Presentation on Software Reliability & Quality

Group 2

Software Reliability

Software Reliability

- Software Reliability is the probability of failure-free software operation for a specified period of time in a specified environment.

Software Reliability

- Key concepts in discussing reliability:
    - Fault
    - Failure
    - Time
    - Three kinds of time intervals: MTTR, MTTF, MTBF

Software Reliability

- Failure
  - A failure is said to occur if the **observable** outcome of a **program execution** is different from the expected outcome.
- Fault
  - The adjudged cause of failure is called a fault.
  - Example: A failure may be cause by a defective block of code.

Software Reliability

- Time
  - Time is a key concept in the formulation of reliability. If the time gap between two successive failures is short, we say that the system is less reliable.
  - Two forms of time are considered.
    - Execution time (τ)
    - Calendar time (t)
    - Clock Time

Software Reliability: Time

- Execution Time
  - The execution time for a software system is the actual time spent by a processor in executing the instructions of the software system.
- Clock Time
  - Clock time includes the wait time of the software system and execution times of other software systems

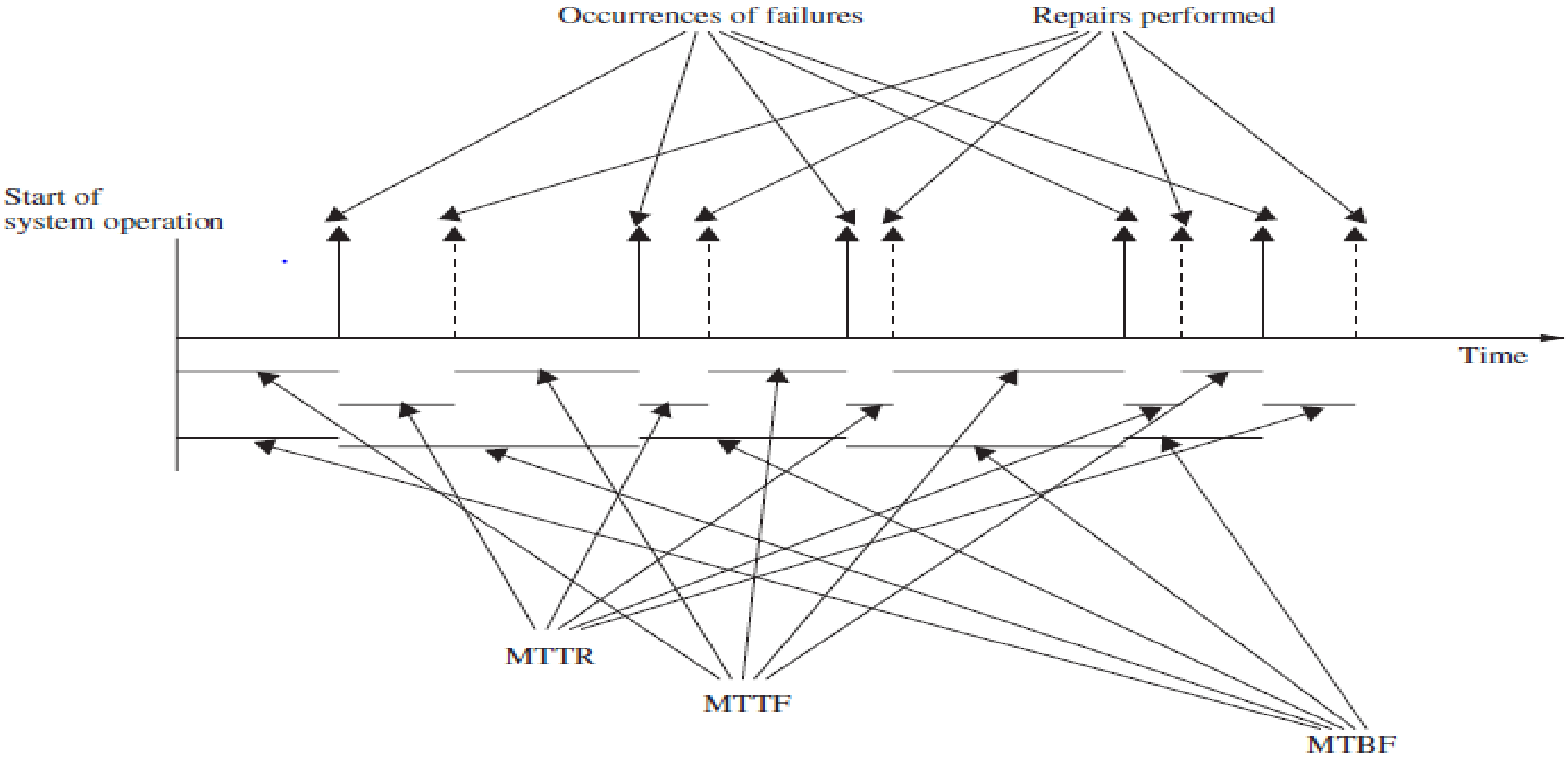
Software Reliability: Time

- Calendar time
  - Calendar time is the time more commonly understood and followed in everyday life by all, including software engineers and project managers.

Software Reliability: Time Interval

- MTTF: Mean Time To Failure
- MTTR: Mean Time To Repair
- MTBF: Mean Time Between Failures
  - MTBF = MTTF + MTTR

Software Reliability: Time Interval



Software Reliability: Time Interval

- Counting failures in periodic intervals $\mu(\tau)$
  - This denotes the total number of failures observed until execution time τ from the beginning of system execution.
- Failure intensity$\lambda(\tau)$
  - This denotes the number of failures observed per unit time after τ time units of executing the system from the beginning.
- Relationship between $\lambda(\tau)$ and $\mu(\tau)$
  - $\lambda(\tau) = d\mu(\tau)/d\tau$

FACTORS INFLUENCING SOFTWARE RELIABILITY

- Size and complexity of code
- Characteristics of the development process used
- Education, experience, and training of development personnel
- Operational environment

Applications of Software Reliability

- Comparison of software engineering technologies
- Measuring the progress of system testing
- Controlling the system in operation
- Better insight into software development processes

