GLOSSARY

Reliability

---

accident — event that caused a harm

ACID
atomicity
consistency
isolation durability

atomicity: all of the operations in the transaction will complete or none will

consistency: the database will be in a consistent state when the transaction begins and ends

isolation: the transaction behaves as if it is the only one being performed

durability: when completed, the transaction's operations will not be reversed (that is a separate action).

see also *eventual consistency.*

airplane safety — The number of hull losses per million flights.

Concord (now: 11.64

737: ~1

IEEE Spectrum *Sept 2000 p24-28; "Faults & Failures: The day Concorde fell to Earth"*

ARM-R — One is upside down and rotate 90deg to eliminate some upsets from common electrical spike.

Ashby's law of requisite variety — "Any controller, to be effective, must have sufficient variety in its coping mechanisms to counter the variety of actions that could be exhibited by the system to be controlled."

See also *fast fail*

assurance — "justified measures of confidence that the system functions as intended" and is reasonably free of flaws.

Uses a clear, comprehensive, defensible argument

analysis — analysis tends to affirm (justify) that the system is safe, versus

analysis that test the system to find the limits of its safety

AT&T — modular redundancy; when it deviates from very narrow constraints, it shuts down; shutdown is announced.

availability — "percentage of time during which the system is operational and conforms to specification."

A Villemeur, Reliability, Availability, Maintainability, and Safety Assessment*, Vol 1 Wiley 1991*

See also *integrity aspects*

$$= \frac{MTTF}{MTTF + MTTR}$$

Gibson 1993

see also *Gompertz law of mortality, MTTF. MTTR, reliability, Weibull power law*

block sparing — Block sparing is a part of a fault recovery and masking technique, usually done at a file-system level. When a block is discovered to be going bad:

1. The block's data is duplicated to another disk block,
2. The data structures referring to the block are updated, and
3. The block is marked bad

see *fault masking, SMART*

Byzantine failure — A component – software module, computer, router, etc. – fails but keeps on operating. This failure might be a software bug, hardware fault, or outright sabotage. In packet

networks, message processing, or data flow systems, this type of failure can be very hard to track down. Examples include misrouting messages, changing fields inappropriately, corrupt data, perform correctly – except for data from one user, or flood the network with garbage

See also *failure*

common mode rejection ratio — A measure of an instruments ability to ignore or reject interference from a voltage common to its input terminals relative to ground. Usually expressed in dB.

complexity — "A high level of complexity, unless it is there to increase robustness, poses more threats to system operation."

"A high level of coupling (sometimes considered a form of complexity) between components indicates a high level of interdependence: a change in a system component will ripple through the system via the coupling paths."

Ogmjen Prnjat, Lionel Sacks, "Integrity Methodology for Interoperable Environments" IEEE Communications, *May 1999 p126-1*

communication complexity — "How many operations or messages need to be generated to perform some action."

computational complexity — "How well a given procedure can be analytically described or determined."

time complexity — "How long (how many iterations or cycles) an operation takes."

conflict catcher — A well-regarded product for the Macintosh, thru System 9; similar techniques are available for other systems as well, and the techniques are useful:

- Rollbacks
- Testing
- Automatic detection of conflicts
- Metrics

See also *error correction*, *journaling*

coupling — Affects flexibility in system, constraints on timing, operation sequencing, acceptable input data ranges.

loosely coupled — If 'A' fails B can go on with it's operations; it isn't critical;

very flexible assumptions, flexible timing and sequencing, flexible time constraints.

tightly coupled — Opposite; less flexible in how it handles errors / faulty. Real time or very tight control systems.

criteria — absolute: compliance with target; often quantitative

relative: relative to previous or existing XYZ (e.g. product)

reductive: "as low as reasonably possible" (reduce it as much)

critical component — Any *component* within a critical device whose failure to perform can reasonably be expected to cause failure or decline in safety or effectiveness in a critical device.

see also *critical device.*

critical device — A critical *device* is one whose failure to perform can reasonably be expected to result in injury.

see also *critical components, critical operation*

critical operation — A critical *operation* is any operation – that if improperly performed – can reasonably be expected to impact a critical device: such as causing failure, impact its safety, or affect its effectiveness.

