Dependable Systems

Reliability Prediction

Dr. Peter Tröger

Krishna B. Misra: Handbook of Performability Engineering. Springer. 2008 (p. 265ff)

Predicting System Reliability - Application Areas

- Feasibility evaluation for given model, identification of potential problem sources
- Comparison of competing designs
- Identification of potential reliability problems - low quality, over-stressed parts
- Input to other reliability-related tasks
  - Maintainability analysis, testability evaluation
  - Failure modes and effects analysis (FMEA)
- Ultimate goal is the prediction of system failures by a reliability methodology
- Approach depends on nature of component (electrical, electronic, mechanical)

Predicting System Reliability - Procedure

- Procedure of system reliability prediction [Misra]
  - Define system and its operating conditions
  - Define system performance criteria (success, failure)
  - Define system model (e.g. with RBDs or fault trees)
  - Compile parts list for the system
  - Assign failure rates and modify generic failure rates with an according procedure
  - Combine part failure rates (see last lecture)
  - Estimate system reliability

Reliability Data

- **Field (operational) failure data**
  - Meaningful source of information, experience from real world operation
  - Operational and environmental conditions may be not fully known
- **Service life data with / without failure**
  - Helpful in assessing time characteristics of reliability issues
- **Data from engineering tests**
  - Example: Accelerated life tests
  - Results from controlled environment
  - Trustworthy for analysis purposes
- Lack of failure information is the ‚greatest deficiency‘ in reliability research [Misra]

Failure Probability Sources (Clemens & Sverdrup)

- Original data
  - Manufacturer‘s Data
  - Industry Consensus Standards, MIL Standards
- Historical Evidence - Same or similar systems
  - Similar product technique - Take experience from old product with similar attributes
  - Similar complexity technique - Estimate reliability by relate product complexity to well-known product
  - Prediction by function technique - Part count prediction or part stress prediction
- Simulation or testing

Reliability Prediction Models

- Economic requirements affect field-data availability
  - Customers do not need generic field data, so collection is extra effort
- Numerical reliability prediction models available for specific areas
  - **Hardware-oriented reliability modeling**
    - MIL-HDBK-217, Bellcore Reliability Prediction Program, ...
  - **Software-oriented reliability modeling**
    - Jelinski-Moranda model, Basic Execution model, software metrics, ...
- Prediction models look for corrective factors, so they are always focused
- Demands confidence in understanding of environmental conditions
- Reconciliation is often needed between the different values of failure data from various sources

MIL-HDBK 217

- Most widely used and accepted prediction data for electronic components
- Military handbook for reliability prediction of electronic equipment
- First published by the U.S. military in 1965, last revision F2 from 1995
- Common ground for comparing the reliability of two or more similar designs
- Empirical failure rate models for various part types used in electronic systems
    - ICs, transistors, diodes, resistors, capacitors, relays, switches, connectors, ...
    - Failures per million operating hours
- Modification for mechanical components
    - Due not follow exponential failure distribution or constant rate (e.g. wear-out)
    - Focus on very long investigation time to include replacement on failure

MIL-HDBK 217 - Part Count Method

- **Part count prediction** in early product development cycle, quick estimate
- Prepare a list of each generic part type and their numbers used
    - Assumes that component setup is reasonably fixed in the design process
- MIL 217 provides set of default tables with generic failure rate per component type
    - Based on intended operational environment
    - **Quality factor** - Manufactured and tested for military, or relaxed environment
    - **Learning factor** - Number of years the component has been in production
- Assumes constant failure rate for each component, assembly and the system

MIL-HDBK 217 - Part Count Method

- Early estimate of a failure rate, given by

$$\lambda = \sum_{i=1}^{n} N_i \lambda_i \pi_{Q_i}$$

- n - Number of part categories
- N_i - Number of parts in category i
- λ_i - Failure rate of category i
- π_{Q_i} - Quality factor for category i
- Literally: Counting similar components of a product and grouping them in types
  - Quality factor from MIL 217 tables
  - Part count assumes all components in a series

MIL-HDBK 217 - Part Stress Method

- **Part stress analysis prediction** supporting 14 different operational conditions
- Accurate method for prediction, as alternative to simulation
- Used with final design - detailed part lists and stress test results are known
  - Demands knowledge of all stresses (temperature, humidity, vibration, ...) and their effect on the failure rate
  - Relies typically on data sheet from component manufacturer
- MIL 217 provides mathematical model for failure rate per component type
- Environment factors influence resulting failure rate
  - Emulation of environments in test chambers
  - Excludes effects of ionizing radiation

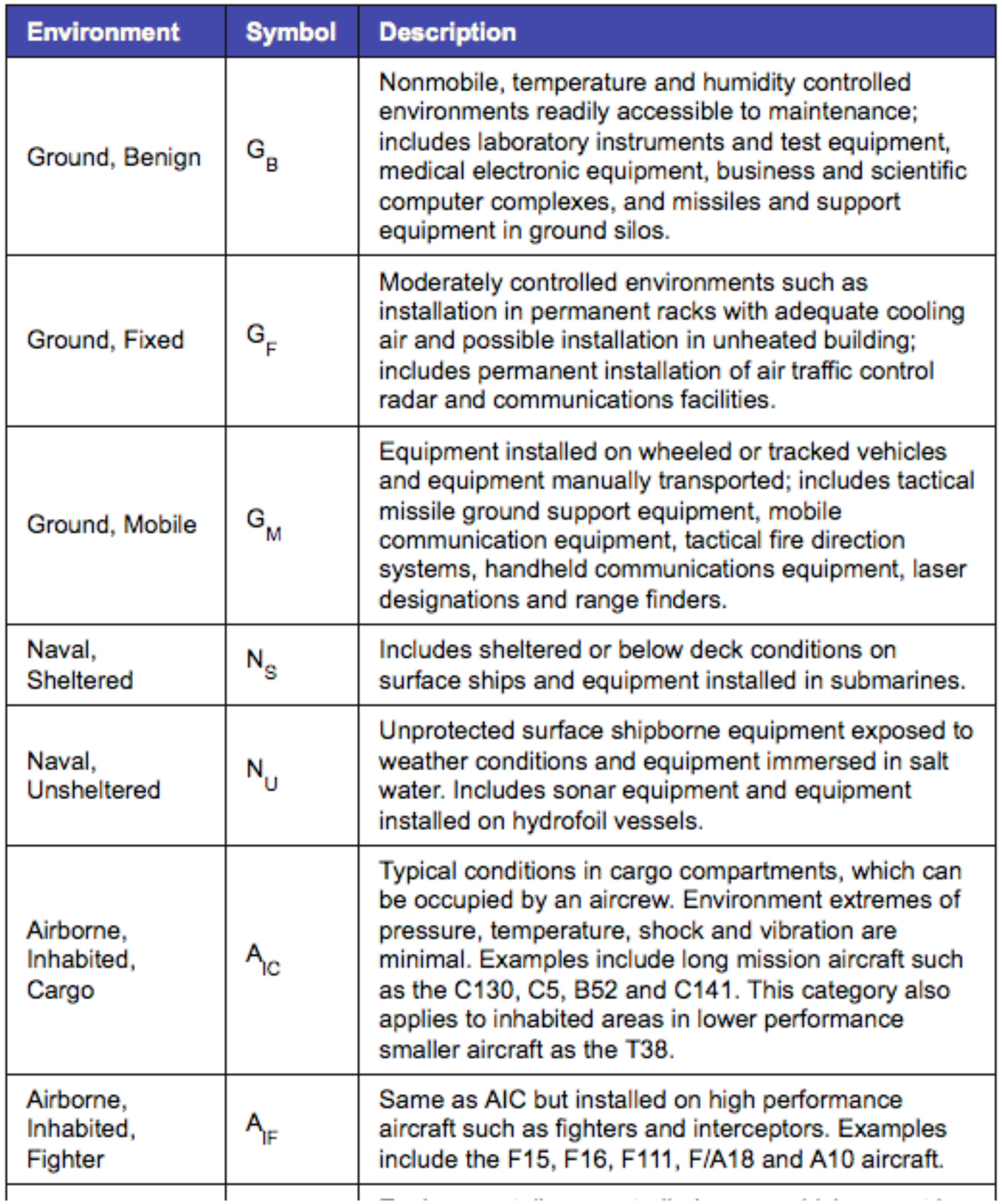
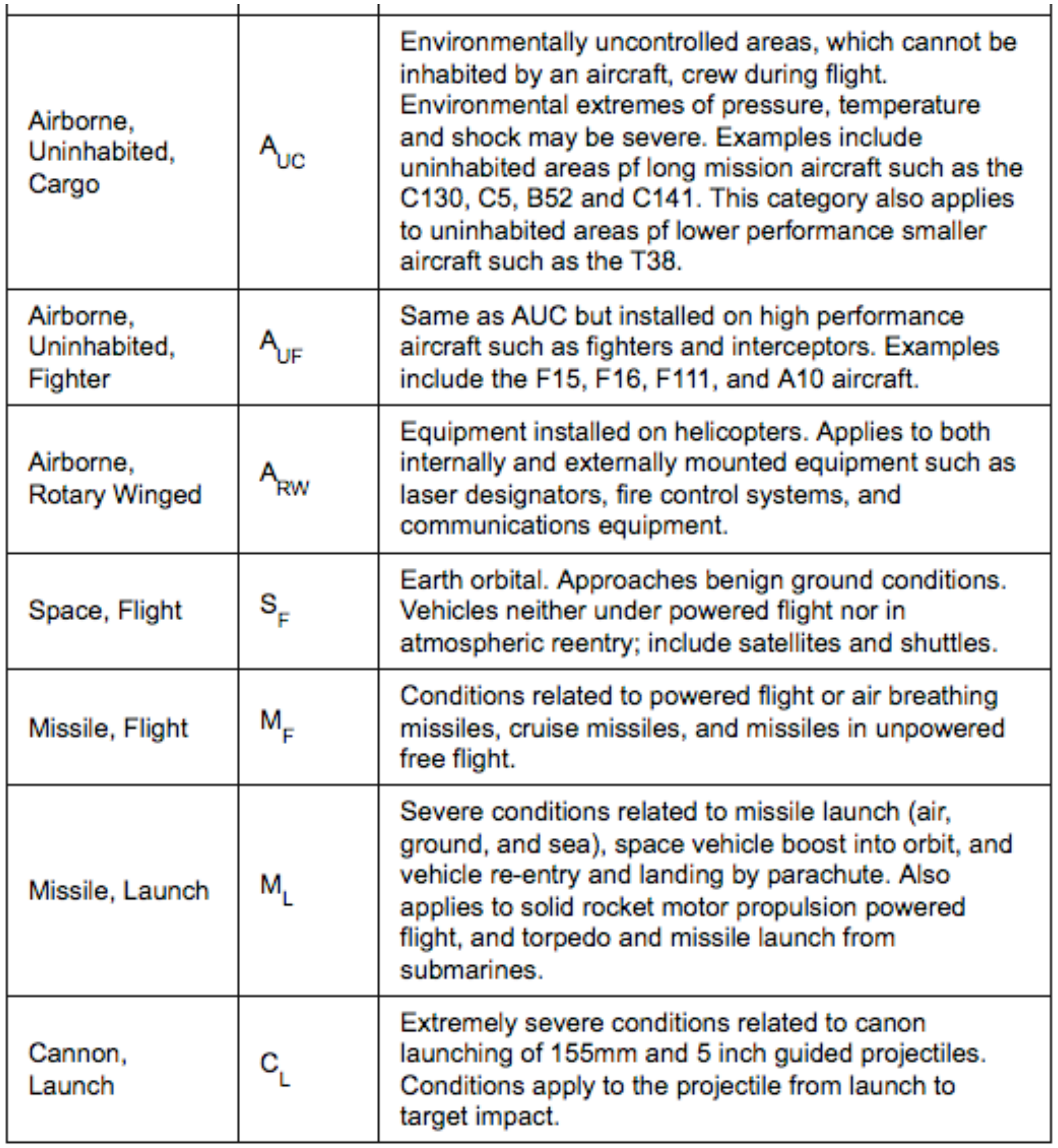
MIL-HDBK 217 - Part Stress Method

- Determine the system failure rate from each part's stress level

$$\lambda = \pi_L \pi_Q (C_1 \pi_T \pi_V + C_2 \pi_E)$$

- π_L - Learning factor (1-2)
- π_Q - Production quality factor (1-100)
- π_T - Temperature acceleration factor (1-100)
- π_V - Voltage stress factor
- π_E - *Application environment factor* (1-20)
- C_1, C_2 - Technology constants for complexity (1-100) and case (1-20)
- λ - Failure rate per million hours
- Supports quantitative assessment of relative cost-benefit of system-level tradeoffs

