

- Succeeds or fails on PSFs.

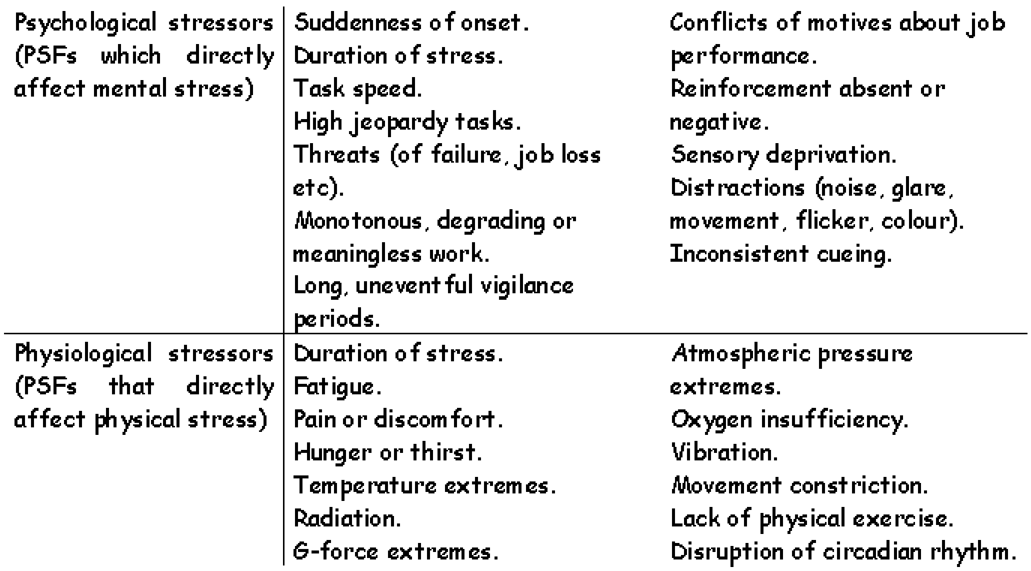
THERP - External PSFs



Situational characteristics (PSFs general to one or more jobs in a work situation)	Architectural features. Quality of environment: (Temperature, humidity, air quality and radiation, lighting, noise and vibration, degree of general cleanliness). Work hours/work breaks. Availability/adequacy of special equipment, tools and supplies. Shift rotation.	Staffing parameters. Organisational structure (authority, responsibility, communication channels). Actions by supervisors, co-workers, union representatives and regulatory personnel. Rewards, recognition and benefits.
Job and task instructions; single most important tool for most tasks.	Procedures required (written or unwritten). Cautions and warnings.	Written or oral communications. Work methods. Plant policies (shop practices).
Task and equipment characteristics (PSFs specific to tasks in a job)	Perceptual requirements. Motor requirements (speed, strength, precision). Control-display relationships. Anticipatory requirements. Interpretation. Decision-making. Complexity (information load). Narrowness of task. Frequency and repetitiveness. Task criticality. Long and short-term memory	Calculation requirements. Feedback (knowledge of results). Dynamic vs step-by-step activities. Team structure and communication. Man-machine interface factors (design of prime/test/manufacturing equipment, job aids, tools, fixtures).

Acknowledgement: A.D. Swain, Comparative Evaluation of Methods for Human Reliability Analysis, (GRS-71), Garching FRG: Gesellschaft fur Reaktorsicherheit.

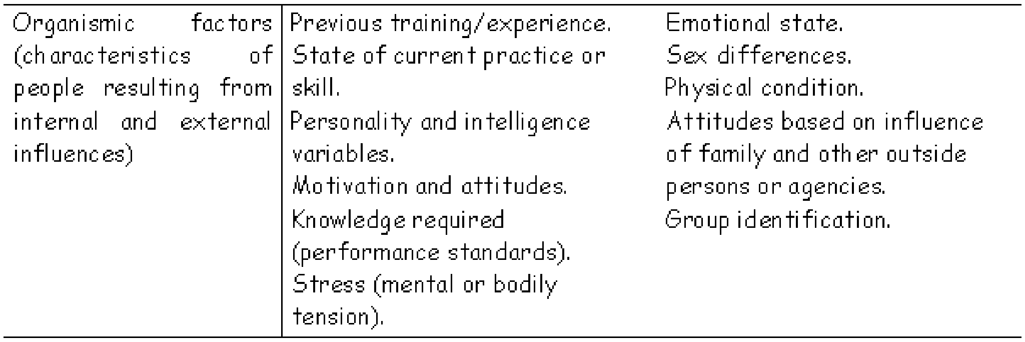
THERP - Stressor PSFs



Psychological stressors (PSFs which directly affect mental stress)	Suddenness of onset. Duration of stress. Task speed. High jeopardy tasks. Threats (of failure, job loss etc). Monotonous, degrading or meaningless work. Long, uneventful vigilance periods.	Conflicts of motives about job performance. Reinforcement absent or negative. Sensory deprivation. Distractions (noise, glare, movement, flicker, colour). Inconsistent cueing.
Physiological stressors (PSFs that directly affect physical stress)	Duration of stress. Fatigue. Pain or discomfort. Hunger or thirst. Temperature extremes. Radiation. G-force extremes.	Atmospheric pressure extremes. Oxygen insufficiency. Vibration. Movement constriction. Lack of physical exercise. Disruption of circadian rhythm.

Acknowledgement: A.D. Swain, Comparative Evaluation of Methods for Human Reliability Analysis, (GRS-71), Garching FRG: Gesellschaft fur Reaktorsicherheit.

THERP - Internal PSFs



Organismic factors (characteristics of people resulting from internal and external influences)	Previous training/experience. State of current practice or skill. Personality and intelligence variables. Motivation and attitudes. Knowledge required (performance standards). Stress (mental or bodily tension).	Emotional state. Sex differences. Physical condition. Attitudes based on influence of family and other outside persons or agencies. Group identification.

Acknowledgement: A.D. Swain, Comparative Evaluation of Methods for Human Reliability Analysis, (GRS-71), Garching FRG: Gesellschaft fur Reaktorsicherheit.

CREAM

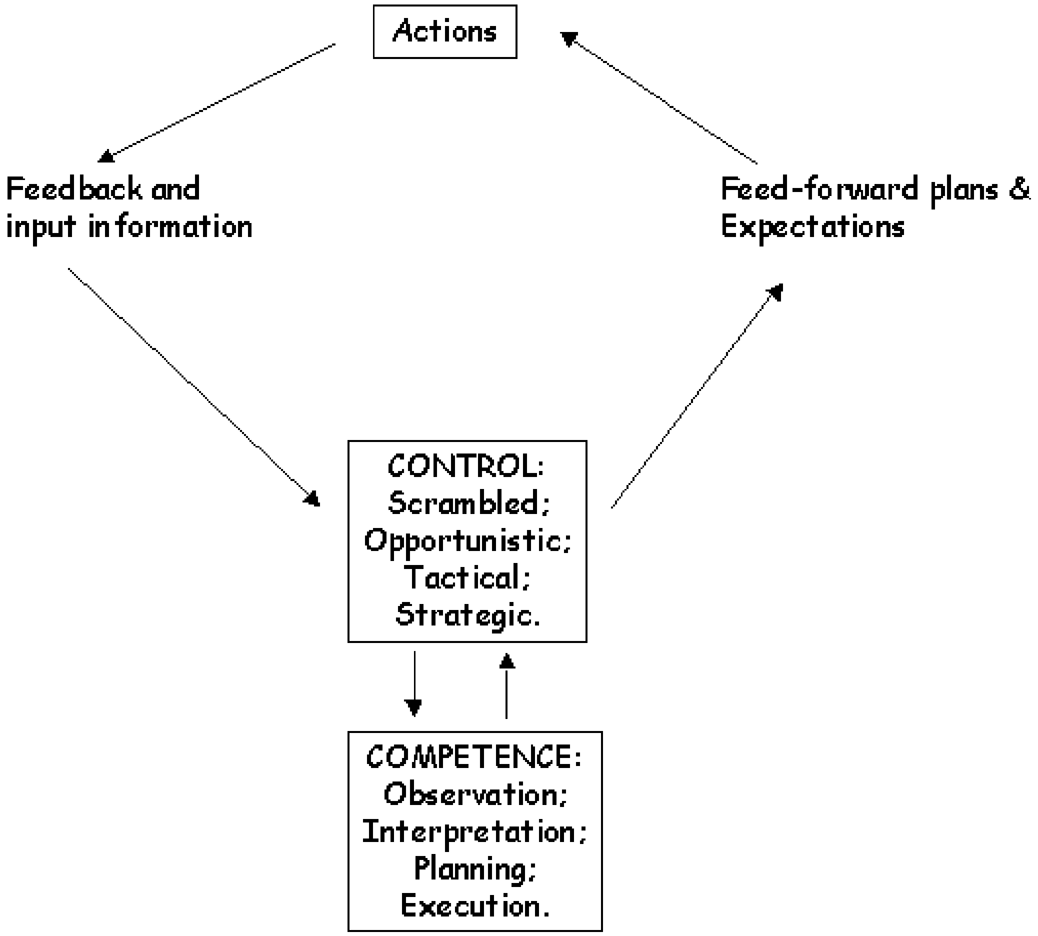
E. Hollnagel, Cognitive Reliability and Error Analysis Method, Elsevier, Holland, 1998.

- HRA + theoretical basis.

- Simple model of control:

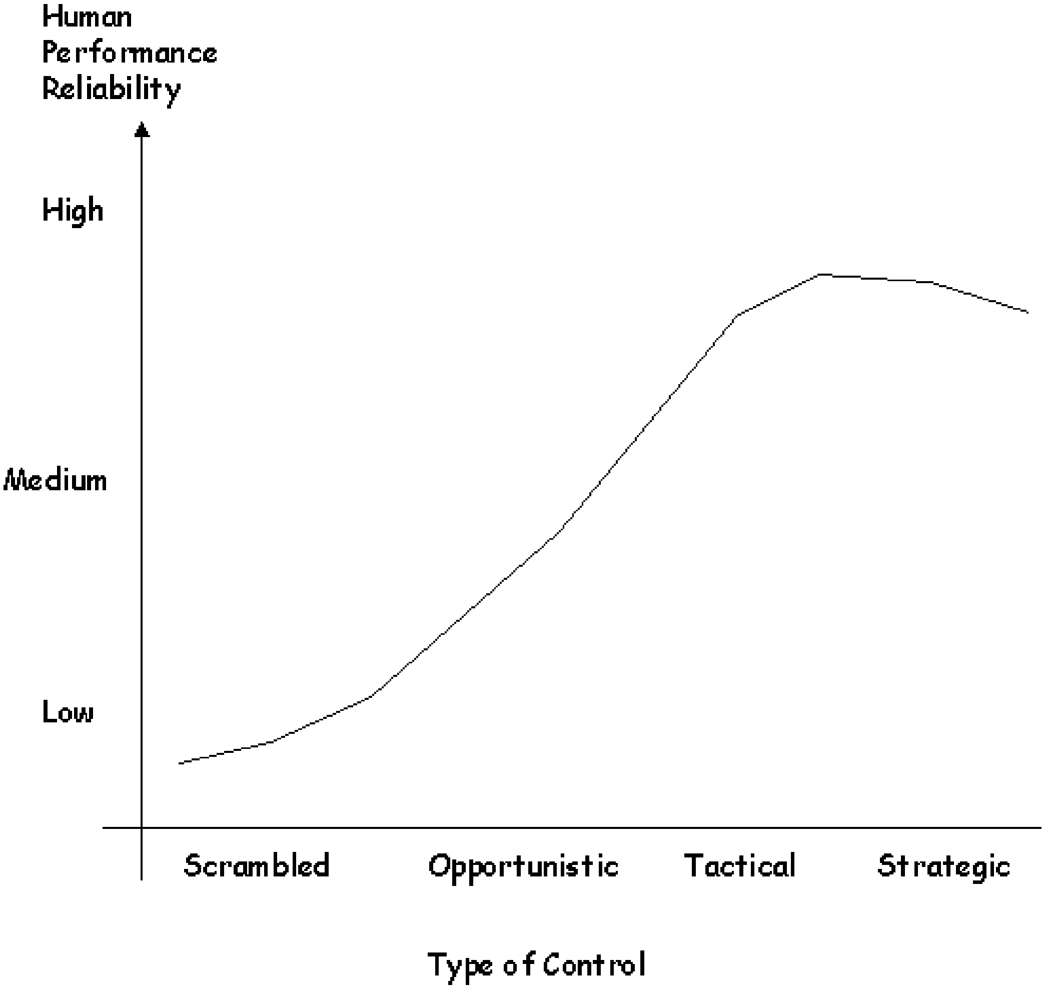
- scrambled - unpredictable actions;
- opportunistic - react dont plan;
- tactical - procedures and rules;
- strategic - consider full context.

CREAM - Simple Model of Control


Actions
Feedback and input information
Feed-forward plans & Expectations
CONTROL: Scrambled; Opportunistic; Tactical; Strategic.
COMPETENCE: Observation; Interpretation; Planning; Execution.

Acknowledgement: E. Hollnagel, Cognitive Reliability and Error Analysis Method, Elsevier, Holland, 1998.

CREAM - Simple Model of Control



Acknowledgement: E. Hollnagel, Cognitive Reliability and Error Analysis Method, Elsevier, Holland, 1998.

CREAM

- Much more to the technique...

- But in the end...

$Strategic = 0.000005 < p < 0.01$
$Tactic = 0.001 < p < 0.1$
$Opportunistic = 0.01 < p < 0.5$
$Scrambled = 0.1 < p < 1.0$

- Common performance conditions to:

- probable control mode then to
- reliability estimate from literature.

Conclusions

- PRA for hardware:

- widely accepted with good data.

- PRA for human performance:

- many are skeptical;
- THERP → CREAM → ???

- PRA for software?

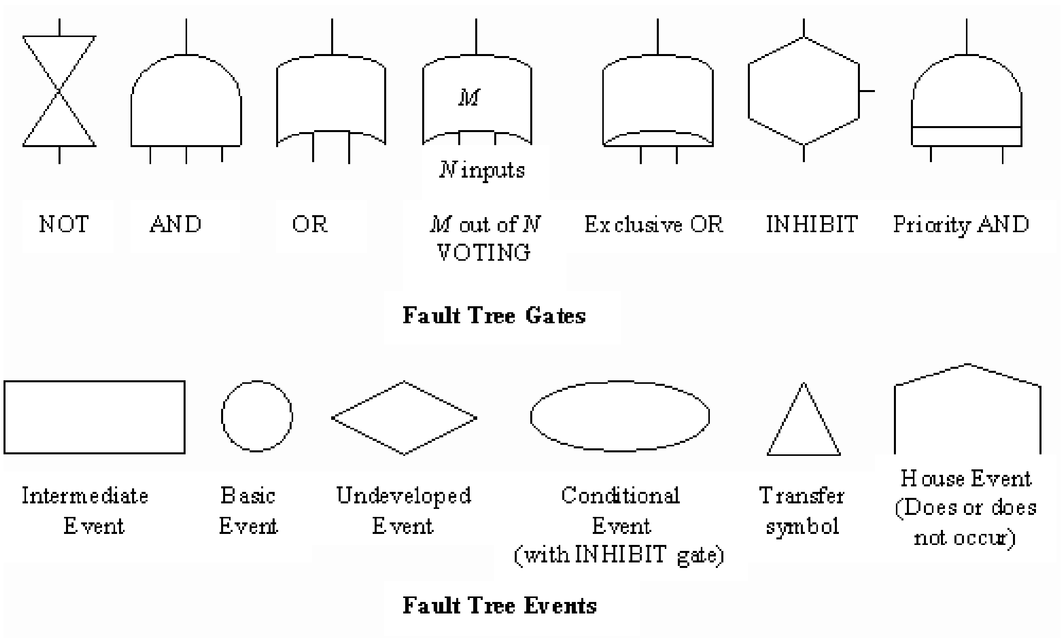
PRA and Fault Tree Analysis

- Fault Trees (recap).

- Software Fault Trees.

- Software PRA.

Fault Trees (Recap)



Fault Tree Analysis

- Each tree considers 1 failure.

- Carefully choose top event.

- Carefully choose system boundaries.

- Assign probabilities to basic events.

Fault Tree Analysis

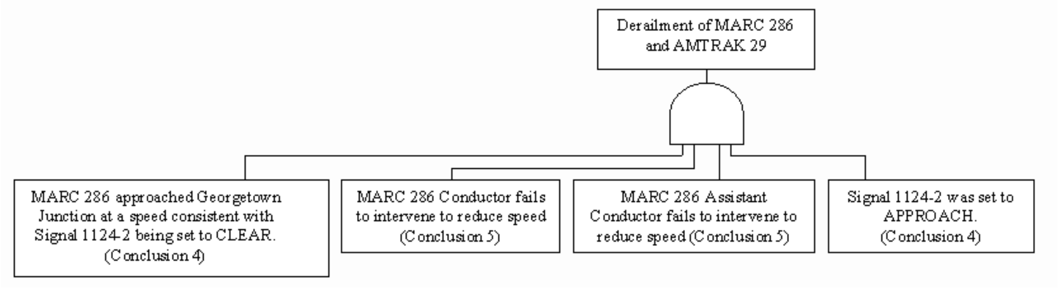
- Assign probabilities to basic events.

- Stop if you have the data.

- Circles denote basic events.

- Even so, tool support is critical.

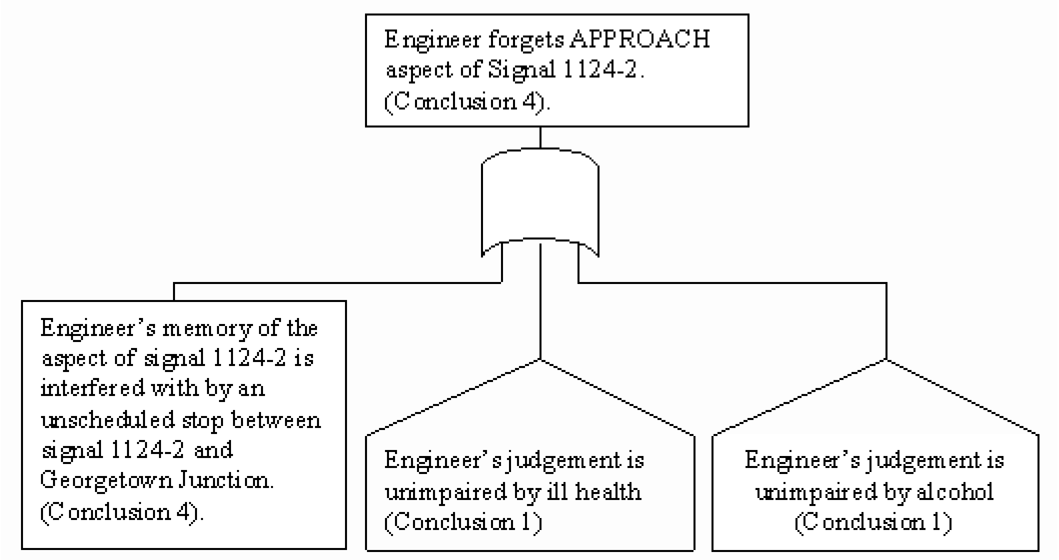
Fault Tree Analysis


Derailment of MARC 286 and AMTRAK 29
MARC 286 approached Georgetown Junction at a speed consistent with Signal 1124-2 being set to CLEAR. (Conclusion 4)
MARC 286 Conductor fails to intervene to reduce speed (Conclusion 5)
MARC 286 Assistant Conductor fails to intervene to reduce speed (Conclusion 5)
Signal 1124-2 was set to APPROACH. (Conclusion 4)

- Usually applied to hardware...

- Can be used for software (later).

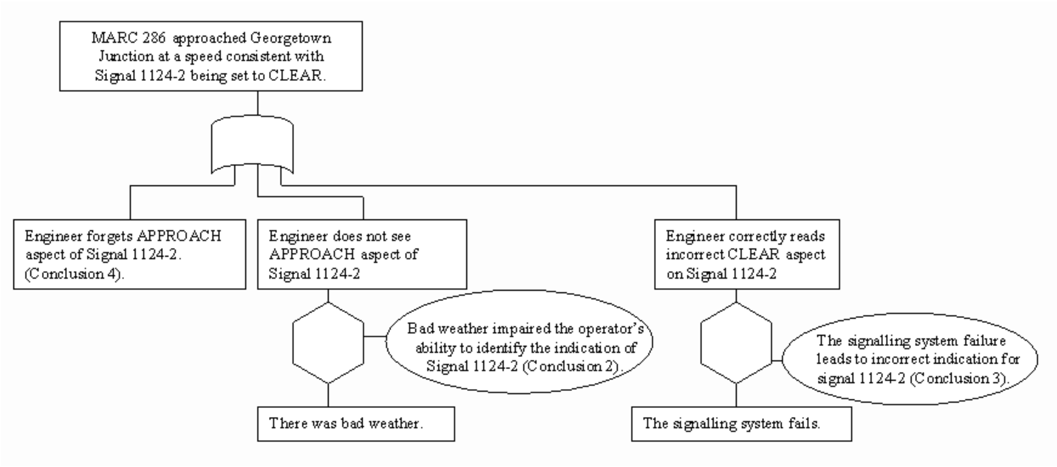
Fault Tree Analysis


Engineer forgets APPROACH aspect of Signal 1124-2. (Conclusion 4).
Engineer's memory of the aspect of signal 1124-2 is interfered with by an unscheduled stop between signal 1124-2 and Georgetown Junction. (Conclusion 4).
Engineer's judgement is unimpaired by ill health (Conclusion 1)
Engineer's judgement is unimpaired by alcohol (Conclusion 1)

- House events; "switch" true or false.

- OR gates - multiple fault paths.

Fault Tree Analysis

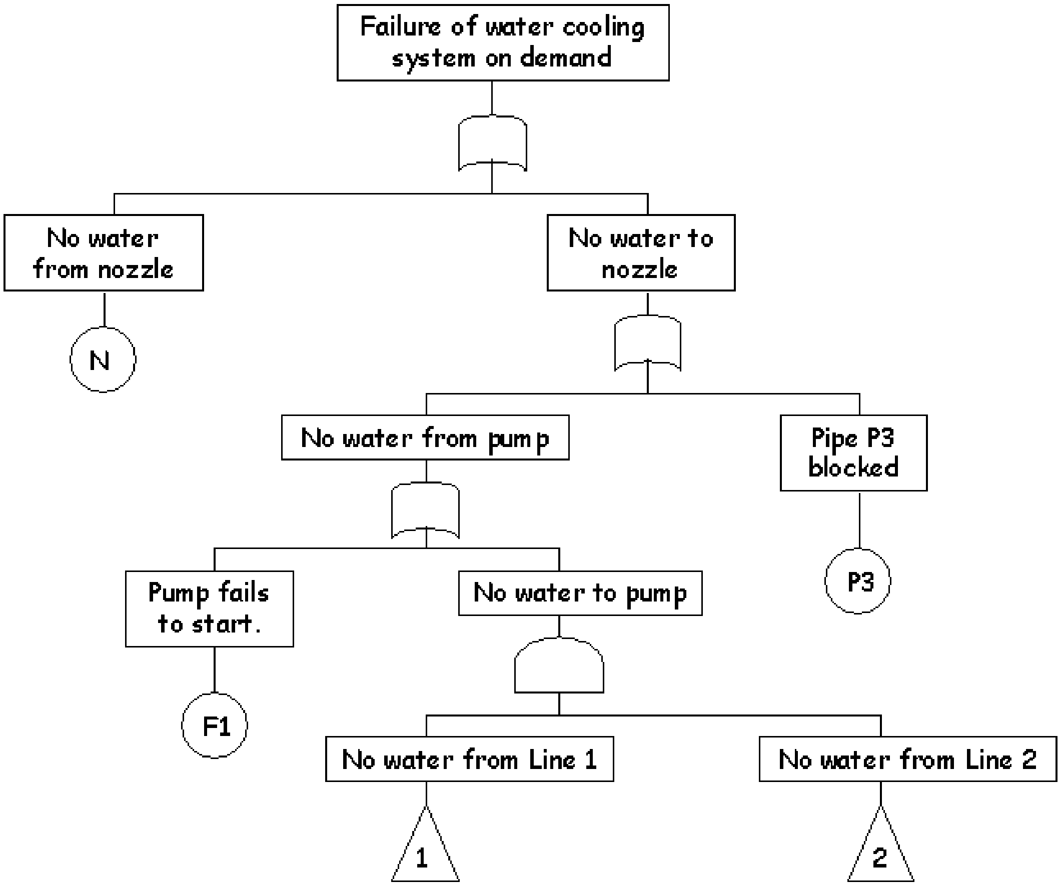

MARC 286 approached Georgetown Junction at a speed consistent with Signal 1124-2 being set to CLEAR.
Engineer forgets APPROACH aspect of Signal 1124-2. (Conclusion 4).
Engineer does not see APPROACH aspect of Signal 1124-2
Engineer correctly reads incorrect CLEAR aspect on Signal 1124-2
Bad weather impaired the operator's ability to identify the indication of Signal 1124-2 (Conclusion 2).
The signalling system failure leads to incorrect indication for signal 1124-2 (Conclusion 3).
There was bad weather.
The signalling system fails.

- Probabilistic inhibit gates.