see also *critical component, critical device*

danger index — = Safety Index + C
where C is a normalizing constant (say 10)

see also *safety index*

deadlock "Block in a state expecting a message or event which will not or cannot occur." *Prnjat* ibid

See also *livelock*

defect prevention Functional specification defects, system design defects, programming defects, maintainability, root cause analysis. Critical success factors for peer reviews. Inspections, how to do them. How create and use specifications to minimize defects, ways to track and control defects.

dependability Decide what components (software, hardware, etc) will be relied on. Identify the critical properties, determine the level of confidence required. The properties: what are they and how atomized are they?

dual-string configuration any failed component in one string can be replaced by its twin in the other string.

dynamic fault recovery If a fault in a module is detected, the modules is disabled and a spare module is enabled. A series of actions is then performed to resume the normal operation.

electronic components failure rate t_0 -> t_s (settling time)

t_s -> oo R(t)

Reliability with time

$$R(t) = Ge^{-\lambda t}$$

G = constant

Lambda = sum of the failure rates of all constituent components

1/lambda = mean time between failure

environmental testing Compatibility testing, performance and stress testing. Security & controls testing, database testing, software testing. Error and disaster recovery testing. GUI testing, Client/server testing. Internet based app testing. Reliability testing.

error control strategy Specifies how error are determined (eg time-outs, explicit notifications) and what will be done when an error occurs (usually a retry).

error handling systematic Detecting when an error/exception condition occurs

Determining where and what should handle such conditions

The actual stop or recovery actions. This should be as graceful as possible. Should not exit the program except when absolutely necessary.

Some techniques employ a combination of codes must uniquely identify the path to the original failure.

error codes Error codes are often returned back through several levels of function calls until they are handled appropriately. The need to preserve various error states for higher layers (push/pop errno).

error messages There may be a clear relationship between user action and error. Such as the system's design anticipated this kind of error and reports cryptic messages, deceptive messages, misused, unspecific. Worse, these messages and errors are never human tested. The other kinds are errors the design didn't anticipate: Here is something bad, here are tea leaves. Hope you can figure it out.

exception handling A type of error and distributing.

longjmp/setjmp. Return through several layers of calls, leaves global state as it was of at longjmp() time, not setjmp() time.

Exceptions – try catch, throw. Like longjmp/setjmp in many regards. Calls destructors of locally declared objects for each level until gets back to catch spot.

error rate

For a given span of operation, testing only provide reliability estimates to ~104^-4 errors/hour (Littlewood & Stringini 1993)

Avionics, nuclear power require better than 10^-9 errors/hour

eventual consistency

consistency: client perceives that a set of operations has occurred all at once

availability: every operation must terminate in a defined response (pass, fail, etc.)

partition tolerance: operations will complete, even if individual components are unavailable (e.g. it fails).

see also *ACID*

fail safe

Under failure modes do safe things. Default deny/permit

Weakest link :strength in proportion to risk

Choke point: allows focus to within bounds that can be dealt with

Least privilege: minimal privilege to do the task, no more.

Defense in depth: a variety of protective mechanisms to produce defense.

Diversity of defenses: a variety of methods and approaches to defense.

Universal participation: all must participate.

Simplicity.

failure
engineering for

FMEA, fault tree

Employees tools and approaches simultaneously. (Note: simultaneous events are often so hard to analyze rationally that they are examined only as an exception—if they are likely to occur)

Reduces probability of fault event

Reduces direct impact of fault

Recover from fault

Note the deterioration or degradation of components and the functionality is often talked about separately.

see also *error correction, error handling*

fast fail

Determine out-of-bounds and disengage. This is often very useful, during development, for making the underlying mechanism robust, rather than the fault-handling mechanism; this is useful when the range of variety of disturbances is unknown. It is discouraged with released products.

see also *Ashby's law of requisite variety*

fast recovery

Identify the 20% you need for fast recovery 1 vs the 80% infrequently used. Residency bits, locality of reference tools, used to identify frequency of use.

fault

What you wanted didn't happen.

