In Pursuit of Software Faults

Status and Challenges

Bojan Cukic

Lane Department of CSEE

West Virginia University

IFIP WG 10.4

January 2009

A view of a cynic

- **A Short History of Software Engineering**
  - Succession of Stampedes.
  - A succession of prefixes:
    - 1970’s: Structured.
    - 1980’s: Knowledge Based.
    - 1990’s: Object Oriented.
    - 2000’s: Web Based / Service Oriented.

Stampede Lifecycle

- Characteristics:
  - **Unrealistic Expectations.**
  - **Unsubstantiated claims, promises.**
  - **Euphoria, Unwarranted Optimism.**
  - **Excessive Hype.**
  - **Sudden death (Problem solved? Not a problem? Not solvable?)**
- And we are left with (a few):
  - **Unsolved problems.**
  - **Unfulfilled promises.**
  - **Unused solutions.**

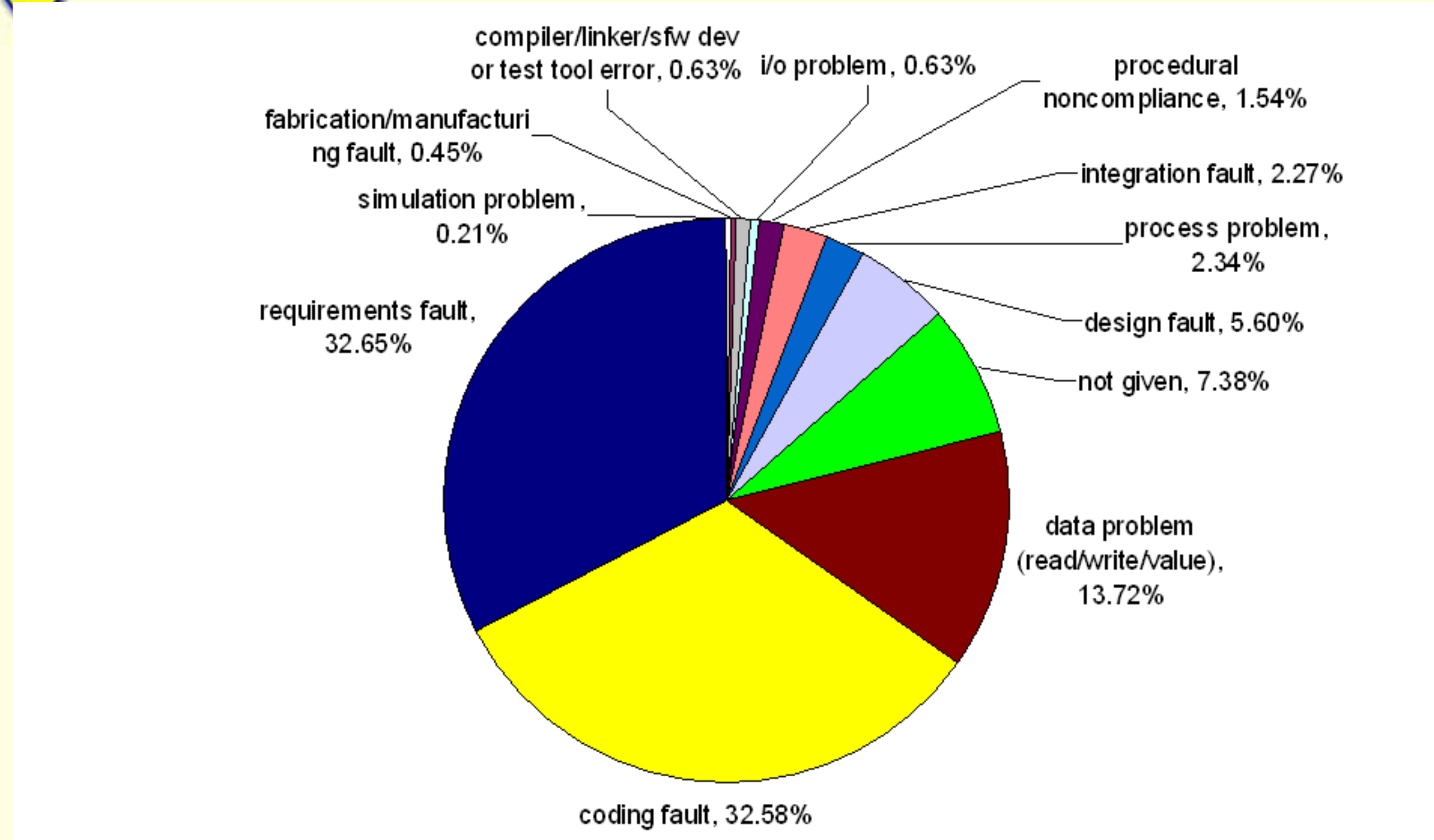
Outline

- **Fault distributions**
  - Large scale, long term projects.
- **Software verification and validation methods and their effectiveness**
  - Implications on failure detection and forecasting.
- **Streamlining V&V**
- **Summary**

Fault Analysis: Pilot study

- **Work by Katerina Goseva – Popstojanova and her students.**
- **A large NASA mission**
  - 21 Computer Software Configuration Items (CSCIs)
  - millions of lines of code
  - over 8,000 files
  - developed at two different locations
- **Analysis includes**
  - over 2,800 Software Change Requests (SCRs) entered due to non-conformance with requirements
    - collected through the software life cycle (i.e., development, testing and on-orbit)
    - over a period of almost 10 years