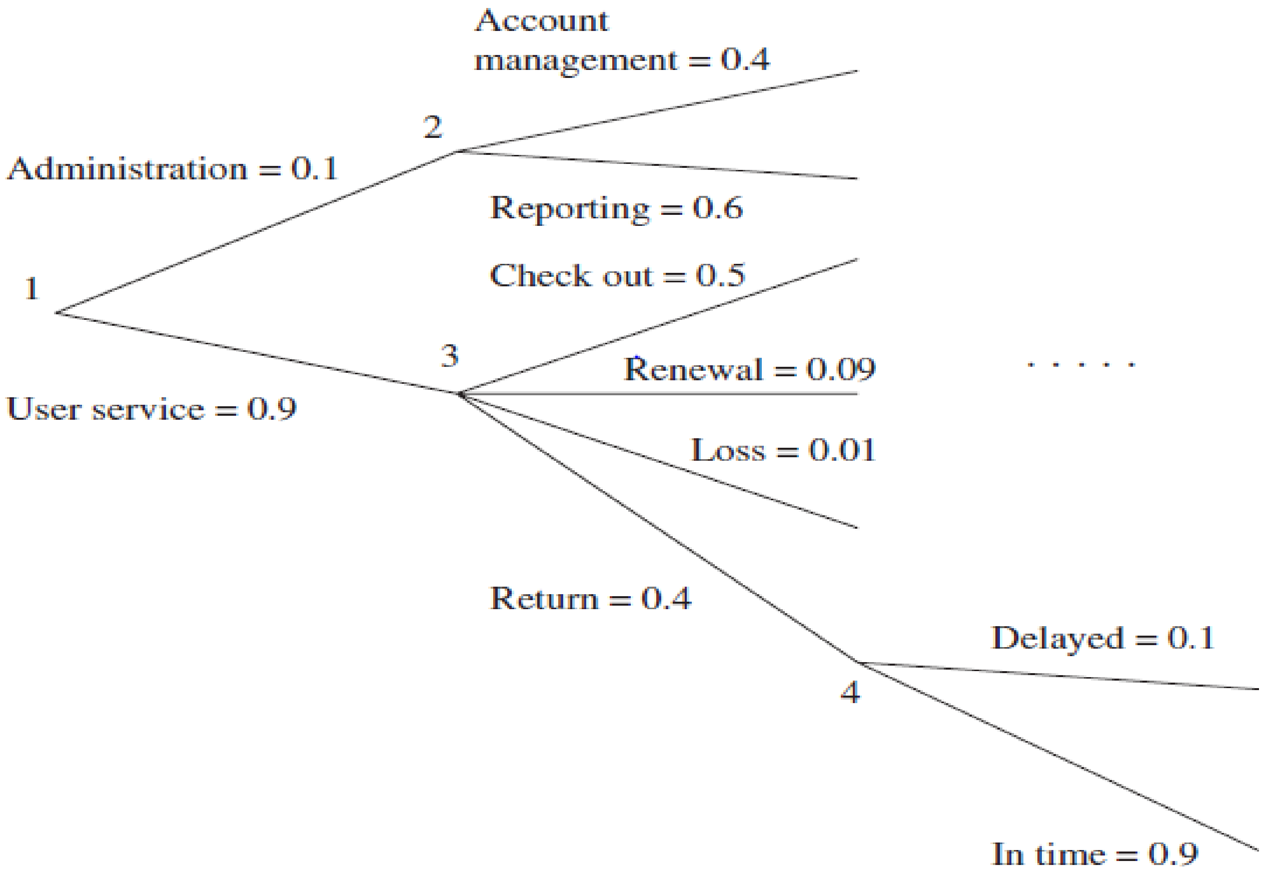
Operational Profiles

- An OP describes how actual users operate a system
- Two ways to represent operational profiles:
  - Tabular representation
  - Graphical representation

Operational Profiles: Tabular representation

Operation	Operations per Hour	Probability
Book checked out	450	0.45
Book returned in time	324	0.324
Book renewed	81	0.081
Book returned late	36	0.036
Book reported lost	9	0.009