- Used with Monte Carlo techniques

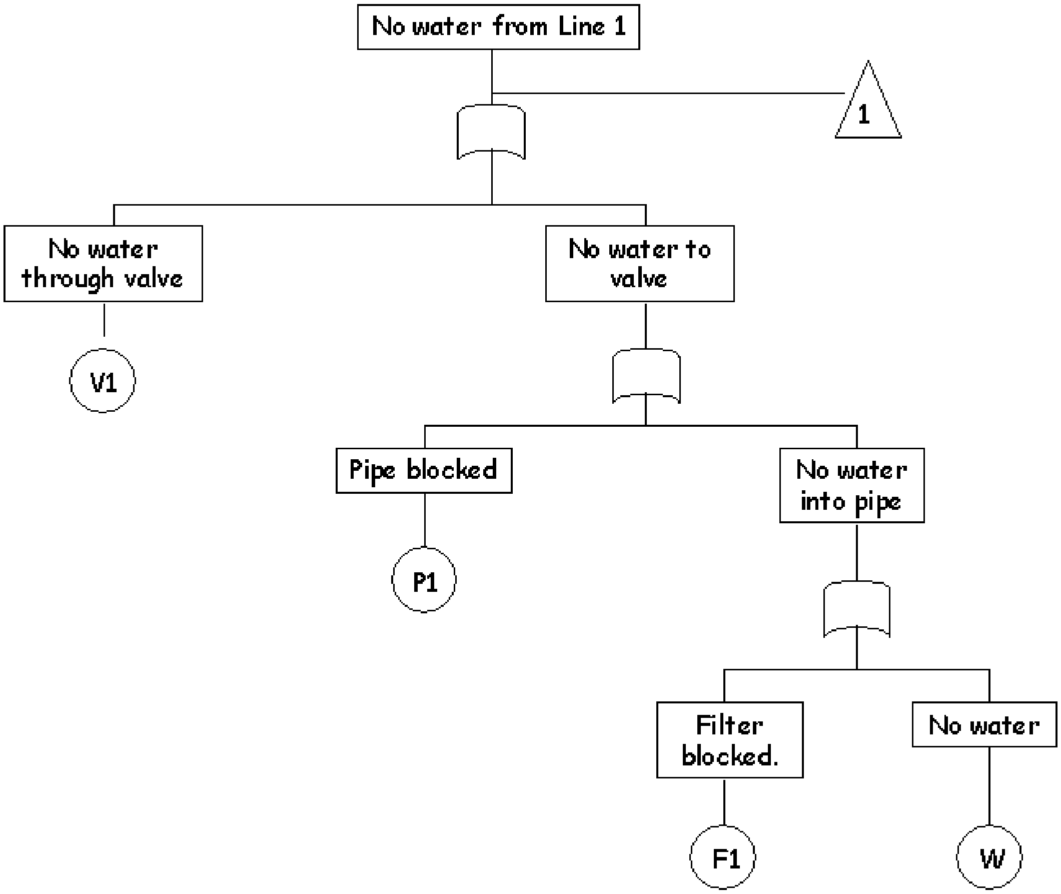
- True if random number < prob.

Fault Tree Analysis



- Usually applied to hardware...

Fault Tree Analysis



Acknowledgement: J.D. Andrews and T.R. Moss, Reliability and Risk Assessment, Longman Scientific and Technical, Harlow, 1993.

Fault Tree Analysis - Cut Sets

- Each failure has several modes

- 'different routes to top event'.

- Cut set:

basic events that lead to top event.

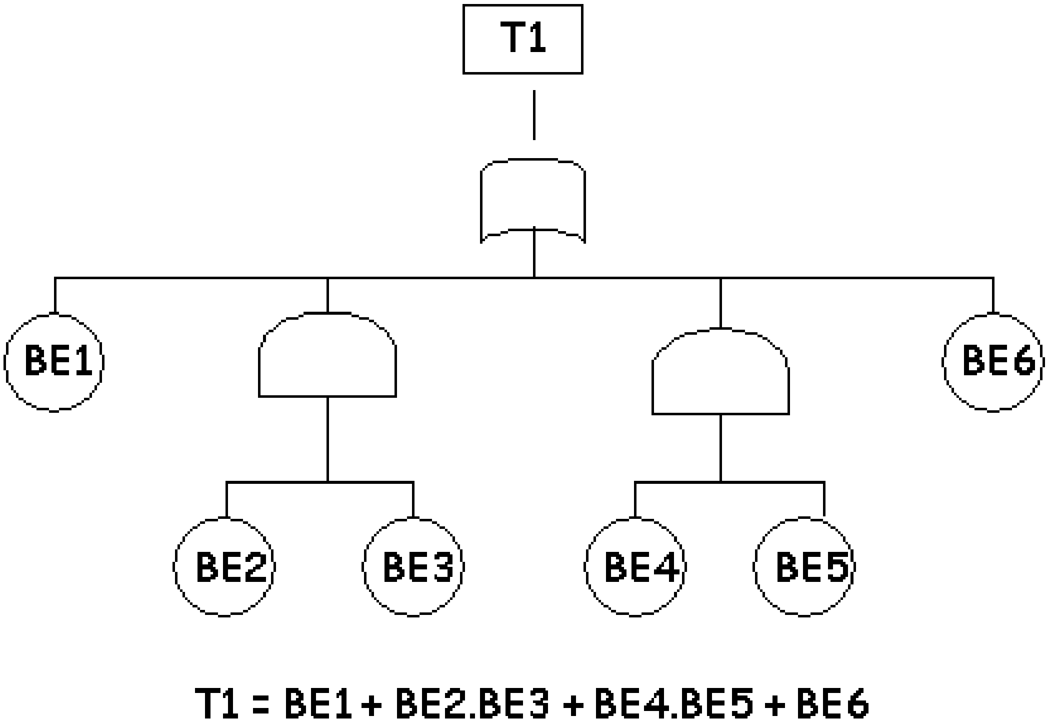
- Minimal cut set:

removing a basic event avoids failure.

- Path set:

basic events that avoid top event;
list of components that ensure safety.

Fault Tree Analysis - Cut Sets



- Top_Event = K1 + K2 + ... K_n

K_i minimal cut sets, + is logical OR.

- K_i = X_1 . X_2 . X_n

MCS are conjuncts of basic events.

Fault Tree Analysis - Cut Sets

- Top-down approach:

- replace event by expression below;
- simply if possible (C.C = C).

- Can use Karnaugh map techniques;

- cf logic circuit design;
- recruit tool support in practice.

- Notice there is no negation.

- Notice there is no XOR.

MOCUS Cut Set Algorithm

1. Assign unique label to each gate.

2. Label each basic event.

3. Create a two dimensional array A.

4. Initialise A(1,1) to top event.

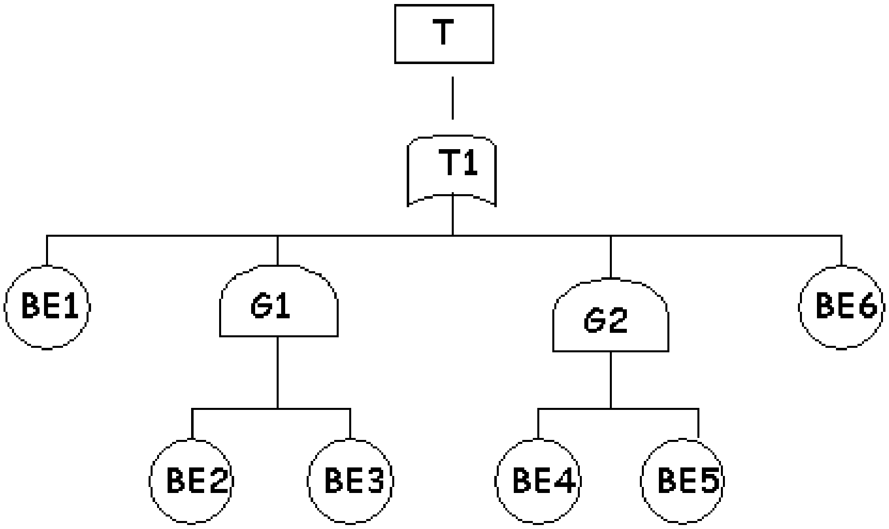
5. Scan array to find an OR/AND gate:

If current position in A is OR gate... - replace current position with a column; - put gate's input events in new row of that column. - replace current position with a row; - put gate's input events in new column of that row.

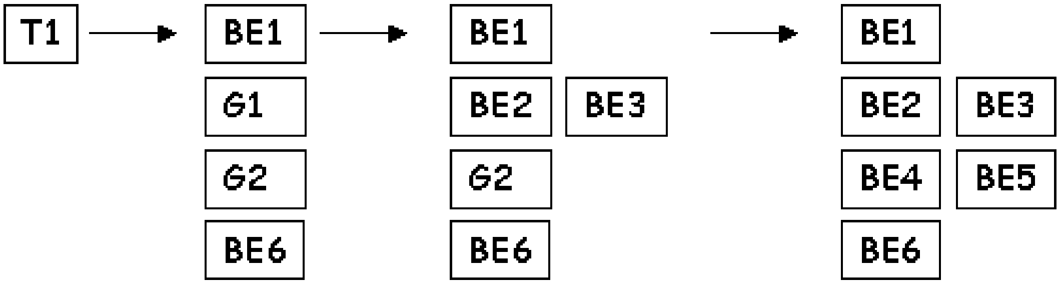
6. Repeat 5 until no gates remain in array.

7. Remove any non-minimal cut sets.

MOCUS

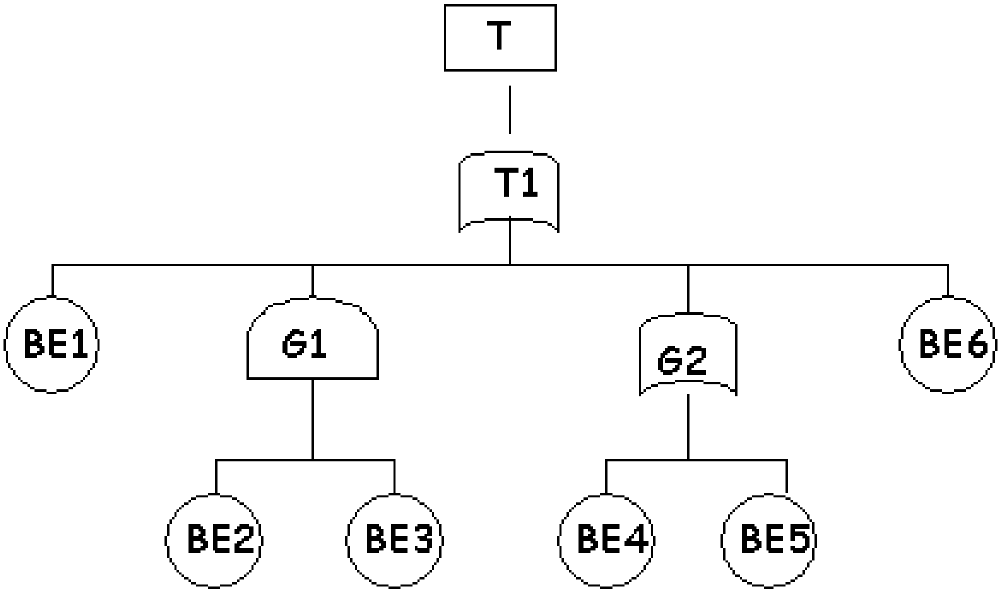


T1 = BE1 + BE2.BE3 + BE4.BE5 + BE6



T1 → BE1, G1, G2, BE6 → BE1, BE2 BE3, G2, BE6 → BE1, BE2 BE3, BE4 BE5, BE6

Probabilistic Analysis



For simplicity assume probability of all basic events is 0.1

P(G1) = P(BE2).P(BE3) = 0.1 * 0.1 = 0.01.

P(G2) = P(BE4).P(BE5) = 0.1 + 0.1 = 0.2.

P(T1) = 0.01 + 0.2 + P(BE1) + P(BE2)
= 0.01 + 0.2 + 0.1 + 0.1
= 0.41

Probabilistic Analysis

- Beware: independence assumption.

“If the same event occurs multiple times/places in a tree, any quantitative calculation must correctly reduce the boolean equation to account for these multiple occurrences. Independence merely means that the event is not caused due to the failure of another event or component, which then moves into the realm of conditional probabilities.”

Clif Ericson, Boeing.

- Inclusion-exclusion expansion (Andrews & Moss).

Software Fault Trees

- As you'd expect.

- Starts with top-level failure

- Trace events leading to failure.

- But: dont use probabilistic assessments;

- If you find software fault path REMOVE IT!

Software Fault Trees

Leveson, N.G., Cha, S.S., Shimeall, T.J. "Safety Verification of Ada Programs using Software Fault Trees," IEEE Software, July 1991.

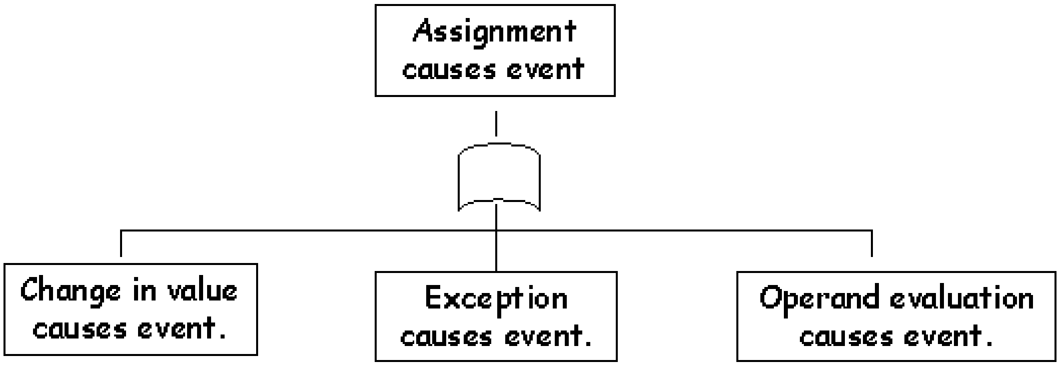
- Backwards reasoning.

- Weakest pre-condition approach.

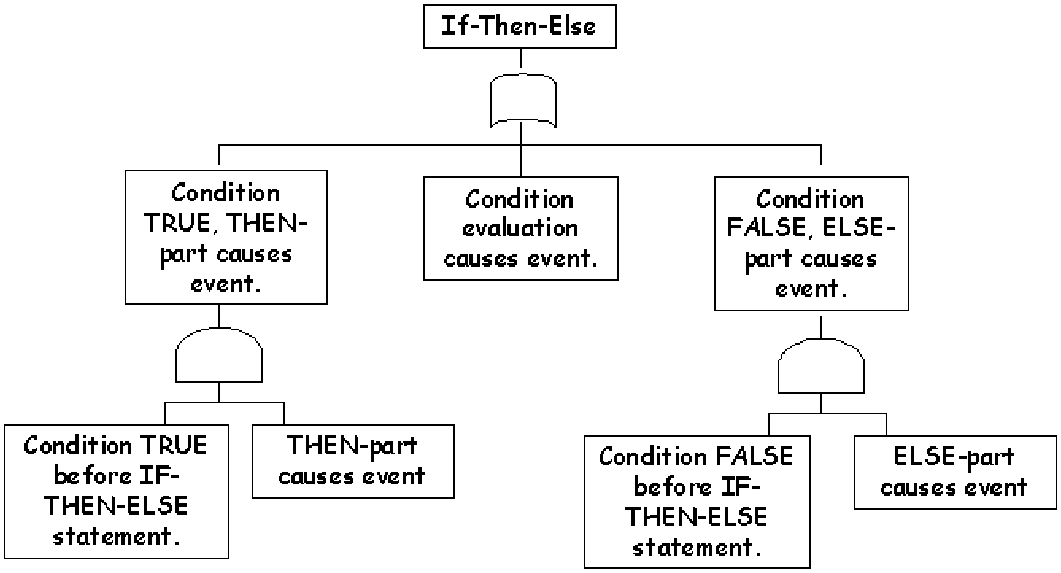
- Similar to theorem proving.

- Uses language dependent templates.

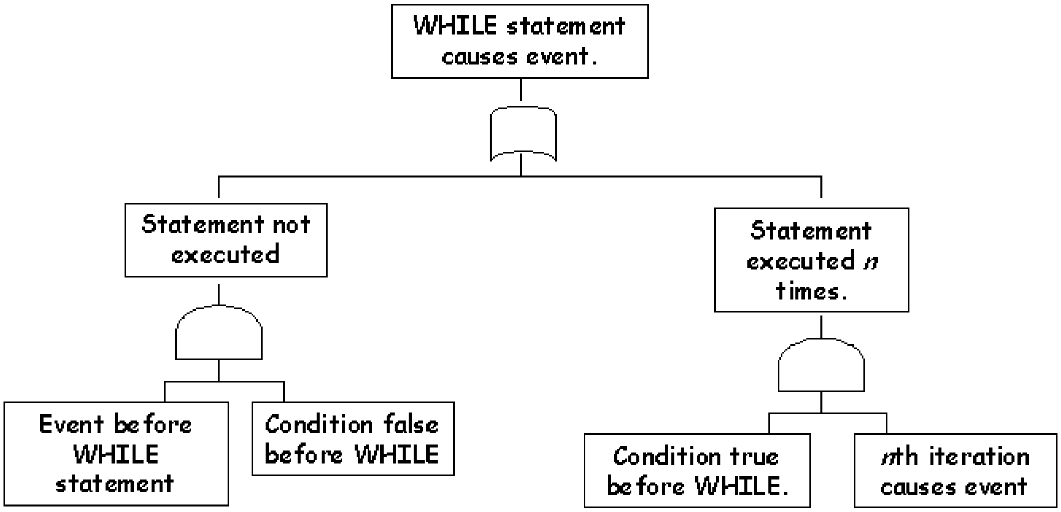
Software Fault Trees


Assignment causes event
Change in value causes event.
Exception causes event.
Operand evaluation causes event.

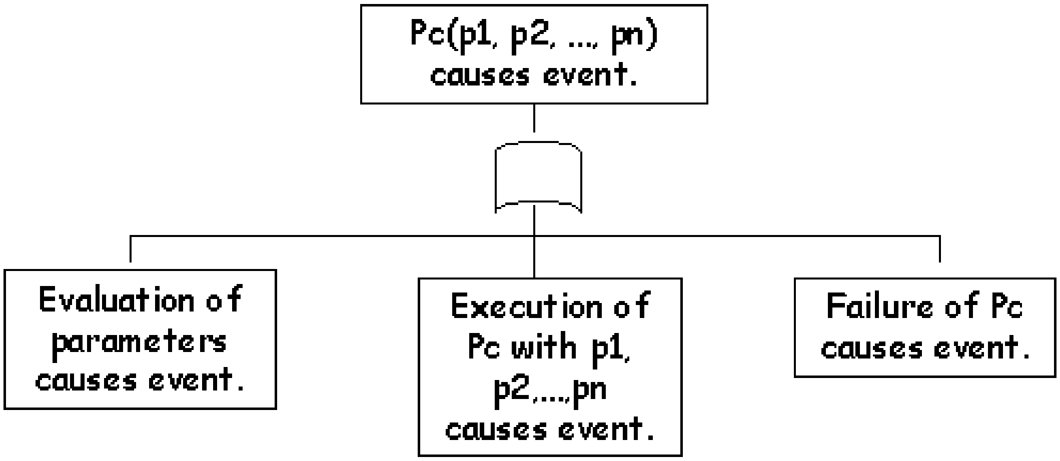
Software Fault Trees


If-Then-Else
Condition TRUE, THEN-part causes event.
Condition evaluation causes event.
Condition FALSE, ELSE-part causes event.
Condition TRUE before IF-THEN-ELSE statement.
THEN-part causes event
Condition FALSE before IF-THEN-ELSE statement.
ELSE-part causes event

Software Fault Trees

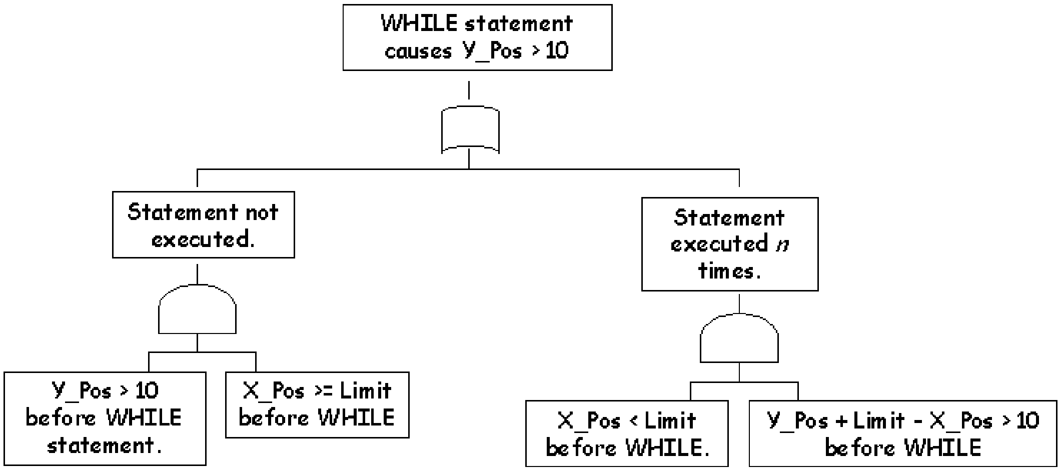

WHILE statement causes event.
Statement not executed
Statement executed n times.
Event before WHILE statement
Condition false before WHILE
Condition true before WHILE.
nth iteration causes event

Software Fault Trees



Pc(p1, p2, ..., pn) causes event.
Evaluation of parameters causes event.
Execution of Pc with p1, p2,...,pn causes event.
Failure of Pc causes event.

Software Fault Trees



WHILE statement causes Y_Pos > 10

Statement not executed.

Statement executed *n* times.

Y_Pos > 10 before WHILE statement.

X_Pos >= Limit before WHILE

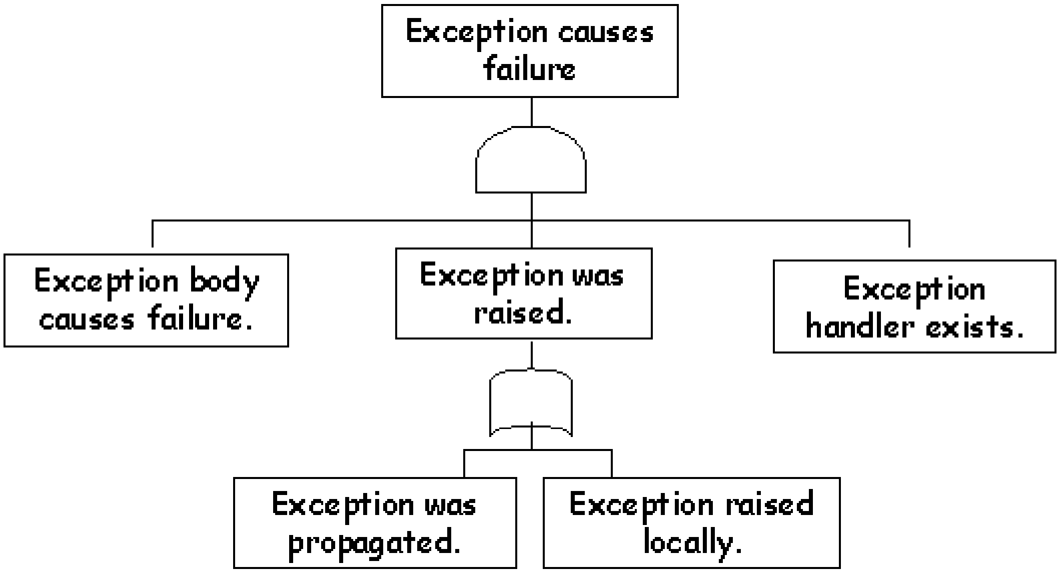
X_Pos < Limit before WHILE.

Y_Pos + Limit - X_Pos > 10 before WHILE

```
while X_Pos < Limit do
begin
        X_Pos := X_Pos + 1;
        Y_Pos := Y_Pos + 1;
end;
```

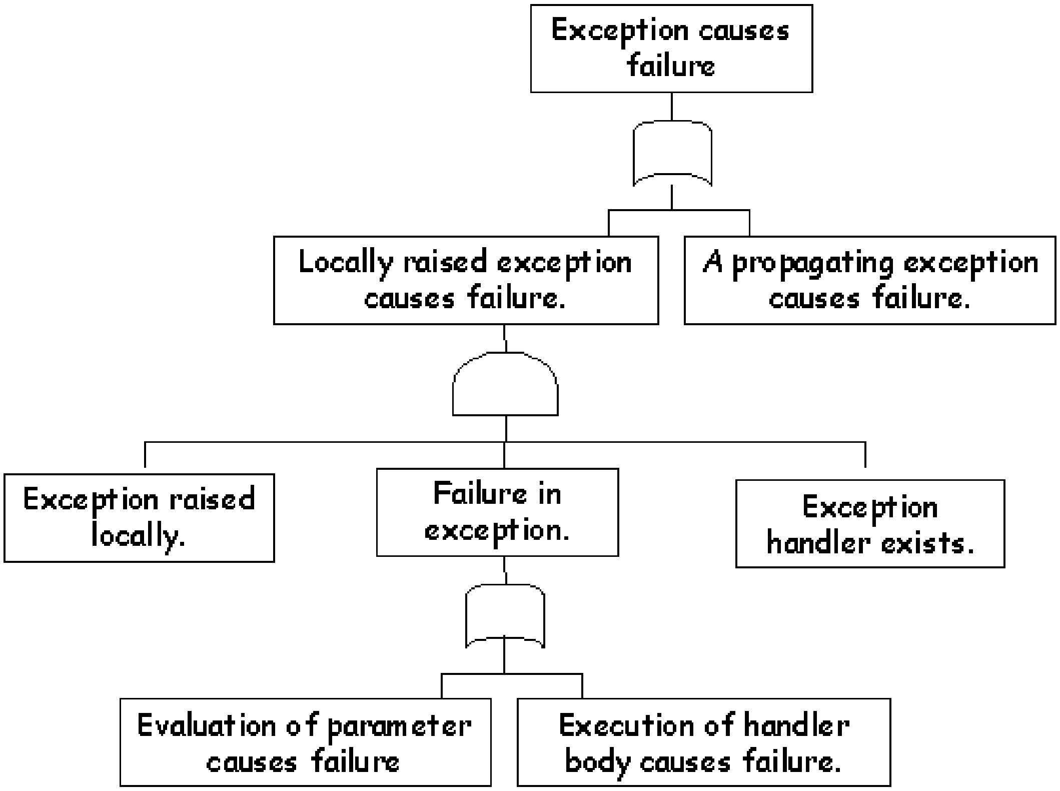
```
-- Let's assume dangerous condition
-- Y_Pos > 10
```

Software Fault Trees


Exception causes failure
Exception body causes failure.
Exception was raised.
Exception handler exists.
Exception was propagated.
Exception raised locally.