Application Environment Factor

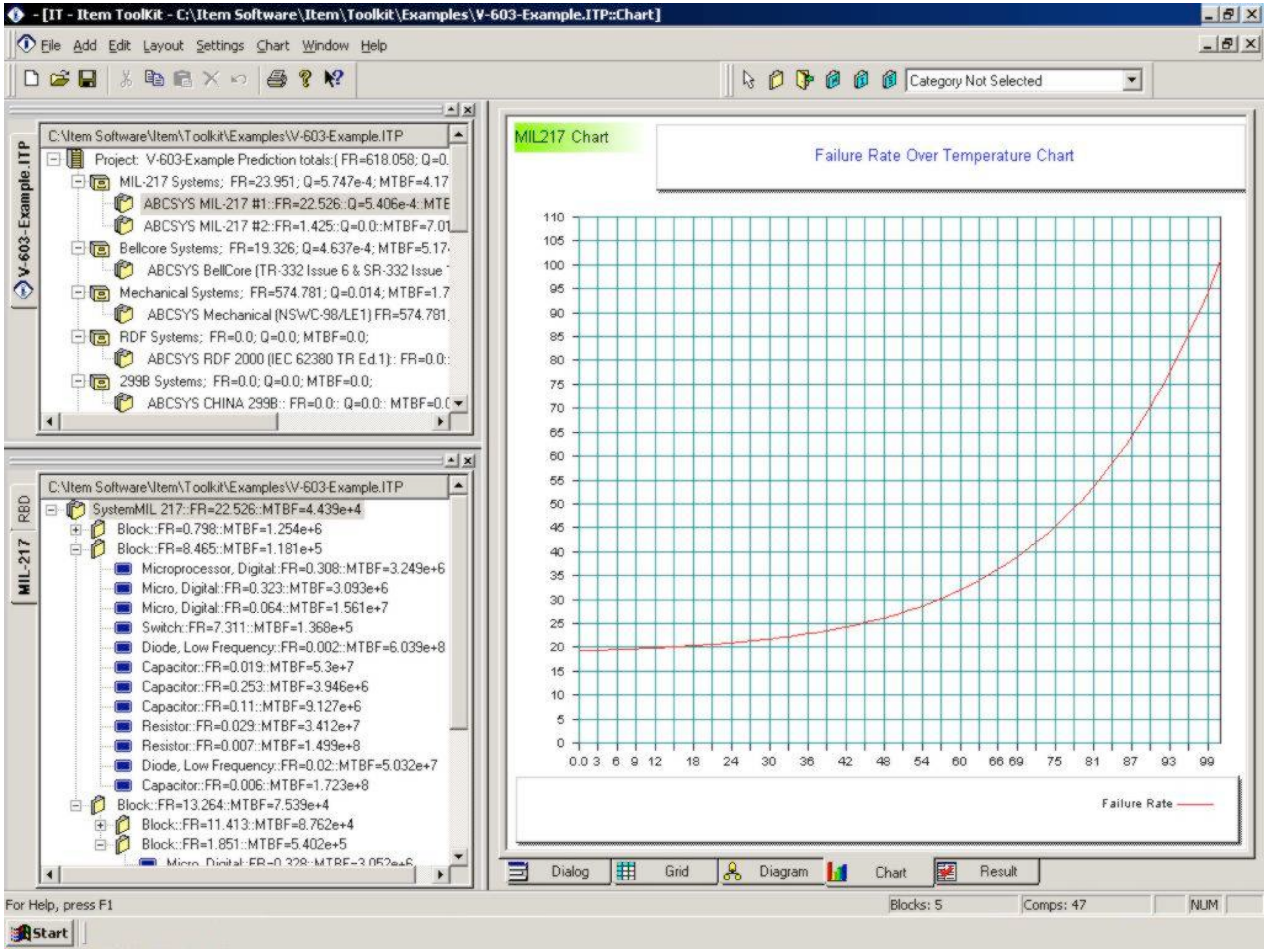
Environment	Symbol	Description
Ground, Benign	G_B	Nonmobile, temperature and humidity controlled environments readily accessible to maintenance; includes laboratory instruments and test equipment, medical electronic equipment, business and scientific computer complexes, and missiles and support equipment in ground silos.
Ground, Fixed	G_F	Moderately controlled environments such as installation in permanent racks with adequate cooling air and possible installation in unheated building; includes permanent installation of air traffic control radar and communications facilities.
Ground, Mobile	G_M	Equipment installed on wheeled or tracked vehicles and equipment manually transported; includes tactical missile ground support equipment, mobile communication equipment, tactical fire direction systems, handheld communications equipment, laser designations and range finders.
Naval, Sheltered	N_S	Includes sheltered or below deck conditions on surface ships and equipment installed in submarines.
Naval, Unsheltered	N_U	Unprotected surface shipborne equipment exposed to weather conditions and equipment immersed in salt water. Includes sonar equipment and equipment installed on hydrofoil vessels.
Airborne, Inhabited, Cargo	A_{IC}	Typical conditions in cargo compartments, which can be occupied by an aircrew. Environment extremes of pressure, temperature, shock and vibration are minimal. Examples include long mission aircraft such as the C130, C5, B52 and C141. This category also applies to inhabited areas in lower performance smaller aircraft as the T38.
Airborne, Inhabited, Fighter	A_{IF}	Same as AIC but installed on high performance aircraft such as fighters and interceptors. Examples include the F15, F16, F111, F/A18 and A10 aircraft.
Airborne, Uninhabited, Cargo	A_{UC}	Environmentally uncontrolled areas, which cannot be inhabited by an aircraft, crew during flight. Environmental extremes of pressure, temperature and shock may be severe. Examples include uninhabited areas pf long mission aircraft such as the C130, C5, B52 and C141. This category also applies to uninhabited areas pf lower performance smaller aircraft such as the T38.
Airborne, Uninhabited, Fighter	A_{UF}	Same as AUC but installed on high performance aircraft such as fighters and interceptors. Examples include the F15, F16, F111, and A10 aircraft.
Airborne, Rotary Winged	A_{RW}	Equipment installed on helicopters. Applies to both internally and externally mounted equipment such as laser designators, fire control systems, and communications equipment.
Space, Flight	S_F	Earth orbital. Approaches benign ground conditions. Vehicles neither under powered flight nor in atmospheric reentry; include satellites and shuttles.
Missile, Flight	M_F	Conditions related to powered flight or air breathing missiles, cruise missiles, and missiles in unpowered free flight.
Missile, Launch	M_L	Severe conditions related to missile launch (air, ground, and sea), space vehicle boost into orbit, and vehicle re-entry and landing by parachute. Also applies to solid rocket motor propulsion powered flight, and torpedo and missile launch from submarines.
Cannon, Launch	C_L	Extremely severe conditions related to canon launching of 155mm and 5 inch guided projectiles. Conditions apply to the projectile from launch to target impact.

PT 2014

MIL-HDBK 217

- Till today the most commonly used method for MTBF computation
- Was based on historical component failure data, mostly exponential distribution
- 1996 announced as discontinued by the U.S. Army
    - „has been shown to be unreliable“
    - Main driver in product reliability is no longer hardware component reliability
    - Failure rates in the specification are too high for todays electronic
- Specific failure model per component
    - Example: Solid tantalum fixed electrolytic capacitor
    - Failure rate depends on base failure rate, series resistance factor, quality level and environment factor

MIL-HDBK 217 Tool Support


File Add Edit Layout Settings Chart Window Help
Category Not Selected
MIL217 Chart
Failure Rate Over Temperature Chart
Failure Rate
Dialog Grid Diagram Chart Result
V-603-Example.ITP
RBD
MIL-217
Start

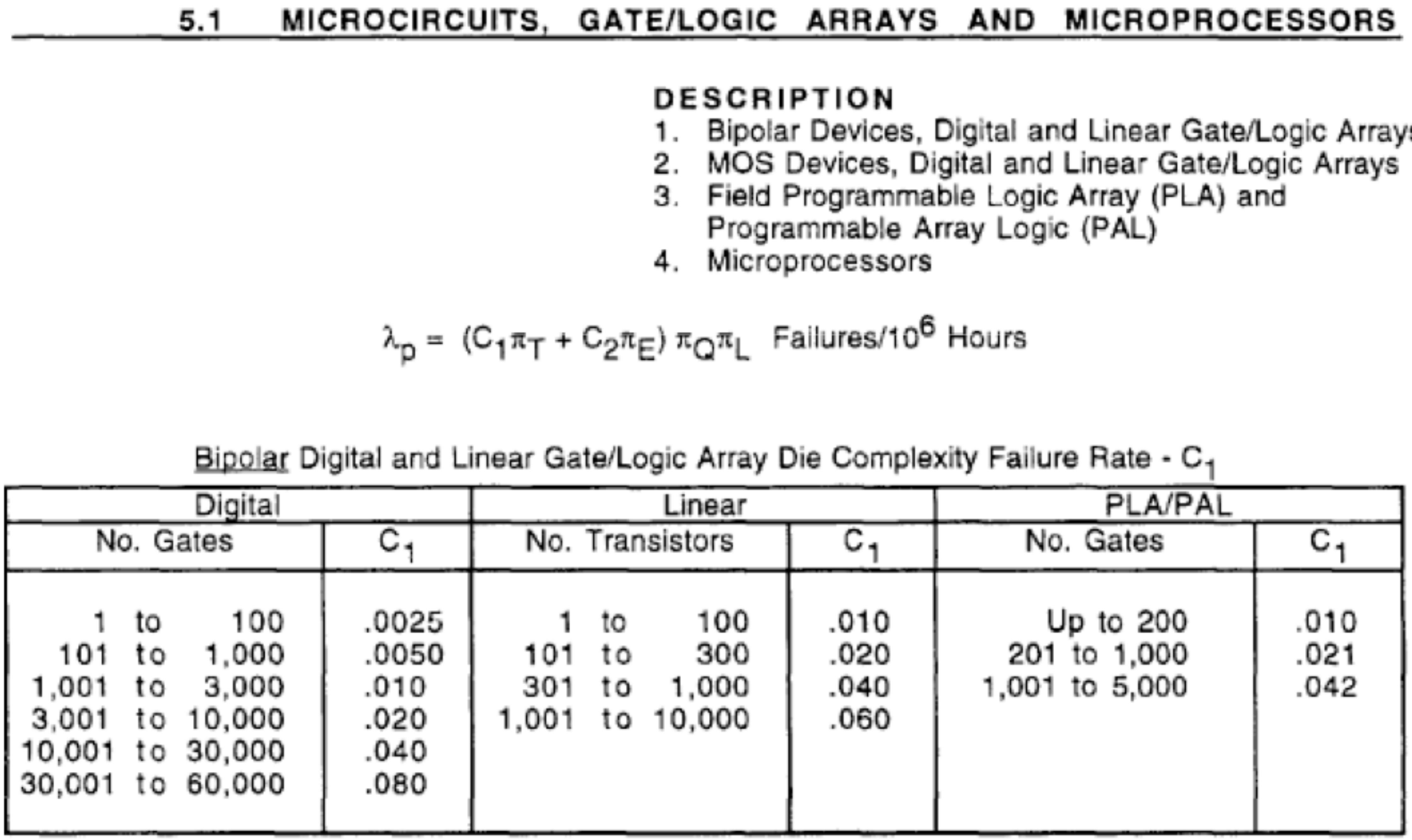
- [IT - Item ToolKit - C:\Item Software\Item\Toolkit\Examples\V-603-Example.ITP::Chart]

C:\Item Software\Item\Toolkit\Examples\V-603-Example.ITP
- Project: V-603-Example Prediction totals:(FR=618.058; Q=0.
  - MIL-217 Systems; FR=23.951; Q=5.747e-4; MTBF=4.17
    - ABCSYS MIL-217 #1::FR=22.526::Q=5.406e-4::MTE
    - ABCSYS MIL-217 #2::FR=1.425::Q=0.0::MTBF=7.01
  - Bellcore Systems; FR=19.326; Q=4.637e-4; MTBF=5.17
    - ABCSYS BellCore (TR-332 Issue 6 & SR-332 Issue
  - Mechanical Systems; FR=574.781; Q=0.014; MTBF=1.7
    - ABCSYS Mechanical (NSWC-98/LE1) FR=574.781
  - RDF Systems; FR=0.0; Q=0.0; MTBF=0.0;
    - ABCSYS RDF 2000 (IEC 62380 TR Ed.1):: FR=0.0::
  - 299B Systems; FR=0.0; Q=0.0; MTBF=0.0;
    - ABCSYS CHINA 299B:: FR=0.0:: Q=0.0:: MTBF=0.0

C:\Item Software\Item\Toolkit\Examples\V-603-Example.ITP
- SystemMIL 217::FR=22.526::MTBF=4.439e+4
  - Block::FR=0.798::MTBF=1.254e+6
  - Block::FR=8.465::MTBF=1.181e+5
    - Microprocessor, Digital::FR=0.308::MTBF=3.249e+6
    - Micro, Digital::FR=0.323::MTBF=3.093e+6
    - Micro, Digital::FR=0.064::MTBF=1.561e+7
    - Switch::FR=7.311::MTBF=1.368e+5
    - Diode, Low Frequency::FR=0.002::MTBF=6.039e+8
    - Capacitor::FR=0.019::MTBF=5.3e+7
    - Capacitor::FR=0.253::MTBF=3.946e+6
    - Capacitor::FR=0.11::MTBF=9.127e+6
    - Resistor::FR=0.029::MTBF=3.412e+7
    - Resistor::FR=0.007::MTBF=1.499e+8
    - Diode, Low Frequency::FR=0.02::MTBF=5.032e+7
    - Capacitor::FR=0.006::MTBF=1.723e+8
  - Block::FR=13.264::MTBF=7.539e+4
    - Block::FR=11.413::MTBF=8.762e+4
    - Block::FR=1.851::MTBF=5.402e+5
      - Micro, Digital::FR=0.328::MTBF=3.052e+6

