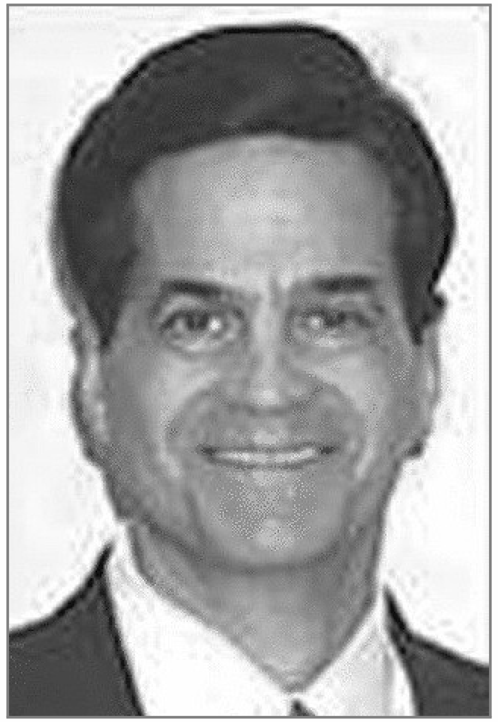
Defects Density

Yegor Bugayenko

Lecture #18 out of 24
80 minutes



Michael Fagan

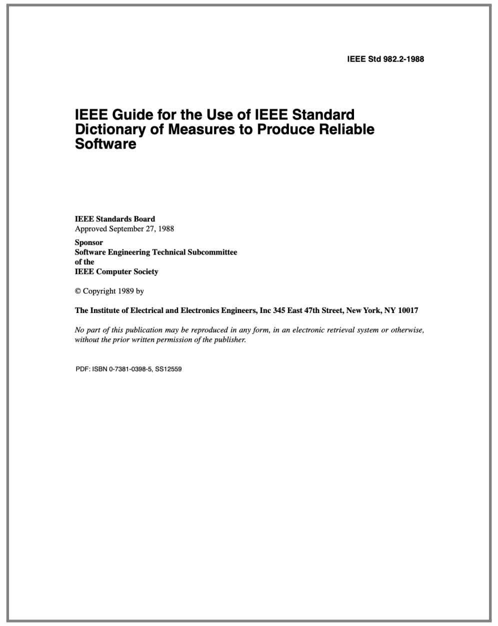
"Feedback of results from inspections must be counted for the programmer's use and benefit: they should not under any circumstances be used for programmer performance appraisal."

— Michael Fagan. Design and Code Inspections to Reduce Errors in Program Development. *IBM Systems Journal*, (3), 1976

Figure 8 Example of most error-prone modules based on I_1 and I_2

Module name	*Number of errors*	*Lines of code*	*Error density, Errors/K. Loc*
Echo	4	128	31
Zulu	10	323	31
Foxtrot	3	71	28
Alpha	7	264	27←Average
Lima	2	106	19 Error
Delta	3	195	15 Rate
⋮	⋮	⋮	⋮
	67		

Source: Michael Fagan. Design and Code Inspections to Reduce Errors in Program Development. *IBM Systems Journal*, (3), 1976



IEEE Std 982.2-1988

IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software

IEEE Standards Board
Approved September 27, 1988

Sponsor
Software Engineering Technical Subcommittee
of the
IEEE Computer Society

PDF: ISBN 0-7381-0398-5, SS12559

"A defect is a product anomaly. Examples include such things as 1) omissions and imperfections found during early life cycle phases and 2) faults contained in software sufficiently mature for test or operation."

— IEEE Standards Board. IEEE Std 982.2-1988: Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software, 1989

$$I = 7$$
$$KSLOD = 8$$

Then,

$$\sum_{i=1}^{7} D_i = 78 \text{ (total defects found)}$$

$$DD = \frac{78}{8} = 9.8 \text{ (estimated defect density)}$$

Source: IEEE Standards Board. IEEE Std 982.2-1988: Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software, 1989

“This measure has a degree of indeterminism. For example, a low value may indicate either a good process and a good product or it may indicate a bad process. If the value is low compared to similar past projects, the inspection process should be examined. If the inspection process is found to be adequate, it should then be concluded that the development process has resulted in a relatively defect-free product.”