Topics: Detection, Isolation, Localization (finding it), Notification, Mitigation (Protection and Restoration)

Fault model; fault hypothesis, fault containment unit

fault avoidance

Prevents faults (so they don't have to be handled) thru estimation, forecasting and prevention. Limit the introduction of faults during the system construction. The approach includes good specifications, good programming practices & discipline, extensive & repetitive reviews & analysis, testing.

prevention: reduces likelihood of faults occurring before the system becomes operational
removal: find and remove causes of errors
forecasting: techniques that estimate the presence, creation and consequences of faults

fault capture — Often system trace. In emulation environment; in production environment with hardware support; in production with no hardware support. *See trace*

fault domain

fault handling — What you do to make it right. This may require detection ,identification, correction (diagnosis, masking, recovery), and communication (documentation, labeling, notification, and logging)

Error: A component did not deliver results it was expected to. Either does task or provides that it did not and a summation of why. Multi-case: provides for another route to get result originally requested.

The failure should be conservative and do no harm. Roll back to an unharmed state.

Tradeoff: how much time are you allowed between detecting and the time you must have the issue corrected?

Responses to restore after a fault:

1. Well-defined procedures that require overall coordination (within area of restoration)
2. Tools to aid operators & area coordinators execute operating procedures & make proper decisions
3. Regular training sessions of a varied nature

Handling:

1. Detect fault e.g. usually with a redundancy of some form, timeout, or a degradation in performance. Note exceptions have issues; try to construct them in a safe manner.
2. Go back to safe state. Dump out crud, eg trace buffers. Try alternate logic. If nothing worked report to caller. Eventually that is step 3.
3. Error messages. See *error handling*

fault masking — The customer doesn't know what you did to do right by him, but gets good service. Usually accomplished thru fault handling.

fault prevention — Structured design, static analysis (e.g. lint), Rigorous design rules, Quality Control, good Design practices.

Normal conditions, and continued operation. Parameter specifications.

Shutdown operations. Parameter specifications.

fault tolerance — Fault tolerance keeps system going in the face of faults by protecting against and compensating for faults. The techniques include: watchdog timers, periodic resets, reasonableness and acceptability checks, design diversity, and data diversity

n-fault tolerant: can tolerate *n* failures without degradation.

FMEA
failure mode effects analysis — Look at schematic and graph of hardware. For each element, ask:

What would if element failed? e.g. Completely fails, of 25% value shift.

What if link to element fails (e.g. stuck high, stuck low, or open)? Looks at connection to node.

Does not ask how it could fail; assumes it does. Only a SPOF method

Because schematics can get large, often only a subset is examined. The output or safety related sections. The items of designer or developer concern. Items that came out of the above.

Some components (to its input) have defined FMEA results. Many do not.

formal methods — "Formal methods used in developing computer systems are mathematically based techniques for describing system properties. Such formal methods provide frameworks within which people can specify, develop, and verify systems in a systematic… manner." *IEEE Computer, September 1990 p8*

"Formal methods are based on mathematics but are not entirely mathematical… [They attempt to] codify the customer's informally state requirements… [and] map the real world to some abstract representation of it." *IEEE Computer, September 1990 p19-20*