For Help, press F1 — Blocks: 5 — Comps: 47 — NUM

MIL-HDBK 217 Examples



5.1 MICROCIRCUITS, GATE/LOGIC ARRAYS AND MICROPROCESSORS

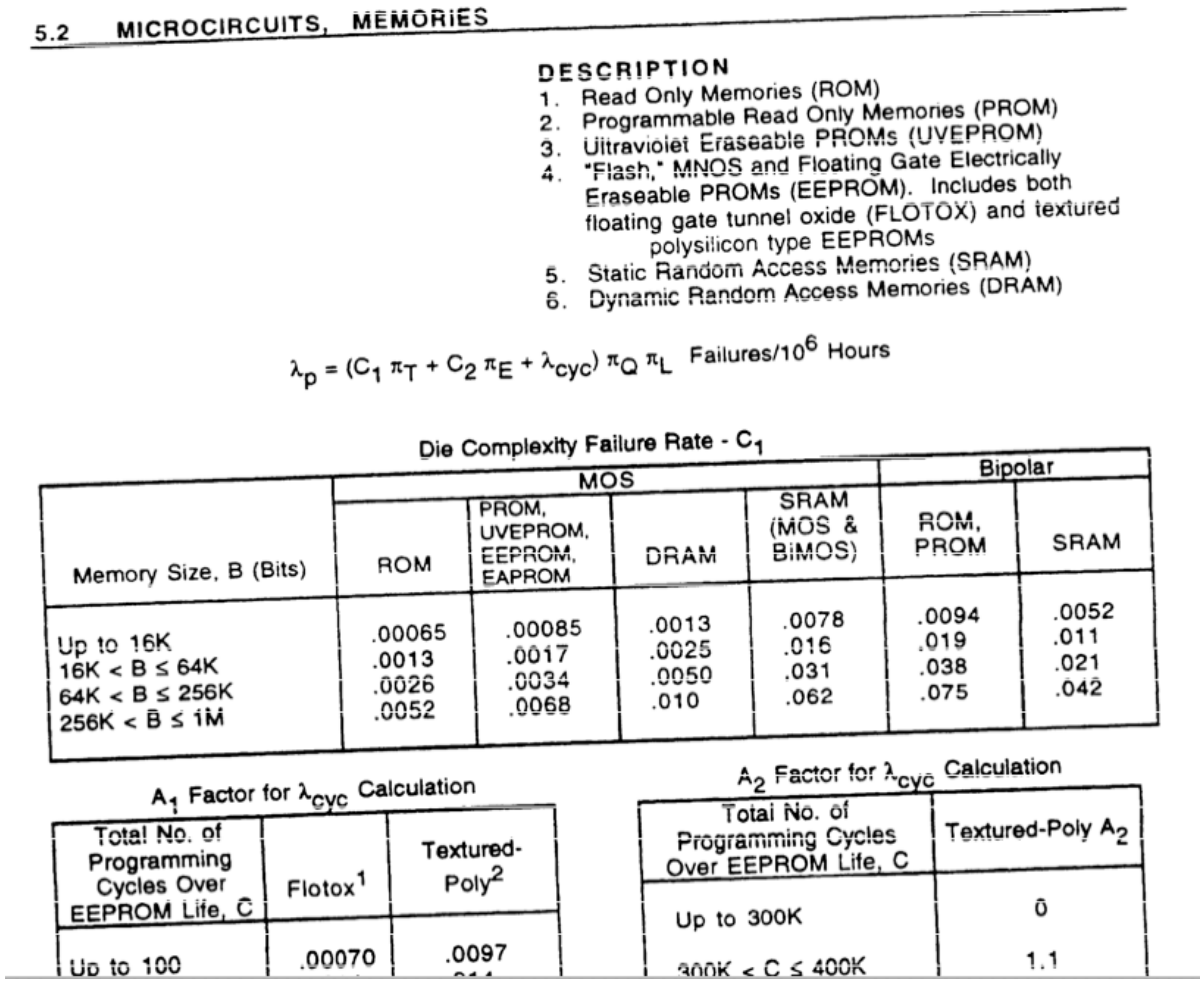
DESCRIPTION
1. Bipolar Devices, Digital and Linear Gate/Logic Arrays
2. MOS Devices, Digital and Linear Gate/Logic Arrays
3. Field Programmable Logic Array (PLA) and Programmable Array Logic (PAL)
4. Microprocessors

$$\lambda_p = (C_1\pi_T + C_2\pi_E)\,\pi_Q\pi_L \quad \text{Failures}/10^6 \text{ Hours}$$

Bipolar Digital and Linear Gate/Logic Array Die Complexity Failure Rate - C_1

Digital		Linear		PLA/PAL	
No. Gates	C_1	No. Transistors	C_1	No. Gates	C_1
1 to 100	.0025	1 to 100	.010	Up to 200	.010
101 to 1,000	.0050	101 to 300	.020	201 to 1,000	.021
1,001 to 3,000	.010	301 to 1,000	.040	1,001 to 5,000	.042
3,001 to 10,000	.020	1,001 to 10,000	.060		
10,001 to 30,000	.040				
30,001 to 60,000	.080				

MIL-HDBK 217 Examples



5.2 MICROCIRCUITS, MEMORIES

DESCRIPTION

1. Read Only Memories (ROM)
2. Programmable Read Only Memories (PROM)
3. Ultraviolet Eraseable PROMs (UVEPROM)
4. "Flash," MNOS and Floating Gate Electrically Eraseable PROMs (EEPROM). Includes both floating gate tunnel oxide (FLOTOX) and textured polysilicon type EEPROMs
5. Static Random Access Memories (SRAM)
6. Dynamic Random Access Memories (DRAM)

$\lambda_p = (C_1 \pi_T + C_2 \pi_E + \lambda_{cyc}) \pi_Q \pi_L$ Failures/10^6 Hours

Die Complexity Failure Rate - C_1

	MOS				Bipolar	
Memory Size, B (Bits)	ROM	PROM, UVEPROM, EEPROM, EAPROM	DRAM	SRAM (MOS & BiMOS)	ROM, PROM	SRAM
Up to 16K	.00065	.00085	.0013	.0078	.0094	.0052
16K < B ≤ 64K	.0013	.0017	.0025	.016	.019	.011
64K < B ≤ 256K	.0026	.0034	.0050	.031	.038	.021
256K < B ≤ 1M	.0052	.0068	.010	.062	.075	.042

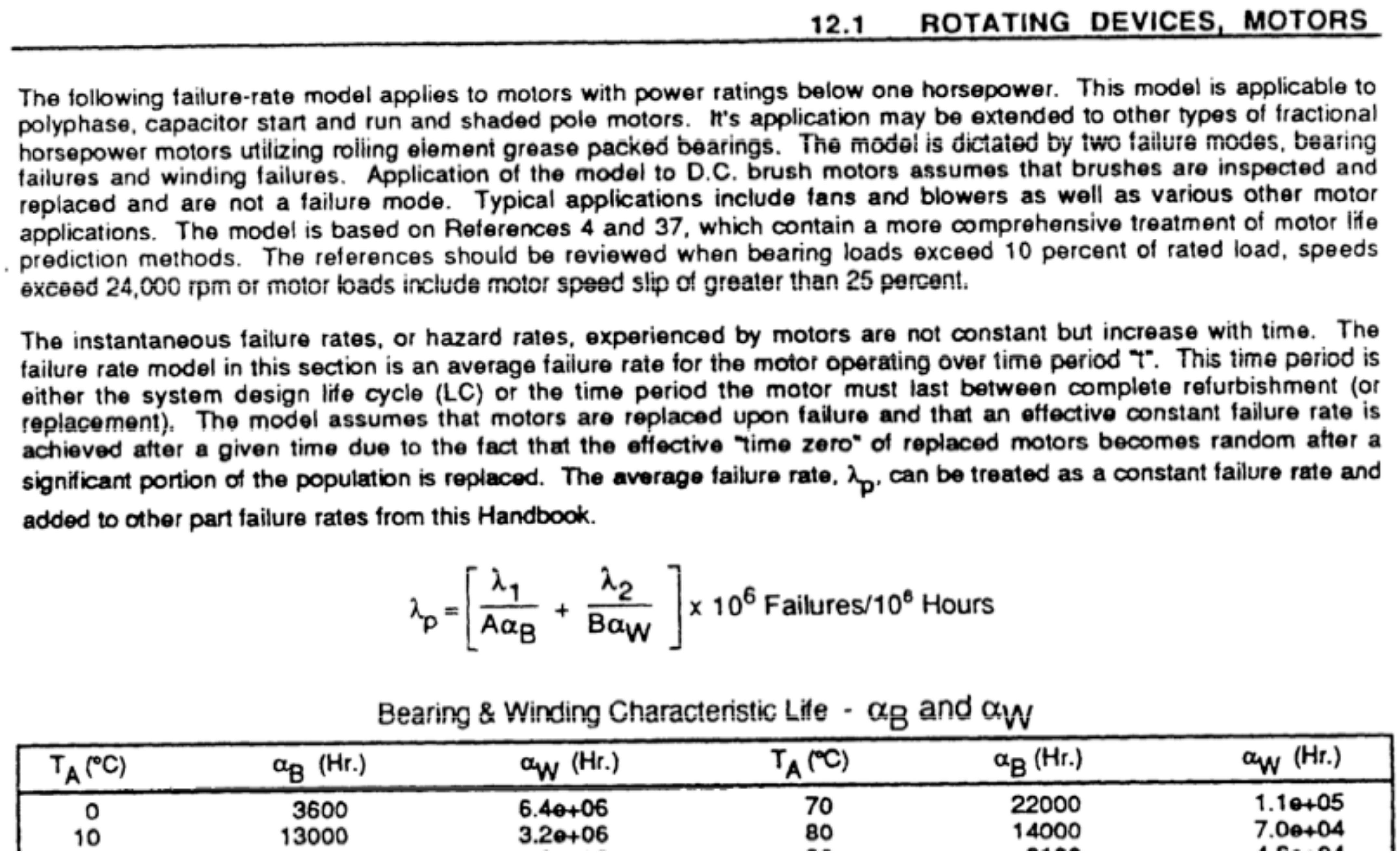
A_1 Factor for λ_{cyc} Calculation

Total No. of Programming Cycles Over EEPROM Life, C	Flotox[1]	Textured-Poly[2]
Up to 100	.00070	.0097

A_2 Factor for λ_{cyc} Calculation

Total No. of Programming Cycles Over EEPROM Life, C	Textured-Poly A_2
Up to 300K	0
300K < C ≤ 400K	1.1

Mechanical Parts



The following failure-rate model applies to motors with power ratings below one horsepower. This model is applicable to polyphase, capacitor start and run and shaded pole motors. It's application may be extended to other types of fractional horsepower motors utilizing rolling element grease packed bearings. The model is dictated by two failure modes, bearing failures and winding failures. Application of the model to D.C. brush motors assumes that brushes are inspected and replaced and are not a failure mode. Typical applications include fans and blowers as well as various other motor applications. The model is based on References 4 and 37, which contain a more comprehensive treatment of motor life prediction methods. The references should be reviewed when bearing loads exceed 10 percent of rated load, speeds exceed 24,000 rpm or motor loads include motor speed slip of greater than 25 percent.

The instantaneous failure rates, or hazard rates, experienced by motors are not constant but increase with time. The failure rate model in this section is an average failure rate for the motor operating over time period "t". This time period is either the system design life cycle (LC) or the time period the motor must last between complete refurbishment (or replacement). The model assumes that motors are replaced upon failure and that an effective constant failure rate is achieved after a given time due to the fact that the effective "time zero" of replaced motors becomes random after a significant portion of the population is replaced. The average failure rate, λ_p, can be treated as a constant failure rate and added to other part failure rates from this Handbook.

$$\lambda_p = \left[\frac{\lambda_1}{A\alpha_B} + \frac{\lambda_2}{B\alpha_W}\right] \times 10^6 \text{ Failures}/10^6 \text{ Hours}$$

Bearing & Winding Characteristic Life - α_B and α_W

T_A (°C)	α_B (Hr.)	α_W (Hr.)	T_A (°C)	α_B (Hr.)	α_W (Hr.)
0	3600	6.4e+06	70	22000	1.1e+05
10	13000	3.2e+06	80	14000	7.0e+04

Telcordia (Bellcore) SR-332 / TR-332

- Reliability prediction model evolved from telecom industry equipment experience
- First developed by Bellcore Communications research, based on MIL217
    - Lower and more conservative values for MTBF
- Availability with relation to failures per billion component operation hours
- Factors: Static failure rate of basic component, quality, electrical load, temperature
- Often observed that calculations are more optimistic than in MIL 217
- Requires fewer parameters than MIL 217 (four unified quality levels)
- Different components supported - Printed circuit boards, lasers, tubes, magnetic memories (217) vs. gyroscopes, batteries, heaters, coolers, computer systems

Telcordia (Bellcore) SR-332 / TR-332

- Three methods with empirically proven data
  - Method 1: Part count approach, no field data available
    - Supports first year multiplier for infant mortality modeling
  - Method 2: Method 1 extended with laboratory test data
    - Supports Bayes weighting procedure for burn-in results
  - Method 3: Method 2 including field failure tracking
    - Different Bayes weighting based on the similarity of the field data components