⋮	⋮	⋮
Total	1000	1.0

Operational Profiles: Graphical representation



Reliability Models

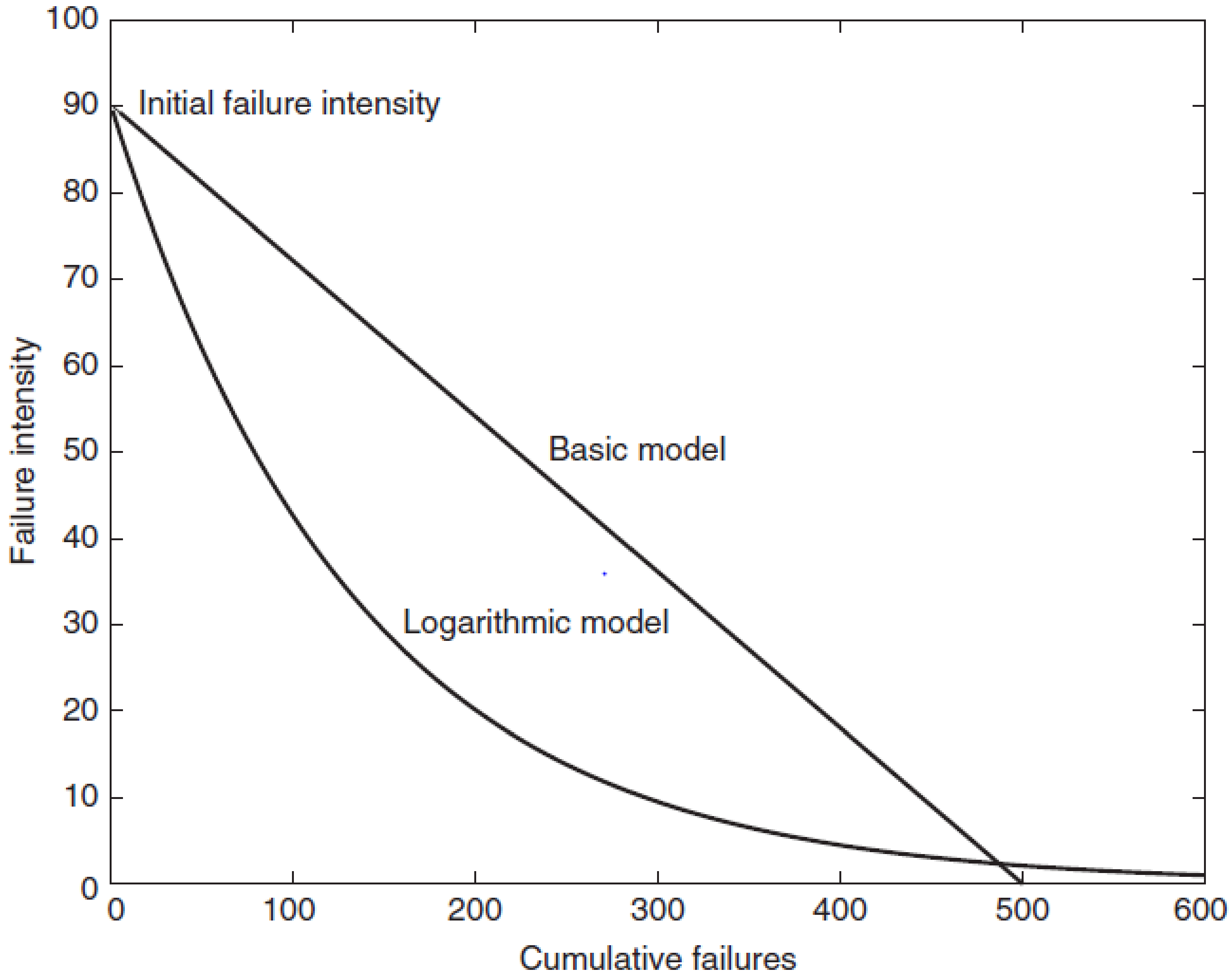
- Basic Model
  - The decrease in failure intensity after observing a failure and fixing the corresponding fault is **constant**.
- Logarithmic Model
  - The decrease in failure intensity after observing a failure and fixing the corresponding fault is **smaller** than the previous decrease.