hazard — circumstances causing an incident or accident

incident	event that did not result in a harm	
integrity	See *availability, feature integration, journaling, liveness, performance, reliability, resilience, robustness, safety, scalability, security*	*Prnjat* ibid
operational level	Requirements to "minimize feature interactions can be achieved through thorough interconnection and interoperability testing, following a defined set of scenarios. A high level of resilience and robustness can be achieved performing extensive test in test beds, simulating possible behaviour of the environment."	
system level	Behaviour modeling, reachability analysis, livelock/deadlock detection techniques. "data coherence policies supporting atomicity, consistency, isolation, and durability of transactions."	
integrated level	"to maximize the computational and data complexity system metrics"	
unit level	Performance measurements.	
interactions	Dependencies between components. Linear (simple) interaction: components only affect others functionally downstream. Complex: interact with many others in many other parts of the system; hard to understand & predict behaviour. Complex + tightly coupled: promotes accidents.	
interruptions	▪ "The nature of the interruption ▪ "Any effects of the interruption ▪ "And parameters that can vary from one instance of the interruption to the next ▪ "How the interruption can be detected ▪ "Which actions the interruption can interrupt ▪ "How to respond to the interruption; especially how or whether to resume the interrupted process."	*kovitz*
journaling roll-back	Why? ▪ To catch conflicts with device driver and configurations ▪ System recovery ▪ Databases ▪ Backup emergency restore ▪ Scroller for going back in time.	
livelock	"oscillates between a closed set of states" See also *deadlock*	*Prnjat* ibid
liveness	"That something will, eventually, happen."	*Prnjat* ibid
load	What to do when the load is too high (or will become too high) to service. 1: load shedding, direct. RED, disable lower priority services, stop servicing some customers. 2. indirect: raise rates, RED which indirectly signals to others to stop, ask to reduce load. 3. Add more capacity, lease or purchase processors, etc.	
shedding	Needs to decide what to shed	
monitor	A control program that oversees the allocation of resources among a set of user programs.	
boot monitor	A control program used in the boot process, and may include a user directed mode	
debug monitor	A user directed, simple 'debugger' tool	
hypervisor	Hierarchy of supervisors in IBM mainframes	
kernel security	Tracks privileges, users identifiers, decides if an action can be carrier out; manages page	

monitor — allocations and privileges; sets, responds to and effects resource limits.

security monitor — Programmer defined monitor within an application. This is used within Java to decide if connection (or other action) definitely should not be allowed.

supervisor — ?

range checks — Parameter validation. The trick is to recognize that most parameters will be correct. Pointers shouldn't ne null, and should point to valid address; numbers should be within acceptable ranges.

Return codes. especially when a pointer or handle is returned. Array indices should be before end; use of buffer, end of buffer

asserts — Conditional tests regarding the validity of the data, may trigger debugger

structure checking — Checks that pointers reference expected structure; possibly via magic #. Is an assertion that must be ensured by the creator of the class, must be preserved by every exported routine of the class. The entire data structure be test validity.

MTTDL
mean time to data loss

$$= \frac{MTBF^2}{N(N/G + 1)MTTR_G}$$

Used to describe a RAID system with all failures assumed to be independent. N is the number of data disks in the raid group, G is the number of parity disks (eg possibly done with multiple RAID groups)

MTTF
mean time to failure

$$= \int_0^\infty dt\, Reliability\,(t)$$

Black's Equation

$$MTTF = AJ^{-N} e^{\frac{E_A}{kT}}$$

where
A = empirical coefficient
J = current density
N = 1..3, metal composition and thickness coefficient
Ea= activation energy
k = Boltzman's constant
T = temperature of metal connection

multiple modular redundancy — Each module is replaced by a series of identical modules Their outputs are compared to derive a correct majority should one of the modules fail.

operating system
deterministic — The worst-case execution time of each its system calls is calculable.

real time

1. The process currently running is guaranteed to be highest-priority task
2. Context switches have a bounded worst-case time
3. Interrupt latency has a bounded worst-case time
4. Prevention of unbounded priority inversion

opposing-traffic sensor — This is a fail-to-a-safe-state mechanism used in traffic lights. It is a sensor that checks to see if the signal lights ever allow vehicles to enter the intersection from opposite directions. When sensor detects such an event, it shuts down the traffic signal's primary controller and has the lights, in all directions, blink red; the condition must be cleared manually after a person figures out what went wrong. Such a sensor must completely separate from the signal controller, and consider green, yellow, turn-lane arrows, pedestrian controls, etc.

see also *fail safe*, *trace buffer, watchdog timer*

Traffic intersection design standards: http://www.urbandalelibrrary.org/Urban_Design_Standards/chapter13.pdf
New York State specifications for traffic controllers: http://www.dot.state.ny.us/traffic/files/dotmes.pdf

patch table — Delivery of patch, verify its authenticity, examining it, storing it in non-volatile memory, handling its failure.

power immunity specifications — IEC 61000-6-1

program maintenance

Fred Brooks p121-112

"program maintenance… consists chiefly of changes that repair design defects. Much more often than with hardware, these changes include added functions. Usually they are visible to the user."