Quality Factor

$$\lambda = \pi_g \lambda_b \pi_S \pi_t$$

Temperature Factor, default 40 deg

Environmenta
l Factor

Base Failure Rate from Transistor Count

MIL-HDBK-217 vs. Telcordia

- Often observed that Telcordia calculations are much more optimistic
- Telecordia requires fewer parameters for components, has additional capabilities for considering burn-in data, laboratory test data, and field data
  - Helps to calculate failure rates based on historical data
  - Can model infant mortality effects
- Telcordia uses *Failures-In-Time (FIT)* - per billion hours, MIL uses *failures per million hours*
- Operating environments in Telcordia are limited, since only designed for telco users
- Telcordia quality levels are simplified

PRISM

- PRISM released by Reliability Analysis Center (RAC) in 2000
- Can update predictions based on test data
- Addresses factors such as development process robustness and thermal cycling
- Basic consideration of system software part
- Seen as industry replacement for MIL 217
- Factors in the model: Initial failure rate, infant mortality, environment, design processes, reliability growth, manufacturing processes, item management processes, wear-out processes, software failure rate, ...
- Quantitative factors are determined through an interactive process

RIAC 217Plus

- Predictive models for integrated circuits (Hermetic and non-hermetic), diodes, transistors, capacitors, resistors and thyristors
- Based on over 2x1013 operation hours and 795,000 failures
- Includes also software model
- System level modeling part for non-component variables
- Failure rate contribution terms: Operating, non-operating, cycling, induced
- Process grading factors: Parts, Design, Manufacturing, System Management, Wear-out, Induced and No Defect Found
    - Model degree of action for failure mitigation
- Still only series calculation, no active-parallel or standby redundancy support

Other Sources

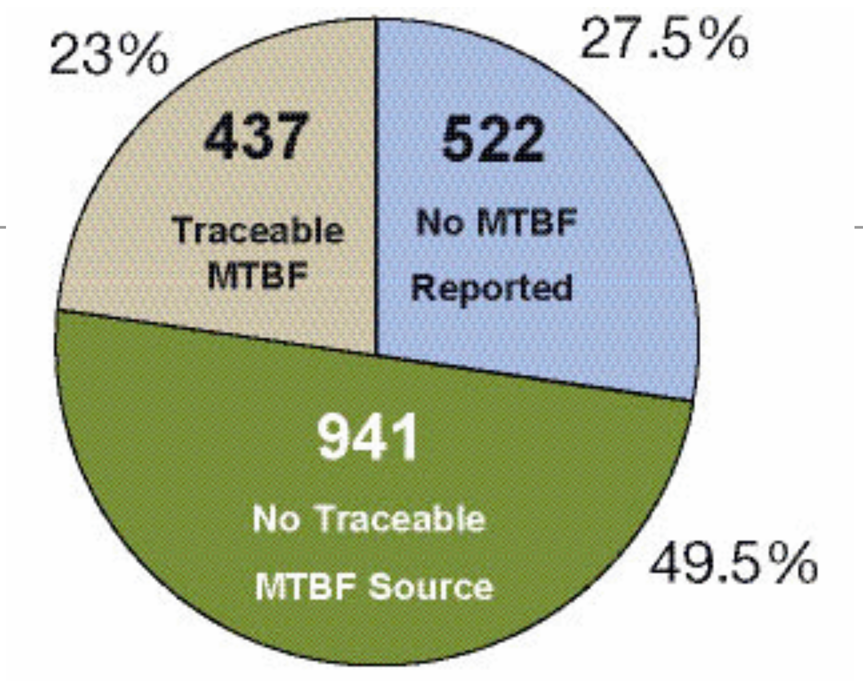
- IEC TR62380 - Technical report of the International Electrotechnical Commission, considers thermal behavior and system under different environmental conditions
- IEC 61709 - Parts count approach, every user relies on own failure rates
- NPRD-95 - Failure rates for > 25.000 mechanical and electromechanical parts, collected from 1970s till 1994
- HRD5 - Handbook for reliability data of electronic components used in Telco industry, developed by British Telecom, similar to MIL 217
- VZAP95 - Electrostatic discharge susceptibility for 22.000 devices (e.g. microcircuits)
- RDF 2000 - Different approach for failure rates
  - Cycling profiles - power on/off, temperature cycling
  - Demands complex information (e.g. thermal expansion, transistor technology, package related failure rates, working time ratio)

Other Sources

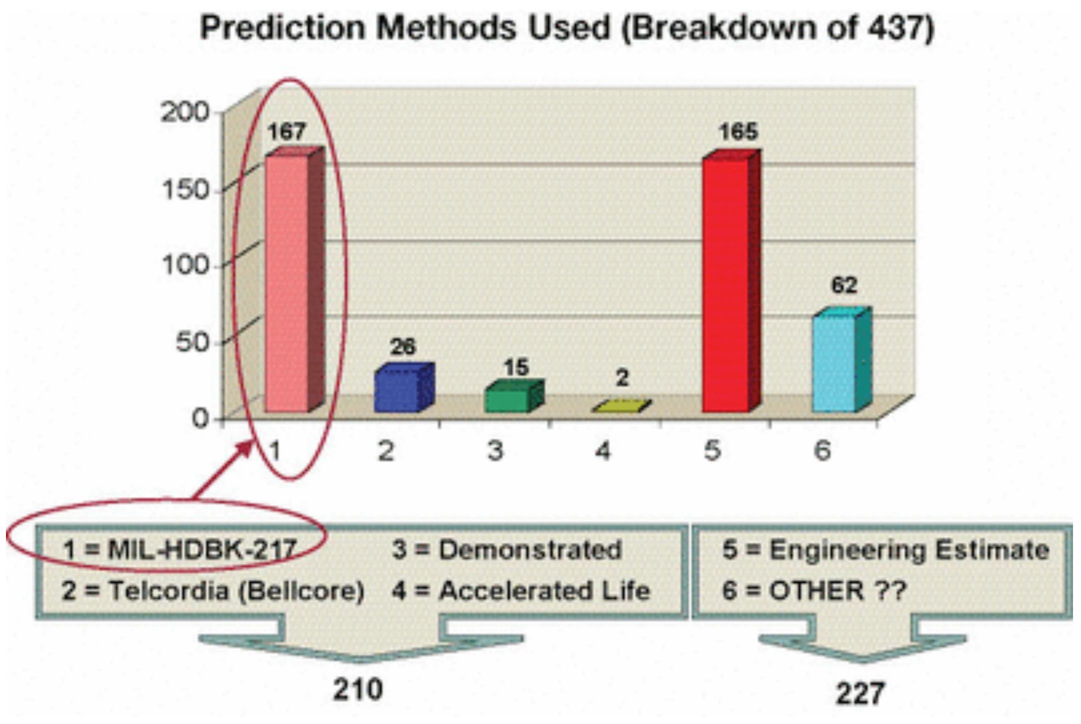
- WASH-1400 (NUREG-75/015) - „Reactor Safety Study - An Assessment of Accident Risks in U.S. Commercial Nuclear Power Plants“ 1975
- IEEE Standard 500
- Government-Industry Data Exchange Program (GIDEP)
- NUREG/CR-1278; „Handbook of Human Reliability Analysis with Emphasis on Nuclear Power Plant Applications;“ 1980
- SAE model - Society of automotive engineers, similar equations to MIL 217
    - Modifying factors are are specific for automotive industry
    - Examples: Component composition, ambient temperature, location in the vehicle

Comparison

- Naval Surface Warfare Center (NSWC) Crane Division established working group for modification factors of MIL 217
- Industry survey - Reliability prediction methods used
  - From survey of 1900 component tests, less than 25% had used prediction models
  - 38% of them used MIL 217

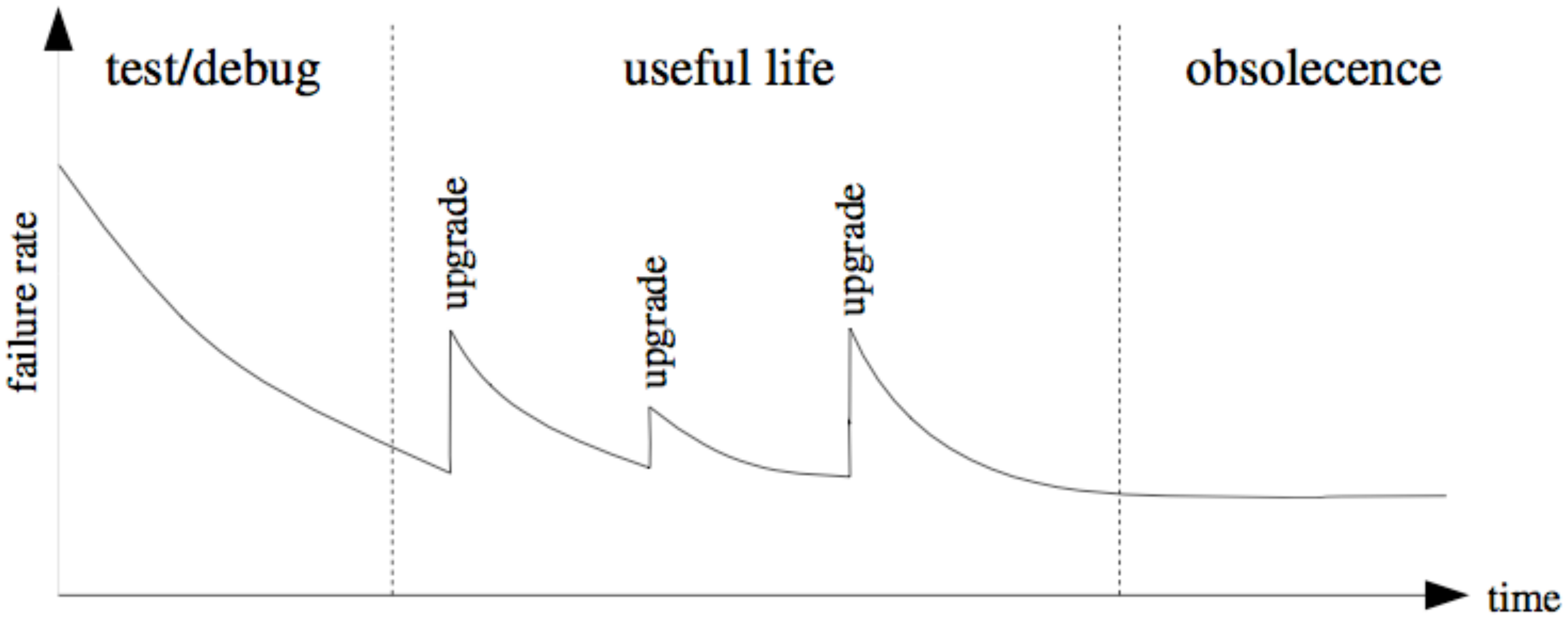


1,900 COTS Items (1,378 with MTBF)



(C)VMECritical.com

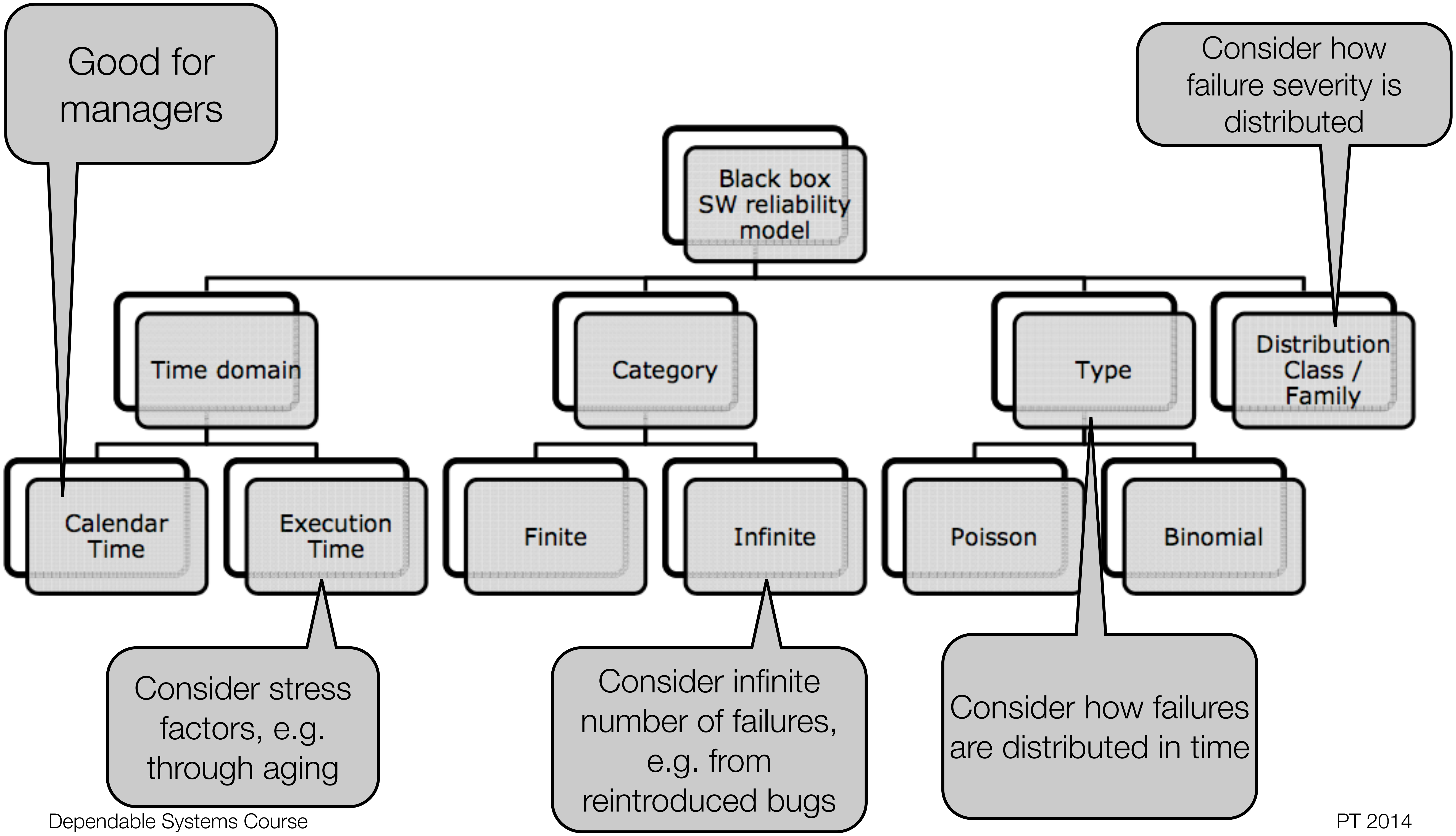
Software - A Different Story



Software Reliability Assessment

- Software bugs are permanent faults, but behave as transient faults
  - Activation depends on software usage pattern and input values
  - Timing effects with parallel or distributed execution
  - Software aging effects
- Fraction of system failures reasoned by software increased in the last decades
- Possibilities for software reliability assessment
  - **Black box models** - No details known, observations from testing or operation
    - Reliability growth models
  - **Software metric models** - Derive knowledge from static code properties