- Exception template for Ada83.

Software Fault Trees


Exception causes failure
Locally raised exception causes failure.
A propagating exception causes failure.
Exception raised locally.
Failure in exception.
Exception handler exists.
Evaluation of parameter causes failure
Execution of handler body causes failure.

See: S.-Y. Min, Y-K. Jang, A-D Cha, Y-R Kwon and D.-H. Bae, Safety Verification of Ada95 Programs Using Software Fault Trees. In M. Felici, K. Kanoun and A. Pasquini (eds.) Computer Safety, Reliability and Security, Springer Verlag, LNCS 1698, 2002.

- Exception template for Ada95.

PRA for Software

- John Musa's work at Bell Labs.

- Failure rate of software before tests.

- Faults per unot of time (λ_0):

- function of faults over infinite time.

- Based on execution time:

- not calendar time as in hardware;
- so no overall system predictions.

Musa's PRA for Software

$\lambda\_0 = KxPxW\_0$

k
Constant that accounts for the dynamic structure of the program and the varying machines
k = 4.2E-7

p
Estimate of the number of executions per time unit
p = r/SLOC/ER

r
Average instruction execution rate, determined from the manufacturer or benchmarking
Constant

SLOC
Source lines of code (not including reused code).

Musa's PRA for Software

$\lambda\_0 = KxPxW\_0$

ER

Expansion ratio, a constant dependent upon programming language

Assembler, 1.0; Macro Assembler, 1.5; C, 2.5; COBAL, FORTRAN, 3; Ada, 4.5

W_0

Estimate of the initial number of faults in the program

Can be calculated using: w0 = N x B or a default of 6 faults/1000 SLOC can be assumed

N

Total number of inherent faults

Estimated based upon judgment or past experience

B

Fault to failure conversion rate; proportion of faults that become failures. Proportion of faults not corrected before the product is delivered. Assume B = .95; i.e., 95undetected at delivery become failures after delivery

PRA for Software

- Considerable debate about this.

- Many variants on the theme.

- Metrics are crude...

- In meantime, be skeptical.

FTA - Conclusions

- Fault Trees:

- cut sets, cut paths;
- quantitative analysis.

- Software Fault Trees:

- language dependent templates;
- if you see faults, remove them!

- Software PRA.

Introduction

- Why is software different?

- Software requirements:

- Leveson's completeness criteria.

- Software design (summary):

MIL-338B preliminary design;
MIL-338B detailed design.

Why is Software Different?

Software is an abstract concept in that it is a set of instructions on a piece of paper or in computer memory. It can be torn apart and analyzed in piece parts like hardware, yet unlike hardware it is not a physical entity with physical characteristics which must comply with the laws of nature (i.e., physics and chemistry).

Since software is not a physical entity it does not wear out or degrade over time. This means that software does not have any failure modes per se. Once developed it always works the same without variation

Unlike hardware, once a software program is developed it can be duplicated or manufactured into many copies without any manufacturing variations.

Software is much easier to change than is hardware. For this reason many system fixes are made by modifying the software rather than the hardware.

There are no standard parts in software as there are with hardware. Therefore there are no high reliability software modules, and no industry alerts on poor quality software items.

If software has anything which even resembles a failure mode, it is in the area of hardware induced failures.

Why is Software Different?

Hardware reliability prediction is based upon random failures, whereas software reliability prediction is based upon the theory that predestined errors exist in the software program.

Hardware reliability modeling is well established, however, there is no uniform, accurate or practical approach to predicting and measuring software reliability.

Since software does not have any failure modes, a software problem is referred to as a software error. A software error is defined as a situation when the software does not perform to specifications or as reasonably expected, that is when it performs unintended functions. This definition is fairly consistent with that of a hardware failure, except that the mechanisms or causes of failure are very different.

Hardware primarily fails due to physical or chemical mechanisms and seldom fails due to human failure mechanisms (e.g., documentation errors, coding errors, specification oversights), whereas just the opposite is true with software.

Software has many more failure paths than hardware, making it difficult to test all paths.

By itself software can do nothing and is not hazardous. Software must be combined with hardware in order to do anything.

Clif Ericosn, Boeing.

Software Defects (Initial Views)

A software defect is either a fault or discrepancy between code and documentation that compromises testing or produces adverse effects in installation, modification, maintenance, or testing.

Requirements Defects: Failure of software requirements to specify the environment in which the software will be used, or requirements documentation that does not reflect the design of the system in which the software will be employed.

Design Defects: Failure of designs to satisfy requirements, or failure of design documentation to correctly describe the design.

Code Defects: Failure of code to conform to software designs.

Robert Dunn, Software Defect Removal, McGraw-Hill, 1984.

Software Hazard Analysis

- Already seen software fault trees.

1. Trace identified software hazards to the software-hardware interface. Translate the identified software related hazards into requirements and constraints on software behaviour.

2. Show the consistency of the software safety constraints with the software requirements specification. Demonstrate the completeness of the software requirements, including the human-computer interface requirements, with respect to system safety properties.

Acknowledgement: Nancy Leveson, Safeware: System Safety and Computers, Addison Wesley, Reading Massachusetts, 1995.

- Point 2 links to safety case slides?

Software Requirements Analysis

- Leveson identifies 3 components.

- Basic function or objective.

- Constraints on operating conditions.

- Prioritised quality goals;

- to help make tradeoff decisions.

- Same as general hazard analysis?

Kernel Requirements and Intent Specifications

- Kernel or core set of requirements.

- Determined by current knowledge of:

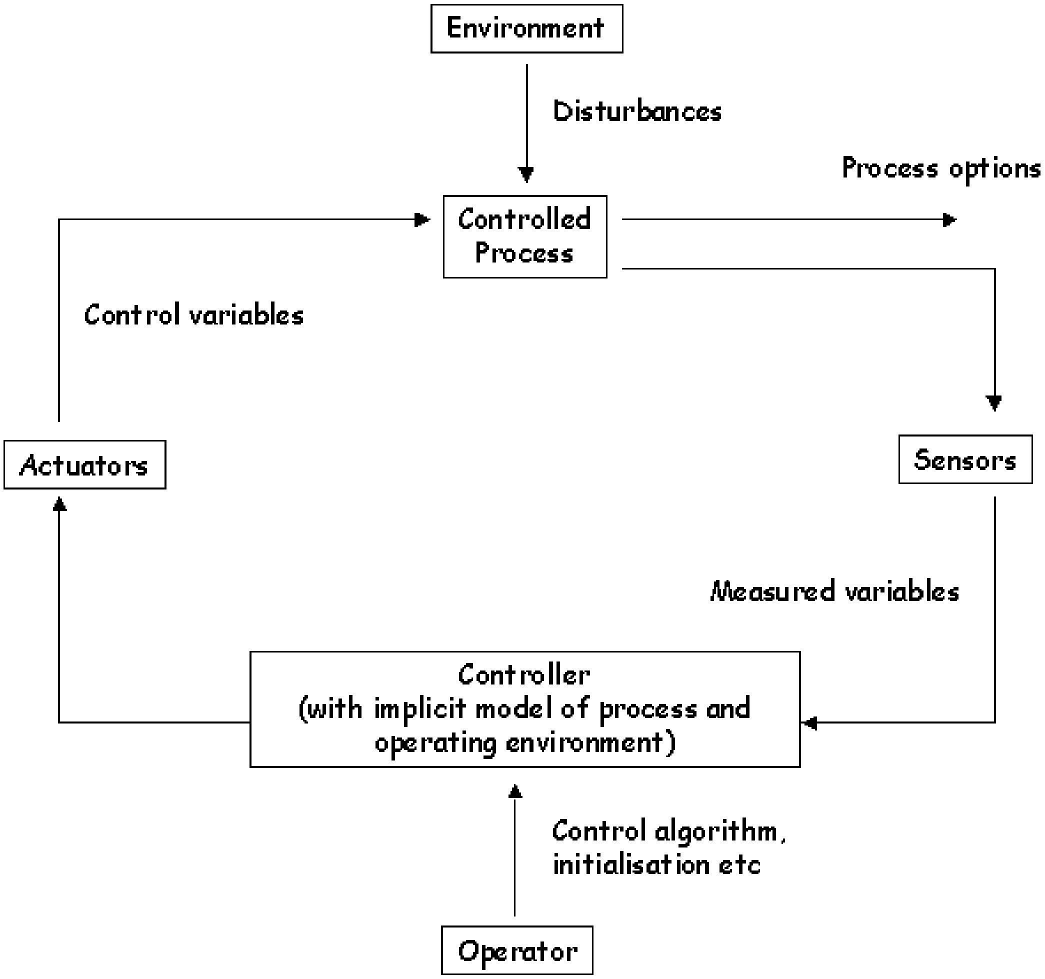
- intended application functionality;
- environment & constraints.

- Analytically independent.

- Only know they are complete if

- we know specification intent...

Leveson's Completeness Criteria



- Remember - ‘Black Box’ architecture.

Leveson's Completeness Criteria

- Human Computer Interface Criteria.
- State Completeness.
- Input/Output Variable Completeness.
- Trigger Event Completeness.
- Output Specification Completeness.
- Output to Trigger Relationships.
- State Transitions.

Leveson's Completeness Criteria

- Human Computer Interface Criteria.

- Criteria depend on task context.

- Eg in monitoring situation:

- what must be observed/displayed?
- how often is it sampled/updated?
- what is message priority?

- Not just when to present but also

- when to remove information...

Leveson's Completeness Criteria

- State Completeness Criteria.

- Consider input effect when state is:
- normal, abnormal, indeterminate.

- Start-up, close-down are concerns.

- Process will change even during
- intervals in which software is ‘idle’.

- Checkpoints, timeouts etc.

Leveson's Completeness Criteria

- Input/Output Variable Completeness.

- Input from sensors to software.

- Output from software to actuators.

- Specification may be incomplete if:

- sensor isnt refered to in spec;
- legal value isnt used in spec.

Leveson's Completeness Criteria

- Trigger Event Completeness.

Robustness:
every state has a transition defined for every possible input.

Non-determinism:
only 1 transition is possible from a state for each input.

Value and Timing assumptions:
- what triggers can be produced from the environment?
- what ranges must trigger variables fall within?
- what are the real-time requirements...
- specify bounds for responses to input (timeouts)

- And much, much more...

Leveson's Completeness Criteria

- Output Specification Completeness.

- from software to process actuators.

- Check for hazardous values.

- Check for hazardous timings;

- how fast do actuators take events?
- what if this rate is exceeded?

Leveson's Completeness Criteria

- Output to Trigger Relationships.

- Links between input & output events.

- For any output to actuators:

- can effect on process be detected?
- if output fails can this be seen?

- What if response is:

- missing, too early or too late?

- If response recieved without trigger

- then erroneous state.

Leveson's Completeness Criteria

- State Transitions.

Reachability:
all specified states can be reached from initial state.

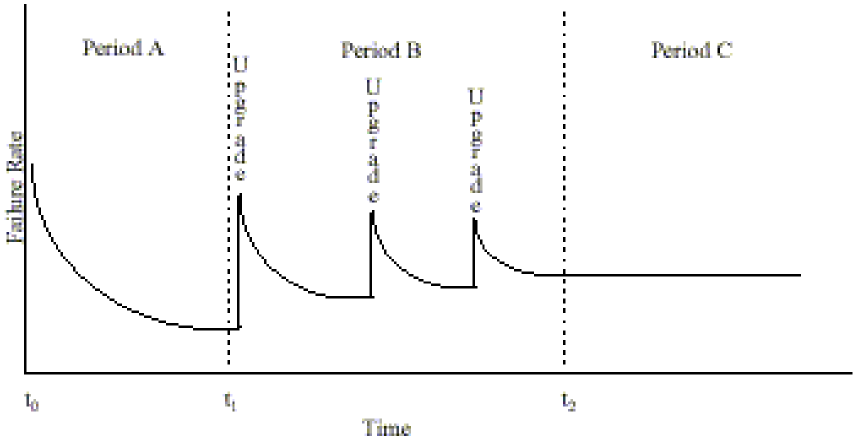
Recurrent behaviour:
desired recurrent behaviour must execute for at least one cycle and be bounded by exit condition.

Reversibility:
output commands should wherever possible be reversible and those which are not must be carefully controlled.

Preemption:
all possible preemption events must be considered for any non-atomic transactions.

- Again more complexity here...

Reality Check...



- Completeness criteria change.

- Environment and functions change.

From Requirements to Design

Once the requirements have been detailed and accepted, the design will process of allocating and arranging the functions of the system so that the aggregate meets all customer needs. Since several different designs may meet the requirements, alternatives must be assessed based on technical risks, costs, schedule, and other considerations. A design developed before there is a clear and concise analysis of the systems objectives can result in a product that does not satisfy the requirements of its customers and users. In addition, an inferior design can make it very difficult for those who must later code, test, or maintain the software. During the course of a software development effort, analysts may offer and explore many possible design alternatives before choosing the best design.

US Department of Defence: Electronic Reliability Design Handbook

Preliminary Design

Preliminary or high-level design is the phase of a software project in which the major software system alternatives, functions, and requirements are analyzed. From the alternatives, the software system architecture is chosen and all primary functions of the system are allocated to the computer hardware, to the software, or to the portions of the system that will continue to be accomplished manually.

US Department of Defence: Electronic Reliability Design Handbook

Preliminary Design

Develop the architecture:

system architecture - an overall view of system components hardware architecture - the systems hardware components and their interrelations

software architecture - the systems software components and their interrelations

Investigate and analyze the physical alternatives for the system and choose solutions

Define the external characteristics of the system

Refine the internal structure of the system by decomposing the high-level software architecture

Develop a logical view or model of the systems data

US Department of Defence: Electronic Reliability Design Handbook

Detailed Design

Detailed design or low-level design determines the specific steps required for each component or process of a software system. Responsibility for detailed design may belong to either the system designers (as a continuation of preliminary design activities) or to the system programmers. Information needed to begin detailed design includes: the software system requirements, the system models, the data models, and previously determined functional decompositions. The specific design details developed during the detailed design period are categories: for the system as a whole (system specifics), for individual processes within the system (process specifics), and for the data within the system (data specifics).

US Department of Defence: Electronic Reliability Design Handbook

Detailed Design (Example concerns)

System specifics:

- Physical file system structure
- Interconnection records or protocols between software and hardware
- Packaging of units as functions, modules or subroutines
- Interconnections among software functions and processes
- Control processing
- Memory addressing and allocation
- Structure of compilation units and load modules

US Department of Defence: Electronic Reliability Design Handbook

Detailed Design (Example concerns)

Process specifics:

- Required algorithmic details
- Procedural process logic
- Function and subroutine calls
- Error and exception handling logic

US Department of Defence: Electronic Reliability Design Handbook

Detailed Design (Example concerns)

Data specifics:

- Global data handling and access
- Physical database structure
- Internal record layouts
- Data translation tables
- Data edit rules
- Data storage needs

US Department of Defence: Electronic Reliability Design Handbook

Don't Forget the Impact of Standards

Sean Matthews sean@aipna.edinburgh.ac.uk Fri, 30 Jun 89 13:49:12 BST

I have just seen a copy of the UK department of defence draft standard for safety critical software (00-55).

Here are a few high (and low) points.

1. There should be no dynamic memory allocation (This rules out explicit recursion - though a bounded stack is allowed).

2. There should be no interupts except for a regular clock interupt.

3. There should not be any distributed processing (i.e. only a single processor).

4. There should not be any multiprocessing.

5. NO ASSEMBLER.

6. All code should be at least rigourously checked using mathematical methods.

7. Any formally verified code should have the proof submitted as well, in machine readable form, so that an independent check can be performed.

8. All code will be formally specified.

9. There are very strict requirements for static analysis (no unreachable code, no unused variables, no unintialised variables etc.).

10. No optimising compilers will be used.

11. A language with a formally defined syntax and a well defined semantics, or a suitable subset thereof will be used.

Don't Forget the Impact of Standards

Comments.

1. means that all storage can be statically allocated. In fact somewhere it says that this should be the case.

2-4 seem to leave no option but polling. This is impractical, especially in embedded systems. No one is going to build a fly by wire system with those sorts of restrictions. (maybe people should therefore not build fly by wire systems, but that is another matter that has been discussed at length here already). it also ignores the fact that there are proof methods for dealing with distributed systems.

5. This is interesting, I seem to remember reading somewhere that Nasa used to have the opposite rule: no high level languages, since they actually read the delivered binary to check that the software did what it was supposed to do.

8. this is an excellent thing, though it does not say what sort of language should be used. Is a description in terms of a Turing machine suitable? After all that is a well understood formal system.

10. Interestingly, there is no requirement that the compiler be formally verified, just that it should conform to international standards (though strictly), and not have any gross hacks (i.e. optimisation) installed. There is also no demand that the target processor hardware be verified (though such a device exists here already: the Royal Signals Research Establishment's Viper processor).

11. seems to be a dig at Ada and the no subsets rule. It also rules out C.

Don't Forget the Impact of Standards

Conclusions.

I find the idea of the wholesale mayhem and killing merchants being forced to try so much harder to ensure that their products maim and kill only the people they are supposed to maim and kill, rather amusing.

The standard seems to be naive in its expectations of what can be achieved at the moment with formal methods (That is apparently the general opinion around here, and there is a *lot* of active research in program verification in Edinburgh), and impossibly restrictive.

An interesting move in the right direction but too fast and too soon. And they might blow the idea of Formal verification by tring to force it too soon. And I would very much like to see these ideas trickle down into the civil sector.

I might follow this up with a larger (and more coherent) description if there is interest (this was typed from memory after seeing it yesterday) there is quite a bit more in it.

Sean Matthews Dept. of Artificial Intelligence, University of Edinburgh.

Conclusion

- Why is software different?

- Software requirements:

- Leveson's completeness criteria.

- Software design (summary):

MIL-338B preliminary design;
MIL-338B detailed design.

Safety-Critical Software Development

- Software design by:

- hazard elimination;
- hazard reduction;
- hazard control.

- Software implementation issues:

- dangerous practices;
- choice of 'safe' languages.

- The DO-178B Case Study.

Leveson's Taxonomy of Design Techniques

- Hazard elimination/avoidance.
- Hazard reduction (see 4?).
- Hazard control.
- Hazard minimization (see 2?).

Software Design and Hazard Elimination

- Substitution:

hardware interlocks before software.

- Simplification:

new software features add complexity.

- Decoupling :

computers add common failure point.

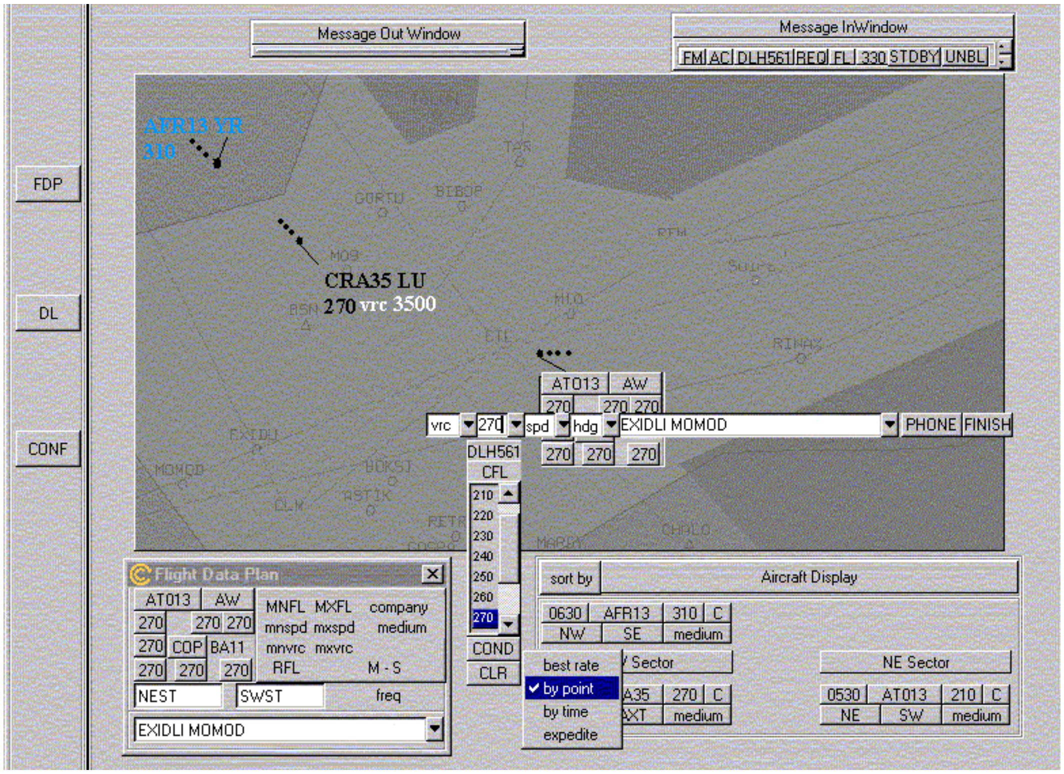
- Human Error ‘Removal’ :

readability of instruments etc.

- Removal of hazardous materials :

eliminate UNUSED code (Ariane 5).

Hazard Elimination: Datalink Example



Software Design and Hazard Reduction

- Design for control:

- incremental control;
- intermediate states;
- decision aids;
- monitoring.

- Add barriers:

- hard/software locks;

- Minimise single point failures:

- increase safety margins;
- exploit redundancy;
- allow for recovery.

Hazard Reduction: Interlock Example

This heavy duty solenoid controlled tongue switch controls access to hazardous machines with rundown times.

Olympus withstands the arduous environments associated with the frequent operation of heavy duty access guards. The unit also self adjusts to tolerate a high degree of guard misalignment. The stainless steel tongue actuator is self-locking and can only be released after the solenoid receives a signal from the machine control circuit. This ensures that the machine has completed it's cycle and come to rest before the tongue can be disengaged and machine access obtained.

Software Design and Hazard Control

- Limit exposure.

- back to 'normal' fast (exceptions).

- Isolate and contain.

- dont let things get worse...

- Fail-safe.

- panic shut-downs, watchdog code.

Hazard Control: Watchdog Example

- Hardware or software (beware).

- Check for processor activity:

- 1. load value into a timer;
- 2. decrement timer every interval;
- 3. if value is zero then reboot.

- Processor performs 1 at a frequency

- great enough to stop 3 being true;
- unless it has crashed.

Software Design Techniques: Fault Tolerance

- Avoid common mode failures.

- Need for design diversity.

- Same requirements:

- different programmers?
- different contractors?
- homogenous parallel redundancy?
- microcomputer vs PLC solutions?

Software Design Techniques: Fault Tolerance

- Redundant hardware may duplicate

- any faults if software is the same.

- N-version programming:

- shared requirements;
- different implementations;
- voting ensures agreement.

- What about timing differences?

- comparison of "continuous" values?
- what if requirements wrong?
- costs make N¿2 very uncommon;
- performance costs of voting.

- A340 primary flight controls.

Software Design Techniques: Fault Tolerance

- Exception handling mechanisms.

- Use run-time system to detect faults:

- raise an exception;
- pass control to appropriate handler;
- could be on another processor.

- Propagate to outmost scope then fail.

- Ada...

Software Design Techniques: Fault Tolerance

- Recovery blocks:

- write acceptance tests for modules;
- if it fails then execute alternative.

- Must be able to restore the state:

- take a snapshot/checkpoint;
- if failure restore snapshot.

- But:

- if failed module have side-effects?
- eg effects on equip under control?
- recovery block will be complicated.

- Different from execptions:

- dont rely on run-time system.

Software Design Techniques: Fault Tolerance

- Control redundancy includes:

- N-version programming;
- recovery blocks;
- exception handling.

- But data redundancy uses extra data

- to check the validity of results.

- Error correcting/detecting codes.

- Checksum agreements etc.

Software Implementation Issues

- Restrict language subsets.

- Alsys CSMART Ada kernel etc.

- Or just avoid high level languages?

- No task scheduler - bare machine.

- Less scheduling/protection risks

- more maintenance risks;
- less isolation (no modularity?).

Software Implementation Issues

- Memory jumps:

- control jumps to arbitrary location?

- Overwrites:

- arbitrary address written to?

- Semantics:

- established on target processor?

- Precision:

- integer, floating point, operations...

Software Implementation Issues

- Data typing issues:

- strong typing prevents misuse?

- Exception handling:

- runtime recovery supported?

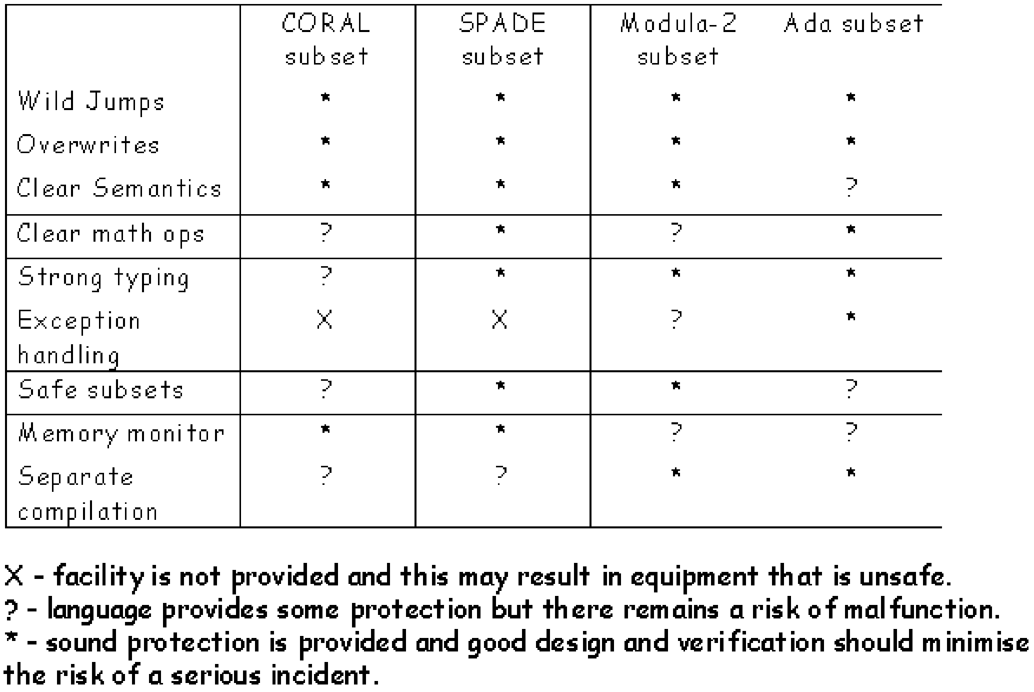
- Memory monitoring:

-guard against memory depletion?