Reliability Models: Basic assumptions

- Faults in the program are independent.
- Execution time between failures is large with respect to instruction execution time.
- Potential test space covers its use space.
- The set of inputs per test run is randomly chosen.
- The fault causing a failure is immediately fixed or else its re-occurrence is not counted again.

Reliability Models: Parameters of the model

- Parameters of the models
  - λ_0: The initial failure intensity observed at the beginning of system testing.
  - v_0: The total number of system failures that we expect to observe over infinite time starting from the beginning of system testing.
  - θ: A parameter representing n0n-linear drop in failure intensity in the Logarithmic model.



MUSA'S BASIC MODEL

- Assumption: Decrement in failure function is constant.
- Result: Failure intensity is function of average number of failures experienced at any given point in time (= failure probability).

$$\lambda(\mu) = \lambda_0 \left[1 - \frac{\mu}{v_0} \right]$$

y=mx+c
f(x)=mx+c
$f(\mu)=m\mu+c=(-\lambda_0 /V_0)\ \mu+ \lambda_0$
That is
$f(\mu)=m\mu+c=\lambda_0 (1- (\mu /V_0))$

$\lambda(\mu)$: failure intensity.

λ_0: initial failure intensity at start of execution.

μ: average total number of failures at a given point in time.

v_0: total number of failures over infinite time.

EXAMPLE

Assume that we are at some point of time *t* time units in the life cycle of a software system after it has been deployed.

Assume the program will experience 100 failures over infinite execution time. During the last *t* time unit interval 50 failures have been observed (and counted). The initially failure intensity was 10 failures per CPU hour.

Compute the current (at *t*) failure intensity:

$$\lambda(\mu) = \lambda_0 \left[1 - \frac{\mu}{v_0} \right]$$

$$\lambda(50) = 10 \left[1 - \frac{50}{100} \right] = 5$$

Failure per CPU hour

Reliability Models

- Example

  Assume that a software system is undergoing system level testing. The initial failure intensity of the system was 25 failures/CPU hours, and the current failure intensity is 5 failures/CPU hour. It has been decided by the project manager that the system will be released only after the system reaches a reliability level of at most 0.001 failures/CPU hour. From their experience the management team estimates that the system will experience a total of 1200 failures over infinite time. Calculate the additional length of system testing required before the system can be released.