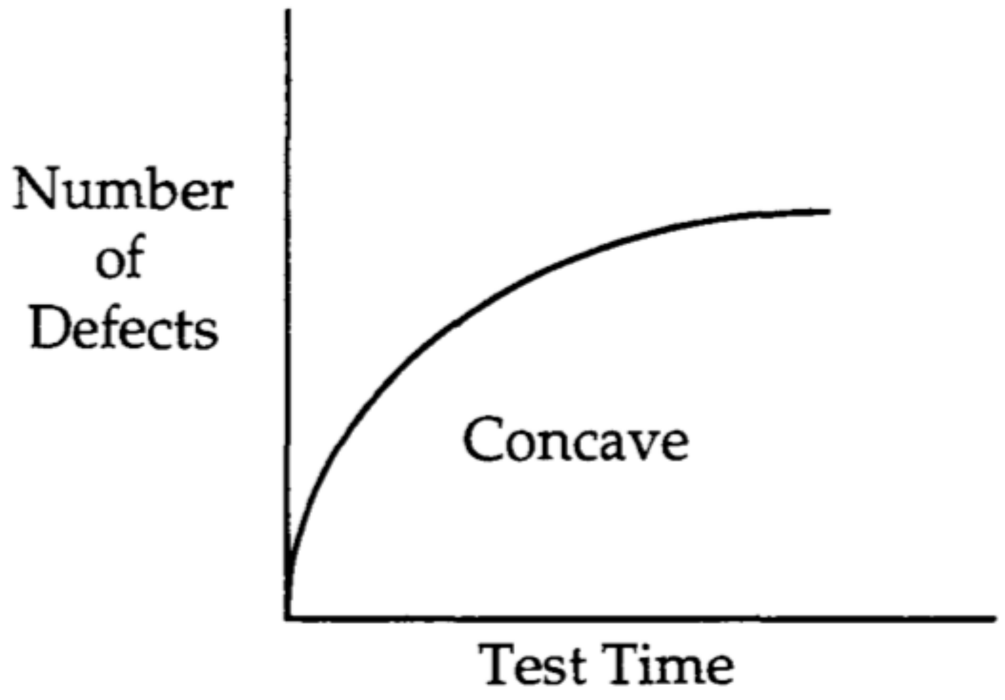
Dimensions of Black Box Models (Musa et al.)

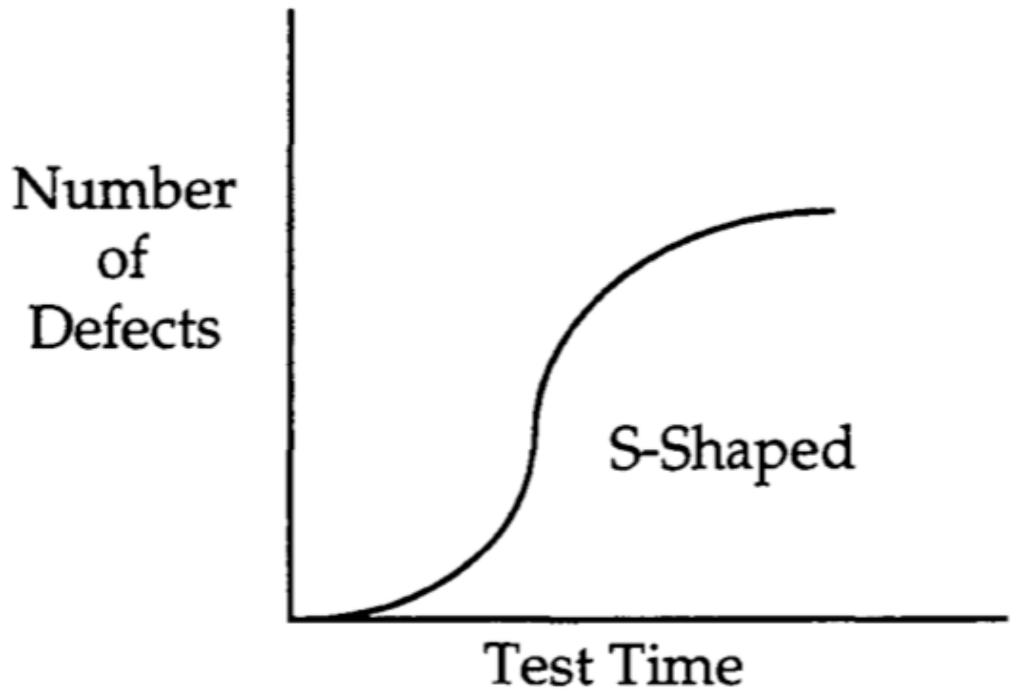

Good for managers
Consider how failure severity is distributed
Black box SW reliability model
Time domain
Category
Type
Distribution Class / Family
Calendar Time
Execution Time
Finite
Infinite
Poisson
Binomial
Consider stress factors, e.g. through aging
Consider infinite number of failures, e.g. from reintroduced bugs
Consider how failures are distributed in time

Software Reliability Growth Models

- Fundamental concept for software reliability modeling
- Basic assumptions
  - Faults are removed **immediately** after being discovered
  - Failure frequency tends to decrease over time, so reliability increases
  - Software reliability can be predicted by interpolating the progress of testing-based or bug fixing-based reliability improvement with a growth model
- Typical question for the model: *When will we arrive at some reliability target?*
  - Supports planning of testing and service operation efforts
- Hundreds of proposed models, only a few tested with sufficient field data
  - Collect software failure data, analyze for distribution characteristics, choose model
  - Active empirical research in software engineering

Reliability Growth Model Analysis by Tandem [Wood]





- Concave models: Fault detection rate decreases over time with repair activities
- S-Shaped models: Early testing is more efficient than later testing
  - Ramp-up phase were error detection rate increases
  - Example: Early phase detects faults that prevent the application from running

Software Reliability Growth Models

- Classification by Singpurwalla and Wilson (1994)
  - **Type I: Model time between successive failures**
    - Should get longer as faults are removed from the software
    - Time is assumed to follow a function, related to number of non-fixed faults
    - Type I-1: Based on failure rate (Example: Jelinski-Moranda Model (1972))
    - Type I-2: Model inter-failure time based on previous inter-failure times
  - **Type II: Counting process to model the number of failures per time unit**
    - As faults are removed, observed number of failures per unit should decrease
    - Typically derivation of remaining number of defects / failures
    - Example: Basic execution time model by Musa (1984)

Jelinksi-Moranda Model

- Finite number of bugs, inter-failure time has exponential distribution
- Assumptions: Faults are equal, occur randomly and independent from each other, detection rate is constant, repairs are perfect, repair time is zero, software operation during test is the same as in production use
- Constant function for failure rate in the i-th time interval, based on constant factor and number of bugs:

$$\lambda_i = \Phi * N_i$$

- Estimation of λ_i : Number of failures in interval i / length of interval i
    - Failure rate estimation based on testing efforts done so far
    - Allows to reason about (remaining) number of dormant faults in the software

Example: Jelinski-Moranda

- Time between failures 7.5 min at start, 2h after investigated testing / fixing period
- 75 faults were removed
- Looking for number of initial faults and proportionality constant

$$\lambda_0 = \Phi * N \qquad \lambda_{end} = \Phi * (N - 75) \qquad \lambda_0 \approx \frac{1}{0,125}$$

$$\lambda_{end} \approx \frac{1}{2} \qquad \frac{\lambda_0}{\lambda_{end}} = \frac{N}{N - 75} = 16 \qquad N = 80, \Phi = 0,1$$

- Reliability after 10h of operation with the improved version of the software

$$R(t_i) = e^{-\lambda_i t_i} \qquad R(10h) = e^{-\lambda_{end} 10h} = 7\%$$

Basic Execution Time Model (Musa)

- Another reliability growth model
  - Determine cumulative number of failures by time i
  - Start with estimated number of initial faults, get number of corrected fault after τ
  - Fundamental assumption that correction of faults does not introduce new ones
- One of the first models to rely on execution time, instead of calendar time
  - Resources limit execution time spent
  - Expression in terms of CPU time can indicate ‚stress' on software
- Random process that represents the number of failures experienced at some point
  - Exponential distribution of resulting failure intensity function
  - Failure **intensity changes** with a **constant rate**, depending on the number of observed failures

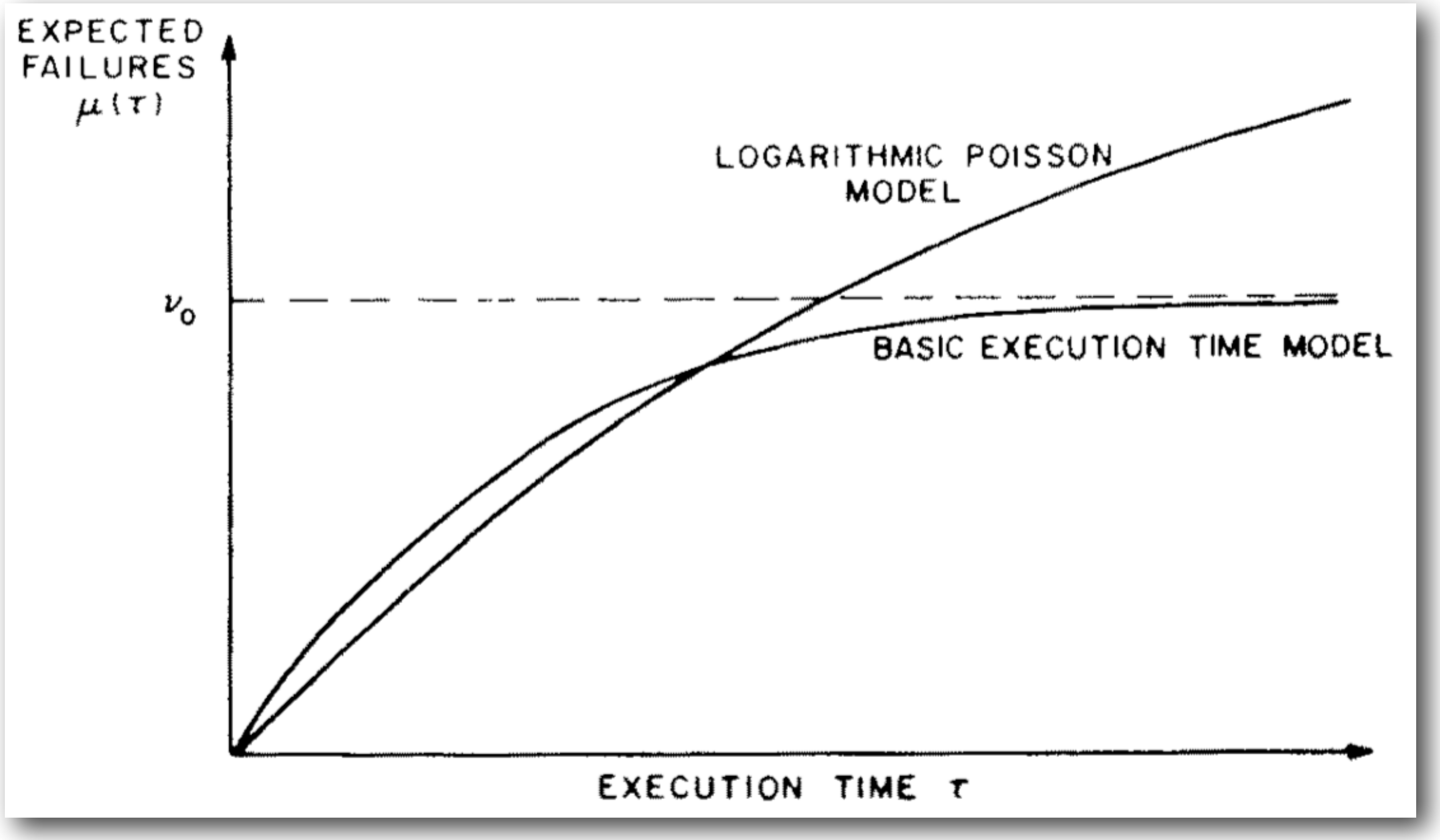
Basic Execution Time Model (Musa)