"The fundamental problem with program maintenance is that fixing a defect has a substantial (20-50 percent) chance of introducing another… Why aren't defects fixed more cleanly? First, even a subtle defect shows itself as a local failure of some kind. In fact, it often has system-wide ramifications, usually non-obvious. Any attempt to fix it with minimum effort will repair the local and the obvious, but unless the structure is pure or the documentation very fine, the far-reaching effects of the repair will be overlooked. Second, the repairer is usually not the man who wrote the code."

quality baked in with good project management

1. Thoughtful planning. List of activities, milestones (to focus, and to track progress), temper with past data
2. Commitment by company, but don't over commit on deliverables
3. Tracking: examine 'why there are delays, and when to re-evaluate'
4. Teams are important
5. Review work and progress: detect defects early, review at ones own pace
6. Quality mechanisms, step. Requirements specification preparation, design & algorithm choice, coding and unit testing, integration testing, peer review meetings, documentation
7. Prevent & catch defects. Avoid adding new defects, use checklists, simplicity hides fewer defects and easier to maintain, complexity lends to poor testability, test as you implement, changes should support testing, maintenance, debugging, design should center on its use.

quality metrics

Testability

Maintainability

Clarity

Common complexity vs essential complexity

Which sections should be rewritten

Guide to rewriting

software quality

Difficult due to ego reasons and ease of longevity
Tight definition of interface

- What goes in definition?
- How to tell if something is missing?
- Versioning an interface
- How an interface is allowed to change with time
- Variables, calls, call tables, call sequence, allocation and memory use

Common elements of bad design:

- Change setting X (directly or indirectly)
- Data flow – linking failure

radiation hard total ionizing dose

accumulates radiation damage, changing threshold, leakage, etc.

single event latch up

a single event upset that creates a CMOS latch up, requiring a power cycle.

single event upset

soft error caused by an ion triggering an electrical transient

rate monotonic analysis

Optimal means of assigning relative priorities. Can demonstrate that a set of tasks will always meet their deadlines, even under worst case situation. Priority inversions must be bounded (i.e., unbounded inversions must be prevented). Strict system use a preemption system to guarantee. In the analysis tasks are assigned a fixed priority (either absolute # or

relative to other task), that is not changeable at runtime.

The priority assigned based on how often the task runs in the worst case.

Tasks that are given the same priority, any of the following can be done:

1. Merge the tasks, and just run Task 1, Task 2, etc.
2. Give them equal priority, with round robin or run-to-completion behavior.
3. Give them adjacent unequal priorities

See also *real-time, scheduling*

real-time

Real-time software measures, monitors, analyzes and controls real-world events as they occur; often it must respond within in strict time constraints. This includes

- Monitoring or data capture from the external environment
- Analysis of data in order to transform it into forms required by the application
- Controls to respond to external events
- Coordinating system components.

operating system design

Low interrupt latency is often achieved by never (ever) disabling interrupts in the kernel. Does this mean never turning off the "global interrupt enable" or never turning of the specific peripheral's interrupt enable?

See also *rate monotonic analysis*

hard real time

soft real time

RED
random early discard

Used only when there is automatic retry/retransmission. Discards elements when queue length exceeds a threshold. This works, not because it temporarily sheds load, but because the sender scales back the rate at which it sends. At first the sender slows because it hasn't recvd enough acks to continue; but it also adjusts its estimate of network load and sacles back the send rate.

reliability

"Probability of a system performing its purpose adequately for the period of time intended under the operation conditions encountered."

A. L. Reibman, M. Veeraghavn, "Reliability Modeling: An Overview for System Designers," IEEE Computer *April 1991*

Optimistic: assumes that none or few conflicts are likely to be experienced by any particular user of the shared resources.

Pessimistic: tend to assume the worst possible case and to defend against it by rather austere measures that often end up limiting concurrency.