Sources of failures



Most common sources of failure for all 21 CSCIs grouped together

– Requirements faults (incorrect, changed & missing requirements): 33%

– Coding faults: 33%

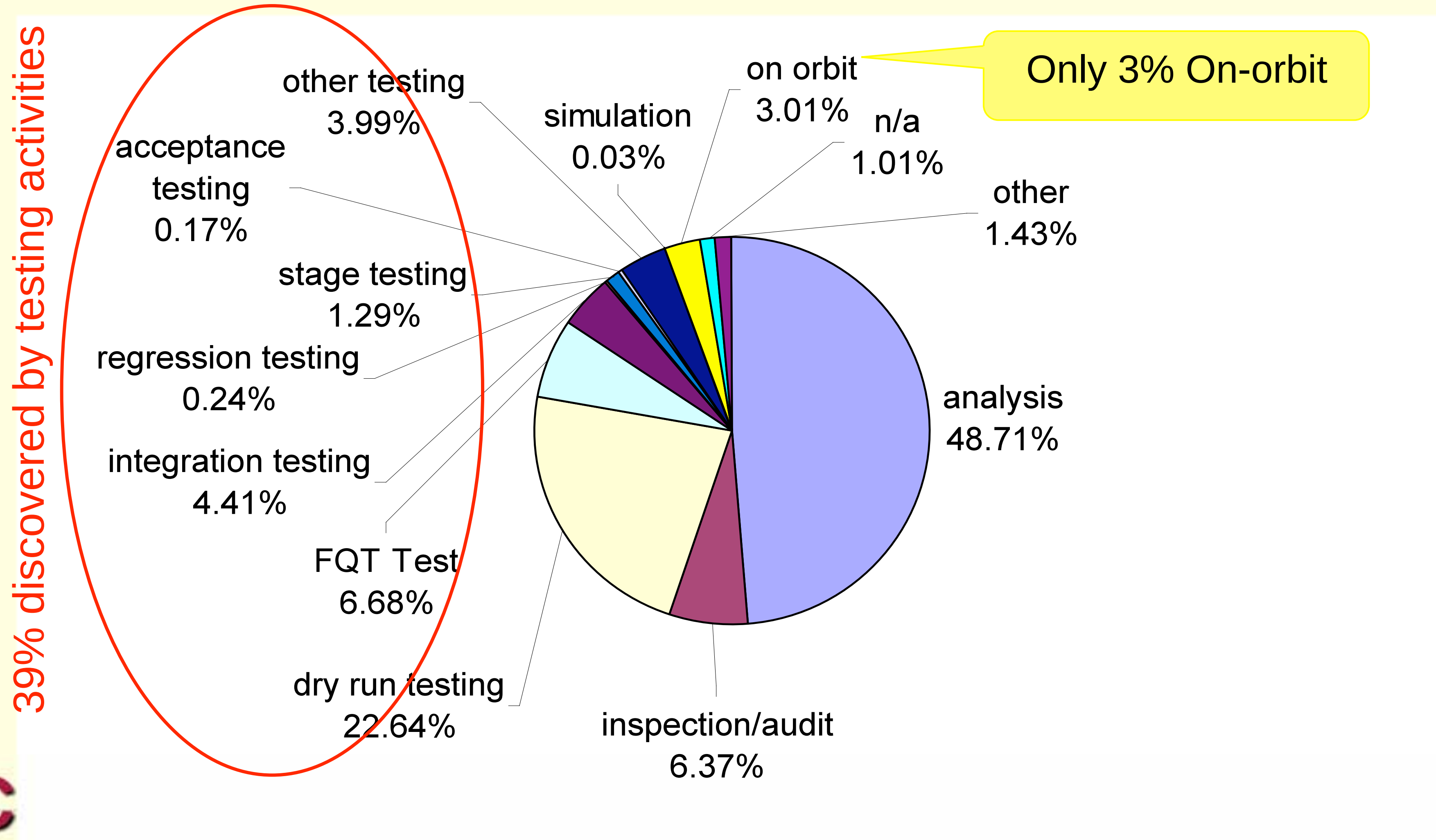
– Data problems: 14%

Source of failures: Early vs. Late life cycle activities

- **Distribution of sources of failures (i.e., fault types)**
  - Requirements & Design: 38.25%
    - Requirements faults: 32.65%
    - Design faults: 5.60%
  - Coding, Interface & Integration: 48.57%
    - Coding Faults: 32.58%
    - Data Problems: 13.72%
    - Integration Faults: 2.27%
  - 'Other' 5.80% and 'Not given' 7.38%
- **This distribution of faults across life cycle activities contradicts the common belief that majority of faults are introduced during early life cycle activities, i.e., requirements and design, which dates back to some of the earliest empirical studies [Boehm et. al 75, Endres 1975, Basili et. al 1984]**

Activity that discovered problem

The activity being performed when the problem was discovered is identified for 99% of the non-conformance SCRs