- Separate compilation:

- type checking across modules etc?

Software Implementation Issues



	CORAL subset	SPADE subset	Modula-2 subset	Ada subset
Wild Jumps	*	*	*	*
Overwrites	*	*	*	*
Clear Semantics	*	*	*	?
Clear math ops	?	*	?	*
Strong typing	?	*	*	*
Exception handling	X	X	?	*
Safe subsets	?	*	*	?
Memory monitor	*	*	?	?
Separate compilation	?	?	*	*

X - facility is not provided and this may result in equipment that is unsafe.
? - language provides some protection but there remains a risk of malfunction.
* - sound protection is provided and good design and verification should minimise the risk of a serious incident.

Acknowledgement: W.J. Cullyer, S.J. Goodenough, B.A. Wichmann, The choice of a Computer Language for Use in Safety-Critical Systems, Software Engineering Journal, (6)2:51-58, 1991.

Software Implementation Issues

- CORAL subset:

staff training issues?

- SPADE Pascal:

Praxis version of ISO Pascal.

- Modula 2 subset:

SACEM trains in Paris.

- Ada subset:

attempts at formal verification.

- Meta question:

programmer more important than language?
or protect all programmers from themselves?

Software Implementation Issues

Newsgroups: comp.lang.ada,comp.lang.c++,comp.lang.misc,comp.software-eng From: cpp@netcom.com (Robin Rowe) Subject: Safety-Critical Survey (Results) Message-ID: Organization: NETCOM On-line Communication Services (408 261-4700 guest) Date: Sun, 13 Nov 1994 22:34:10 GMT Lines: 180

==================================

Here are the results of my recent informal survey of computer languages used in safety-critical embedded systems and other interesting systems. In responses, Ada was by far the most popular language for these systems followed by assembler. There is a list describing 722 Ada projects that is available via ftp from the Ada Information Clearinghouse. The current version is 213K in size (contact adainfo@ajpo.sei.cmu.edu). I did not attempt to integrate that data into this report.

No assertion is intended here that any language is necessarily superior to any other.

Software Implementation Issues

Aerospace:

- Boeing: Mostly Ada with assembler. Also: Fortran, Jovial, C, C++. Onboard fire extinguishers in PLM. 777 seatback entertainment system in C++ with MFC (in development by Microsoft). 757/767: approximately 144 languages used. 747-400: approximately 75 languages used. 777: approximately 35 languages used.
- Boeing Defense & Space Group: (777 cabin mgmt. system in Ada?)
- DAINA/Air Force: Aircraft mission manager in Ada.
- Chandler Evans: Engine Control System in Ada (386 DOS).
- Draper Labs/Army/NASA: Fault tolerant architecture in Ada/VHDL.
- European Space Agency: mandates Ada for mission critical systems. ISO (Infrared Space Observatory) SOHO (Solar and Heliospheric Observatory) Huygens/Cassini (a joint ESA/NASA mission to Saturn) Companies involved: British Aerospace (Space Systems) - Bristol, UK Fokker Space Systems - Amsterdam, Holland Matra-Marconi Espace - Toulouse, France Saab - Sweden Logica - UK DASA - Germany MBB - Germany
- Ford Aerospace: Spacecraft in Ada with assembler. GEOS and INSAT spacecraft in FORTRAN. (Ford Aerospace is now Space Systems/Loral.)
- Hamilton-Standard: (777 air cowling icing protection system in Ada?).

Software Implementation Issues

Aerospace (Continued):

- Honeywell: Aircraft navigation data loader in C. (777 airplane information mgmt. system in Ada?)
- Intermetrics/Houston: space shuttle cockpit real-time executive in Ada '83 with 80386 assembly
- Lockheed Fort Worth: F-22 Advanced Tactical Fighter program in Ada 83 (planning to move to Ada 94) with a very small amount in MIL-STD-1750A assembly. Maintain older safety-critical systems for the F-111 and F-16/F-16 variant airframes primarily done in JOVIAL.
- NASA: Space station in Ada. (Sources differed on whether it was Ada only, or Ada with some C and assembler.)
- NASA Lewis: March 1994 space shuttle experiment in C++ on 386.
- Rockwell Space Systems Div.: Space shuttle in Hal/s and Ada. Defense Initiative in Ada. Other systems in Ada and C.
- Space Systems/Loral: Spacecraft in Ada with assembler.
- Teledyne: Aircraft flight data recorder in C.
- TRW/Air Force: Realtime avionics OS in Ada.
- Wilcox Electric: Navigation aids in C prior to 1990, Ada after. VOR-DME in Ada. Microwave landing system in Ada. Wide Area GPS in C and C++.

Software Implementation Issues

Air Traffic Control:

- Hughes: Canadian ATC system in Ada.
- Loral FSD: U.S. ATC system in Ada.
- Thomson-CSF SDC: French ATC system in Ada.

Land Vehicles:

- Delco: Engine controls and ABS in 68C series (Motorola) assembler. C++ used for data acquisition in GM research center. '93+ GM trucks vehicle controllers mostly in Modula-GM (Modula-GM is a variant of Modula-2. A typical 32-bit integrated vehicle controller may control the engine, the transmission, the ABS system, the Heating/AC system, as well as the associated integrated diagnostics and off-board communications systems.)
- Ford: Assembler.
- General Dynamic Land Systems: M1A2 tank tank software in Ada with time-critical routines in 68xxx assembler. Tank software simulators in C.
- Lucas: Many systems in Lucol (Lucas control language). Diesel engine controls in C++. ABS in 68xxx assembler.

Software Implementation Issues

Ships:

- Vosper Thornycroft Ltd (UK): navigation control in Ada.

Trains:

- CSEE Transports (France): TGV Braking system in Ada (68K). Denver Airport baggage system: This well publicized problem system is written in C++. (A source familiar with the system said the problems were political and managerial, not directly related to C++.)
- European Rail: Switching system in Ada.
- EuroTunnel: in Ada.
- Extension to the London Underground: in Ada.
- GEC Alsthom (France): Railway and signal control systems for trains and the TGV (north lines and Chunnel) in Ada. Subway network control systems (Paris, Calcutta, and Cairo).
- TGV France: Switching system in Ada.
- Union Switch & Signal, Pittsburgh: (Switching system in ?)
- Westinghouse Signals Ltd (UK): Railway signalling systems in Ada.
- Westinghouse Brake & Signal UK: Automatic Train Protection (ATP) systems for Westrace project in PASCAL.
- Westinghouse Australia: ATP systems in PASCAL and ADA.

Software Implementation Issues

Medical:

- Baxter: Left Ventricular Heart Assist in C with 6811 assembler.
- Coulter Corp.: ONYX hematology analyzer in Ada.

Nuclear Reactors:

- Core and shutdown systems in assembler, migrating to Ada.

SURVEY METHODOLOGY

I operated under the theory that, with regard to what languages are really in use, the recollections of the engineers themselves are probably the most accurate and open source. In general, I did not have enough sources that I could cross check the information. In cases where I could, the most interesting discrepancy was that companies that thought they had adopted one language as the total solution for all their software designs often had something in assembler or some other language somewhere.

Software Implementation Issues

Every response to the survey was positive except one. An individual at Rockwell Collins said: “The language(s) we do/don’t use is a matter best left to us, our customers, and the appropriate regulatory agencies governing our businesses and markets. All of these parties also look out for the public’s interests in safety, cost, etc. as well.” This individual took me to task for not contacting the PR department of his company, but was unwilling to help me do so. Per his request, I have omitted his company.

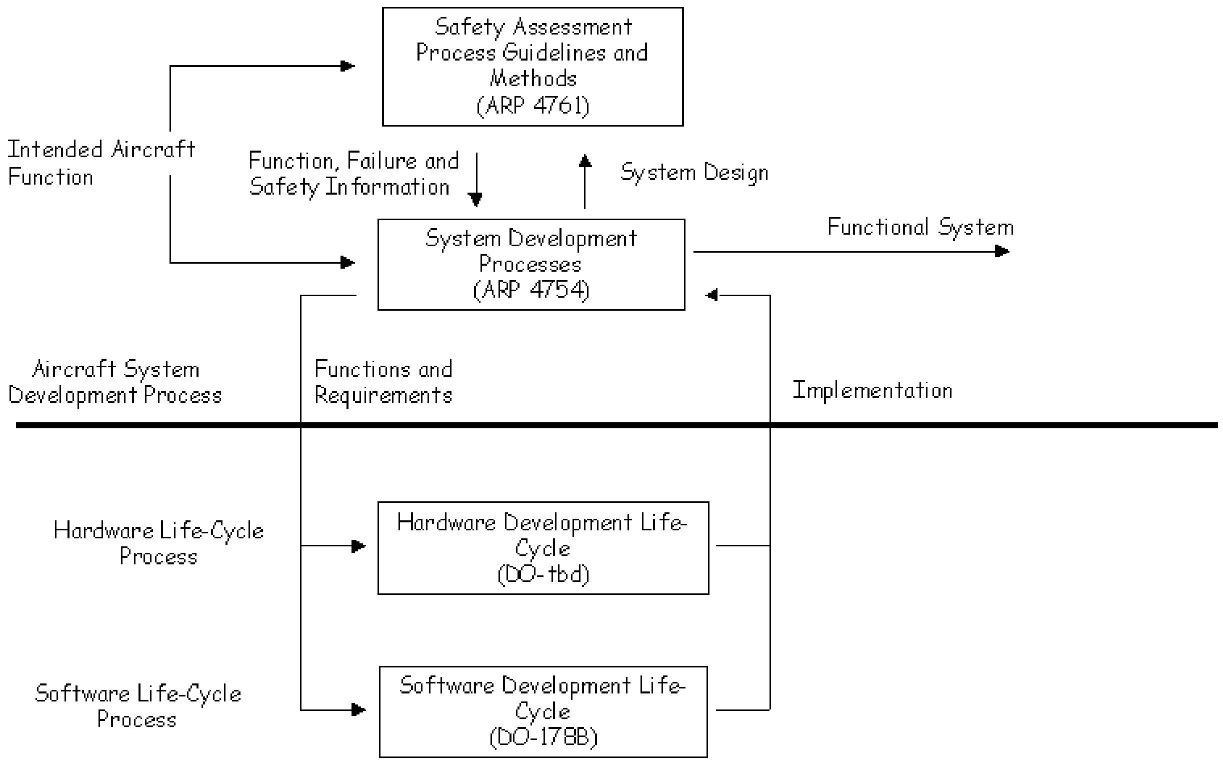
If you wish to add information or make a correction please send mail to cpp@netcom.com. I’d like to fill in the companies that have question marks by them. I’m particularly interested in systems written in C++. Names of respondents are held confidential. If you respond with a public follow-up on the net, please cc via e-mail to me so that I don’t miss you.

Thanks to everyone who helped with this. I meant to post this in August, but got busy with work and relocating to Monterey and forgot. Sorry for the delay.

Robin

cpp@netcom.com, Rowe Technology.

Software Development: DO-178B


Safety Assessment Process Guidelines and Methods (ARP 4761)
Intended Aircraft Function
Function, Failure and Safety Information
System Design
System Development Processes (ARP 4754)
Functional System
Aircraft System Development Process
Functions and Requirements
Implementation
Hardware Life-Cycle Process
Hardware Development Life-Cycle (DO-tbd)
Software Life-Cycle Process
Software Development Life-Cycle (DO-178B)

- Software Considerations in

in Airborne Systems and Equipment Certification.

Software Development: DO-178B

- Planning Process:

- coordinates development activities.

- Software Development Processes:

- requirements process
- design process
- coding process
- integration process.

- Software Integral Processes:

- verification process
- configuration management
- quality assurance
- certification liaison.

Software Development: DO-178B

(a) A detailed description of how the software satisfies the specified software high-level requirements, including algorithms, data-structures and how software requirements are allocated to processors and tasks.

(b) The description of the software architecture defining the software structure to implement the requirements.

c) The input/output description, for example, a data dictionary, both internally and externally throughout the software architecture.

(d) The data flow and control flow of the design.

(e) Resource limitations, the strategy for managing each resource and its limitations, the margins and the method for measuring those margins, for example timing and memory.

(f) Scheduling procedures and interprocessor/intertask communication mechanisms, including time-rigid sequencing, pre-emptive scheduling, Ada rendez-vous and interrupts.

Software Development: DO-178B

(g) Design methods and details for their implementation, for example, software data loading, user modifiable software, or multiple-version dissimilar software.

(h) Partitioning methods and means of preventing partitioning breaches.

(i) Descriptions of the software components, whether they are new or previously developed, with reference to the baseline from which they were taken.

(j) Derived requirements from the software design process.

(k) If the system contains deactivated code, a description of the means to ensure that the code cannot be enabled in the target computer.

(l) Rationale for those design decisions that are traceable to safety-related system requirements.

- Deactivated code (k) (see Ariane 5).

- Traceability issues interesting (l).

Software Development: DO-178B - Key Issues

- Traceability and lifecycle focus.

- Designated engineering reps.

- Recommended practices.

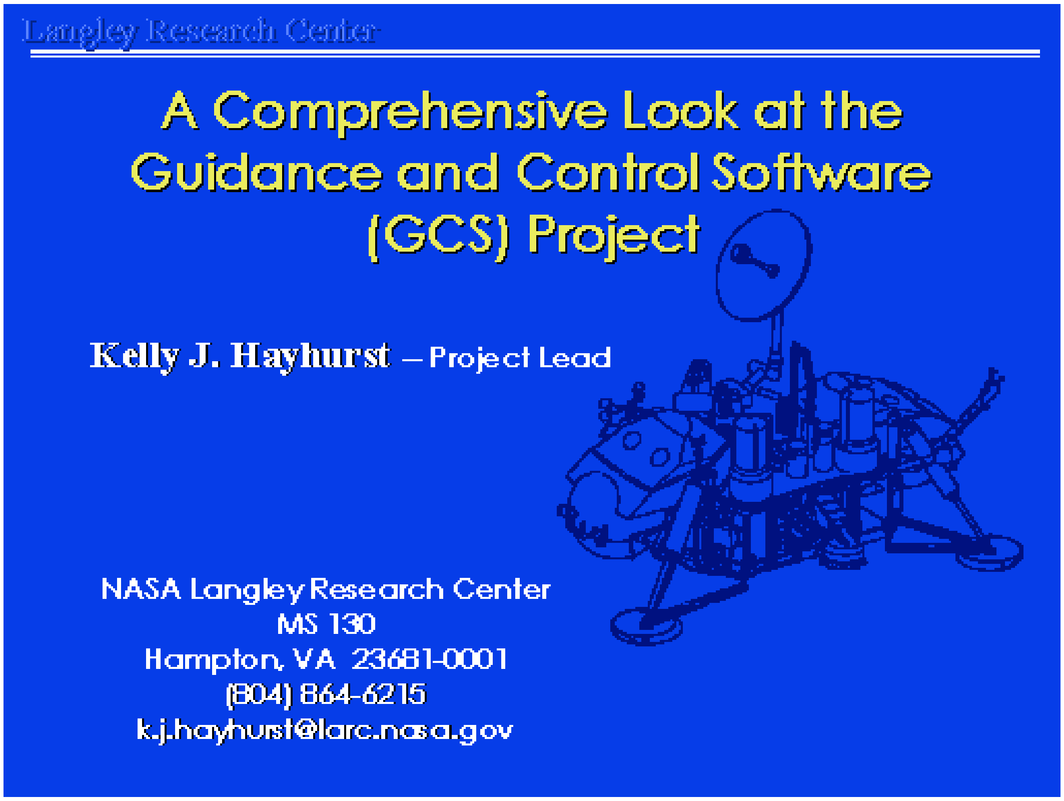
- Design verification:

- formal methods “alternative” only;
- “inadequate maturity”;
- limited applicability in aviation.

- Design validation:

- use of independent assessors etc.

DO-178B - NASA GCS Case Study



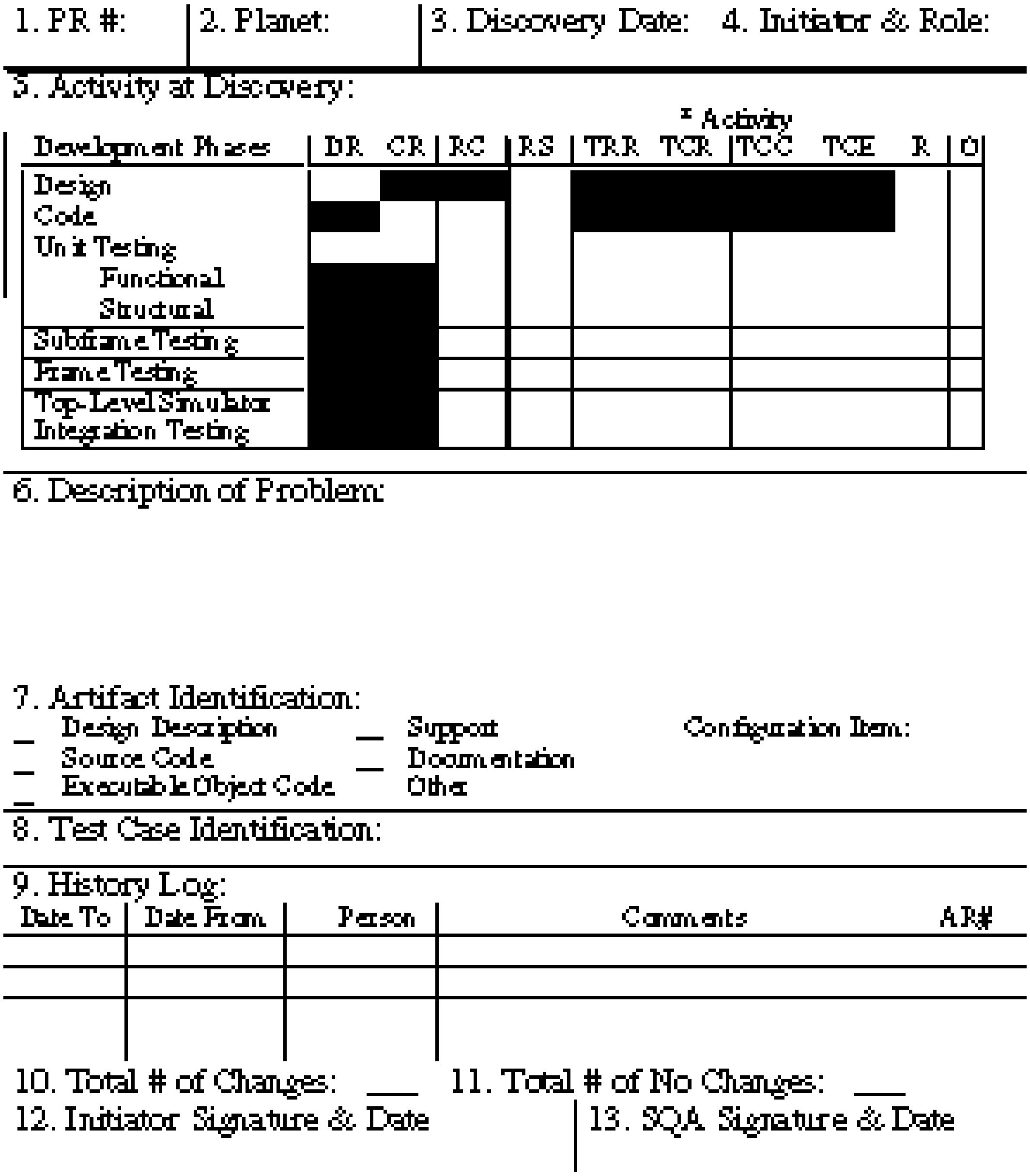
Langley Research Center

A Comprehensive Look at the Guidance and Control Software (GCS) Project

Kelly J. Hayhurst – Project Lead

NASA Langley Research Center
MS 130
Hampton, VA 23681-0001
(804) 864-6215
k.j.hayhurst@larc.nasa.gov

DO-178B - NASA GCS Case Study



1. PR #: | 2. Planet: | 3. Discovery Date: 4. Initiator & Role:

5. Activity at Discovery:

* Activity

Development Phases	DR	CR	RC	RS	TRR	TCR	TCC	TCE	R	O
Design		■	■		■	■	■	■		
Code	■				■	■	■	■		
Unit Testing										
Functional	■	■								
Structural	■	■								
Subframe Testing	■	■								
Frame Testing	■	■								
Top-Level Simulator Integration Testing	■	■								

6. Description of Problem:

7. Artifact Identification:

_ Design Description _ Support Configuration Item:
_ Source Code _ Documentation
_ Executable Object Code Other

8. Test Case Identification:

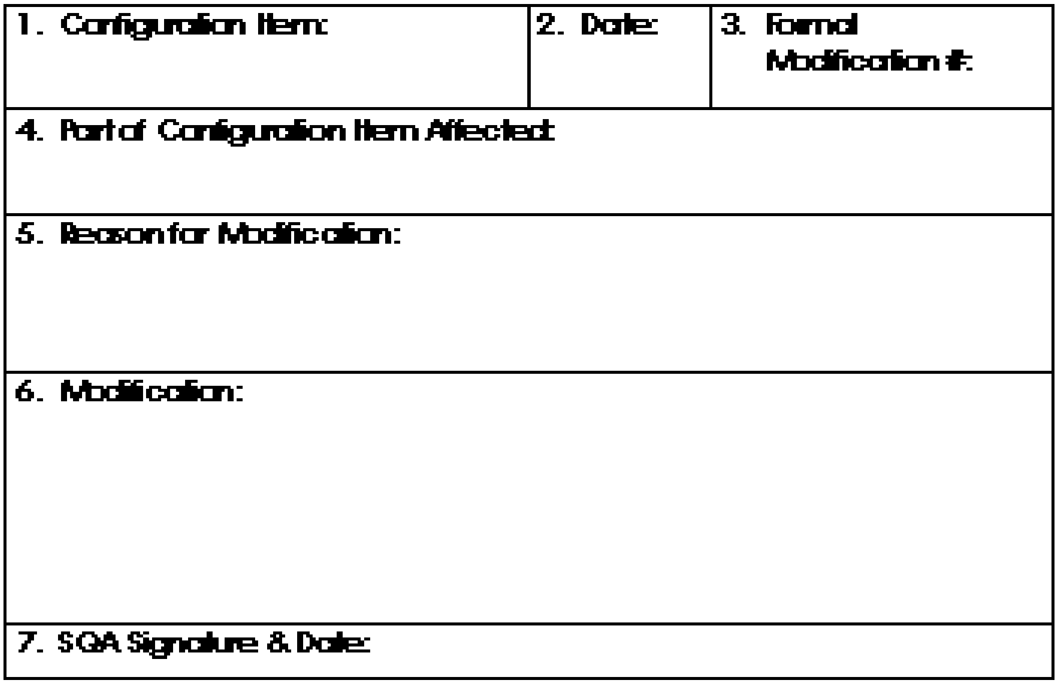
9. History Log:

Date To	Date From	Person	Comments	AR#

10. Total # of Changes: ___ 11. Total # of No Changes: ___

12. Initiator Signature & Date | 13. SQA Signature & Date

DO-178B - NASA GCS Case Study



1. Configuration Item:	2. Date:	3. Formal Modification #:
4. Part of Configuration Item Affected:		
5. Reason for Modification:		
6. Modification:		
7. SQA Signature & Date:		

DO-178B - NASA GCS Case Study

- Project compared:

- faults found in statistical tests;
- faults found in 178B development.

- Main conclusions:

- such comparisons very difficult;
- DO-178B hard to implement;
- lack of materials/examples.

Software Development: DO-178B Practitioners' View

The difficulties that have been identified are the DO-178 requirements for evidence and rigorous verification... Systematic records of accomplishing each of the objectives and guidance are necessary. A documentation trail must exist demonstrating that the development processes not only were carried out, but also were corrected and updated as necessary during the program life cycle. Each document, review, analysis, and test must have evidence of critique for accuracy and completeness, with criteria that establishes consistency and expected results. This is usually accomplished by a checklist which is archived as part of the program certification records. The degree of this evidence varies only by the safety criticality of the system and its software.

...Engineering has not been schooled or trained to meticulously keep proof of the processes, product, and verification real-time. The engineers have focused on the development of the product, not the delivery. In addition, program durations can be from 10 to 15 years resulting in the software engineers moving on by the time of system delivery. This means that most management and engineers have never been on a project from "cradle-to-grave."

Original source on http://stsc.hill.af.mil/crosstalk/1998/oct/schad.asp

Software Development: DO-178B Practitioners' View

The weakness of commercial practice with DO-178B is the lack of consistent, comprehensive training of the FAA engineers/DERs/foreign agencies affecti ng:

- the effectiveness of the individual(s) making findings; and,
- the consistency of the interpretations in the findings.

Training programs may be the answer for both the military and commercial environments to avoid the problem of inconsistent interpretation and the results of literal interpretation.

Original source on http://stsc.hill.af.mil/crosstalk/1998/oct/schad.asp

Safety-Critical Software Development - Conclusions

- Software design by:

- hazard elimination;
- hazard reduction;
- hazard control.

- Software implementation issues:

- dangerous practices;
- choice of ‘safe’ languages.

- The DO-178B Case Study.

Conclusions

- Software design by:

- hazard elimination;
- hazard reduction;
- hazard control.

- Software implementation issues:

- dangerous practices;
- choice of ‘safe’ languages.

- The DO-178B Case Study.

Hardware Design: Fault Tolerant Architectures

- The basics of hardware management.

- Fault models.

- Hardware redundancy.

- Space Shuttle GPC Case Study.

Parts Management Plan

- MIL-HDBK-965

- help on hardware acquisition.

- General dependability requirements.

- Not just about safety.

- But often not considered enough...