Table 4.1-1 — Measure Classification Matrix

	Product Measures						Process Measures		
Measures (Experience)	Errors, Faults, Failures	Mean Time to Failure; Failure Rate	Reliability Growth & Projection	Remaining Product Faults	Completeness & Consistency	Complexity	Management Control	Coverage	Risk, Benefit, Cost Evaluation
1. Fault density (2)	X								
2. Defect density (3)	X								
3. Cumulative failure profile (1)	X								
4. Fault-days number (0)	X						X		
5. Functional or modular test coverage (1)					X			X	X
6. Cause and effect graphing (2)					X			X	
7. Requirements traceability (3)	X				X			X	
8. Defect indices (1)	X						X		
9. Error distribution(s) (1)							X		
10. Software maturity index (1)			X						X
11. Man hours per major defect detected (2)							X		X
12. Number of conflicting requirements (2)	X				X			X	
13. Number of entries/exists per module (1)					X	X			
14. Software science measures (3)				X		X			
15. Graph-theoretic complexity for architecture (1)						X			
16. Cyclomatic complexity (3)					X	X			
17. Minimal unit test case determination (2)					X	X			
18. Run reliability (2)			X						
19. Design structure (1)						X			
20. Mean time to discover the next K faults (3)									X
21. Software purity level (1)			X						
22. Estimated number of faults remaining (seeding) (2)				X					
23. Requirements compliance (1)	X				X			X	
24. Test coverage (2)					X			X	
25. Data or information flow complexity (1)						X			
26. Reliability growth function (2)			X						
27. Residual fault count (1)				X					
28. Failure analysis using elapsed time (3)			X	X					
29. Testing sufficiency (0)			X					X	
30. Mean-time-to-failure (3)		X	X						
31. Failure rate (3)		X							
32. Software documentation & source listings (2)					X				
33. RELY – (Required Software Reliability) (1)								X	X
34. Software release readiness (0)									X
35. Completeness (2)					X				
36. Test accuracy (1)				X	X			X	
37. System performance reliability (2)			X						
38. Independent process reliability (0)			X						
39. Combined HW/SW system operational availability (0)			X						

Source: IEEE Standards Board. IEEE Std 982.2-1988: Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software, 1989

39 Measures for Reliable Software

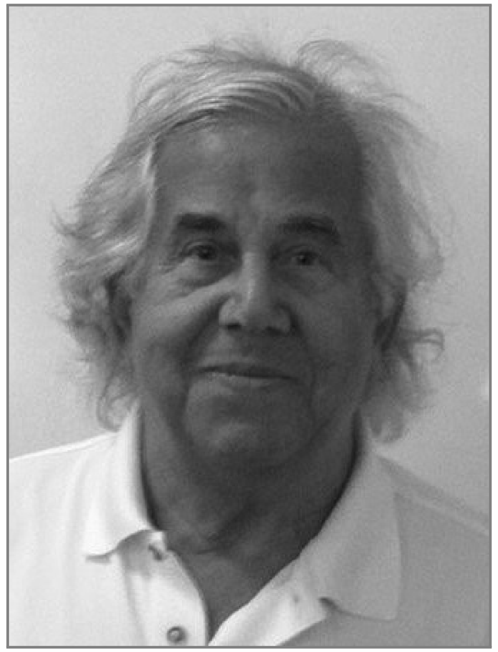
1. Fault Density
2. Defect Density
3. Cumulative Failure Profile
4. Fault-Days Number
5. Functional or Modular Test Coverage
6. Cause and Effect Graphing
7. Requirements Traceability
8. Defect Indices
9. Error Distribution(s)
10. Software Maturity Index
11. Manhours per Major Defect Detected
12. Number of Conflicting Requirements
13. Number of Entries and Exits per Module
14. Software Science Measures
15. Graph-Theoretic Complexity for Arch.
16. Cyclomatic Complexity
17. Minimal Unit Test Case Determination
18. Run Reliability
19. Design Structure
20. Mean Time to Discover the Next K Faults
21. Software Purity Level
22. Estimated Num. of Faults Remaining
23. Requirements Compliance
24. Test Coverage
25. Data or Information Flow Complexity
26. Reliability Growth Function
27. Residual Fault Count
28. Failure Analysis Using Elapsed Time
29. Testing Sufficiency
30. Mean Time to Failure
31. Failure Rate
32. Software Docmtn and Source Listings
33. RELY-Required Software Reliability
34. Software Release Readiness
35. Completeness
36. Test Accuracy
37. System Performance Reliability
38. Independent Process Reliability
39. Combined H&S Operational Availability