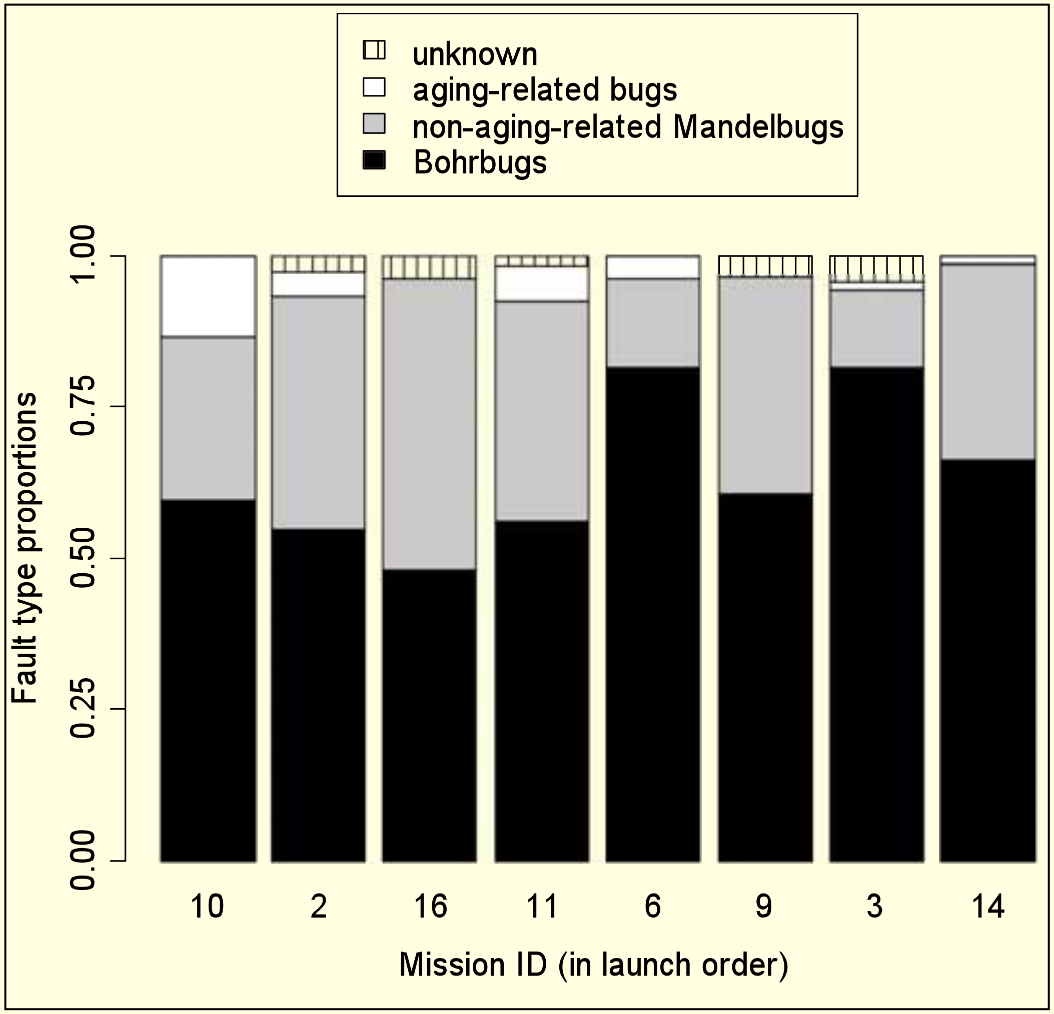
Different fault classification

- Classification scheme used by A Nikora:
  - ***Mandelbug*** *:*= A fault whose activation and/or error propagation are complex, caused by interactions of the software with its system environment (hardware, operating system, other applications), or by a time lag between the fault activation and the occurrence of a failure. Difficult to isolate, and/or the failures caused by it are not systematically reproducible.
  - ***Bohrbug*** *:*= Easily isolated and that manifests consistently under a well-defined set of conditions,
  - ***Aging-related bug*** := A fault that leads to the accumulation of internal error states, resulting in an increased failure rate and/or degraded performance. Sub-type of Mandelbug.

[Grottke05a] M. Grottke and K. S. Trivedi, “Software faults, software aging and software rejuvenation,” *Journal of the Reliability Engineering Association of Japan* 27(7):425–438, 2005.

[Grottke05b] M. Grottke and K. S. Trivedi, “A classification of software faults,” *Supplemental Proc. Sixteenth International Symposium on Software Reliability Engineering*, 2005, pp. 4.19-4.20.

Analysis Results



Fault type proportions for the eight projects with the largest number of unique faults

Fault Distributions: Summary

- **Faults are introduced in all phases of the life cycle.**
  – Detection methods should acknowledge this.
- **Effectiveness of detection methods varies.**
  – Although the numbers can be specific for this study, including diverse techniques seems necessary.
- **Fault distributions and detection methods seem to defeat the stampede lifecycle.**

Outline

- **Fault distributions**
  - Large scale, long term projects.
- **Software verification and validation**
  - Implications on failure detection and forecasting.
- **Streamlining V&V**
- **Summary**

Software Verification

- **Have we solved "hard problems" in software verification?**
  – The question appears to be goal dependent and domain dependent.
- **Goals are typically related to reliability requirements.**
  – Almost always, the decision to release / deploy made through engineering judgment, supported by partial qualitative and quantitative arguments.
- **Some domains, seemingly complex, very successful.**
  – Processor design, iris recognition…

Fault forecasting

- **Validation testing reveals (or does not reveal) failures.**
  – A sound approach to reliability assessment.
  – Stopping rules for system tests (Littlewood)
  – Key factors of failure-free fault forecasting:
    - Operational environment
    - Known operational profile
    - "Adequate" number of tests, typically impractical.