- λ - Failure rate (number of failures per time unit)
- τ - Execution time (time since the program is running)
- μ - Mean number of expected failures in an execution time interval
- λ_0 - Initial failure rate at the start of execution
- ν_0 - Total number of failures over an infinite time period

$$\mu(\tau) = \nu_0(1 - e^{-\frac{\lambda_0}{\nu_0}\tau})$$

$$\lambda(\tau) = \lambda_0 e^{-\frac{\lambda_0}{\nu_0}\tau}$$

- Assumed maximum number of failures ν_0 leads to decreasing μ over time



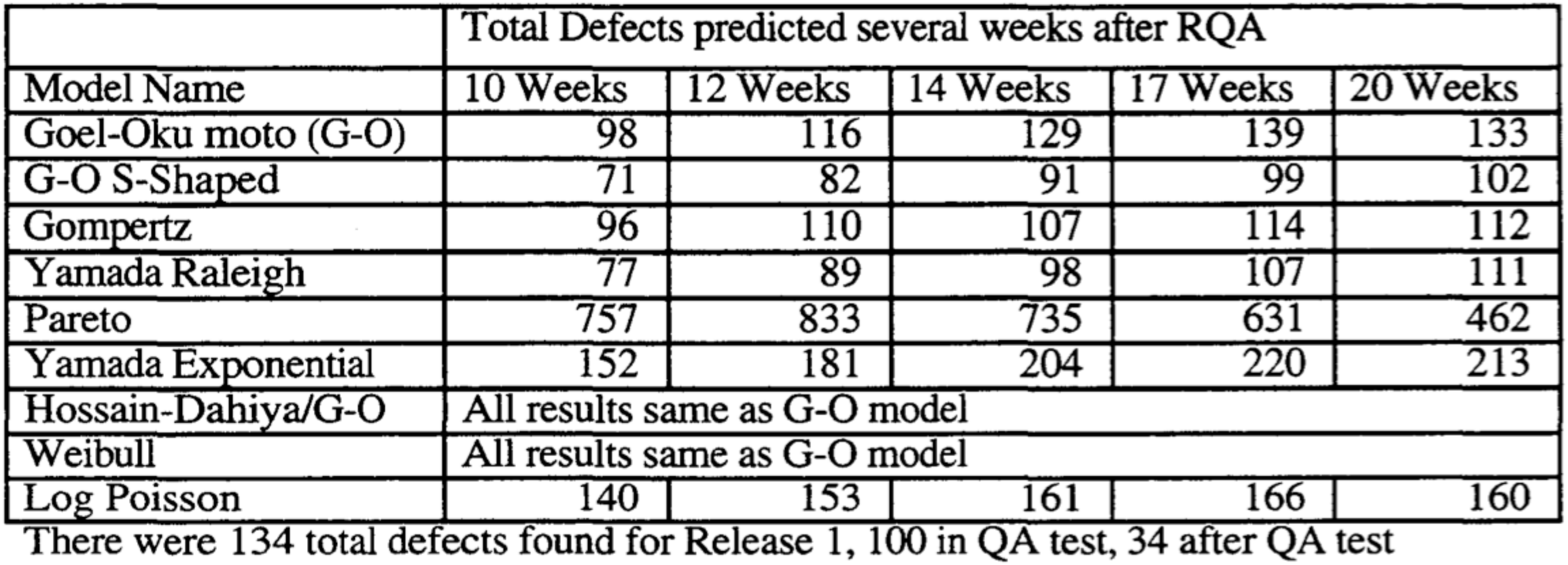
Reliability Growth Model Analysis by Tandem [Wood]

- Technical report 96.1 by Tandem, 1996
- Analysis of 9 different growth models in comparison with real data
  - Methodology needs to be adjusted with every new release
  - Execution time is the best measure for test amount, calendar time does not work
  - Weekly data was sufficient for the prediction models
  - Simple (exponential) models outperformed others in stability and predictive ability
- Failure count estimation relevant for planning amount of testing and customer support levels

Reliability Growth Model Analysis by Tandem [Wood]

- Reality check for typical assumptions
  - *„Repair is perfect“*: New defects are less likely to be detected, since regression is not as comprehensive as the original tests
  - *„Each unit of time is equivalent“*: Corner tests are more likely to find defects, test re-runs for fixed bugs are less likely to find new defects
  - *„Tests represent operational profile“*: Size and transaction volume of production system is typically not reproducible
  - *„Failures are independent“*: Reasonable, except for badly tested portions of code
- During testing, the model should predict additional test effort for some quality level
- After test, the model should predict the number of remaining defects in production
- Most models do not become stable until about 60% of the tests

Reliability Growth Model Analysis by Tandem [Wood]



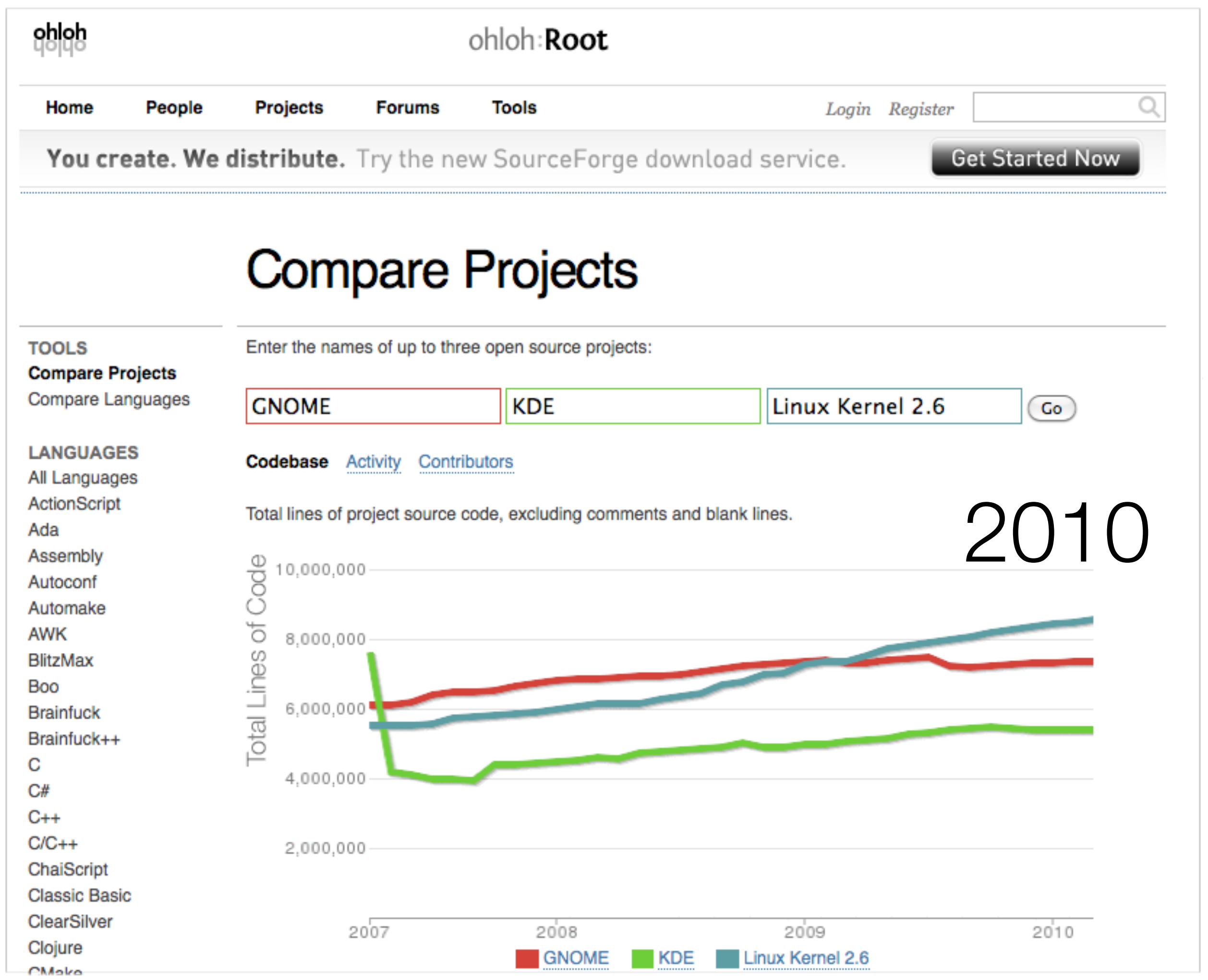
	Total Defects predicted several weeks after RQA				
Model Name	10 Weeks	12 Weeks	14 Weeks	17 Weeks	20 Weeks
Goel-Oku moto (G-O)	98	116	129	139	133
G-O S-Shaped	71	82	91	99	102
Gompertz	96	110	107	114	112
Yamada Raleigh	77	89	98	107	111
Pareto	757	833	735	631	462
Yamada Exponential	152	181	204	220	213
Hossain-Dahiya/G-O	All results same as G-O model				
Weibull	All results same as G-O model				
Log Poisson	140	153	161	166	160

There were 134 total defects found for Release 1, 100 in QA test, 34 after QA test

- Most models become stable at the same time as the G-O model
- S-shaped models tend to underestimate the number of effects
- G-O model (which is the Musa model) is the best choice for Tandem

White Box Approach - Software Metric Models

- Larger software is more complex, which brings more dormant faults
- Idea: Rank software reliability by complexity measures
    - ... for the code itself
    - ... for the development process
- Complexity metrics should rank degree of difficulty for design and implementation of program structures, algorithms, and communication schemes
- Hundreds of different metrics proposed, variety of software tools
- Most basic approach - lines of code (LOC)
    - Industry standard from 0,3 to 2 bugs per 1000 lines of code
- Better data for structured analysis available with large open source projects


ohloh

ohloh:Root

Home People Projects Forums Tools Login Register

You create. We distribute. Try the new SourceForge download service. Get Started Now

Compare Projects

TOOLS
Compare Projects
Compare Languages

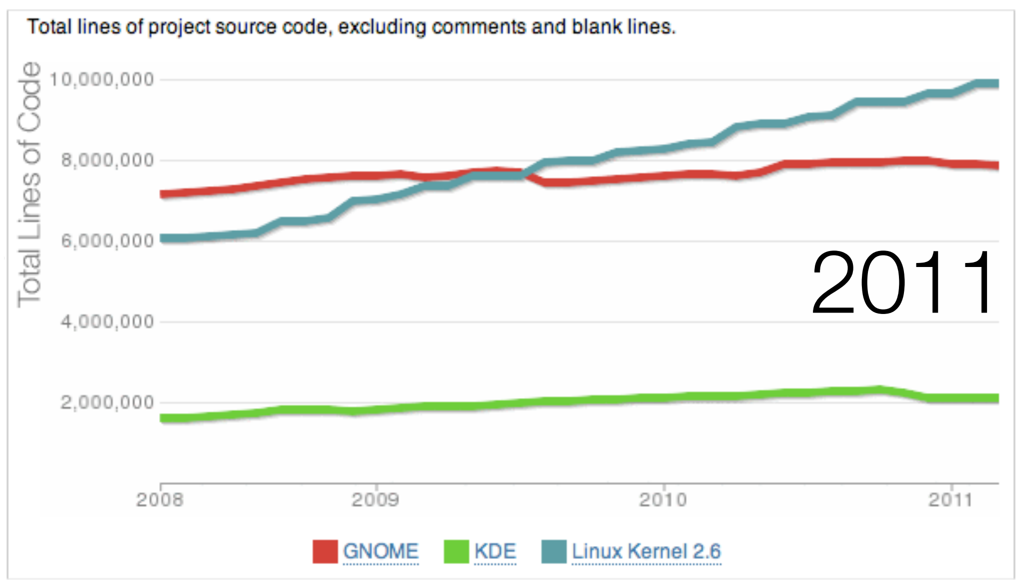
LANGUAGES
All Languages
ActionScript
Ada
Assembly
Autoconf
Automake
AWK
BlitzMax
Boo
Brainfuck
Brainfuck++
C
C#
C++
C/C++
ChaiScript
Classic Basic
ClearSilver
Clojure