  - The system will experience a total of 1200 failures over infinite time. Thus, we use the Basic model.
  - λ_c and λ_r are the current failure intensity and the failure intensity at the time of release.
  - Assume that the current failure intensity has been achieved after executing the system for τ_c hours.
  - Let λ_r be achieved after testing the system for a total of τ_r hours.

Reliability Models

- (Example continued)
  - $(\tau_r - \tau_c)$ denotes the additional execution time requires to achieve λ_r.

We can write λ_c and λ_r as follows.

$\lambda_c = \lambda_0.e^{-\lambda 0\ \tau c/v0}$

$\lambda_r = \lambda_0.e^{-\lambda 0\ \tau r/v0}$

$\lambda_c / \lambda_r = (\lambda_0.e^{-\lambda 0\ \tau c/v0})/(\lambda_0.e^{-\lambda 0\ \tau r/v0})$

$= e^{(\tau r - \tau c)\ \lambda 0/v0}$

$\ln(\lambda_c / \lambda_r) = (\tau_r - \tau_c)\ \lambda_0/v_0$

$(\tau_r - \tau_c) = (v_0/ \lambda_0)\ln(\lambda_c / \lambda_r)$

$= (1200/25)\ln(5/0.001)$

$= 408.825$ hours

It is required to test the system for more time so that the CPU runs for another 408.825 hours to achieve the reliability level of 0.001 failures/hour.

LOGARITHMIC MODEL

- Decrement per encountered failure decreases

$$\lambda(\mu) = \lambda_0 e^{-\theta\mu}$$

θ : failure intensity decay parameter.

- Example

λ_0 = 10 failures per CPU hour.

θ =0.02/failure.

50 failures have been experienced (μ = 50).

Current failure intensity:

$$\lambda(50) = 10e^{(-0.02\times 50)} = 10e^{-1} = 3.68$$

Model Extension

- Failure intensity as a function of execution time.
- For basic model:

$$\lambda(\tau) = \lambda_0 e^{\left(-\frac{\lambda_0}{v_0}\tau\right)}$$

- For logarithmic model

$$\lambda(\tau) = \frac{\lambda_0}{\lambda_0 \theta \tau + 1}$$

Example (Basic Model Extension)

∀ λ_0 = 10 [failures/CPU hour].

- v_0 = 100 (number of failures over infinite execution time).

  τ = 10 CPU hours: $\lambda(\tau) = \lambda_0 e^{\left(-\frac{\lambda_0}{v_0}\tau\right)}$

$$\lambda(10) = 10e^{\left(-\frac{10}{100}\times 10\right)} = 10e^{-1} = 3.68$$

Failure per CPU hour

∀ τ = 100 CPU hours:

$$\lambda(100) = 10e^{\left(-\frac{10}{100}\times 100\right)} = 10e^{-10} = 0.000454$$

Failure per CPU hour

Example (Logarithmic Model Extension)

- $\lambda_0 = 10$ [failures/CPU hour]. $\theta = 0.02$ / failure.
- $\tau = 10$ CPU hours: $\lambda(\tau) = \dfrac{\lambda_0}{\lambda_0 \theta \tau + 1}$

$$\lambda(10) = \frac{10}{10 \times 0.02 \times 10 + 1} = 3.33$$ Failure per CPU hour (3.68 in basic model)

- $\tau = 100$ CPU hours:

$$\lambda(100) = \frac{10}{10 \times 0.02 \times 100 + 1} = 0.467$$ Failure per CPU hour

(0.000454 in basic model)

MODEL DISCUSSION

Comparison of basic and logarithmic model

- Basic model assumes that there is a 0 failure intensity, logarithmic model assumes convergence to 0 failure intensity.
- Basic model assumes a finite number of failures in the system, logarithmic model assumes infinite number.

Software Quality

Quality Assurance

- **Quality assurance** (**QA**) is a way of preventing mistakes and defects in manufactured products and avoiding problems when delivering products or services to customer.

VIEWS OF SOFTWARE QUALITY

The five viewpoints help us in understanding different aspects of the quality concept

- Transcendental View
- User View
- Manufacturing View
- Product View
- Value-Based View

Transcendental View

- Quality is something that can be recognized through experience, but not defined in some tractable form.
- A good quality object stands out, and it is easily recognized.