Ultra reliability in iris recognition

Fig. 1. *Even dark brown eyes reveal rich iris texture when illuminated in near-infrared light (700–900-nm band). The randomness of this texture and its complexity, spanning at least 3 octaves in usable scales of analysis, enables the discriminating power of the IrisCode.*

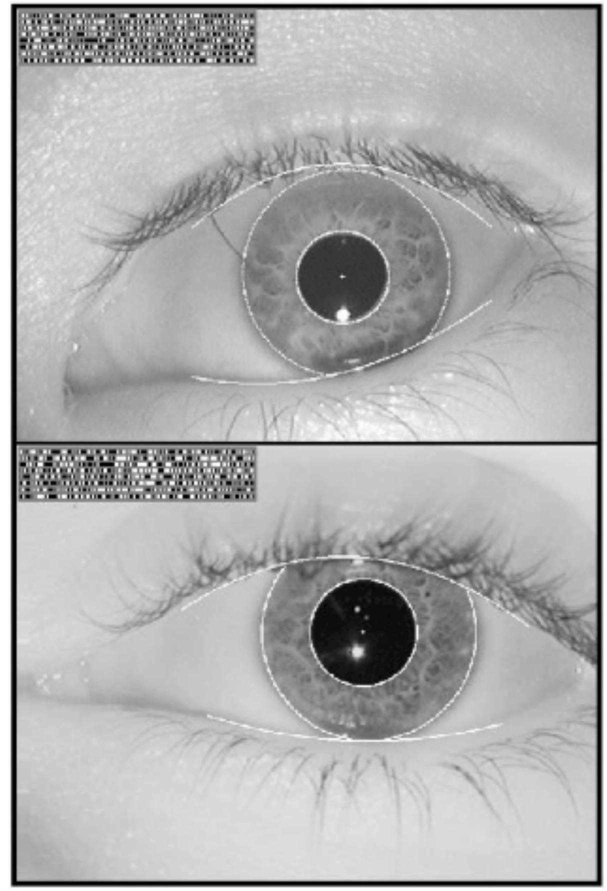


Fig. 2. *Isolation of the iris from the rest of the image. The white graphical overlays signify detected iris boundaries resulting from the segmentation process. For these images, circular inner and outer iris boundaries were assumed, but this is an overly restrictive assumption.*

Matching score is a Hamming Distance between two iris codes (2048 bits).

From Daugman, Proc IEEE, V94(11)

Demonstrating ultra reliability

- **632,500 unique irises → 200,027,808,750 comparisons (tests).**

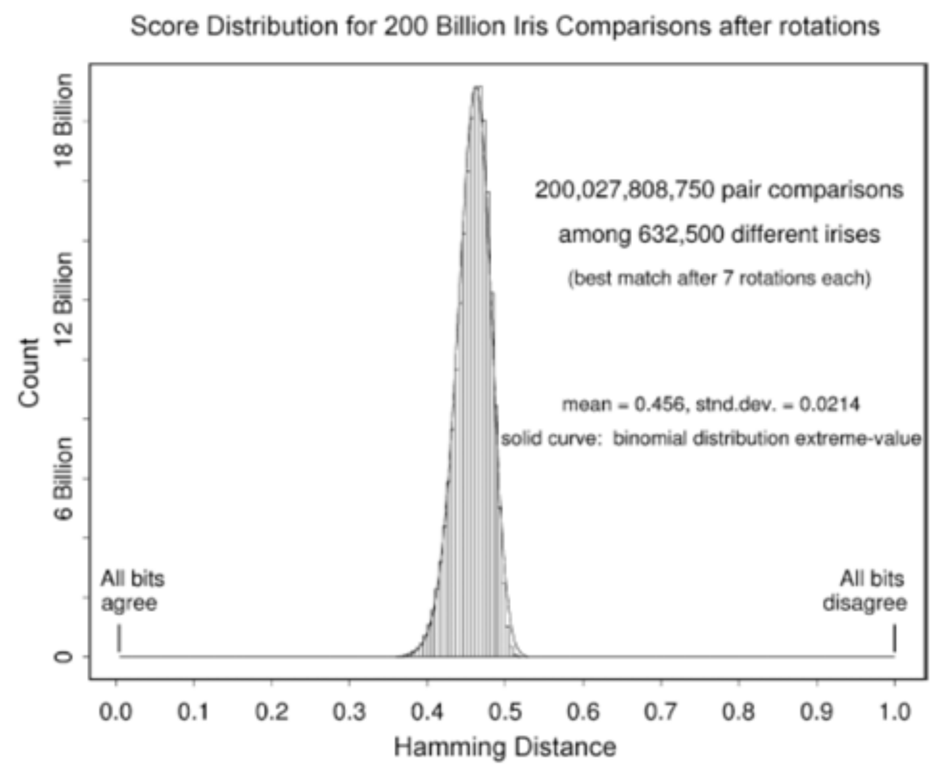


Fig. 6. *Distribution of dissimilarity scores between the same 200 billion iris pair comparisons as given in Fig. 5, but showing only the best match after seven relative rotations of each pair because of uncertainty about actual iris orientation. The solid curve is (11).*

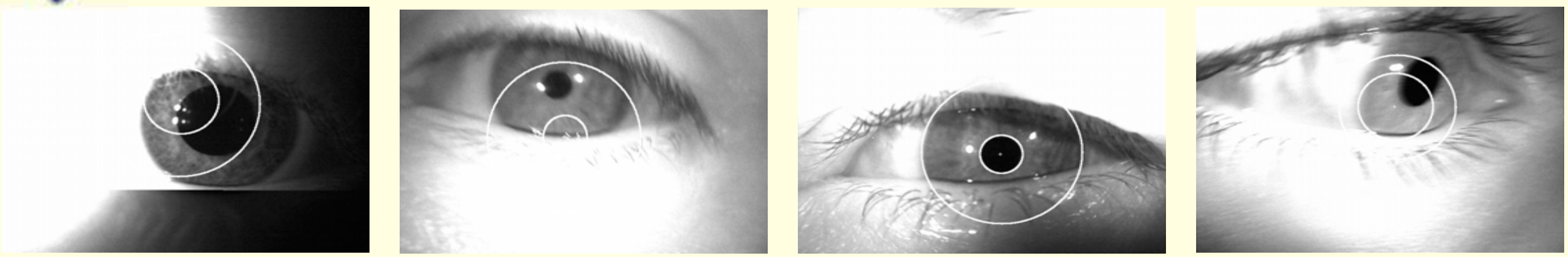
Probability of failure (false match)