The Basics: Hardware Management

- MIL-HDBK-965

Acquisition Practices for Parts Management

- Preferred Parts List

- Vendor and Device Selection

- Critical Devices, Technologies & Vendors

- Device Specifications

- Screening

- Part Obsolescence

- FRACAS:

- Failure Reporting, Analysis and Corrective Action

Types of Faults

- Design faults:

- erroneous requirements;
- erroneous software;
- erroneous hardware.

- These are systemic failures;

- not due to chance but design.

- Dont forget management/regulators!

Types of Faults

- Intermittent faults:

- fault occurs and recurrs over time;
- fault connections can recur.

- Transient faults:

- fault occurs but may not recurr;
- electromagnetic interference.

- Permanent faults:

- fault persists;
- physical damage to processor.

Fault Models

- Single stuck-at models.

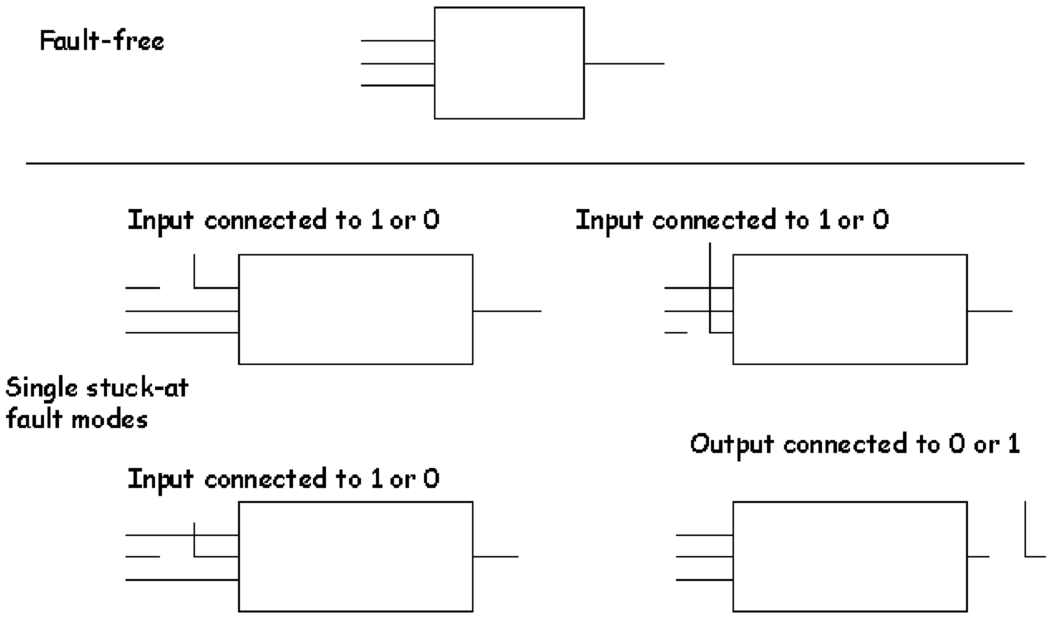
- Hardware seen as 'black-box'.

- Fault modelled as:

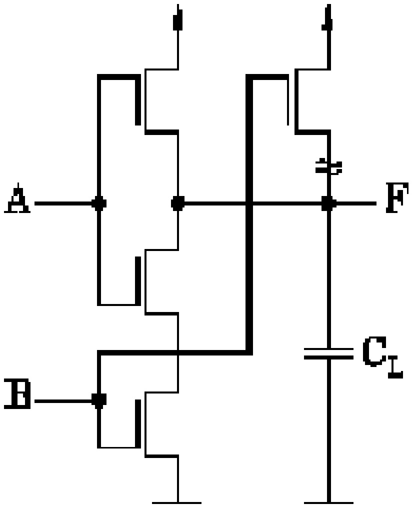
- input or output error;
- stuck at either 1 or 0.

- Models permanent faults.

Fault Models - Single Stuck-At...


Fault-free
Input connected to 1 or 0
Input connected to 1 or 0
Single stuck-at
fault modes
Output connected to 0 or 1
Input connected to 1 or 0

Fault Models



A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

CMOS - NAND Gate - fault detection on:
(A1, B1, A1, B0), (A1, B1, A0, B1) and (A1, B1, A0, B0).

Fault Models

- Bridging Model:

- input not 'stuck-at' 1 or 0;
- but shorting of inputs to circuit;
- input then is wired-or/wired-and.

- Stuck-open model:

- both CMOS output transistors off;
- results is neither high nor low...

- Transition and function models.

Software Faults (Aside...)

- Much more could be said...

- see Leveson or Storey.

- Huge variability:

- specification errors;
- coding errors;
- translation errors;
- run-time errors...

Redundancy

- Adds:

- cost;
- weight;
- power consumption;
- complexity (most significant).

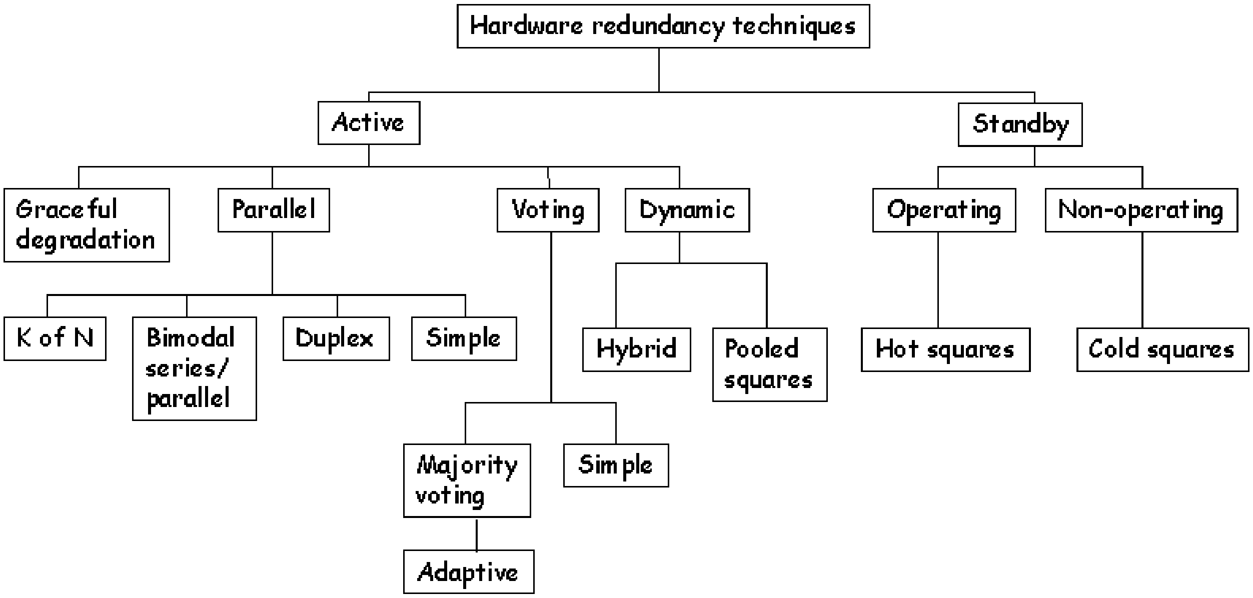
- These can outweigh safety benefits.

- Other techniques available:

- improved maintenance;
- better quality materials;

- Sometimes no choice (Satellites).

Hardware Redundancy Techniques



Active Redundancy

- When component fails...

- Redundant components do not have:

- to detect component failure;
- to switch to redundant resource.

- Redundant units always operate.

- Automatically pick up load on failure.

Standby Redundancy

- Must detect failure.

- Must decide to replace component.

- Standby units can be operating.

- Stand-by units may be brought-up.

Triple Modular Redundancy (TMR)

- Possibly most widespread.

- In simple voting arrangement,

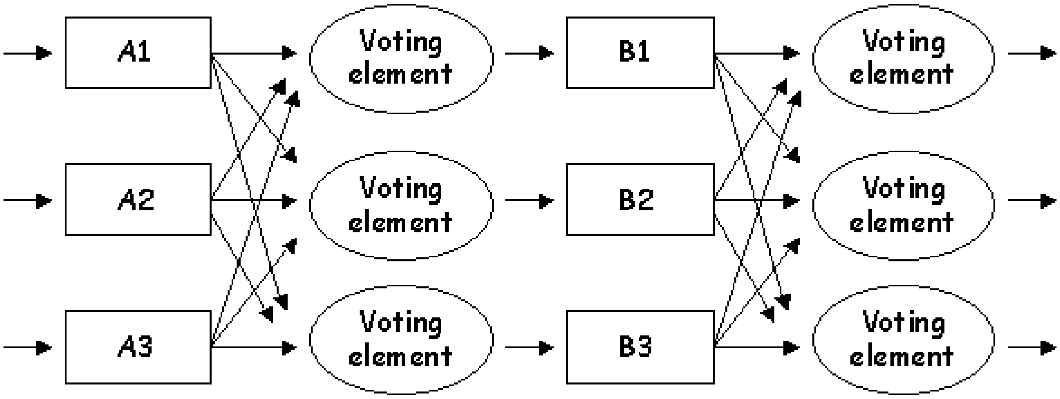
- voting element → common failure;
- so triplicate it as well.

- Multi-stage TMR architectures.

- More cost, more complexity...

Multilevel Triple Modular Redundancy (TMR)

- No protection if 2 fail per level.



- No protection from common failure

- eg if hard/software is duplicated.

Fault Detection

- Functionality checks:

- routines to check hardware works.

- Signal Comparisons:

- compare signal in same units.

- Information Redundancy:

- parity checking, M out of N codes...

- Watchdog timers:

- reset if system times out.

- Bus monitoring:

- check processor is ‘alive’.

- Power monitoring:

- time to respond if power lost.

Space Shuttle General Purpose Computer (GPC) Case Study

“GPCs running together in the same GN&C (Guidance, Navigation and Control) OPS (Operational Sequence) are part of a redundant set performing identical tasks from the same inputs and producing identical outputs. Therefore, any data bus assigned to a commanding GN&C GPC is heard by all members of the redundant set (except the instrumentation buses because each GPC has only one dedicated bus connected to it). These transmissions include all CRT inputs and mass memory transactions, as well as flight-critical data. Thus, if one or more GPCs in the redundant set fail, the remaining computers can continue operating in GN&C. Each GPC performs about 325,000 operations per second during critical phases. ”

http://spaceflight.nasa.gov/shuttle/reference/shutref/orbiter/avionics/dps/s

Space Shuttle General Purpose Computer (GPC) Case Study

"GPC status information among the primary avionics computers. If a GPC operating in a redundant set fails to meet two redundant multiplexer interface adapter receiver during two successive reads of response data and does not receive any data while the other members of the redundant set do not receive the data, they in turn will vote the GPC out of the set. A failed GPC is halted as soon as possible."

"GPC failure votes are annunciated in a number of ways. The GPC status matrix on panel O1 is a 5-by-5 matrix of lights. For example, if GPC 2 sends out a failure vote against GPC 3, the second white light in the third column is illuminated. The yellow diagonal lights from upper left to lower right are self-failure votes. Whenever a GPC receives two or more failure votes from other GPCs, it illuminates its own yellow light and resets any failure votes that it made against other GPCs (any white lights in its row are extinguished). Any time a yellow matrix light is illuminated, the GPC red caution and warning light on panel F7 is illuminated, in addition to master alarm illumination, and a GPC fault message is displayed on the CRT. "

http://spaceflight.nasa.gov/shuttle/reference/shutref/orbiter/avionics/dps/s

Space Shuttle General Purpose Computer (GPC) Case Study

"Each GPC power on , off switch is a guarded switch. Positioning a switch to on provides the computer with triply redundant normally, even if two main or essential buses are lost. "

"(There are) 5 identical general-purpose computers aboard the orbiter control space shuttle vehicle systems. Each GPC is composed of two separate units, a central processor unit and an input/output processor. All five GPCs are IBM AP -101 computers. Each CPU and IOP contains a memory area for storing software and data. These memory areas are collectively re ferred to as the GPC's main memory.

The IOP of each computer has 24 independent processors, each of which controls 24 data buses use d to transmit serial digital data between the GPCs and vehicle systems, and secondary channels between the telemetry system a nd units that collect instrumentation data. The 24 data buses are connected to each IOP by multiplexer interface adapt ers that receive, convert and validate the serial data in response to discrete signals calling for available data to be transmitted or received from vehicle hardware."

http://spaceflight.nasa.gov/shuttle/reference/shutref/orbiter/avionics/dps/s

Space Shuttle General Purpose Computer (GPC) Case Study

“A GPC on orbit can also be ”freeze-dried;” that is, it can be loaded with the software for a particular memory configuration and then moded to standby. It can then be moded to halt and powered off. Since the GPCs have non-volatile memory, the software is retained. Before an OPS transition to the loaded memory configuration, the freeze-dried GPC can be moded back to run and the appropriate OPS requested.

A failed GPC can be hardware-initiated, stand-alone-memory-dumped by switching the powered computer to terminate and halt and then selecting the number of the failed GPC on the GPC memory dump rotary switch on panel M042F in the crew”

http://spaceflight.nasa.gov/shuttle/reference/shutref/orbiter/avionics/dps/s

Space Shuttle General Purpose Computer (GPC) Case Study

“A simplex GPC is one in run and not a member of the redundant set, such as the BFS (Backup Flight System) GPC. Systems management and payload major functions are always in a simplex GPC.”

“Even though the four primary avionics software system GPCs control all GN&C functions during the critical phases of the mission, there is always a possibility that a generic failure could cause loss of vehicle control. Thus, the fifth GPC is loaded with different software created by a different company than the PASS developer. This different software is the backup flight system. To take over control of the vehicle, the BFS monitors the PASS GPCs to keep track of the current state of the vehicle. If required, the BFS can take over control of the vehicle upon the press of a button. The BFS also performs the systems management functions during ascent and entry because the PASS GPCs are operating in GN&C. BFS software is always loaded into GPC 5 before flight, but any of the five GPCs could be made the BFS GPC if necessary.”

http://spaceflight.nasa.gov/shuttle/reference/shutref/orbiter/avionics/dps/s

Conclusion

- The basics of hardware management.
- Fault models.
- Hardware redundancy.
- Space Shuttle GPC Case Study.

Hardware Implementation Issues

- COTS Microprocessors.
- Specialist Microprocessors.
- Programmable Logic Controllers
- Electromagnetic Compatability

COTS Microprocessors

- As we have seen:

- safety of software jeopardised
- if flaws in underlying hardware.

- Catch-22 problem:

- best tools for COTS processors;
- most experience with COTS;
- least assurance with COTS...

- Redundancy techniques help...

- but danger of common failures;
- vs cost of heterogeneity;

COTS Microprocessors

- Where do the faults arise?

1. fabrication failures;
2. microcode errors;
3. documentaiton errors.

- Can guard against 1:

- using same processing mask;
- tests then apply to all of batch;
- high cost (specialist approach).

- Cannot distinguish 2 from 3?

- Undocumented instructions...

COTS Microprocessors

“Modern microprocessor chips are getting very complex indeed. The current gate count can exceed 2.5 million. One must therefore expect that new versions of such chips will contain logical bugs. A common form of bug is in the microcode, but since the distinction between a microcode fault and another form of design bug is difficult to define, the distinction is not made here. We are *not* concerned with fabrication faults.”

“Attempts to report bugs openly have not been successful. A consequence of the above is that it is very difficult of users undertaking a critical application to protect themselves against a potential design bug. One approach that has been tried with one project is to use identical chips from the same mask so that rig and development testing will extrapolate to the final system. In some cases, the suppliers have provided information under a non-disclosure agreement, be this seems to be restricted to major projects. In contrast, quite a few software vendors have an open bug reporting scheme — and almost all provide a version number to the user. Hence it appears in this area, software is in ‘advance’ of hardware.”

Brian Wichmann, Microprocessor design faults (comp.risks)

COTS Microprocessors

"The key issues extracted are as follows:

- Early chips are unreliable:

  There have been some dramatic errors in very early releases of chips.

- Rarely used instructions are unreliable:

  One report sent to me reported that some instructions not generated by the 'C' compiler were completely wrong. Another report noted that special instructions for 64-bit integers did not work, and when this was reported, the supplier merely removed them from the documentation!

- Undocumented instructions are unreliable:

  Obviously, such instructions must be regarded with suspicion.

- Case handling is unreliable:

  A classic instance of this problem is an error which has been reported to me several times of the jump instructions on the 6502. When such an instruction straddled a page boundary, it did not work correctly. This issue potentially gives the user most cause for concern, since it may be very difficult to avoid the issue. For instance, with machine generated code form a compiler, the above problem with the 6502 would be impossible to avoid."

Brian Wichmann, Microprocessor design faults (comp.risks)

Specialist Microprocessors

- Commercial microprocessor flaws.

- What happens if illegal opcode?

- or result may be undefined?

- Motorola 6801 test instruction

- fetches infinite bytes from memory;
- good to test for faults on bus;
- but could be executed erroneously;
- see Storey or comp.risks for more.

Specialist Microprocessors - AAMP2

- Collins Avionics/Rockwell group.

- AAMP2

- 30+ in every Boeing 747-400.

- High criticality implies cost

- can you sell enough to cover input?

- What is money spent on?

- extra time spent on design?
- bench testing (see later);
- formal verification...

Specialist Microprocessors - AAMP5

"The AAMP5 verification was a project conducted to explore how formal techniques for specification and verification could be introduced into an industrial process. Sponsored by the Systems Validation Branch of NASA Langley and Collins Commercial Avionics, a division of Rockwell International, it was conducted by Collins and the Computer Science Research Lab at SRI International. The project consisted of specifying in the PVS language developed by SRI a portion of a Rockwell proprietary microprocessor, the AAMP5, at both the instruction set and register-transfer levels and using the PVS theorem prover to show the microcode correctly implemented the specified behavior for a representative subset of instructions. The formal verification was performed in parallel with the development of the AAMP5 and did not replace any production verification activities."

"This methodology was used to formally verify a core set of eleven AAMP5 instructions representative of several instruction classes. The core set did not include floating point instructions. Although the number of instructions verified is small, the methodology and the formal machinery developed are adequate to cover most of the remaining AAMP5 microcode. The success of this project has lead to a sequel in which the same methodology is being reused to verify another member of the AAMP family of processors"

Steve Miller and Manadayam Srivas, Formal Verification of AAMP5, Comp.risks

Specialist Microprocessors - AAMP5

“Another key result was the discovery of both actual and seeded errors. Two actual microcode errors were discovered during development of the formal specification, illustrating the value of simply creating a precise specification. Both errors were specific to the AAMP5 and corrected prior to first fabrication. Two additional errors seeded by Collins in the microcode were systematically uncovered by SRI while doing correctness proofs. One of these was an actual error that had been discovered by Collins during testing of an early prototype but left in the microcode provided to SRI. The other or simulation.”

Steve Miller and Manadayam Srivas, Formal Verification of AAMP5, Comp.risks

- Formal verification can be done,

- but still very expensive;
- need techniques and tools;
- reduce costs and increase subsets.

Specialist Microprocessors - Verifiable Integrated Processor for Enhanced Reliability (VIPER)

- This is an old story - but still very controversial.

- Royal Signals & Radar Establishment.

- LCF-LSM and Ella.

- Big claims about confidence levels:

- did MOD claim "fully proven"?
- proof from spec to production?

Specialist Microprocessors - Verifiable Integrated Processor for Enhanced Reliability (VIPER)

- Charter technologies market the chip.

- Sue MOD over “ungrounded” claims.

- Charter into liquidation as costs rise.

- Key lesson:

- general ignorance about proof;
- argument not absolute guarantee.

Specialist Microprocessors - 1750A, 1750B

- Introduced in 1979, Revised in 1982.
- Deactivated in 1996.
- Well documented/understood.
- But dont forget safety of language.
- Not just processor reliability.
- Started but never completed?
- 1750A remains a de facto standard.

Specialist Microprocessors - ERC32

- Reliable not safety-critical?

- Space and radiation tolerant.

- Ada development tools.

- integer unit (IU);
- floating-point unit (FPU);
- memory controller (MEC).

- Single-chip (TSC695E) June 2006.

Microprocessor

- MIL-STD1750 - expensive.

- Select processor for application.

- Storey cites widespread use;
- range of less critical areas.
- specifically for airbag applications;
- "general purpose in-system programmable microcontrollers".

Programmable Logic Controllers (PLCs)

- Self contained:

- power supply;
- interface circuitry;
- 1+ processors.

- Different from GPCs (eg Shuttle):

- replace electromechanical relays;
- perform simple logic functions.

- Designed for high MTBFs:

- kernels provide trusted functions;
- proprietary source for firmware.

Programmable Logic Controllers (PLCs)

- Widely used, well tested

- But hard/software proprietary.

- Certification by trusted bodies.

- However...

Programmable Logic Controllers (PLCs)

"Lin Zucconi" lin_zucconi@lccmail.ocf.llnl.gov
3 Mar 1993 16:50:50 U

People using Modicon 984 Series programmable controllers with Graysoft Programmable Logic Controller (PLC) software Version 3.21 are advised to contact Graysoft (414) 357-7500 to receive the latest version (3.50) of the software. A bug in Version 3.21 can corrupt a controller's logic and cause equipment to operate erratically. PLCs are frequently used in safety-related applications. Users often assume that if their "logic" is correct then they are ok and forget that the underlying logic is implemented with software which may not be correct.

Lin Zucconi zucconi@llnl.gov

Programmable Logic Controllers (PLCs)

- PC emulators to develop software:

- download to target PLC;
- volaile store is dangerous;
- wide use of EEPROMS.

- Fail safe PLC's:

- two or more independent CPU's;
- voting forms of redundancy;
- if conflict close down in safe state

- Several graphical design techniques:

ladder & function block diagrams...

- See Storey for more detail.

Electromagnetic Compatability (EMC)

- Work in presence of interference.

- AND not create interference.

- Interference from external noise

- Interference from external source

2 radio signals on same frequency.

Electromagnetic Compatability (EMC)

- Difficult to predict.

- Intensity changes over time;

- eg with work patterns;

- Sources may also be mobile;

- or you may be mobile!

- Mobile telephones, car ignitions...

Electromagnetic Compatability (EMC)

- Protection.

- Screening:

- use conductive cage/enclosure.

- Check design of PCBs if possible:

- (power) loops form antennas;
- check use of ground planes.

- Check CMOS output capacitance;

- can buy chips (Philips 8051);
- help discriminate signal edges.

- Seek help from a specialist...

Hardware Implementation Issues

- COTS Microprocessors.
- Specialist Microprocessors.
- Programmable Logic Controllers
- Electromagnetic Compatability

Validation and Verification

- What are the differences?

- When, why and who?

- UK MOD DEF STAN 00-66

Definitions and Distinctions

- Verification:

- does it meet the requirements?

- Validation:

- are the requirements any good?

- Testing:

- process used to support V&V.

Definitions and Distinctions

B.5.3.6 Verification and Validation.

This sub-process evaluates the products of other Software Modification sub-processes to determine their compliance and consistency with both contractual and local standards and higher level products and requirements. Verification and validation consists of software testing, traceability, coverage analysis and confirmation that required changes to software documentation are made. Testing subdivides into unit testing, integration testing, regression testing, system testing and acceptance testing.

MOD DEF STAN 00-66
Integrated Logistic Support: Part 3, Guidance for Software Support.

Definitions and Distinctions

- Misuse of terms?

A. The certification/validation process should confirm that hazards identified by hazard analysis, (HA), failure mode effect analysis (FMEA), and other system analyses have been eliminated by design or devices, or special procedures. The certification/validation process should also confirm that residual hazards identified by operational analysis are addressed by warning, labeling safety instructions or other appropriate means.

OSHA Regulations (Standards - 29 CFR)
Nonmandatory guidelines for certification/validation of safety systems for presence sensing device initiation of mechanical power presses - 1910.217 App B

- More like verification?

Validation

- During design

- external review before commission;
- external review for certification.

- During implementation:

- additional constraints discovered;
- additional requirements emmerge.

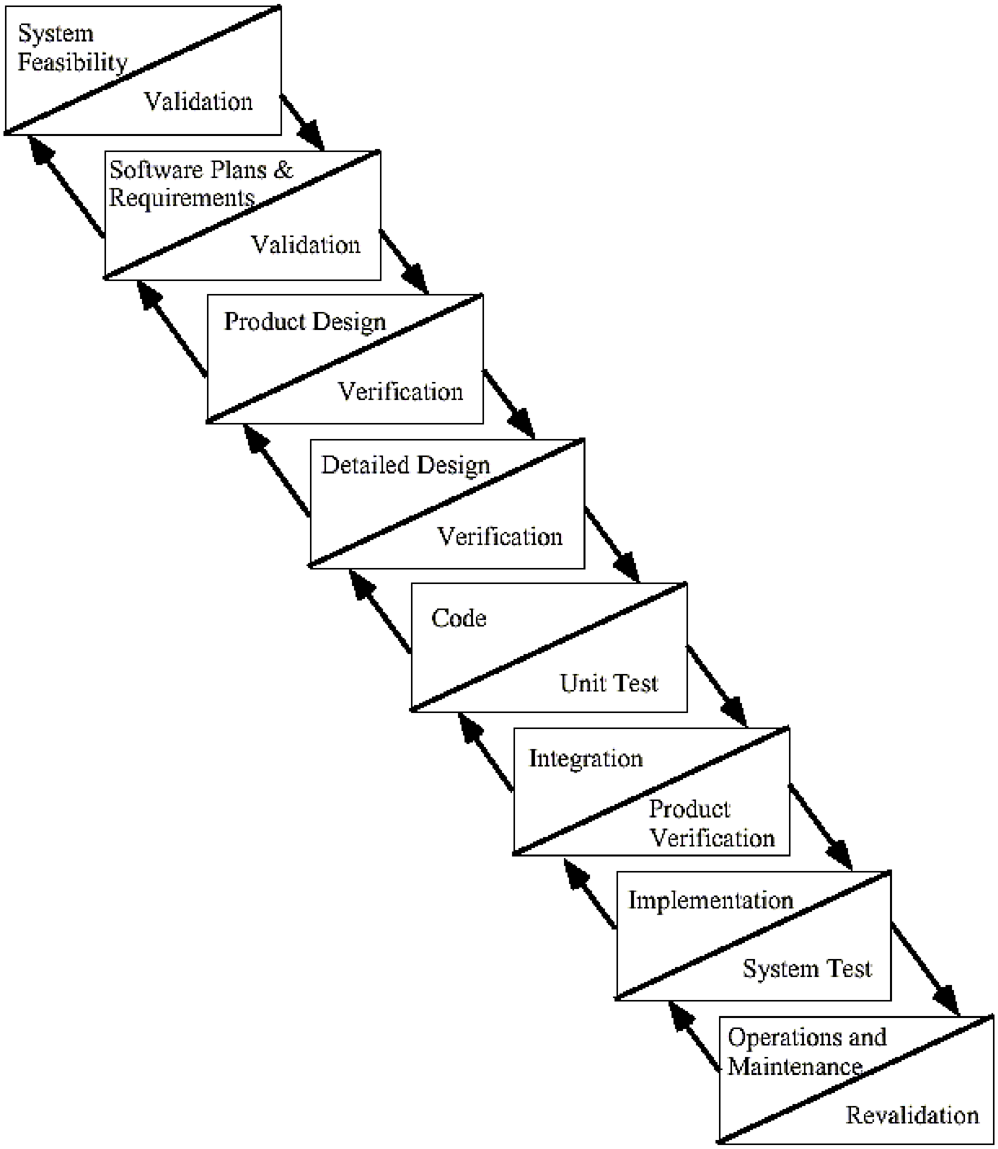
- During operation:

- were the assumptions valid?
- especially environmental factors.