User View

- Quality concerns the extent to which a product meets user needs and expectations.
- This view may encompass many subject elements, such as usability, reliability, and efficiency.

Manufacturing View

- This view has its genesis in the manufacturing industry – auto and electronics.
- The concept of process plays a key role.
- Conformance to requirements leads to uniformity in products. Some argue that such uniformity does not guarantee quality.

Product View

- If a product is manufactured with good internal properties, then it will have good external properties.
- One can explore the causal relationship between internal properties and external qualities.

Value-Based View

- This represents the merger of two concepts: excellence and worth.
- Quality is a measure of excellence, and value is a measure of worth.

McCall’s Quality Factors and Criteria

- A quality factor represents the behavioral characteristic of a system.

  Examples: correctness, reliability, efficiency, testability, portability

- A quality criterion is an attribute of a quality factor that is related to software development.

  Example:

  - Modularity is an attribute of the architecture of a software system.

MCCALL'S QUALITY FACTORS AND CRITERIA

Quality Factors	Definition
Correctness	Extent to which a program satisfies its specifications and fulfills the user's mission objectives
Reliability	Extent to which a program can be expected to perform its intended function with required precision
Efficiency	Amount of computing resources and code required by a program to perform a function
Integrity	Extent to which access to software or data by unauthorized persons can be controlled
Usability	Effort required to learn, operate, prepare input, and interpret output of a program
Maintainability	Effort required to locate and fix a defect in an operational program
Testability	Effort required to test a program to ensure that it performs its intended functions
Flexibility	Effort required to modify an operational program
Portability	Effort required to transfer a program from one hardware and/or software environment to another
Reusability	Extent to which parts of a software system can be reused in other applications
Interoperability	Effort required to couple one system with another

MCCALL'S QUALITY FACTORS AND CRITERIA

Quality Categories	**Quality Factors**	**Broad Objectives**
Product Operation	Correctness	Does it do what the customer wants?
	Reliability	Does it do it accurately all of the time?
	Efficiency	Does it quickly solve the intended problem?
	Integrity	Is it secure?
	Usability	Can I run it?
Product Revision	Maintainability	Can it be fixed?
	Testability	Can it be tested?
	Flexibility	Can it be changed?
Product Transition	Portability	Can it be used on another machine?
	Reusability	Can parts of it be reused?
	Interoperability	Can it interface with another system?