Table 1 The False Match Rates, Either Observed in the Distribution of Scores or Predicted Theoretically From Eqns (7)-(10), Are Tabulated as a Function of Possible Decision Policy Match Criteria Imposed on the Normalized Hamming Distance Scores HD_{norm}

HD Criterion	*Observed False Match Rate*
0.220	0 (*theor: 1 in* 5×10^{15})
0.225	0 (*theor: 1 in* 1×10^{15})
0.230	0 (*theor: 1 in* 3×10^{14})
0.235	0 (*theor: 1 in* 9×10^{13})
0.240	0 (*theor: 1 in* 3×10^{13})
0.245	0 (*theor: 1 in* 8×10^{12})
0.250	0 (*theor: 1 in* 2×10^{12})
0.255	0 (*theor: 1 in* 7×10^{11})
0.262	1 in 200 billion
0.267	1 in 50 billion
0.272	1 in 13 billion
0.277	1 in 2.7 billion
0.282	1 in 284 million
0.287	1 in 96 million
0.292	1 in 40 million
0.297	1 in 18 million
0.302	1 in 8 million
0.307	1 in 4 million
0.312	1 in 2 million
0.317	1 in 1 million

- **For UK population (~60 M), all to all pairing (about 10^{15}), setting the threshold at 0.22 would imply ~1% false non match rate with “a low” expectation of a false match.**

However…



- Reliability conditioned on "ideality" of samples.
- Short distance (<1m), full subject cooperation, reacquisition (if needed), failure to acquire rate…
- → Such restrictions impractical in most applications.

- **When such assessments impractical, use diverse sources of evidence.**

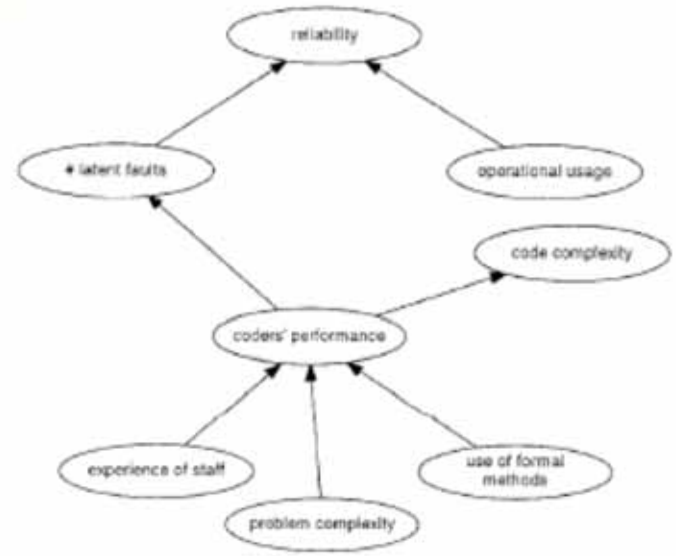
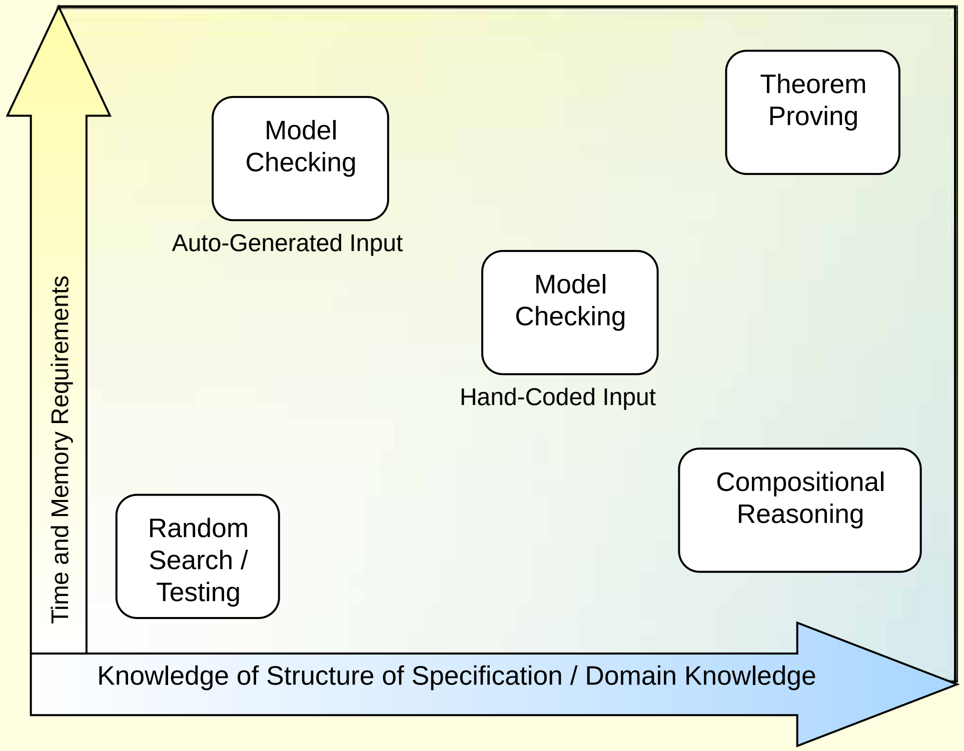


Fig. 1 Simple BBN for software reliability taking account of process and product information

Fenton et al,

Proc IEEE Soft., V145 (1)

Diverse verification methods



- **Complementary nature of verification methods.**
- **But combining them is not a trivial exercise.**

A Case Study

- **SCR specification of a personnel access control system (PACS)**
- **Five tools used to verify 15 assertions present in the specification**
  - Salsa Invariant Checker
  - Cadence SMV and NuSMV Symbolic Model Checker
  - SPIN Explicit-State Model Checker
  - Lurch Random Search Tool
- **323 fault-seeded specifications used in experiments**
  - 229 with one mutation, 94 with two mutations.
  - 90 found to be equivalent mutants.