- Validate:

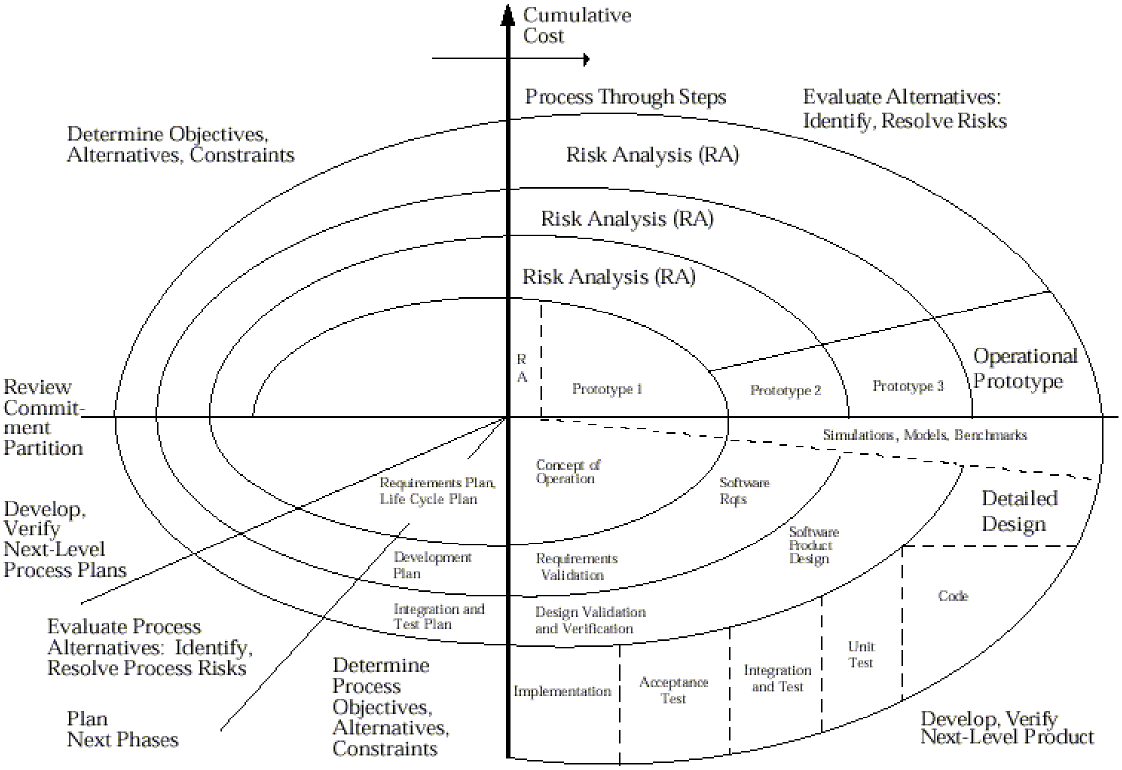
- PRA's; development processes etc.

Validation: Waterfall Model



- Validation at start and end.

Validation: Spiral Model



- Validation more continuous.

Validation: IEC 61508 (Draft)

The following should be considered in an overall safety validation plan:

- Details of when the validation should take place.
- Details of who should carry out the validation.
- Identification of the relevant modes of the system operation, including:
  - preparation for use, including setting up and adjustment
  - start up
  - teach
  - automatic
  - manual
  - semi-automatic
  - steady-state operation
  - resetting
  - shutdown
  - maintenance
  - reasonably foreseeable abnormal conditions

Validation: IEC 61508 (Draft)

- Identification of the safety-related systems and external risk reduction facilities that need to be validated for each mode of the system before commissioning commences.
- The technical strategy for the validation, for example, whether analytical methods or statistical tests are to be used.
- The measures, techniques and procedures that shall be used to confirm that each safety function conforms with the overall safety requirements documents and the safety integrity requirements.
- The specific reference to the overall safety requirements documents.
- The required environment in which the validation activities are to take place.
- The pass/fail criteria.
- The policies and procedures for evaluating the results of the validation, particularly failures.

Validation: MOD DEF STAN 00-60

D.4.1.6 Validation.

At the earliest opportunity support resource requirements should be confirmed and measurements should be made of times for completion of all software operation and support tasks. Where such measurements are dependent upon the system state or operating conditions, averages should be determined over a range of conditions. If measurements are based on non-representative hardware or operating conditions, appropriate allowances should be made and representative measurements carried out as soon as possible. The frequency of some software support tasks will be dependent upon the frequency of software releases and the failure rate exhibited by the software.

MOD DEF STAN 00-66
Integrated Logistic Support: Part 3, Guidance for Software Support.

Validation: MOD DEF STAN 00-60

D.4.1.6 Validation (cont.)

Measurements of software failure rates and fault densities obtained during software and system testing might not be representative of those that will arise during system operation. However, such measurements may be used, with caution, in the validation of models and assumptions. For repeatable software engineering activities, such as compilation and regression testing, the time and resource requirements that arose during development should be recorded. Such information may be used to validate estimates for equivalent elements of the software modification process. For other software engineering activities, such as analysis, design and coding, the time and resource requirements that arose during development should be recorded. However, such information should only be used with some caution in the validation of estimates for equivalent elements of the software modification process. The preceding clauses might imply the need for a range of metrics

MOD DEF STAN 00-66
Integrated Logistic Support: Part 3, Guidance for Software Support.

Validation: Summary of Key Issues

- Who validates validator?

- External agents must be approved.

- Who validates validation?

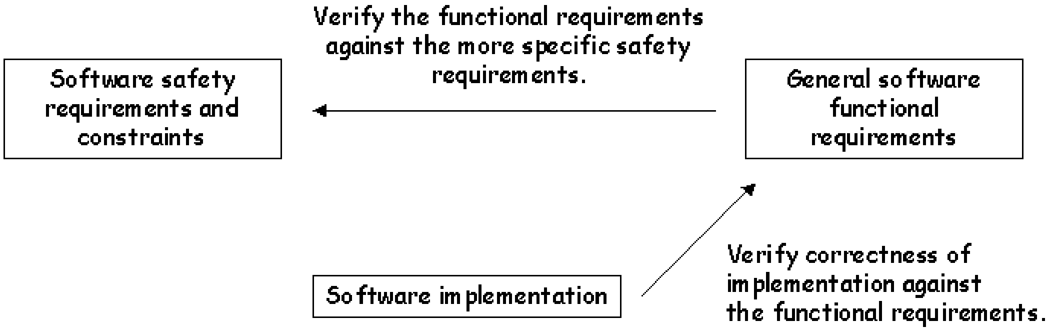
- Clarify links to certification.

- What happens if validation fails?

- Must have feedback mechanisms;
- Links to process improvement?

- NOT the same as verification!

Verification: Leveson's Strategies


Verify the functional requirements against the more specific safety requirements.
Software safety requirements and constraints
General software functional requirements
Software implementation
Verify correctness of implementation against the functional requirements.

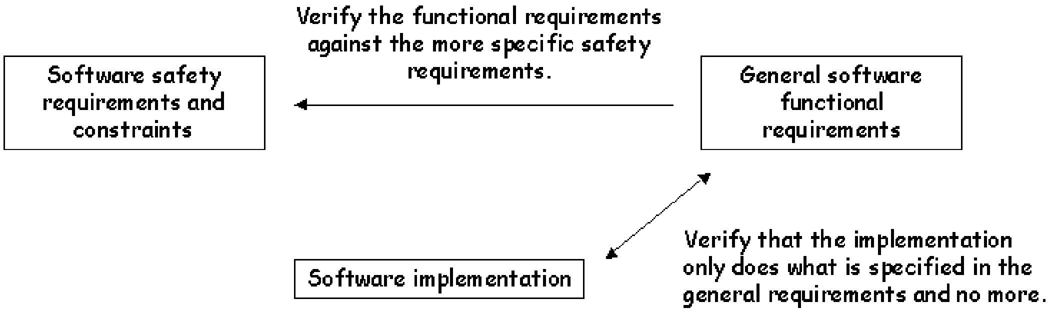
- Show that functional requirements

- are consistent with safety criteria ?

- Implementation may include hazards

not in safety/functional requirements.

Verification: Leveson's Strategies


Verify the functional requirements against the more specific safety requirements.
Software safety requirements and constraints
General software functional requirements
Software implementation
Verify that the implementation only does what is specified in the general requirements and no more.

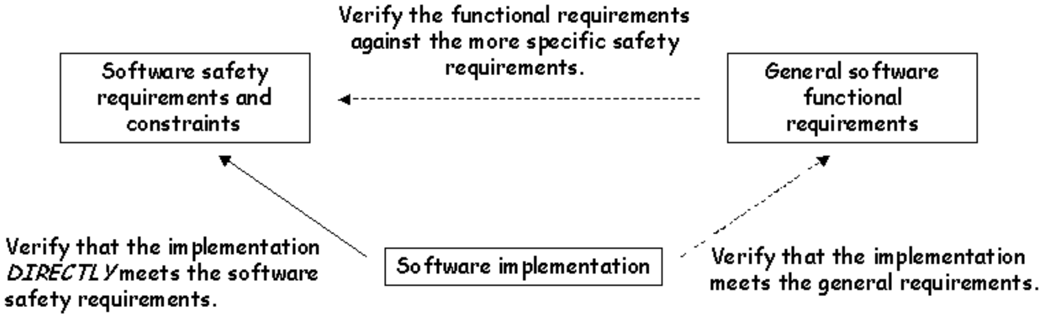
- Show that implementation is

- same as functional requirements?

- Too costly and time consuming

all safety behaviour in specification?

Verification: Leveson's Strategies


Verify the functional requirements against the more specific safety requirements.
Software safety requirements and constraints
General software functional requirements
Verify that the implementation *DIRECTLY* meets the software safety requirements.
Software implementation
Verify that the implementation meets the general requirements.

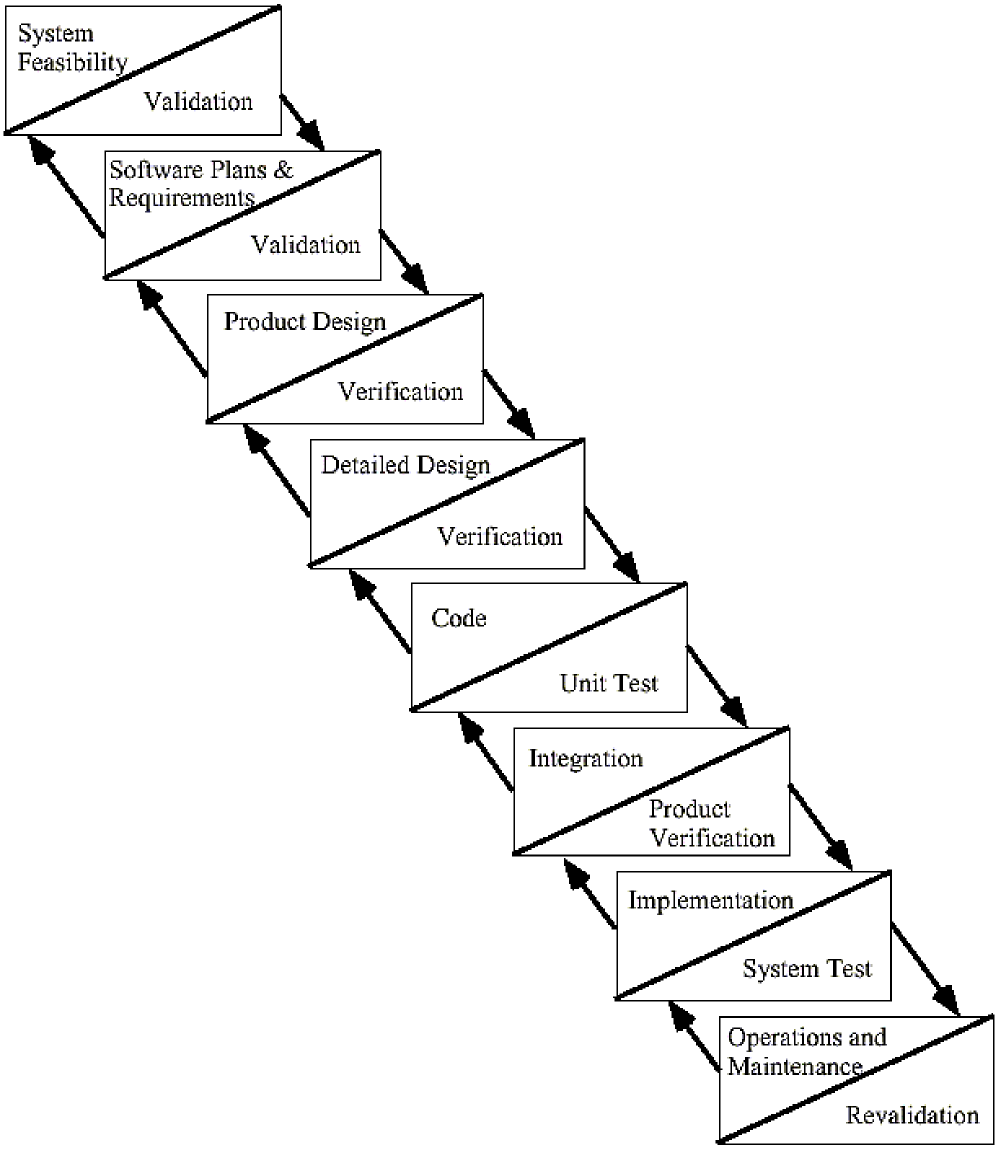
- Or show that the implementation

- meets the safety criteria.

- Fails if criteria are incomplete...

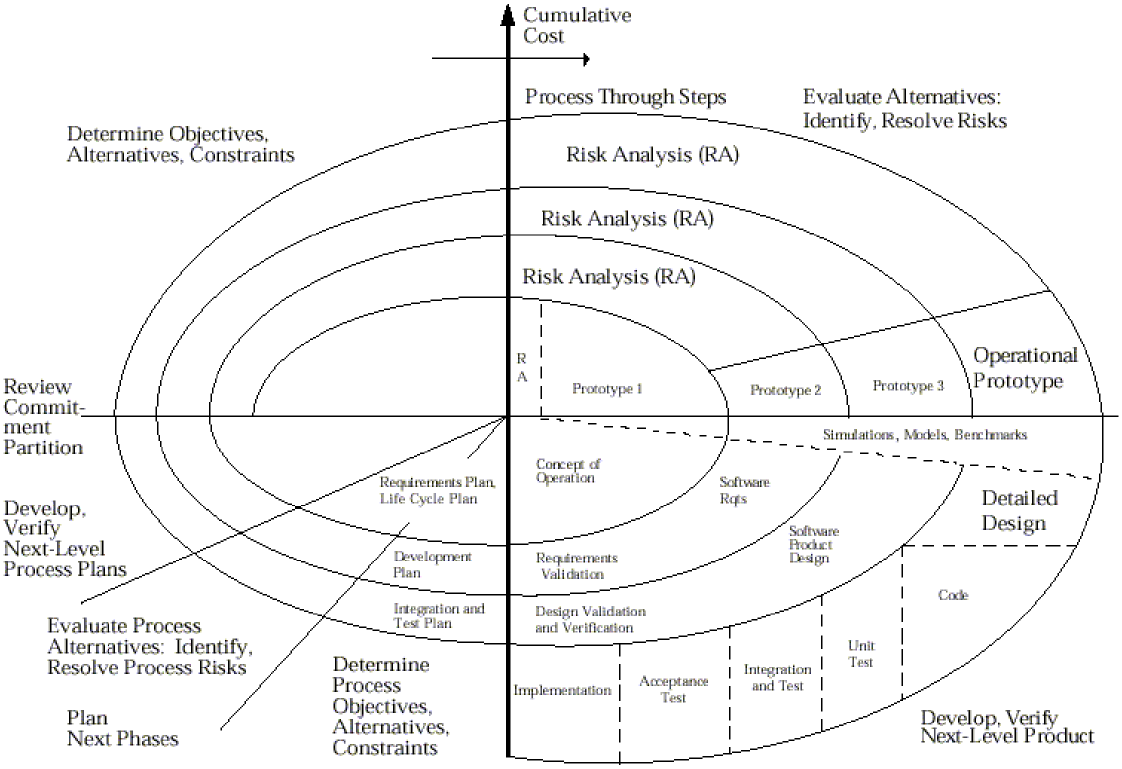
- but can find specification errors.

Verification: Lifecycle View



- At several stages in waterfall model.

Verification: Lifecycle View


Cumulative Cost
Process Through Steps
Evaluate Alternatives: Identify, Resolve Risks
Determine Objectives, Alternatives, Constraints
Risk Analysis (RA)
Risk Analysis (RA)
Risk Analysis (RA)
R A
Prototype 1
Prototype 2
Prototype 3
Operational Prototype
Review Commitment Partition
Simulations, Models, Benchmarks
Concept of Operation
Requirements Plan, Life Cycle Plan
Software Rqts
Detailed Design
Develop, Verify Next-Level Process Plans
Software Product Design
Development Plan
Requirements Validation
Code
Integration and Test Plan
Design Validation and Verification
Evaluate Process Alternatives: Identify, Resolve Process Risks
Unit Test
Integration and Test
Acceptance Test
Implementation
Determine Process Objectives, Alternatives, Constraints
Plan Next Phases
Develop, Verify Next-Level Product

Verification

- Verification as a catch-all?

- A recurrent cost, dont forget...

- verification post maintenance.

- Verification supported by:

- determinism (repeat tests);
- separate safety-critical functions;
- well defined processes;
- simplicity and decoupling.

Verification

D.5.1 Task 501 Supportability Test, Evaluation and Verification

D.5.1.1 Test and Evaluation Strategy.

Strategies for the evaluation of system supportability should include coverage of software operation and software support. Direct measurements and observations may be used to verify that all operation and support activities - that do not involve design change - may be completed using the resources that have been allocated. During the design and implementation stage measurements may be conducted on similar systems, under representative conditions. As software modification activity is broadly similar to software development the same monitoring mechanism might be used both pre- and post-implementation. Such a mechanism is likely to be based on a metrics programme that provides information, inter alia, on the rate at which software changes are requested and on software productivity.

MOD DEF STAN 00-66
Integrated Logistic Support: Part 3, Guidance for Software Support.

Verification

D.5.1.3 Objectives and Criteria.

System test and evaluation programme objectives should include verification that all operation and support activities may be carried out successfully -within skill and time constraints - using the PSE and other resources that have been defined. The objectives, and associated criteria, should provide a basis for assuring that critical software support issues have been resolved and that requirements have been met within acceptable confidence levels. Any specific test resources, procedures or schedules necessary to fulfil these objectives should be included in the overall test programme. Programme objectives may include the collection of data to verify assumptions, models or estimates of software engineering productivity and change traffic.

MOD DEF STAN 00-66
Integrated Logistic Support: Part 3, Guidance for Software Support.

Verification

D.5.1.4 Updates and Corrective Actions.

Evaluation results should be analyzed and corrective actions determined as required. Shortfalls might arise from:

- Inadequate resource provision for operation and support tasks.
- Durations of tasks exceeding allowances.
- Software engineering productivity not matching expectations.
- Frequencies of tasks exceeding allowances.
- Software change traffic exceeding allowances.

Corrective actions may include: increases in the resources available; improvements in training; additions to the PSE or changes to the software, the support package or, ultimately, the system design. Although re-design of the system or its software might deliver long term benefits it would almost certainly lead to increased costs and programme slippage.

MOD DEF STAN 00-66
Integrated Logistic Support: Part 3, Guidance for Software Support.

Verification: Summary of Key Issues

- What can we affoard to verify?

- Every product of every process?

- MIL HDBK 338B...

- Or only a few key stages?

- If the latter, do we verify :

- specification by safety criteria?
- implementation by safety criteria?
- or both...

Verification: Summary of Key Issues

- Above all....

- Verification is about proof.

- Proof is simply an argument.

- Argument must be correct but

- not a mathematical ‘holy grail’...

Validation and Verification

- What are the differences?

- When, why and who?

- UK MOD DEF STAN 00-66

Testing

- The processes used during - validation and verification.

- White and black boxes.

- Static and Dynamic techniques

- Mode confusion case study.

Definitions and Distinctions

- Black box tests:

- tester has no access to information
- about the system implementation.

- Good for independence of tester.

- But not good for formative tests.

- Hard to test individual modules...

Definitions and Distinctions

- White box tests:

- tester can access information about
- the system implementation.

- Simplifies diagnosis of results.

- Can compromise independence?

- How much do they need to know?

Definitions and Distinctions

- Module testing:

- tests well-defined subset.

- Systems integration:

- tests collections of modules.

- Acceptance testing:

- system meets requirements?

- Results must be documented.

- Changes will be costly.

Dynamic Testing - Process Issues

- Functional testing:

- test cases examine functionality;
- see comments on verification.

- Structural testing:

- knowledge of design guides tests;
- interaction between modules...
- test every branch (coverage)?

- Random testing:

- choose from possible input space;
- or beyond the “possible”...

Definitions and Distinctions

- Dynamic testing:

- execution of system components;
- is environment being controlled?

- Static testing:

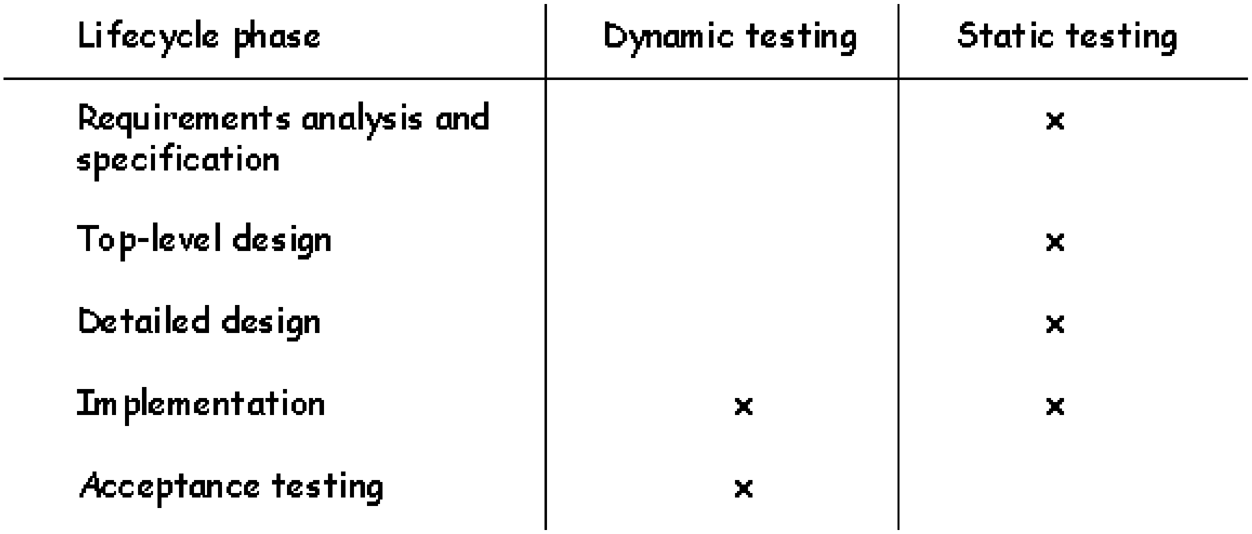
- investigation without operation;
- pencil and paper reviews etc.

- Most approaches use both.

- Guide the test selection by using:

- functional requirements:
- safety requirements;
- (see previous lecture).

Definitions and Distinctions



Lifecycle phase	Dynamic testing	Static testing
Requirements analysis and specification		x
Top-level design		x
Detailed design		x
Implementation	x	x
Acceptance testing	x	

Dynamic Testing

- Where do you begin?

- Look at the original hazard analysis;

- demonstrate hazard elimination?
- demonstrate hazard reduction?
- demonstrate hazard control?

- Must focus both on:

- expected and rare conditions.

- PRA can help - but for software?

Dynamic Testing - Leveson's Process Issues

- All of this will cost time and money.

1. Review test plans.

2. Recommend tests based on the hazard analyses, safety standards and checklists, previous accident and incidents, operator task analyses etc.

3. Specify the conditions under which the test will be conducted.

4. Review the test results for any safety-related problems that were missed in the analysis or in any other testing.

5. Ensure that the testing feedback is integrated into the safety reviews and analyses that will be used in design modifications.

- Must be planned, must be budgeted.

Dynamic Testing Techniques

- Partitioning:

- identify groups of input values;
- do they map to similar outputs?

- Boundary analysis:

- extremes of valid/invalid input.

- Probabilistic Testing:

- examine reliability of system.

- (State) Transition tests:

- trace states, transitions and events.

Dynamic Testing Techniques

- Simulation:

- assess impact on EUC (IEC61508).

- Error seeding:

- put error into implementation;
- see is test discover it (dangerous).

- Performance monitoring:

- check real-time, memory limits.