Source: IEEE Standards Board. IEEE Std 982.2-1988: Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software, 1989

NACHIAPPAN NAGAPPAN

"A case study performed on Windows Server 2003 indicates the validity of the relative code churn measures as early indicators of system defect density. Our code churn metric suite is able to discriminate between fault and not fault-prone binaries with an accuracy of 89%."

— Nachiappan Nagappan and Thomas Ball. Use of Relative Code Churn Measures to Predict System Defect Density. In *Proceedings of the 27th International Conference on Software Engineering*, pages 284–292, 2005



Joseph Sherif

"The analysis showed a significantly higher density of defects during requirements inspections. It was also observed, that the defect densities found decreased exponentialy as the mork products approached the coding phase."

— John C. Kelly, Joseph S. Sherif, and Jonathan Hops. An Analysis of Defect Densities Found During Software Inspections. *Journal of Systems and Software*, 17(2):111–117, 1992

Norman Fenton

"Our critical review of state-of-the-art of models for predicting software defects has shown that many methodological and theoretical mistakes have been made... We recommend holistic models for software defect prediction, using Bayesian Belief Networks, as alternative approaches to the single-issue models used at present."

— Norman E. Fenton and Martin Neil. A Critique of Software Defect Prediction Models. *IEEE Transactions on Software Engineering*, 25(5):675–689, 1999

TABLE 4
DEFECTS DENSITY (F/KLOC) VS. MTTF

F/KLOC	MTTF
> 30	1 min
20–30	4-5 min
5–10	1 hr
2–5	several hours
1–2	24 hr
0.5–1	1 month

"This means we should be very wary of attempts to equate fault densities with failure rates, as proposed for example by Jones [1996]. Although highly attractive in principle, such a model does not stand up to empirical validation."

Source: Norman E. Fenton and Martin Neil. A Critique of Software Defect Prediction Models. *IEEE Transactions on Software Engineering*, 25(5):675–689, 1999

TABLE 1
DEFECTS PER LIFE-CYCLE PHASE PREDICTION USING TESTING METRICS

Defect Origins	Defects per Function Point
Requirements	1.00
Design	1.25
Coding	1.75
Documentation	0.60
Bad fixes	0.40
Total	5.00

"We already see defect density defined in terms of defects per function point, and empirical studies are emerging that seem likely to be the basis for predictive models. For example, Jones [1991] reports the following bench-marking study, reportedly based on large amounts of data from different commercial sources."

Source: Norman E. Fenton and Martin Neil. A Critique of Software Defect Prediction Models. *IEEE Transactions on Software Engineering*, 25(5):675–689, 1999

"Industry average experience is about 1-25 errors per 1000 lines of code for delivered software. Cases that have one-tenth as many errors as this are rare; cases that have 10 times more tend not to be reported. (They probably aren't ever completed!) Microsoft experiences about 10–20 defects per 1000 lines of code during in-house testing and 0.5 defects per 1000 lines of code in released product."