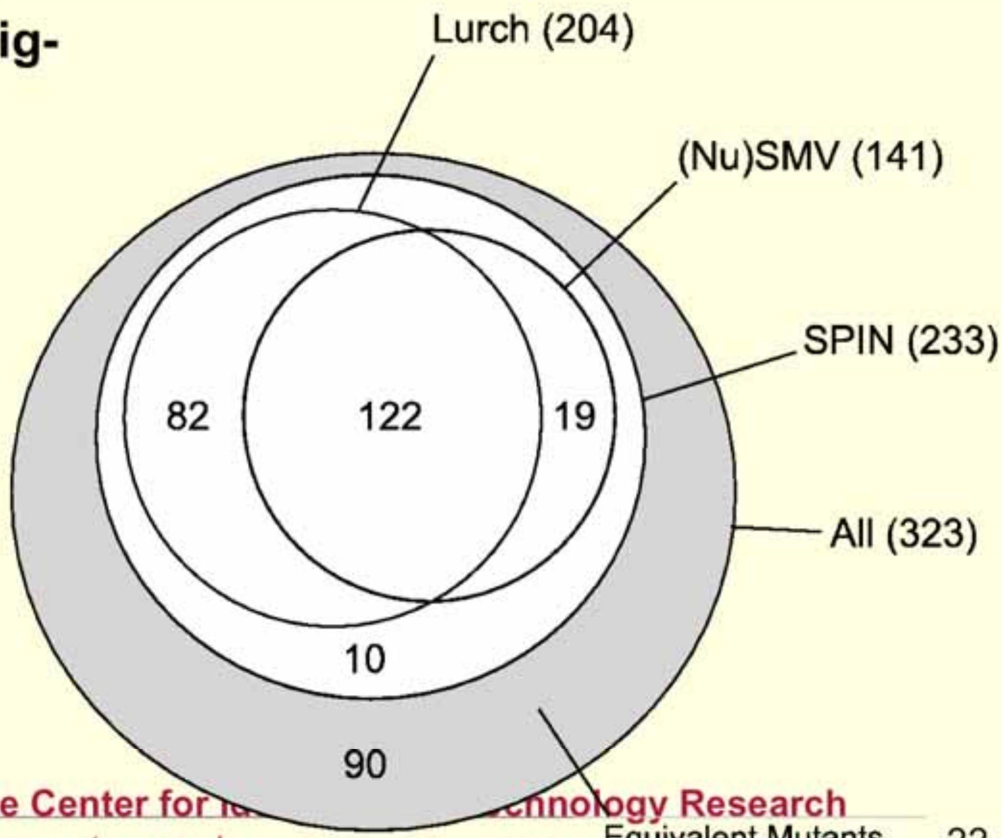
Inconsistent Results
(from alternative verification tools)

- **Cadence SMV and NuSMV**
  - NuSMV missed a property violation detected by Cadence SMV
  - Traced to translator's use of SPEC rather than INVARSPEC in property definition
  - Translator output fine for Cadence SMV, but not (although syntactically correct) for NuSMV
- **SPIN and Lurch**
  - SPIN (complete tool) missed error detected by Lurch (incomplete tool)
  - Translator to SPIN used invalid *d_step*
- **SPIN and Salsa**
  - SPIN reported violation of property proved true by Salsa
  - *NATURE* constraint in SCR model ignored by translator to SPIN
    - Much effort might have been wasted trying to track down the cause of the spurious error reported by SPIN

No indication NuSMV or SPIN had been used incorrectly on these models

Verification Accuracy

- **Salsa accurately proves properties true, but unproven properties may be true or false**
- **SMV works only on single-state assertions**
- **SPIN accuracy depends on configuration options (and NATURE constraints are ignored, so reported property violations must be confirmed)**
- **Lurch accurately detects property violations, but properties not disproved may be true or false (opposite of Salsa).**



Establishing confidence

- **State of the art:**
  - Different forms of success arguments, assurance cases (John Knight).
  - Design diversity, diversity arguments (Littlewood).
- **Needed: Mechanisms to,**
  - Decompose verification goals.
  - Compose verification claims.
    - Between separate verification activities.
    - Between different methods (testing, verification, inspection).
    - Over architectural elements
    - Against different specifications.
    - In different usage scenarios

Outline

- **Fault distributions**
  - Large scale, long term projects.
- **Software verification and validation methods and their effectiveness**
  - Implications on failure detection and forecasting.
- **Streamlining V&V**
  - Using the diversity to our advantage.
- **Summary**