Safe: if all processes already granted resources would be able to complete in some order even if each such processes were to use all resources they are entitled to.

reliability

$$= e^{\frac{-t}{MTTF}}$$

Siewiorek 1992i

reliability

should provide adequate means for its functionality to be unaffected by faults

possibly provide means of reducing probability of fault events.

should provide a means of tolerating faults

should be stable over time – its behaviour and functionality should not evolve to undermine any aspect of the system

it should provide a means of recovering from faults

Fuses that can be used to disconnect defective functional units.

see also *storage reliability*

Reliability: Probability that it works during the desired time period

Availability: probability that it works (over all time)

	Dependability	
	Security	
	Performance: missing deadlines	
	Conformance: meets certain types of specifications	
	Safety:	
issues	Fault levels. Radiation hardness, single event upsets. CMOS latch up. Memory error detection and correction. Reliability and cross strapping.	
replaceable modules issues	Replaceable modules – especially those designed for redundancy in a `reliable' system: ▪ Power distribution ▪ Heat removal ▪ IO interconnects ▪ Storage access ▪ CPU ▪ Switching of communication ▪ Switching tasks to another module ▪ Switching a module off ▪ Switching a module on	
requirements DO-178B	Outcomes are function / effect Emphasis is on writing identifiable requirements, leaving the document with little context and discourse of the requirement. Challenges include the expense, little immediate/direct benefits. Testing that the requirement is properly implemented. How you test that the requirements are the right ones. Analyzing how well the tests test the implementation. Attributes of a requirement's construction Requirement by property.	
resilience	"the system can recover from fault."	*Prnjat ibid*
risk	likelihood of a hazard, and severity of harm Presence (present,extension,new) – risk is technology presence by market presence. see also *hazard, likelihood, servity*	
risk identification	Recognize symptoms: unrealistic specifications and margins in design and test. Reliability analysis, FMEA, fault tree. How do others identify and evaluate risk? Insurance underwriters in setting premiums, NASA, other projects. Analysis techniques. NMI 7120.4. Requirements and Intent DOD 4245.2 templates. Analytic hierarchy process. Information gathering. Expert interviews. Structured questionnaires. Known technical risks. Redundancy calculations. Test margins. Parts, materials, components selection. Analysis. Assessment criteria. Risk baselines. Simulation via crystal ball, prima vera monte carlo.	
risk management	either accept it, mitigate it, or transfer it.	
risk management	Process: preliminary analysis, definition, design & development, operations.	

process

1. Identify & understand your major risks
2. Decide which risks are natural to keep? Which should you seek to transfer?
3. Determine your capacity for risk
4. Embed risk in all decisions / processes
5. Align everything around risk – monitor risks, manage them

Concept. Risk components – technical, schedule, cost, programmatic, political. Measures of risk. Definition of risk factor.

How to assign value to risk?

Focus on the few risks – or few groups of risk – that matter.

Application. Developing risk management plans. Developing risk assessment criteria. Establishing risk baselines. Establishing mitigation actions and contingency planning. Monitoring and reporting risks – lessons learned database, anomaly reporting.

robust design principles

- Fault tolerance
- isolation (containment)
- patching, each independent module has its own corrections area, well-defined structures in memory to allowing restarting without loss of existing flows,
- lifespan being sufficiently shorter than the failure rate,
- bounded behaviour.

robust features

- Update software without shutting down or rebooting
- Handle failures without loss of operations.
- verify existing block contents before performing patch / upgrade
- structured logging

robustness "The stability of the system to handle .. all eventualities and continue to operation" *Prnjat,* ibid

safety nothing 'bad' will occur; that "the risk associated with it is judged to be acceptable." *William Lowrance, 1976*

Table 1: *Distinctions*

Term	*Meaning*
safety	How to prevent or reduce risks or harms
security	Preventing, reducing, etc others from finding information or altering

psychological perceptions of safety

1. People do not believe they are at risk. Believe that they are less vulnerable to risks than others and less likely to be harmed.
2. People are unmotivated
3. Safety is an abstract concept. Outcomes with abstract nature tend to be less persuasive than concrete outcomes ("security" outcomes are selected less because of this?). Costs are real and immediate compared with abstract gains / benefits.
4. People tend to prefer a gamble-for-a-loss over a guaranteed loss.

safety critical

Concerned with the potential harm a machine might do (e.g. should few, if any, machines strong or heavy enough to crush a person be allowed into homes?). The techniques include FMEA and design approaches:

- Avoid contact altogether with people
- Sensors to avoid collisions with people
- Being made from lighter and softer materials
- Backup systems
- Emergency shut off switches and valves

safety index	$= \log N$ where there is probability that 1 in N people will die from the procedure, product, etc.	
safety processor	Secondary processor: logical safety and electrical safety.	
security	"stay in correct operation state… they are able to detect and avoid intentional attack."	*Prnjat,* ibid
single event effects	Effects in digital and analog devices: microprocessors, DSPs, DC/DC converters, opto-electronics, volatile & non-volatile memories Transient & permanent upsets Upset rate Soft errors, latch up, burn-out, SEFI	
underlying phenomena	Ionization by primary particles and secondaries from nuclear collisions Charge collection in small structures, Charge migration and Critical Charge Crystalline lattice deformation Damage thresholds in Silicon and Gallium Arsenide	
specification document	The intent is that the specification is without ambiguity, is complete, consistent, and that the implementation conforms to it. There are formal methods to test the conformance. A specification may be written to preserve properties in an existing implementation/.	
requirements	Includes a means of identifying requirements; we use a TEXTLABEL### (eq REQ123) but it does not have t be that way. Strongly encouraged to ensure that a requirement does not have unbounded time will respond with X, but it is decoupled so could respond 100 usecs form now. Requirement should be free of negative tense: not, doesn't, never, won't, etc. Requirements for hat the user wants, software requirements, functional requirements, system requirements, performance requirements, constraints, etc.	
stack safety	Tracking safe range to write, and regions that are not allowed. For instance, tracking the writing of undefined areas of stack after the current (which can happen when a called subroutine returns a pointer to a variable on the stack).	
stability	failures of interconnection failures of consistency of a part decomposition to a stable intermediary form that can be quickly reassembled.	
static analysis	a body of techniques that can examine source or object code and identify properties – especially certain kinds of bugs – without executing the code.	
storage reliability classification	The reliability of file systems and other storage mechanisms might be as follows: 1. No special reliability mechanism 2. Protects meta-data consistency, but not data consistency 3. Protects content – data consistency Most systems are #1 or moderate #2. see also *information lifecycle management*	
system wear	Reflected in subtle changes throughout system	
testers	Testers find info about quality that helps management in making better informed decisions. Testers have poor ability to prioritize; little empathy with users.	
bug testing	Can manage only a small number of testers, esp. when there is a lot of bugs. A person only reports a few bug: don't want all testers to report the same bug. Start with a small set of testers Expand this set only when you get the most frequent bugs fixed. Will lose some people with each round of beta expansion.	
testing automated	Load & stress testing	

testing modifications — localized change testing, regression testing, change completion checklist

test cases — indentifying what to test, how to design test cases, how to ensure complete coverage: analyzing functional specifications for testability, equivalence, path analysis, input-based tests, risk-based testing

test quality — Attributes of a test's construction

Coverage of branches

MCDC analysis

thread
issues with design of threaded code — Data races, lock granularity, lock ordering, modularity, blocking time, priority inversion, and deadlocks.

No IO's in atomics.

Java RT — Thread stop: ability to force it go up call tree

Aircraft use CORBA

trace buffer — IBM CHRP, z990 Firmware in peripherals store events to a circular buffer dedicated to the peripheral unit. z990 8x64bits. Used to trace activity. Used to alert OS of events, such as overheat. Not clear it is good enough for replay

aka *flight recorder.*

data trace — Traces of data flow through system

instruction tracing — Supported in some hardware. Records the functions that were called, possibly with parameters, preserves order, including return value. Record all of API calls in binary trace file, parameters of function, return value, procedure name. Detailed log of calls, done at runtime.

traceability — Ability to follow the steps from output back to original sources. For products, this allows tracing all of the products design, and features back to the original documents approved by the company. For information, measurements, methodology and standards.

trap instructions
conditional — When hardware and/or software detects unsatisfactory operation control is transferred to a special (possibly interrupt) handler, often with pertinent address & state information. CPU or memory traps are main types. Not all traps can be ignored; complexity, frequency of occurrence, overhead, and response times. Can replace explicit test instructions.