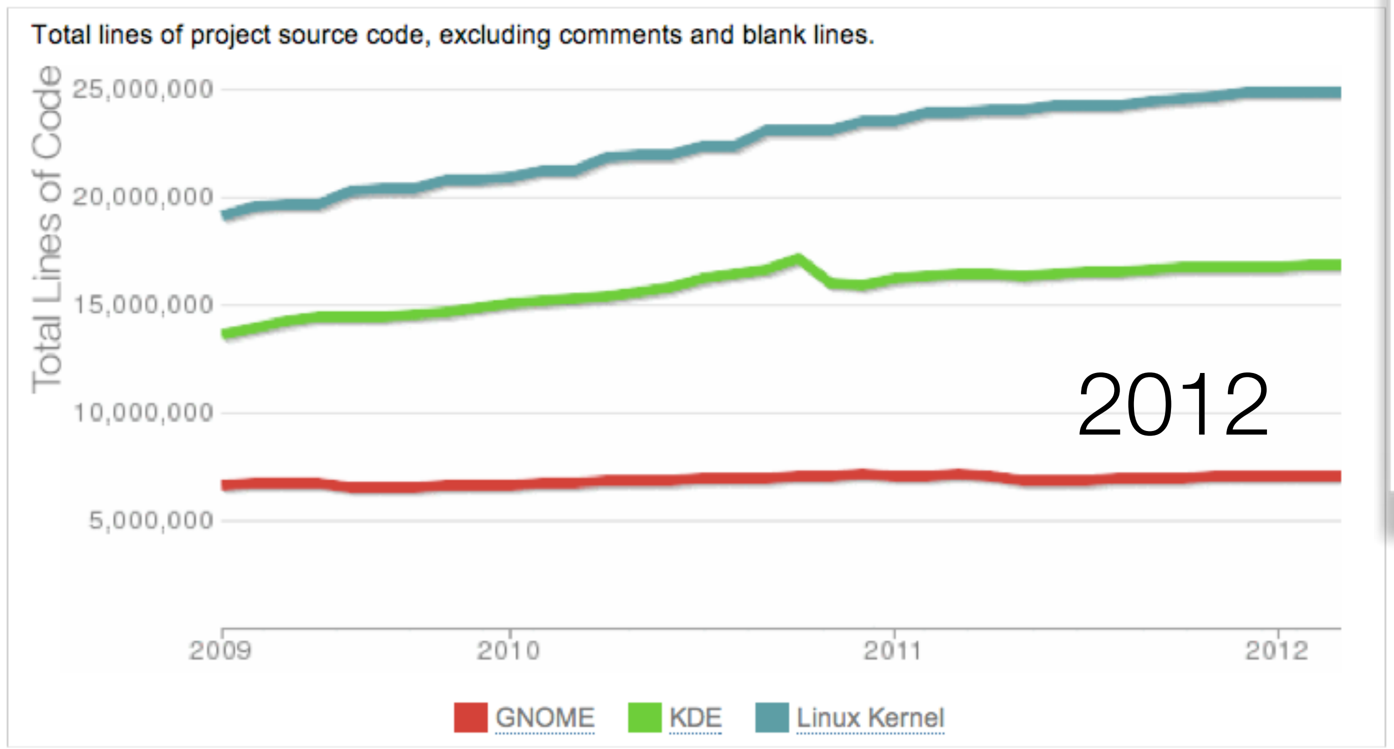
Enter the names of up to three open source projects:

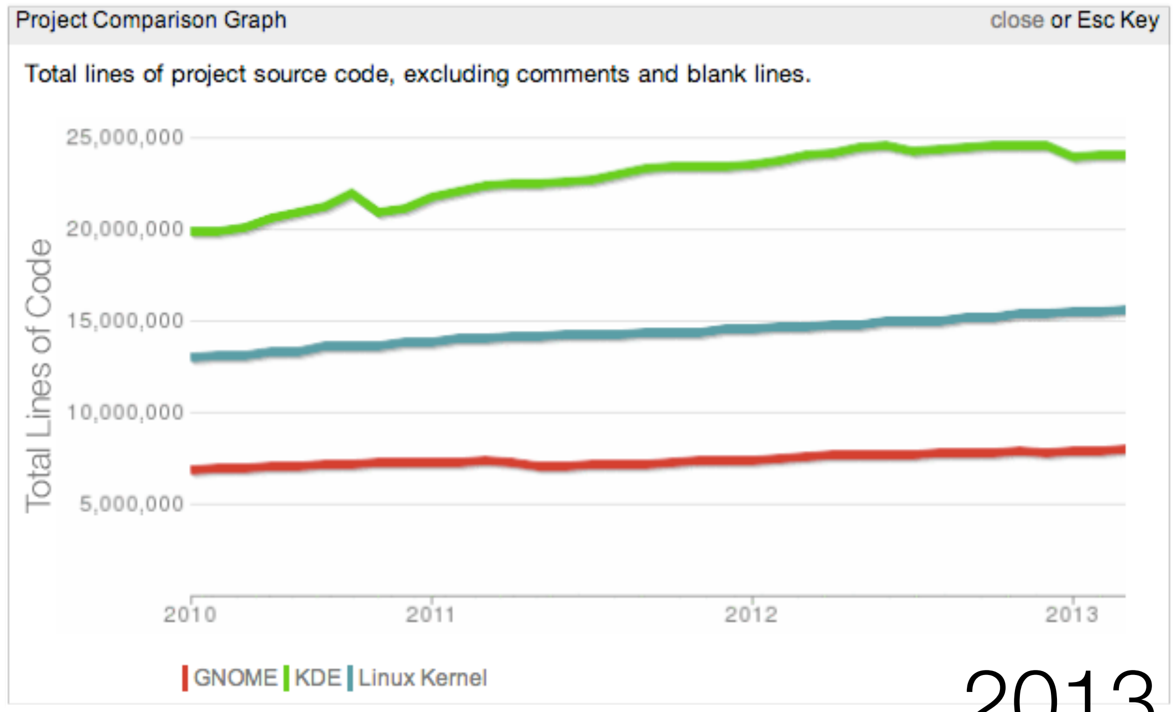
GNOME | KDE | Linux Kernel 2.6 | Go

Codebase Activity Contributors

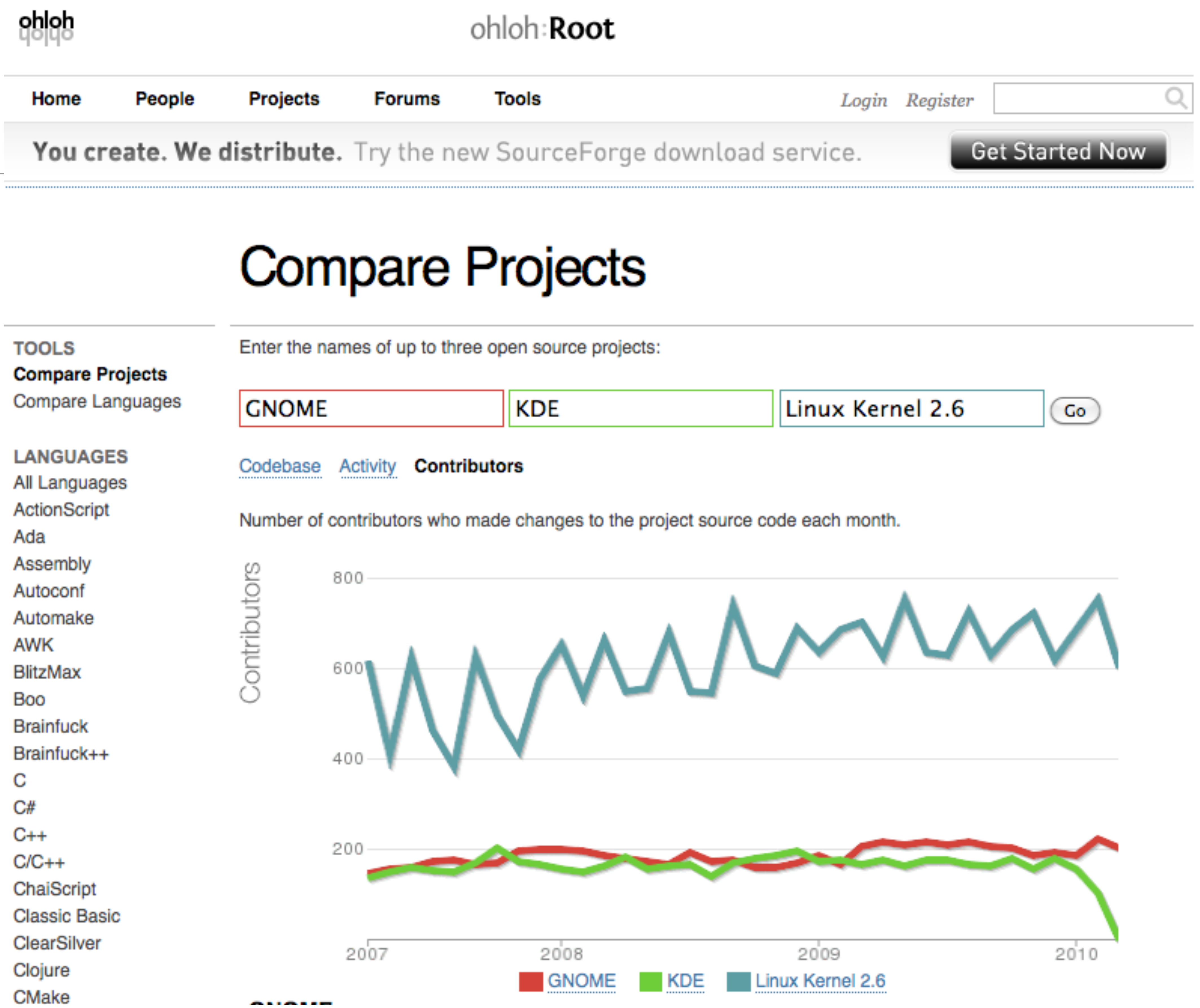
Total lines of project source code, excluding comments and blank lines.


Total lines of project source code, excluding comments and blank lines.


Total lines of project source code, excluding comments and blank lines.


Project Comparison Graph close or Esc Key

Total lines of project source code, excluding comments and blank lines.



ohloh:Root

Home People Projects Forums Tools Login Register

You create. We distribute. Try the new SourceForge download service. Get Started Now

Compare Projects

TOOLS
Compare Projects
Compare Languages

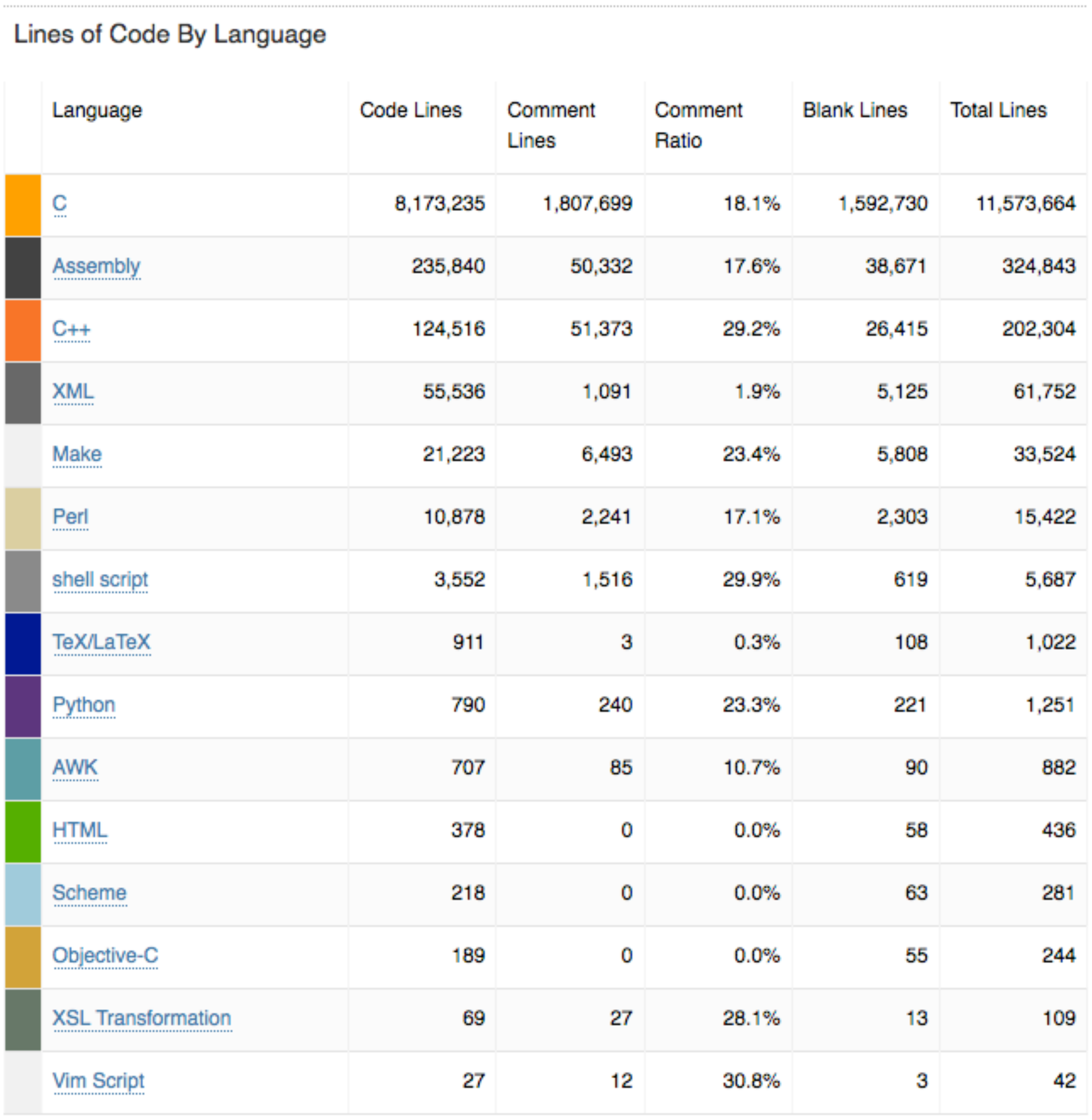
LANGUAGES
All Languages
ActionScript
Ada
Assembly
Autoconf
Automake
AWK
BlitzMax
Boo
Brainfuck
Brainfuck++
C
C#
C++
C/C++
ChaiScript
Classic Basic
ClearSilver
Clojure
CMake