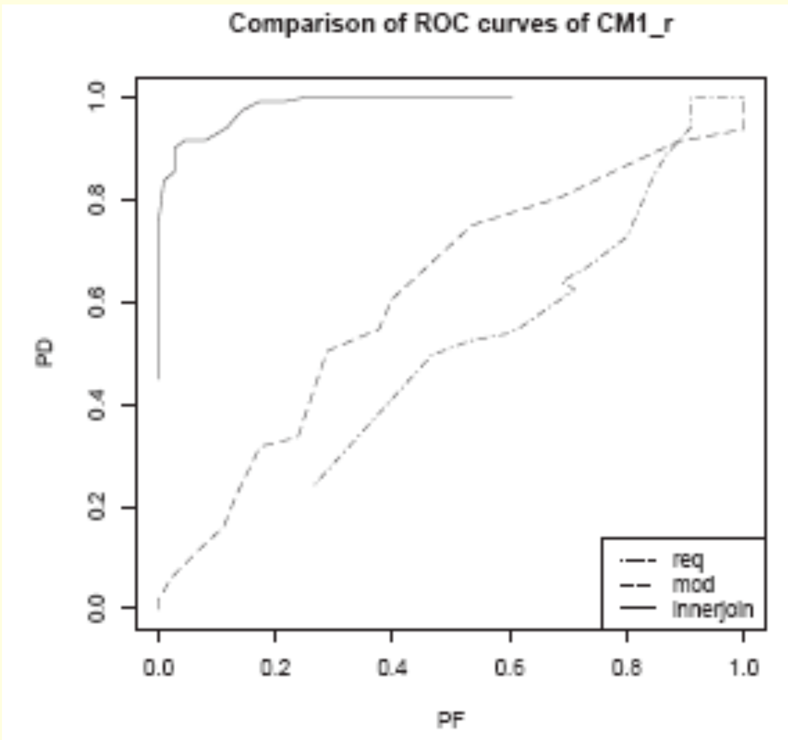
Identify "bad smells" early

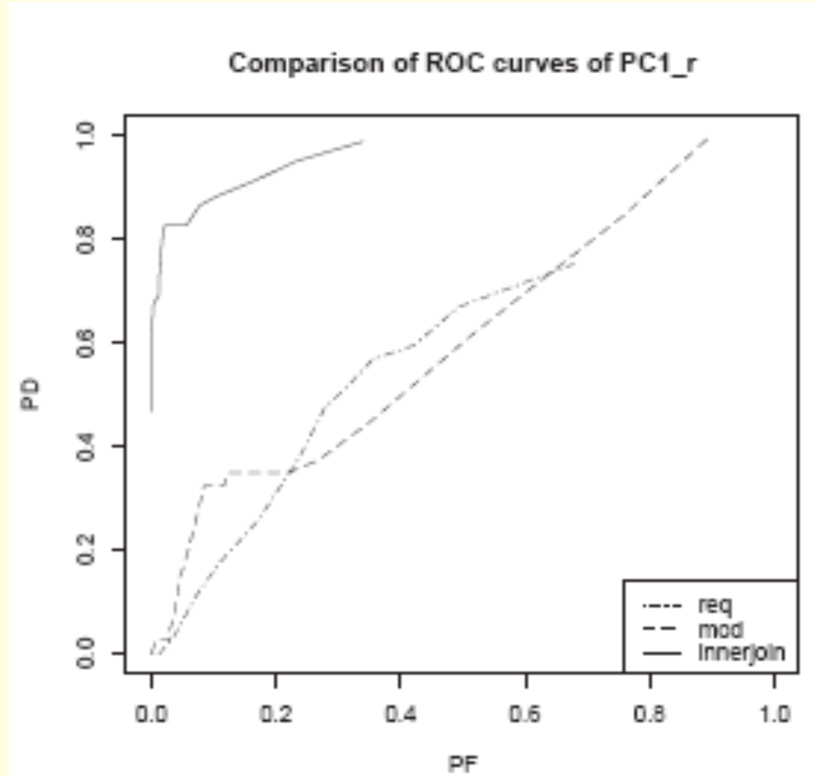
m = Mccabe		$v(g)$	cyclomatic_complexity
		$iv(G)$	design_complexity
		$ev(G)$	essential_complexity
locs	loc		loc_total (one line = one count
	loc(other)		loc_blank
			loc_code_and_comment
			loc_comments
			loc_executable
			number_of_lines (opening to closing brackets)
Halstead	h	N_1	num_operators
		N_2	num_operands
		μ_1	num_unique_operators
		μ_2	num_unique_operands
	H	N	length: $N = N_1 + N_2$
		V	volume: $V = N * log_2 \mu$
		L	level: $L = V^*/V$ where $V^* = (2 + \mu_2{}^*) log_2 (2 + \mu_2{}^*)$
		D	difficulty: $D = 1/L$
		I	content: $I = \hat{L} * V$ where $\hat{L} = \frac{2}{\mu_1} * \frac{\mu_2}{N_2}$
		E	effort: $E = V/\hat{L}$
		B	error_est
		T	prog_time: $T = E/18$ seconds

Can we identify where faults are likely to hide if we combine code metrics AND associated requirements metrics?

Requirement	Definitions
action	Represents the number of actions the requirement needs to be capable of performing.
conditional	Represents whether the requirement will be addressing more than one condition.
continuance	Phrases that follow an imperative and precede the definition of lower level requirement specification.
imperative	Those words and phrases that command that something must be provided.
incomplete	Phrases such as 'TBD' or 'TBR'. They are used when a requirement has yet to be determined.
option	Those words that give the developer latitude in the implementation of the specification that contains them.
risk_level	A calculated risk level metric based on weighted averages from metrics collected for each requirement.
source	Represents the number of sources the requirement will interface with or receive data from.
weak_phrase	Clauses that are apt to cause uncertainty and leave room for multiple interpretations.