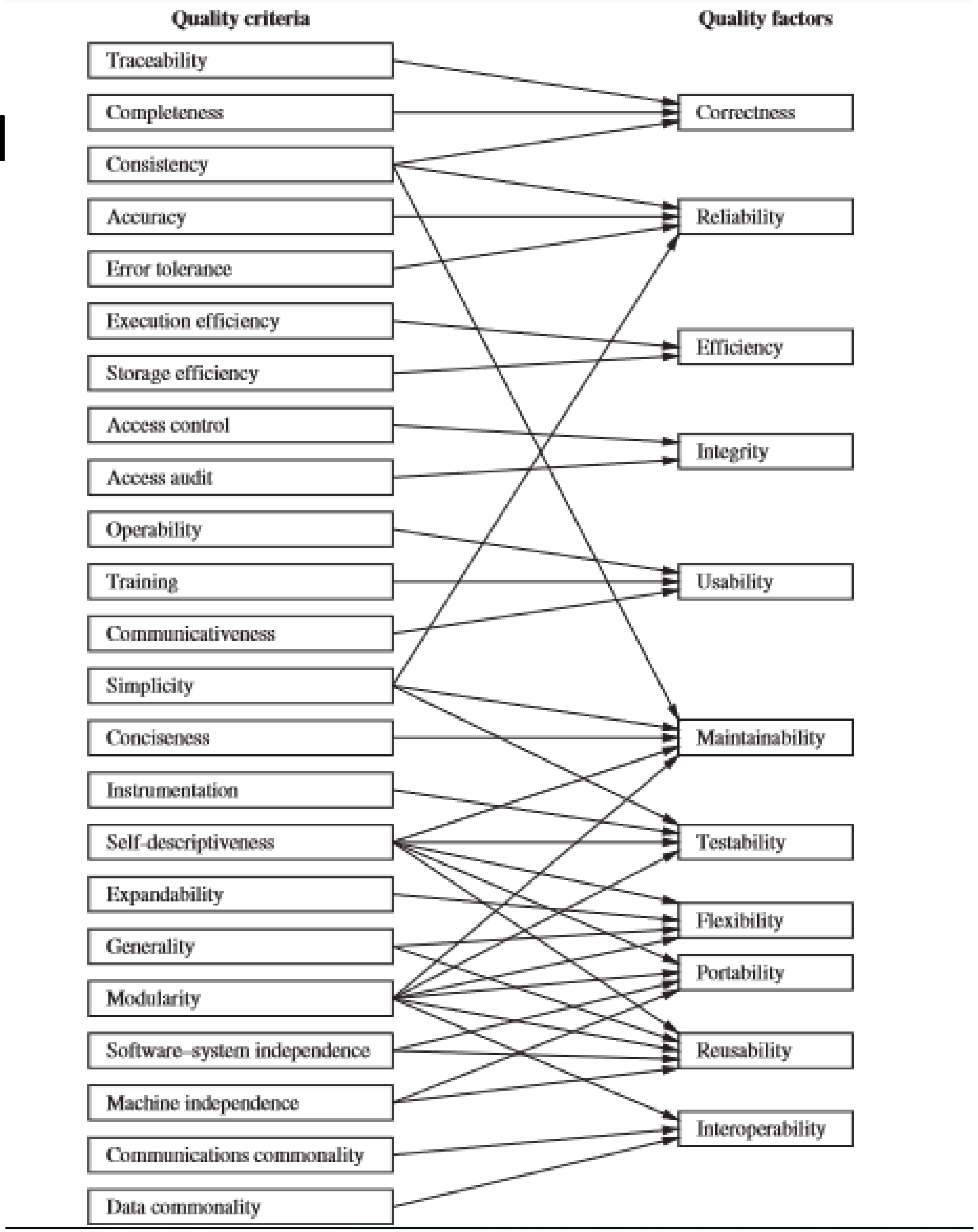
MCCALL’S QUALITY CRITERIA

- **1. Access Audit:** Ease with which the software and data can be checked for compliance with standards.
- **2. Access Control:** Provisions for control and protection of the software
- **3. Accuracy:** Precisions of computations and output.
- **4. Completeness:** Degree to which full implementation of required functionalities have been achieved.
- **5. Communicativeness:** Ease with which the inputs and outputs can be assimilated.
- **6. Conciseness:** Compactness of the source code, in terms of lines of code.
- **7. Consistency:** Use of uniform design and implementation techniques.
- **8. Data commonality:** Use of standard data representation.
- **9. Error tolerance:** Degree to which continuity of operation is ensured under adverse conditions.

MCCALL’S QUALITY CRITERIA

- **10. Execution efficiency:** Run time efficiency of the software.
- **11. Expandability:** Degree to which storage requirements or software functions can be expanded.
- **12. Hardware independence:** Degree to which a software is dependent on the underlying hardware.
- **13. Modularity:** Provision of highly independent modules.
- **14. Operability:** Ease of operation of the software.
- **15. Simplicity:** Ease with which the software can be understood.
- **16. Software efficiency:** Run time storage requirements of the software.
- **17. Traceability:** Ability to link software components to requirements.
- **18. Training:** Ease with which new users can use the system.

Relation between quality factors and quality criteria



APPENDIX

Reliability Models

- Basic model

  Assumption: $\lambda(\mu) = \lambda_0 (1 - \mu/v_0)$
  [y=mx+c]

  $d\mu(\tau)/d\tau = \lambda_0 (1 - \mu(\tau)/v_0)$

  $\mu(\tau) = \lambda_0 (1 - \mu/v_0)$

  $\lambda(\tau) = \lambda_0 . e^{-\lambda 0\ \tau/v 0}$

- Logarithmic model

  Assumption: $\lambda(\mu) = \lambda_0 e^{-\theta\mu}$

  $d\mu(\tau)/d\tau = \lambda_0 e^{-\theta\mu(\tau)}$

  $\mu(\tau) = \ln(\lambda_0\theta\tau + 1)/\theta$

  $\lambda(\tau) = \lambda_0/(\lambda_0\theta\tau + 1)$