Enter the names of up to three open source projects:

GNOME | KDE | Linux Kernel 2.6 | Go

Codebase Activity **Contributors**

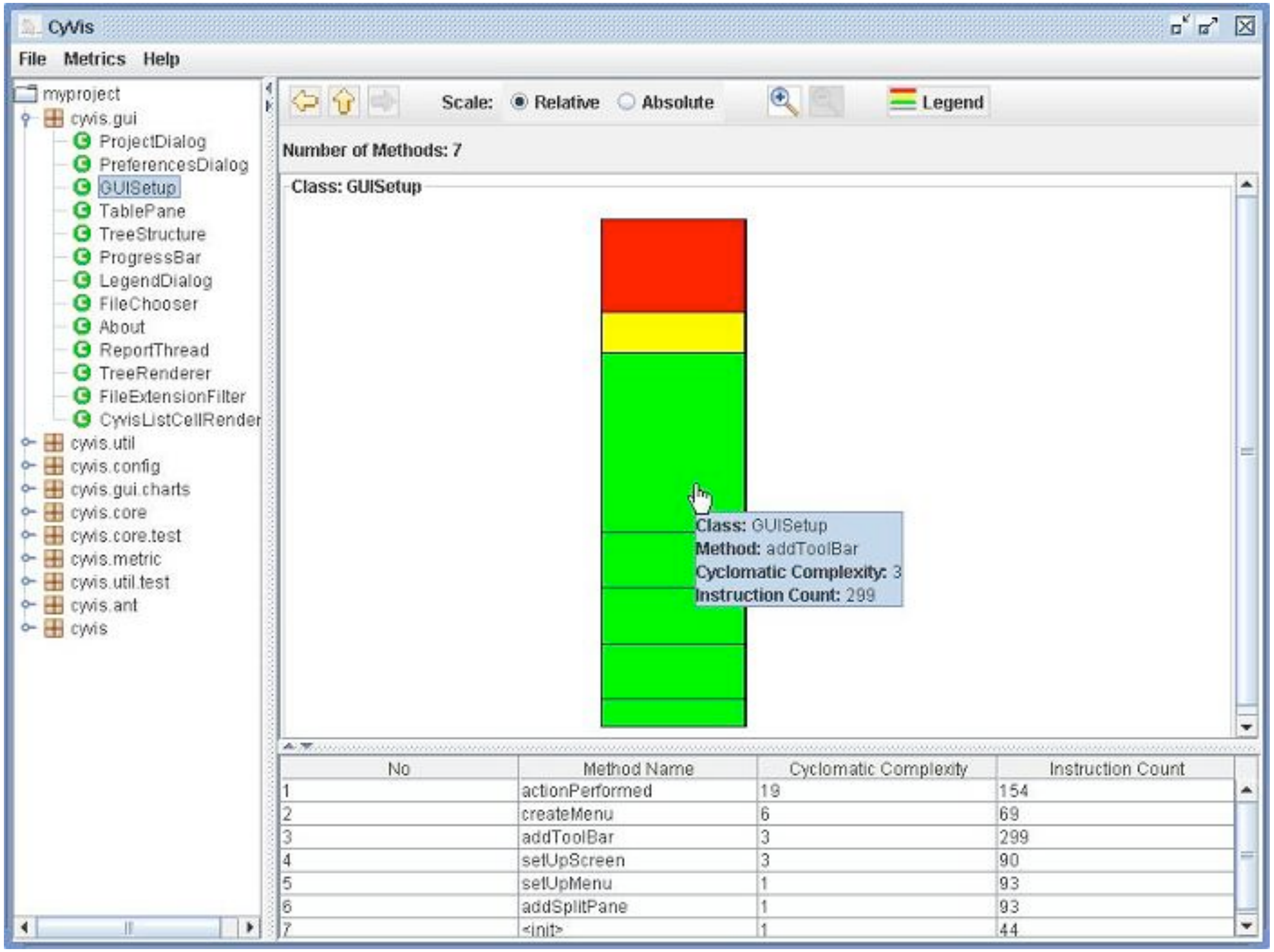
Number of contributors who made changes to the project source code each month.



Lines of Code By Language

	Language	Code Lines	Comment Lines	Comment Ratio	Blank Lines	Total Lines
🟧	C	8,173,235	1,807,699	18.1%	1,592,730	11,573,664
⬛	Assembly	235,840	50,332	17.6%	38,671	324,843
🟧	C++	124,516	51,373	29.2%	26,415	202,304
⬛	XML	55,536	1,091	1.9%	5,125	61,752
⬜	Make	21,223	6,493	23.4%	5,808	33,524
🟨	Perl	10,878	2,241	17.1%	2,303	15,422
⬛	shell script	3,552	1,516	29.9%	619	5,687
🟦	TeX/LaTeX	911	3	0.3%	108	1,022
🟪	Python	790	240	23.3%	221	1,251
🟦	AWK	707	85	10.7%	90	882
🟩	HTML	378	0	0.0%	58	436
🟦	Scheme	218	0	0.0%	63	281
🟨	Objective-C	189	0	0.0%	55	244
⬛	XSL Transformation	69	27	28.1%	13	109
⬜	Vim Script	27	12	30.8%	3	42

Example: CyVis Tool for Java



Scale: [✓ Relative] Absolute

Number of Methods: 7

Class: GUISetup

Class: GUISetup
Method: addToolBar
Cyclomatic Complexity: 3
Instruction Count: 299

No	Method Name	Cyclomatic Complexity	Instruction Count
1	actionPerformed	19	154
2	createMenu	6	69
3	addToolBar	3	299
4	setUpScreen	3	90
5	setUpMenu	1	93
6	addSplitPane	1	93
7	<init>	1	44

```
x = x + 1;
```

$N_{OP}=n_{OP}=3$
$N_{OD}=3$
$n_{OD}=2$

Halstead Metric

- Statistical approach - Complexity is related to number of operators and operands
- Only defined on method level, OO code analysis must consider additional structure
  - Program **length** *N = Number of operators* N_{OP} + *total number of operands* N_{OD}
  - **Vocabulary** size *n = Number of unique operators* n_{OP} + *number of unique operands* n_{OD}
  - Program **volume** $V = N * \log_2(n)$
    - Describes size of an implementation by vocabulary and (mainly) by program length
    - In-place changes are ok ...
  - **Difficulty** level $D = (n_{OP} / 2) * (N_{OD} / n_{OD})$, program level $L = 1 / D$
    - Error proneness is proportional to number of unique operators
      -> since most languages only offer a few, this should be small
    - Also proportional to the level of operand reuse through the code

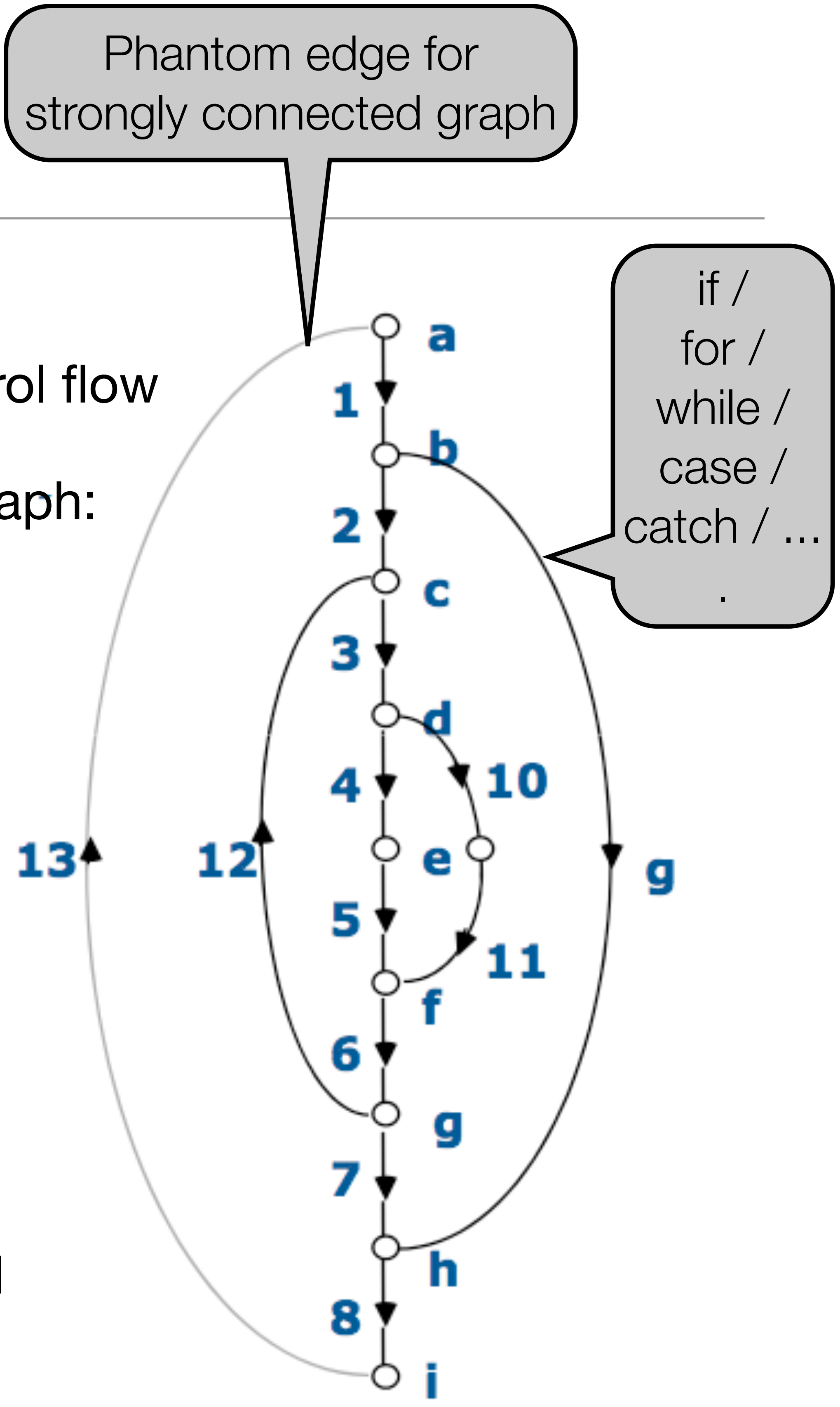
Halstead Metric

- **Effort** to implement / understand the program $E = V * D$
  - Depends on volume and difficulty level of the analyzed software
  - Describes mental effort to re-create the software (only implementation)
  - Questioned by some researchers
- **Time** in seconds to implement / understand the program $T = E / 18$
  - Stroud number 18 by John Stroud, psychologist
  - -> humans can detect between 5 and 20 discrete events per second
  - Widely ignored in software metric tools
- Number of delivered **bugs** $B = V / 3000$
  - Bug count correlates with the complexity / effort to understand the software
  - Statistically proven for object-oriented software

Cyclomatic Complexity

Thomas McCabe‘s cyclomatic complexity (1976)

- Measure based on graph representation of the control flow
- *Cyclomatic number v(G)* for a strongly connected graph:
  - Number of directed edges - number of vertexes + number of connected components
  - Expresses number of independent conditional paths in the control flow
- No consideration of data structure complexity, data flow, or module interface design
- Goal should be max. 15 for functions, 100 for files
- Studies showed correlation between fault count and cyclomatic number


Phantom edge for strongly connected graph
if / for / while / case / catch / ... .
a
1
b
2
c
3
d
10
4
e
5
11
f
6
g
7
h
8
i
12
13
g

State-Based Software Model (Cheung)

- Software modules N_1 ... N_m
- States C and F denote correct and incorrect output
- Each module N_i computes the right result with probability R_i
- P_{ij} denotes probability of control transition from module i to j, allows DTMC model
- System produces correct output if N_n is reached from N_1, and N_n produces C

$$
\begin{array}{c|cccccccc}
 & C & F & N_1 & N_2 & \dots & N_j & \dots & N_n \\
\hline
C & \mathbf{1} & \mathbf{0} & 0 & 0 & \dots & 0 & \dots & 0 \\
F & 0 & \mathbf{1} & 0 & 0 & \dots & 0 & \dots & 0 \\
N_1 & 0 & 1-R_1 & \mathbf{0} & R_1P_{12} & \dots & R_1P_{1j} & \dots & R_1P_{1n} \\
\vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & & \vdots \\
N_i & 0 & 1-R_i & \mathbf{0} & R_iP_{i2} & \dots & R_iP_{ij} & \dots & R_iP_{in} \\
\vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & & \vdots \\
N_{n-1} & 0 & 1-R_{n-1} & \mathbf{0} & R_{n-1}P_{(n-1)2} & \dots & R_{n-1}P_{(n-1)j} & \dots & R_{n-1}P_{(n-1)n} \\
N_n & R_n & 1-R_n & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \dots & \mathbf{0}
\end{array}
$$

"All Models are Wrong - Some are Useful."
George E. P. Box