- Stress tests: - abnormally high workloads?

Dynamic Testing: Software Issues

- Boundary conditions.
- Incorrent and unexpected inputs sequences.
- Altered timings - delays and over-loading.
- Environmental stress - faults and failures.
- Critical functions and variables.
- Firewalls, safety kernels and other safety features.
- Usual suspects...automated tests?

Limitations of Dynamic Testing

- Cannot test all software paths.

- Cannot even text all hardware faults.

- Not easy to test in final environment.

- User interfaces very problematic:

- effects of fatigue/longitudinal use?
- see section on human factors.

- Systems CHANGE the environment!

- How can we test for rare events?

- may have to wait 10^9 years?

Static Testing

- Dont test the system itself.

- Test an abstraction of the system

- Perform checks on requirements?

- Perform checks on static code.

- Scope depends on representation...

Static Testing Techniques

- Walkthroughs:

- peer review by other engineers.

- Fagan inspections:

- review of design documents.

- Symbolic execution:

- use term-rewriting on code;
- does code match specification?

- Metrics:

- lots (eg cyclomatic complexity);
- most very debatable...

Static Testing Techniques

- Sneak Circuit Analysis:

- find weak patterns in topologies;
- for hardware not software.

- Software animation:

- trace behaviour of software model;
- Petri Net animation tools.

- Performance/scheduling theory:

- even if CPU scheduling is static;
- model other resource allocations.

- Formal methods:

- considerable argument even now;
- compare 00-60 with DO-178B...

Formal Methods: The Mode Confusion Case Study

- Recent, novel use formal analysis.

- To guide/direct other testing.

- The mode confusion problem...

- Several groups:

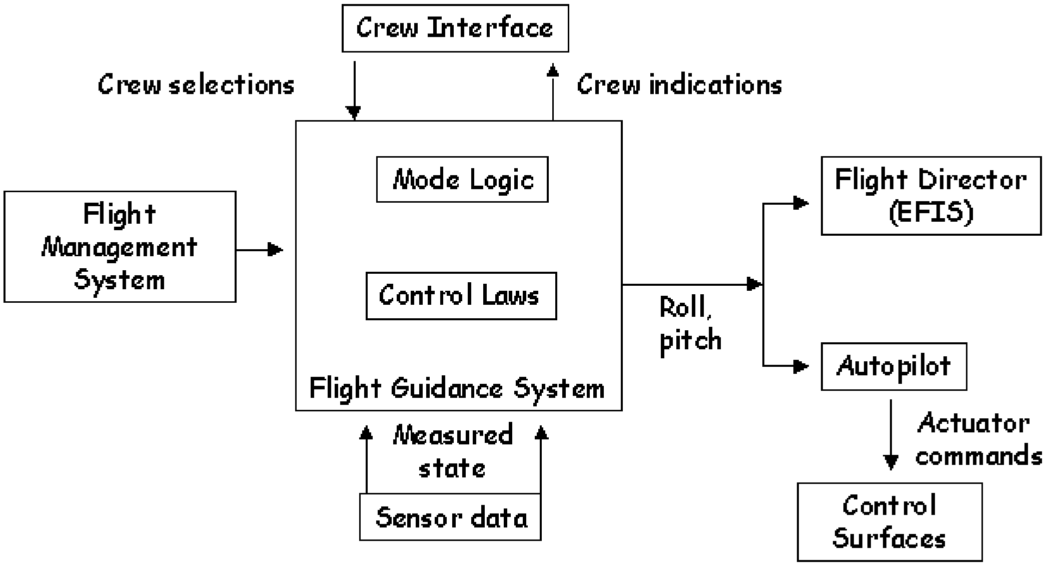
- Steve Miller et al(NASA Langley);
- Denis Javaux(Univ of Liege);
- Nancy Levelson (MIT);
- John Rusby(SRI)

Formal Methods: The Mode Confusion Case Study

The Flight Guidance System (FGS) compares the measured state of an aircraft (position, speed, and attitude) to the desired state and generates pitch and roll guidance commands to minimize the difference between the measured and desired state. When engaged, the Autopilot (AP) translates these commands into movement of the aircrafts control surfaces necessary to achieve the commanded changes about the lateral and vertical axes. An FGS can be further broken down into the mode logic and the flight control laws. The mode logic accepts commands from the flight crew, the Flight Management System (FMS), and information about the current state of the aircraft to determine which system modes are active. The active modes in turn determine which flight control laws are used to generate the pitch and roll guidance commands. The active lateral and vertical modes are displayed (an Instrumentation System (EFIS)).

Acknowledgement: R.W. Butler, S.P. Miller, J.N. Potts and V.A. Carreno, A Formal Methods Approach to the Analysis of Mode Confusion. In AIAA/IEEE Digital Avionics Systems Conference, October, 1998.

Formal Methods: The Mode Confusion Case Study



Acknowledgement: R.W. Butler, S.P. Miller, J.N. Potts and V.A. Carreno, A Formal Methods Approach to the Analysis of Mode Confusion. In AIAA/IEEE Digital Avionics Systems Conference, October, 1998.

Formal Methods: The Mode Confusion Case Study

Mode confusion can be traced to at least three fundamental sources:

- 1. Opacity (i.e., poor display of automation state),
- 2. Complexity (i.e., unnecessarily complex automation),
- 3. incorrect mental model (i.e., the flight crew misunderstands the behaviourr of the automation).

Traditional human factors has concentrated on (1), and made significant progress has been made. However, mitigation of mode confusion will require addressing problem sources (2) and (3) as well. Towards this end, our approach uses two complementary strategies based upon a formal model:

- Visualisation Create a clear, executable model of the automation that is easily understood by flight crew and use it to drive a flight deck mockup from the formal model
- Analysis Conduct mathematical analysis of the model.

Acknowledgement: R.W. Butler, S.P. Miller, J.N. Potts and V.A. Carreno, A Formal Methods Approach to the Analysis of Mode Confusion. In AIAA/IEEE Digital Avionics Systems Conference, October, 1998.

Formal Methods: The Mode Confusion Case Study

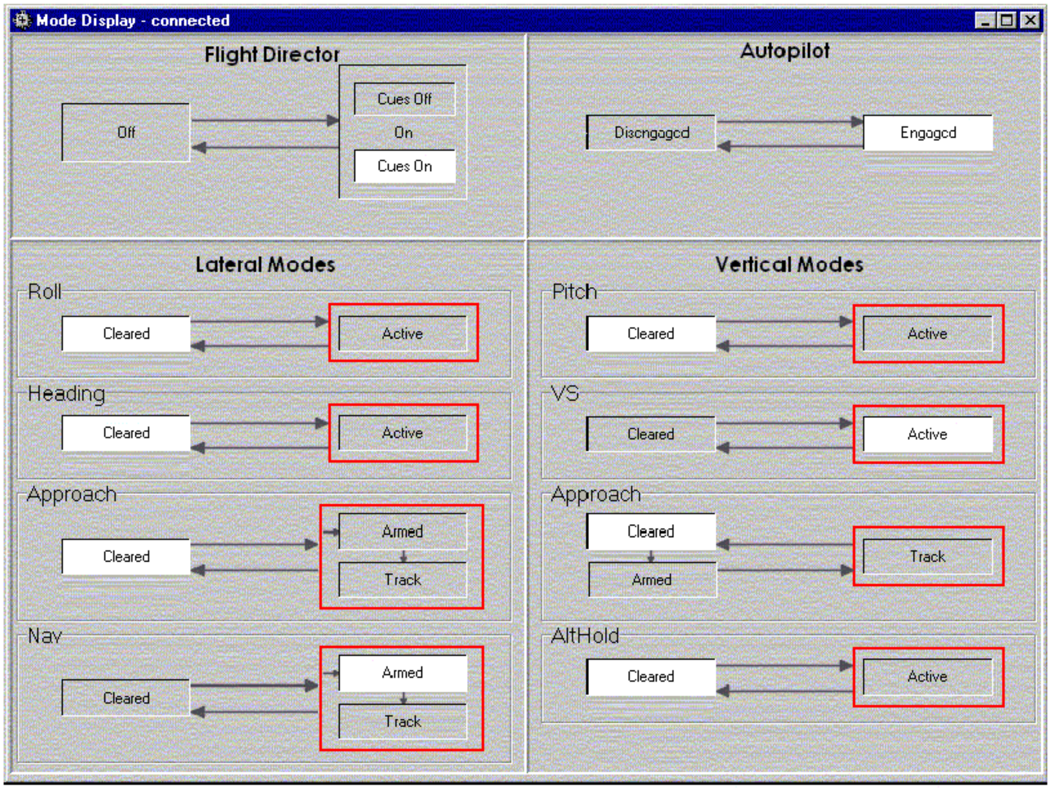
- Problems stemming from modes:

- input has different effect;
- uncommanded mode changes;
- different modes → behaviours;
- different intervention options;
- poor feedback.

- ObjectTime visualisation model...

- Represent finite state machines.

Formal Methods: The Mode Confusion Case Study



Formal Methods: The Mode Confusion Case Study

The state of the Flight Director (FD), Autopilot (AP), and each of the lateral and vertical modes are modeled as In Figure 3 (see previous slide), the FD is On with the guidance cues displayed; the AP is Engaged; lateral Roll, Heading, and Approach modes are Cleared; lat-eral NAV mode is Armed; vertical modes Pitch, Approach, and AltHold are Cleared; and the VS mode is Active. Active modes are those that actually control the aircraft when the AP is en-gaged. These are indicated by the heavy dark boxes around the Active, Track, and lateral Armed modes.

Acknowledgement: R.W. Butler, S.P. Miller, J.N. Potts and V.A. Carreno, A Formal Methods Approach to the Analysis of Mode Confusion. In AIAA/IEEE Digital Avionics Systems Conference, October, 1998.

Formal Methods: The Mode Confusion Case Study

- ObjectTime model:

- give pilots better mental model?
- drive simulation (dynamic tests?).

- Build more complete FGS model

- prove/test for mode problems.

- Discrete maths:

- theorem proving;
- or model checking?

Formal Methods: The Mode Confusion Case Study

The first problem is formally defining what constitutes an indirect mode change. Lets begin by defining it as a mode change that occurs when there has been no crew input:

```
Indirect_Mode_Change?(s,e):  bool =
NOT Crew_input?(e) AND Mode_Change?(s,e)
```

```
No_Indirect_Mode_Change:  LEMMA
Valid_State?(s) IMPLIES
NOT Indirect_Mode_Change?(s,e)
```

Acknowledgement: R.W. Butler, S.P. Miller, J.N. Potts and V.A. Carreno, A Formal Methods Approach to the Analysis of Mode Confusion. In AIAA/IEEE Digital Avionics Systems Conference, October, 1998.

Formal Methods: The Mode Confusion Case Study

We then seek to prove the false lemma above using GRIND, a brute force proof strategy that works well on lemmas that do not involve quantification. The resulting unproved sequents elaborate the conditions where indirect mode changes occur. For example,

```
{-1} Overspeed_Event?(e!1)
{-2} OFF?(mode(FD(s!1)))
{-3} s!1 WITH [FD := FD(s!1) WITH [mode := CUES],
LATERAL := LATERAL(s!1) WITH [ ROLL := (# mode := ACTIVE
#)]
VERTICAL := VERTICAL(s!1) WITH [ PITCH :=
(# mode := ACTIVE #)]] = NS
{-4} Valid_State(s!1)
|-------
{1} mode(PITCH(VERTICAL(s!1))) =
mode(PITCH(VERTICAL(NS)))
```

The situations where indirect mode changes occur are clear from the negatively labeled formulas in each sequent. We see that an indirect mode change occurs when the overspeed event occurs and the Flight Director is off. This event turns on the Flight Director and places the system into modes ROLL and PITCH.

Acknowledgement: R.W. Butler, S.P. Miller, J.N. Potts and V.A. Carreno, A Formal Methods Approach to the Analysis of Mode Confusion. In AIAA/IEEE Digital Avionics Systems Conference, October, 1998.

Formal Methods: The Mode Confusion Case Study

We define an ignored command as one in which there is a crew input and there is no mode change. We seek to prove that this never happens:

```
No_Ignored_Crew_Inputs:  LEMMA
Valid_State(s) AND Crew_Input?(e) IMPLIES
NOT Mode_Change?(s,e)
```

The result of the failed proof attempt is a set of sequents similar to the following:

```
{-1} VS_Pitch_Wheel_Changed?(e!1)
{-2} CUES?(mode(FD(s!1)))
{-3} TRACK?(mode(NAV(LATERAL(s!1))))
{-4} ACTIVE?(mode(VS(VERTICAL(s!1))))
|-------
{1} ACTIVE?(mode(ROLL(LATERAL(s!1))))
{2} ACTIVE?(mode(HDG(LATERAL(s!1))))
```

Acknowledgement: R.W. Butler, S.P. Miller, J.N. Potts and V.A. Carreno, A Formal Methods Approach to the Analysis of Mode Confusion. In AIAA/IEEE Digital Avionics Systems Conference, October, 1998.

Formal Methods: The Mode Confusion Case Study

The negatively labeled formulas in the sequent clearly elaborate the case where an input is ignored, i.e., when the VS/Pitch Wheel is changed and the Flight Director is displaying CUES and the active lateral mode is ROLL and the active vertical mode is PITCH. In this way, PVS is used to perform a state exploration to discover all conditions where the lemma is false, i.e., all situations in which a crew input is ignored.

Acknowledgement: R.W. Butler, S.P. Miller, J.N. Potts and V.A. Carreno, A Formal Methods Approach to the Analysis of Mode Confusion. In AIAA/IEEE Digital Avionics Systems Conference, October, 1998.

Formal Methods: The Mode Confusion Case Study

- Are these significant for user?

- Beware - atypical example of formal methods.

- Haven't mentioned refinement.

- Haven't mentioned implementation.

- Much more could be said...

- see courses on formal methods.

Conclusion

- Testing.
- The processes used in validation & verification.
- White and black boxes.
- Static and Dynamic techniques
- Mode confusion case study.

Individual Human Error

- Slips, Lapses and Mistakes.

- Rasmussen: Skill, Rules, Knowledge.

- Reason: Generic Error Modelling.

- Risk Homeostasis.

What is Error?

- Deviation from optimal performance?

- very few achieve the optimal.

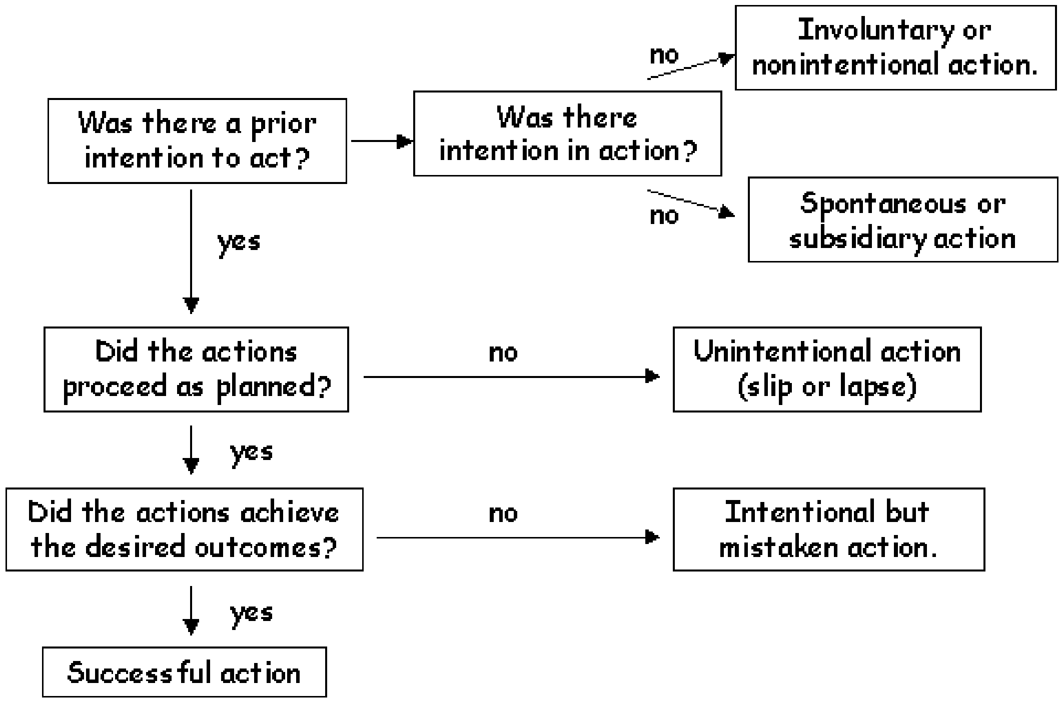
- Failure to achive desired outcome?

- desired outcome can be unsafe.

- Departure from intended plan?

- but environment may change plan...

What is Error?



Was there a prior intention to act?

yes

Was there intention in action?

no

Involuntary or nonintentional action.

no

Spontaneous or subsidiary action

Did the actions proceed as planned?

no

Unintentional action (slip or lapse)

yes

Did the actions achieve the desired outcomes?

no

Intentional but mistaken action.

yes

Successful action

Acknowledgement:J. Reason, Human Error, Cambridge University Press, 1990 (ISBN-0-521-31419-4).

Types of Errors...

- Slips:

- correct plan but incorrect action;
- more readily observed.

- Lapses:

- correct plan but incorrect action;
- failure of memory so more covert?

- Mistakes:

- incorrect plan;
- more complex, less understood.

- Human error modelling helps to:

- analyse/distinguish error types.

Rasmussen: Skill, Rules and Knowledge

- Skill based behaviour:

- sensory-motor performance;
- without conscious control;
- automated, high-integrated.

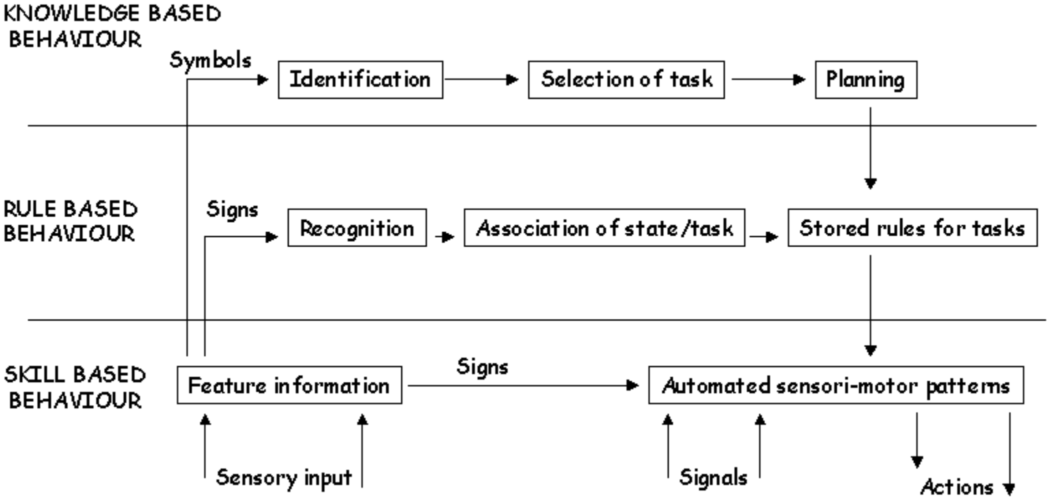
- Rule based behaviour:

- based on stored procedures;
- induced by experience or taught;
- problem solving/planning.

- Knowledge based behaviour:

- in unfamilliar situations;
- explicitly think up a goal;
- develop a plan by selection;
- try it and see if it works.

Rasmussen: Skill, Rules and Knowledge



Acknowledgement: J. Rasmussen, Skill, Rules, Knowledge: Signals, Signs and Symbols and Other Distinctions in Human Performance Models. IEEE Transactions on Systems, Man and Cybernetics (SMC-13)3:257-266, 1983.

Rasmussen: Skill, Rules and Knowledge

- Signals:

- sensory data from environment;
- continuous variables;
- cf Gibson's direct perception.

- Signs:

- indicate state of the environment;
- with conventions for action;
- activate stored pattern or action.

- Symbols:

- can be formally processed;
- related by convention to state.

Rasmussen: Skill, Rules and Knowledge

- Skill-based errors:

- variability of human performance.

- Rule-based errors:

- misclassification of situations;
- application of wrong rule;
- incorrect recall of correct rule.

- Knowledge-based errors:

- incomplete/incorrect knowledge;
- workload and external constraints...

Building on Rasmussen's Work

- How do we account for:

- slips and lapses in SKR?

- Can we distinguish:

- more detailed error forms?
- more diverse error forms?

- Before an error is detected:

- operation is, typically, skill based.

- After an error is detected:

- operation is rule/knowledge based.

- GEMS builds on these ideas...

GEMS: Monitoring Failures

- Normal monitoring:

- typical before error is spotted;
- preprogrammed behaviours plus;
- attentional checks on progress.

- Attentional checks:

- are actions according to plan?
- will plan still achieve outcome?

- Failure in these checks:

- often leads to a slip or lapse.

- Reason also identifies:

- Overattention failures.

GEMS: Problem Solving Failures

- Humans are pattern matchers:

- prefer to use (even wrong) rules;
- before effort of knowledge level.

- Local state information:

- indexes stored problem handling;
- schemata, frames, scripts etc.

- Misapplication of good rules:

- incorrect situation assessment;
- over-generalisation of rules.

- Application of bad rules:

- encoding deficiencies;
- action deficiencies.

GEMS: Knowledge-Based Failures

- Thematic vagabonding:

- superficial analysis/behaviour;
- flit from issue to issue.

- Encysting:

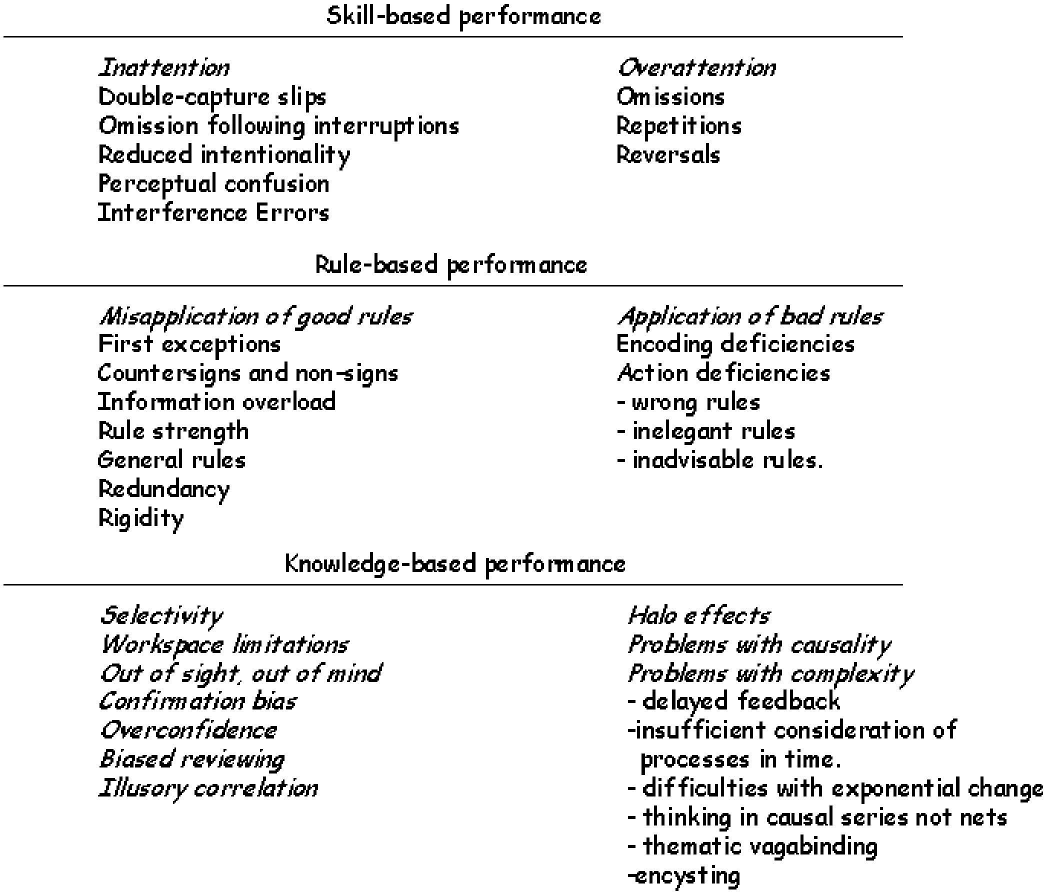
- myopic attention to small details;
- meta-level issues may be ignored.

- Reason:

- individual fails to recognise failure;
- does not face up to consequences.

- Berndt Brehmer & Dietrich Doerner.

GEMS: Failure Modes and the SKR Levels



Skill-based performance

Inattention	*Overattention*
Double-capture slips	Omissions
Omission following interruptions	Repetitions
Reduced intentionality	Reversals
Perceptual confusion	
Interference Errors	

Rule-based performance

Misapplication of good rules	*Application of bad rules*
First exceptions	Encoding deficiencies
Countersigns and non-signs	Action deficiencies
Information overload	- wrong rules
Rule strength	- inelegant rules
General rules	- inadvisable rules.
Redundancy	
Rigidity	

Knowledge-based performance

Selectivity	*Halo effects*
Workspace limitations	*Problems with causality*
Out of sight, out of mind	*Problems with complexity*
Confirmation bias	- delayed feedback
Overconfidence	-insufficient consideration of processes in time.
Biased reviewing	- difficulties with exponential change
Illusory correlation	- thinking in causal series not nets
	- thematic vagabinding
	-encysting

Acknowledgement:J. Reason, Human Error, Cambridge University Press, 1990 (ISBN-0-521-31419-4).

GEMS: Error Detection

- Dont try to eliminate errors:

- but focus on their detection.

- Self-monitoring:

- correction of postural deviations;
- correction of motor responses;
- detection of speech errors;
- detection of action slips;
- detection of problem solving error.

- How do we support these activities?

- standard checks procedures?
- error hypotheses or suspicion?
- use simulation based training?

GEMS: Error Detection

- Dont try to eliminate errors:

- but focus on their detection.