MACH/ OS-X examples at: http://www.clozure.com/cgi-bin/viewcvs.cgi/ccl/lisp-kernel/lisp-exceptions.c http://www.omnigroup.com/mailman/archive/macosx-dev/2000-June/002030.html http://download.plt-scheme.org/scheme/plt-clean-cvs/src/mzscheme/gc2/vm_osx.c

Traps are, usually a fast alternative to a variety of frequently used safety constructions frequently. These constructions typically look like:

```
if (!Ptr) goto HandleError;
if (Idx >= Ary->length) goto HandleError;
if (Idx < 0) goto HandleError;
```

While small and compact, comparison and branch cumulatively adds up into a performance overhead on well-behaved software.

The PowerPC includes a family of conditional trap instructions, such as the following, that throw a trap if R15 is null:

```
teqi R15, 0x00
```

The performance, in the common case, is that there is no branch, and that the comparison happens in parallel with other instructions. (The CPU usually guarantees that their effects will be discard if the trap is thrown). The application needs to install a custom handler to examine why a trap occurred, and to restore the processing if possible. Some even include the ability to emulate some instructions.

Often the handlers are OS and CPU specific, and can quite slow. When a trap is invoked – access violation, manual trap, debugging trap, instruction emulations, FPU error, etc – the CPU goes to the kernels trap table, and passes off to its trap handler. The kernel eventually passes control back to the process.

validation

see also *validation* (infosec)

central tasks of validation:

- to stimulate a design,
- to check that the design produces results according to its specification, and
- coverage: measure how much of a design's possible execution space has been simulated and checked

Table 2: *Between validation and verification*

Term	*Meaning*
validation	test that the behaviour is consistent with the requires, safety and efficaciousness
verification	demonstrates that the implementation matches the stated design.

validity checking test setup

An empirical process to ensure that the collected data properly represents the phenomena with in tolerances.

Chuck Wright,"Expert Column: Automated or not, data-validity check plays a significant role in test" Personal Engineering*, Feb 1998*

1. Examine the Average and RMS levels before the test. What is the normal and noise band? The average value at this step is the zero level.
2. Examine the Average and RMS levels after the test. The sample window is not long enough if the singla has not returned too the settled state.
3. Compute the maximum signal level and check that the full scale is larger, but not too much
4. Calculate the signal to noise ratio: the swing at step 1, and those during the test
5. Check the symmetry of noise around the signal level
6. Calculate figures of merit
7. Integrate signal to check net change
8. Examine the channel's spectra: is there enough bandwidth in the signal? (The signals should drop off at the top end)
9. Check the SRS and peak ratio
10. Check initial slope

verification

"formal verification] FV is the use of tools that mathematically analyze the space of possible behaviors of a design, rather than computing results for particular values."

versioning

Method to distinguish between two or more versions.

Answers questions: Is it ~100% compatible (excluding bug fixes)? Which version is newer? Is it tested and approved?

watchdog timer

Those with selectable intervals, Except at the highest level of privilege, should be only allowed to shorten their time to expiration.

Weibull power law

$$\log(\textit{failure rate}) = c_1 t + c_0$$

$$R(t) = Ge^{-\lambda t}$$

G = constant specific to device

λ = Sum of failure rates of all constituent components

$\frac{1}{\lambda}$ = mean time between failures

See also *Gompertz law of mortality, reliability, risk*

word tearing

unaligned access of machine word causing invalid (not merely out of date) data. Since one part of the word was read before the new value and another part was read after it was written. This is especially true for words that cross-cache boundaries in SMP systems and systems that require exception handlers for

unaligned access.

Once the products reach/exceed a satisfactory level of performance, the emphasis switches to reliability. If a reliability/product is too expensive, customers work around unreliability in current products

SAE ARP 4761 Guidelines and methods for conducting the safety assessment process on Civil Airborne Systems and Equipment. 1996 Society of Automotive Engineers.

MOD Defence Standard 0058 Requirements for Safety Related Software in Defence Equipment. 1996 UK Ministry of Defence

MOD Interim Defence Standard 08-58 Issues 1: HAZOP Studies on Systems Containing Programmable Electronics 1996 UK Ministry of Defence