Diverse metrics seem to work





- **CAVEATS:**
- **Small datasets**
  - Some requirements have no connection to any modules,
  - Few modules have requirements.
- **Models describe sub-projects.**

Combining design and code measures

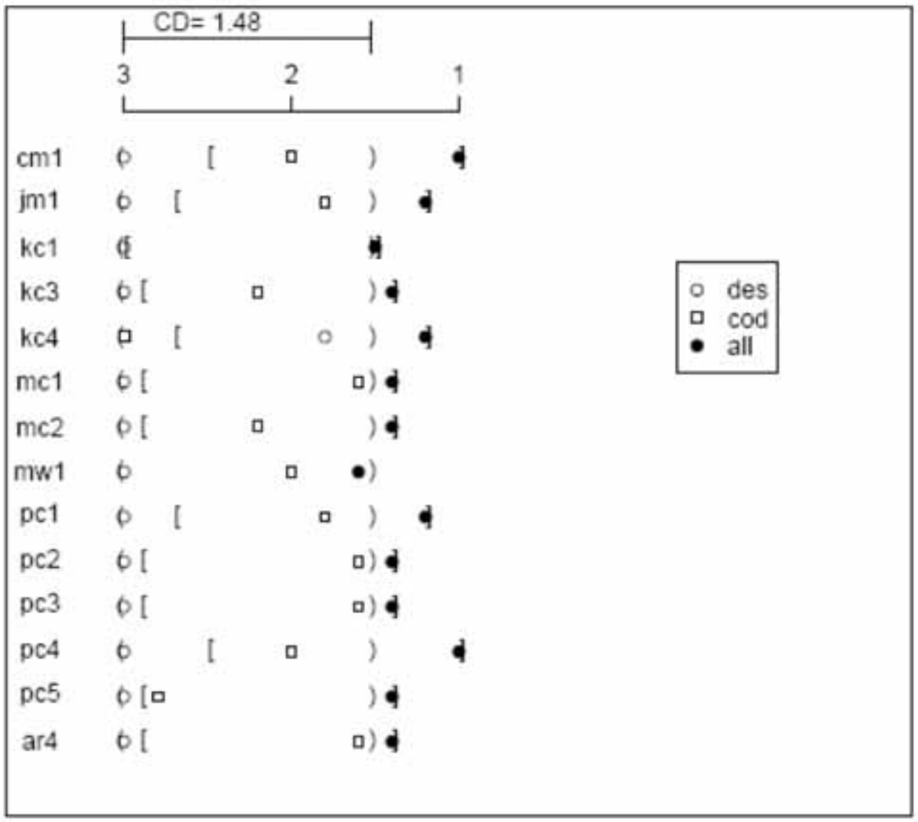


Fig. 13 Statistical performance ranks of *design*, *code*, and *all* models built using 50% of data for training.

- **Combination of *code and design* improves models built from *design* OR *code* metrics separately.**
- **Differences between *design* models and *all* models statistically significant.**

Streamlining?

- **Software projects have different business goals.**
- **Extremes:**
  - Cost averse:
    Release ASAP, deal with consequences later
  - Risk aversion
    Identify fault-prone modules at any cost, because the dramatic consequences of failures
- **How do our models compare to trivial classifiers?**
  - Analyze all or nothing

V&V costs

Table 2. Probability cost ranges.

Data set	Risk	Probability cost range of interest	Best classifier	Significant performance range and $\% \frac{significantRange}{interestedRange}$		Significant μ range
JM1	high	(0.545, 0.960)	rf	(0.545, 0.72)	42.17%	$(\frac{1}{5}, \frac{1}{10})$
MC1	high	(0.031, 0.391)	rf	entire	100%	$(\frac{1}{5}, \frac{1}{100})$
PC2	high	(0.021, 0.297)	rf	(0.15, 0.297)	53.26%	$(\frac{1}{40}, \frac{1}{100})$
PC5	high	(0.134, 0.756)	rf	entire	100%	$(\frac{1}{5}, \frac{1}{100})$
PC1	high	(0.261, 0.876)	rf	(0.261, 0.80)	87.64%	$(\frac{1}{5}, \frac{1}{50})$
PC3	high	(0.368, 0.921)	rf,bst	(0.368, 0.82)	81.74%	$(\frac{1}{5}, \frac{1}{40})$
PC4	high	(0.411, 0.933)	rf	entire	100%	$(\frac{1}{5}, \frac{1}{100})$
CM1	med.	(0.037, 0.489)	rf	(0.11, 0.489)	83.85%	$(1.5, \frac{1}{5})$
MW1	med.	(0.014, 0.264)	bag,bst	(0.13, 0.264)	53.6%	$(\frac{1}{2}, \frac{1}{5})$
KC1	med.	(0.031, 0.447)	rf	(0.14, 0.447)	73.8%	$(1, \frac{1}{5})$
KC3	med.	(0.013, 0.252)	bag	(0.18, 0.252)	30.13%	$(\frac{1}{3.3}, \frac{1}{5})$
KC4	low	(0.009, 0.156)	nb	(0.10, 0.156)	38.10%	(8, 5)
MC2	low	(0.005, 0.087)	nb	no	0	
ar3	low	(0.001, 0.028)	bst	no	0	
ar4	low	(0.002, 0.044)	bst	no	0	
ar5	low	(0.003, 0.054)	bst	no	0	

Streamlining V&V

- **Identifying subsystems which need V&V resources the most.**
  - A process guidance, not assessment.
- **Challenges:**
  - Diversification of information sources
    Metrics, language analysis, human factors and social networks [Weyuker et al., Nagappan et al.]
  - Searching for security vulnerabilities (beyond buffer overflow). [Williams, UIUC…].
  - Fusion of information sources to benefit model performance.
  - How do these techniques play into the overall reliability/ dependability assessment?

Summary

- **"Stampede cycles"**
- **Many problems remain research challenges:**
  - Composition of verification arguments.
    - Necessary, because no one technique can address all the types of commonly occurring faults.
    - Necessary, because of the varying strengths of verification analyses techniques (and assumptions they make).
    - Ever growing complexity necessitates decomposition of verification goals.
    - How to do this without breaking the budget?