— Steve McConnell. *Code Complete.* Pearson Education, 2004

"While our experience in applying statistical quality-control techniques to software development is limited, initial experience indicates that five fixes per thousand lines of code can be tolerated without invalidating the application of statistics to estimate MTTF. This failure rate is low compared to normal development practices, where 20 to 60 fixes per thousand lines of code is not atypical."

— Richard H. Cobb and Harlan D. Mills. Engineering Software under Statistical Quality Control. *IEEE software*, 7(6):45–54, 1990

A Güneş Koru

"We studied four large-scale object-oriented products, Mozilla, Cn3d, JBoss, and Eclipse. We observed that defect proneness increased as class size increased, but at a slower rate; smaller classes were proportionally more problematic than larger classes."

— A Güneş Koru, Dongsong Zhang, Khaled El Emam, and Hongfang Liu. An Investigation into the Functional Form of the Size-Defect Relationship for Software Modules. *IEEE Transactions on Software Engineering*, 35(2):293–304, 2008

Parastoo Mohagheghi

"The analysis showed that reused components have lower defect-density than non-reused ones. Reused components have more defects with highest severity than the total distribution, but less defects after delivery."

— Parastoo Mohagheghi, Reidar Conradi, Ole M. Killi, and Henrik Schwarz. An Empirical Study of Software Reuse vs. Defect-Density and Stability. In *Proceedings of the 26th International Conference on Software Engineering*, pages 282–291. IEEE, 2004

TABLE IX. Complexity and Error Rate for Errored Modules

Module Size	Average Cyclomatic Complexity	Errors/1000 Executable Lines
50	6.2	65.0
100	19.6	33.3
150	27.5	24.6
200	56.7	13.4
>200	77.5	9.7

"One surprising result was that module size did not account for error proneness. In fact, it was quite the contrary—the larger the module, the less error prone it was. This was true even though the larger modules were more complex."

Source: Victor R. Basili and Barry T. Perricone. Software Errors and Complexity: An Empirical Investigation. *Communications of the ACM*, 27(1): 42–52, 1984

?

- ???

References

Victor R. Basili and Barry T. Perricone. Software Errors and Complexity: An Empirical Investigation. *Communications of the ACM*, 27(1): 42–52, 1984.

IEEE Standards Board. IEEE Std 982.2-1988: Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software, 1989.

Richard H. Cobb and Harlan D. Mills. Engineering Software under Statistical Quality Control. *IEEE software*, 7(6):45–54, 1990.

Michael Fagan. Design and Code Inspections to Reduce Errors in Program Development. *IBM Systems Journal*, (3), 1976.

Norman E. Fenton and Martin Neil. A Critique of Software Defect Prediction Models. *IEEE Transactions on Software Engineering*, 25(5): 675–689, 1999.

Capers Jones. *Applied Software Measurement*. McGraw-Hill, 1991.

Capers Jones. The Pragmatics of Software Process Improvements. *Technical Council on Software Engineering*, 14(2), 1996.

John C. Kelly, Joseph S. Sherif, and Jonathan Hops. An Analysis of Defect Densities Found During Software Inspections. *Journal of Systems and Software*, 17(2):111–117, 1992.

A Güneş Koru, Dongsong Zhang, Khaled El Emam, and Hongfang Liu. An Investigation into the Functional Form of the Size-Defect Relationship for Software Modules. *IEEE Transactions on Software Engineering*, 35(2):293–304, 2008.

Steve McConnell. *Code Complete*. Pearson Education, 2004.

Parastoo Mohagheghi, Reidar Conradi, Ole M. Killi, and Henrik Schwarz. An Empirical Study of Software Reuse vs. Defect-Density and Stability. In *Proceedings of the 26th International Conference on Software Engineering*, pages 282–291. IEEE, 2004.

Nachiappan Nagappan and Thomas Ball. Use of Relative Code Churn Measures to Predict System Defect Density. In *Proceedings of the 27th International Conference on Software Engineering*, pages 284–292, 2005.