- Environmental error cueing:

- block users progress;
- help people discover error;
- “gag” or prevent input;
- allow input but warn them;
- ignore erroneous input;
- self correct;
- force user to explain..

- Importance of other operators

GEMS: Error Detection

- Cognitive barriers to error detection.

- Relevance bias:

- users cannot consider all evidence;
- "confirmation bias".

- Partial explanations:

- users accept differences between
- "theory about state" and evidence.

- Overlaps:

- even incorrect views will receive
- some confirmation from evidence.

- "Disguise by familliarity".

GEMS: Practical Application

- So how do we use GEMS?

- Try to design to avoid all error?

- Use it to guide employee selection?

- Or only use it post hoc:

- to explain incidents and accidents?

- No silver bullet, no panacea.

GEMS: Practical Application

- Eliminate error affoardances:

- increase visibility of task;
- show users constraints on action.

- Decision support systems:

- dont just present events;
- provide trend information;
- “what if” subjunctive displays;
- prostheses/mental crutches?

- Memory aids for maintenance:

- often overlooked;
- aviation task cards;
- must maintain maintenance data!

GEMS: Practical Application

- Improve training:

- procedures or heuristics?
- simulator training (contentious).

- Error management:

- avoid high-risk strategies;
- high probability/cost of failure.

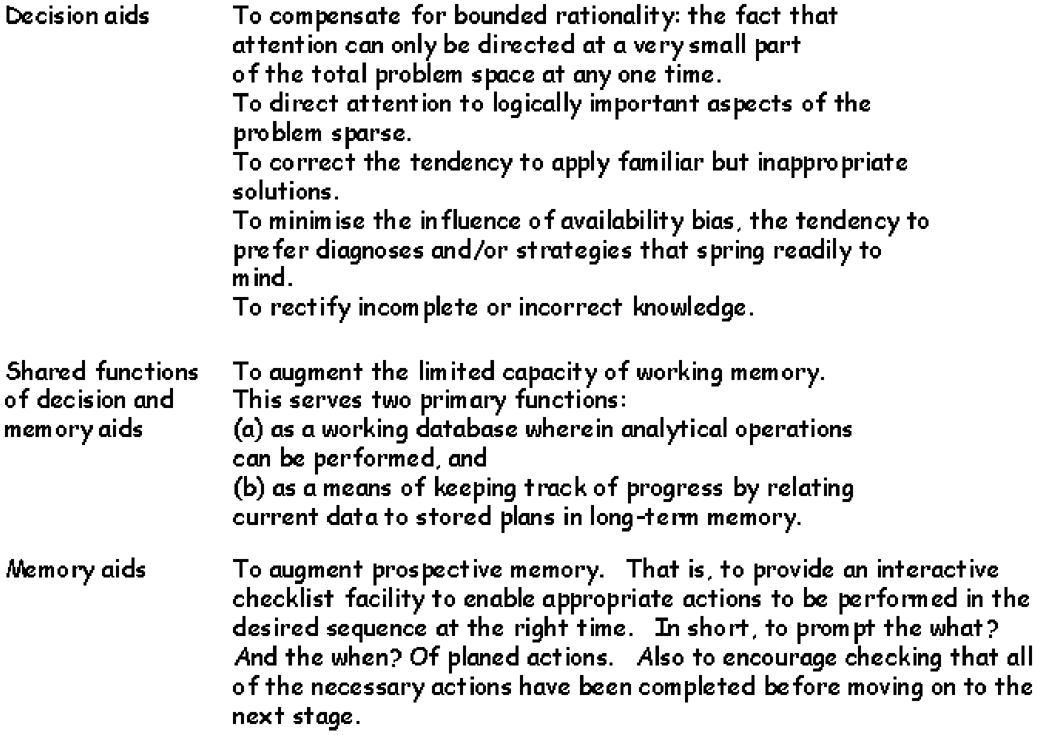
- Ecological interface design:

- Rasmussen and Vincente;
- 10 guidelines (learning issues).

- Self-awareness:

- when might I make an error?
- contentious...

GEMS: Practical Application



Decision aids	To compensate for bounded rationality: the fact that attention can only be directed at a very small part of the total problem space at any one time. To direct attention to logically important aspects of the problem sparse. To correct the tendency to apply familiar but inappropriate solutions. To minimise the influence of availability bias, the tendency to prefer diagnoses and/or strategies that spring readily to mind. To rectify incomplete or incorrect knowledge.
Shared functions of decision and memory aids	To augment the limited capacity of working memory. This serves two primary functions: (a) as a working database wherein analytical operations can be performed, and (b) as a means of keeping track of progress by relating current data to stored plans in long-term memory.
Memory aids	To augment prospective memory. That is, to provide an interactive checklist facility to enable appropriate actions to be performed in the desired sequence at the right time. In short, to prompt the what? And the when? Of planed actions. Also to encourage checking that all of the necessary actions have been completed before moving on to the next stage.

GEMS: Outstanding Issues

- Problem of intention:

- is an error a slip or lapse?
- is an error a mistake of intention?

- Given an observations of error:

- aftermath of accident/incident;
- guilt, insecurity, fear, anger.

- Can we expect valid answers?

- Can we make valid inferences?

GEMS: Outstanding Issues

- GEMS focusses on causation:

- built on Rasmussens SKR model;
- therefore, has explanatory power

- Hollnagel criticises it:

- difficult to apply in the field;
- do observations map to causes?

- Glasgow work has analysed:

- GEMS plus active/latent failures;

- Results equivocal, GEMS:

- provides excellent vocabulary;
- can be hard to perform mapping.

Risk Homeostasis Theory

- What happens if we introduce the

- decision aids Reason suggests?

"Each road user has a target (or accepted) level of risk which acts as a comparison with actual risk. Where a difference exists, one may move towards the other. Thus, when a safety improvement occurs, the target level of risk motivates behaviour to compensate - e.g., drive faster or with less attention. Risk homeostasis theory (RHT) has not beenconcerned with the cognitive or behavioural pathways by which homeostasis occurs, only with the consequences of adjustments in terms of accident loss."

Acknowledgement: T.W. Hoyes and A.I. Glendon, Risk Homeostasis: Issues for Further research, Safety Science, 16:19-33, (1993).

- Will users accept more safety?

- or trade safety for performance?

Risk Homeostasis Theory

- Very contentions.

- Bi-directionality?

- what if safety levels fall
- will users be more cautious?

- Does it affect all tasks?

- Does it affect work/leisure?

- How do we prove/disprove it?

- unlikely to find it in simulators.

Conclusions: Individual Human Error

- Slips, Lapses and Mistakes.

- Rasmussen: Skill, Rules, Knowledge.

- Reason: Generic Error Modelling.

- Risk Homeostasis.

Human Error and Group Work

- Workload.

- Situation Awareness.

- Crew Resource Management

Human Error and Group Work: Workload

- High workload:

- stretches users resources.

- Low workload:

- wastes users resources;
- can inhibit ability to respond.

- Cannot be "seen" directly;

- is inferred from behaviour.

- No widely accepted definition?

Human Error and Group Work: Workload

"Physical workload is a straightforward concept. It is easy to measure and define in terms of energy expenditure. Traditional human factors texts tell us how to measure human physical work in terms of kilocalories and oxygen consumption..."

Acknowledgement: B.H. Kantowitz and P.A. Casper, Human Workload in aviation. In E.L. Wiener and D.C. Nagel (eds.), Human Factors in Aviation, 157-187, Academic Press, London, 1988.

"The experience of workload is based on the amount of effort, both physical and psycholoigcal, expended in response to system demands (taskload) and also in accordance with the operator's internal standard of performance."

Acknowledgement: E.S. Stein and B. Rosenberg, The Measurement of Pilot Workload, Federal Aviation Authority, Report DOT/FAA/CT82-23, NTIS No. ADA124582, Atlantic City, 1983.

Human Error and Group Work: Workload

- Various approaches:

- Wickens on perceptual channels;
- Kantowitz on problem solving;
- Hart on overall experience.

- Holistic vs atomistic approaches:

- FAA (+ Seven) a gestalt concept;
- cannot measure in isolation;
- (many) experimentalists disagree.

- Single-user vs team approaches:

- workload is dynamic;
- shared/distributed between a team;
- many prvious studies ignore this.

Human Error and Group Work: Workload

- How do we measure workload?

- Subjective ratings?

- NASA TLX, task load index;
- consider individual differences.

- Secondary tasks?

- performance on additional task;
- obtrusive & difficult to generalise.

- Physiological measures?

- heart rate, skin temperature etc;
- lots of data but hard to interpret.

Human Error and Group Work: Workload

- How to reduce workload?

- Function allocation?

- static or dynamic allocation;
- to crew, systems or others (ATC?).

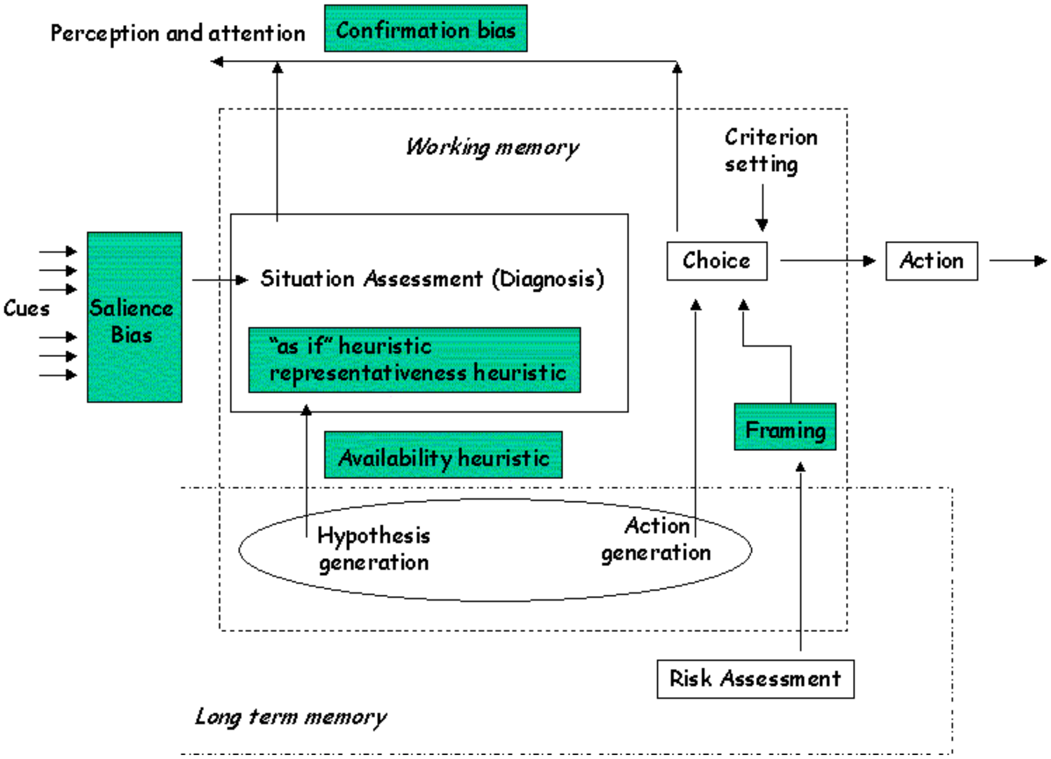
- Automation?

- but it can increase workload;
- or change nature (monitoring).

- Crew resource management?

- coordination, decision making etc;
- see later in this section...

Human Error and Group Work: Workload and Situation Awareness



Acknowledgement: C.D. Wickens and J.M. Flach, Information Processing. In E.L. Wiener and D.C. Nagel (eds.), Human Factors in Aviation, 111-156, Academic Press, London, 1988.

Human Error and Group Work: Situation Awareness

- Rather abstract definition.

"Situation awareness is the perception of the elements of the environment within a volume of time and spcae, the comprehension of their meaning, and the projection of their status in the near future"

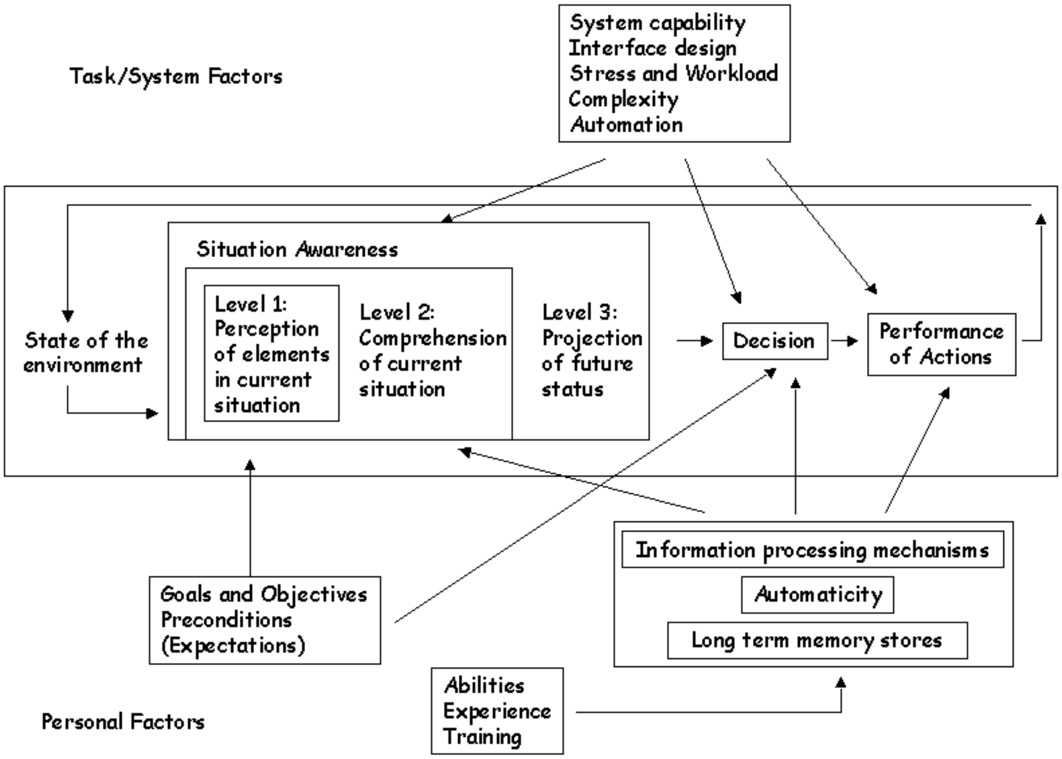
Acknowledgement: M. R. Endsley, Design and Evaluation for Situation Awareness Enhancement. In Proceedings of the Human Factors Society 32nd Annual Meeting, 97-101. Human Factors Society, Santa Monica, CA, 1988.

- Most obvious when it is lost.

- Difficult to explain behaviour:

- beware SA becoming a "catch all";
- just as "high workload" was.

Human Error and Group Work: Situation Awareness



Acknowledgement: M.R. Endsley, Towards a Theory of Situation Awareness, Human Factors, (37)1:32-64, 1995.

Human Error and Group Work: Situation Awareness

- Level 1: perception of environment

- how much can be attended to?
- clearly not everything...

- Level 2: Comprehension of situation

- synthesise the elements at level 1;
- significance determined by goals.

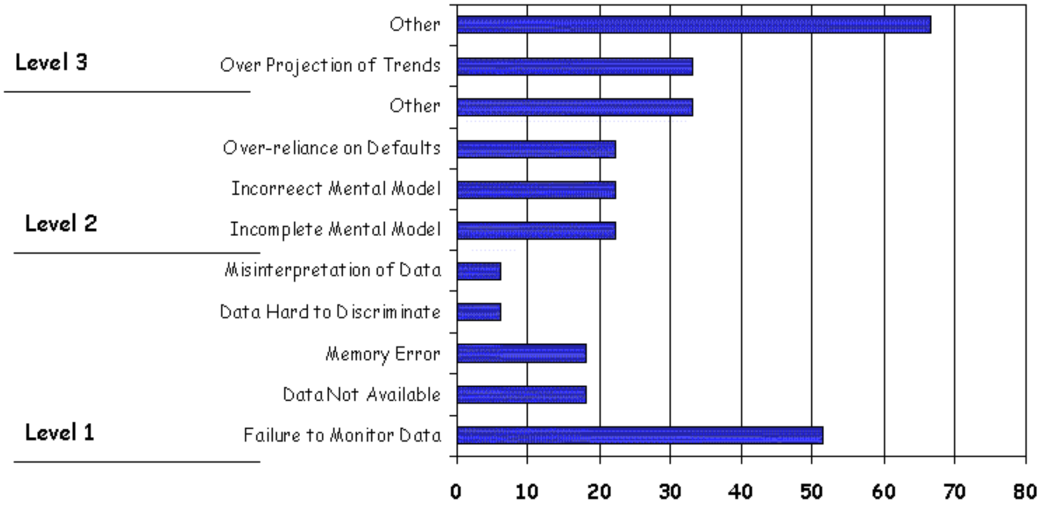
- Level 3: Projection of future.

- knowledge of status and dynamics;
- may only be possible in short term;
- enables strategy not just reaction.

- Novice perceives everything at L1;

- but fails at levels 2 and 3.

Human Error and Group Work: Situation Awareness



Acknowledgement: D.G. Jones and M.R. Endsley, Sources of Situation Awareness Errors in Aviation. Aviation, Space and Evironmental Medecine, 67(6):507-512, 1996.

Human Error and Group Work: Situation Awareness

- Hmm, subjective classification.

- 33 incidents with Air Traffic Control.

- NASA (ASRS) reporting system:

- how typical are reported events?

- I worry about group work:

- colleagues help you maintain SA?
- prompting, reminding, informing?

Human Error and Group Work: Situation Awareness

“Investigators were able to trace a series of errors that initiated with the flight crews acceptance of the controller’s offer to land on runway 19. The flightcrew expressed concern about possible delays and accepted an offer to expedite their approach into Cali... One of the AA965 pilots selected a direct course to the Romeo NDB believing it was the Rozo NDB, and upon executing the selection in the FMS permitted a turn of the airplane towards Romeo, without having verified that it was the correct selection and without having first obtained approval of the other pilot, contrary to AA procedures... The flightcrew had insufficient time to prepare for the approach to Runway 19.”

Acknowledgement: Controlled Flight Into Terrain, American Airlines Flight 965 Boeing 757-223, N651AA Near CALI, Colombia December 20, 1995 Aeronautica Civil of the Republic of Colombia

Human Error and Group Work: Crew Resource Management

"...Among the results were that captains of more effective crews (who made fewer operational or precedural errors) verbalised a greater number of plans than those of lower performing crews and requested and used more information in making their decisions. This raises interesting questions about whether situation awareness can be improved by teaching specific communication skills or even proceduralising certain communications that would otherwise remain in the realm of unregulated CRM (crew resource management behaviour)."

Acknowledgement: S. Dekker and J. Orasanu, Automation and Situation Awareness. In S. Dekker and E. Hollnagel (eds.), Coping with Computers in the Cockpit. 69-85, Ashgate, Aldershot, 1999. ISBN-0-7546-1147-7.

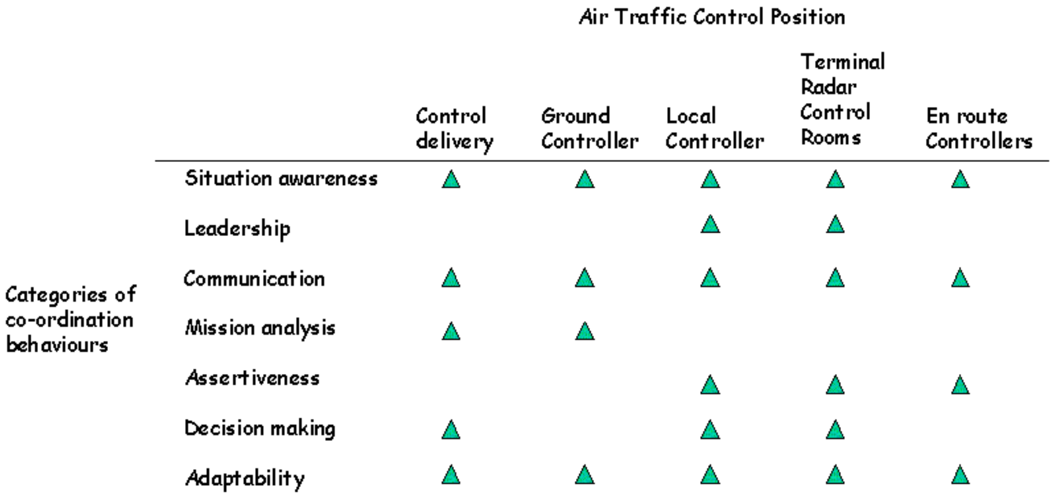
Human Error and Group Work: Crew Resource Management

- Cockpit Resource Management:

- crew coordination;
- decision making;
- situation awareness...

- More review activities inserted into standard operating procedures.

Human Error and Group Work: Crew Resource Management



		Air Traffic Control Position				
		Control delivery	Ground Controller	Local Controller	Terminal Radar Control Rooms	En route Controllers
Categories of co-ordination behaviours	Situation awareness	▲	▲	▲	▲	▲
	Leadership			▲	▲	
	Communication	▲	▲	▲	▲	▲
	Mission analysis	▲	▲			
	Assertiveness			▲	▲	▲
	Decision making	▲		▲	▲	
	Adaptability	▲	▲	▲	▲	▲

Acknowledgement: C.A. Bowers, E.L. Blickensderfer and B.B. Morgan, Air Traffic Control Team Coordination. In M.W. Smolensky and E.S. Stein, Human Factors in Air Traffic Control, 215-237, Academic Press, London, 1998.

Human Error and Group Work: Crew Resource Management

- Cockpit Resource Management:

- based on Foushee and Helmreich.

- Group performance determined by:

- process variables - communication;
- input variables - group size/skill.

- Goes against image of:

- pilot as “rugged individual”;
- showing “the right stuff”.

Human Error and Group Work: Crew Resource Management

- Key objectives:

- alter individual attitudes to groups;
- improve coordination within crew;
- increase team member effort;
- optimise team composition.

- Can we change group norms?

- Does it apply beyond aviation?

- with fewer rugged individuals?

Human Error and Group Work: Crew Resource Management

FAA Advisory Circular 120-51A 1993:

- Briefings are interactive and emphasize the importance of questions, critique, and the offering of information.
- Crew members speak up and state their information with appropriate persistence until there is some clear resolution.
- Critique is accepted objectively and non-defensively.
- The effects of stress and fatigue on performance are recognised.

NASA /UT LOS Checklist

- When conflicts arise, the crew remain focused on the problem or situation at hand. Crew members listen actively to ideas and opinions and admit mistakes when wrong, conflict issues are identified and resolved.
- Crew members verbalize and acknowledge entries to automated systems parameters.
- Cabin crew are included as part of team in briefings, as appropriate, and guidelines are established for coordination between flight deck and cabin.

Cited in: Quality Crew Resource Management, Human Factors Group Of The Royal Aeronautical Society.

Human Error and Group Work: Crew Resource Management

CRM TRAINING METHODS AND PROCESSES
Phase One - Awareness training - 2 days classroom
Objectives:

- Knowledge:
  - Relevance of CRM to flight safety and the efficient operation of an aircraft
  - How CRM reduces stress and improves working environment
  - Human information processing
  - Theory of human error
  - Physiological effects of stress and fatigue
  - Visual & aural limitations
  - Motivation
  - Cultural differences
  - CRM language and jargon.
  - The CRM development process
  - Roles such as leadership and followership
  - Systems approach to safety and man machine interface and SHEL model
  - Self awareness
  - Personality types
  - Evaluation of CRM

Human Error and Group Work: Crew Resource Management

Objectives (phase 1 cont.):

- Skills: Nil
- Attitudes:
  - Motivated to observe situations, others' and own behaviour in future.
  - Belief in the value of developing CRM skills.

Activities:

- Presentations
- Analysis of incidents and accidents by case study or video
- Discussion groups
- Self disclosure
- Personality profiling and processing
- Physiological experience exercises
- Self study

Quality Crew Resource Management, Human Factors Group Of The Royal Aeronautical Society.

Human Error and Group Work: Crew Resource Management

CRM TRAINING METHODS AND PROCESSES

Phase Two - Basic Skills training - 3/4 days classroom residential
Objectives:

- Knowledge:
  - Perceptions
  - How teams develop
  - Problem solving & decision making processes
  - Behaviours and their differences
  - Thought processes
  - Respect and individual rights
  - Development of attitudes
  - Communications toolkits
- Skills: See Appendix B
- Attitudes See Appendix B

Quality Crew Resource Management, Human Factors Group Of The Royal Aeronautical Society.

Human Error and Group Work: Crew Resource Management

CRM TRAINING METHODS AND PROCESSES

Activities:

- Presentations
- Experiential learning - (Recreating situations and experiences, using feelings to log in learning, experimenting in safe environments with cause and effect behaviour exercises)
- Role play
- Videod exercises
- Team exercises
- Giving & receiving positive and negative criticism
- Counselling
- Case studies
- Discussion groups
- Social and leisure activities

Quality Crew Resource Management, Human Factors Group Of The Royal Aeronautical Society.

Human Error and Group Work: Crew Resource Management

CRM TRAINING METHODS AND PROCESSES

Classroom, CPT or simulator

Objectives: Development of knowledge, skills and attitudes to required competency standards.

Activities: Practicing one or more skills on a regular basis under instruction in either the classroom, mock up/ CPT facility or full simulator LOFT sessions. Also considered valuable would be coaching by experienced crews during actual flying operations.

Quality Crew Resource Management, Human Factors Group Of The Royal Aeronautical Society.

Human Error and Group Work: Crew Resource Management

“Under normal conditions, aircraft flying is not a very interdependent task. In many cases, pilots are able to fly their aircraft successfully with relatively little coordination with other crew members, and communication between crew members is rquired primarily during nonroutine situations.”

Acknowledgement: C.A. Bowers, E.L. Blickensderfer and B.B. Morgan, Air Traffic Control Team Coordination. In M.W. Smolensky and E.S. Stein, Human Factors in Air Traffic Control, 215-237, Academic Press, London, 1998.

- Does it work in abnormal events?
- Additional requirements ignored?
- Can it hinder performance?

Human Error and Group Work

- Workload.

- Situation Awareness.

- Crew Resource Management