Software Architecture

5. Availability

Chang-Hyun Jo
Professor, PhD
Department of Computer Science
California State University Fullerton
http://jo.ecs.fullerton.edu

Outlines

Availability

- Availability refers to a property of software that it is there and ready to carry out its task when you need it to be.
  - Reliability: The ability of an item to perform a required function under stated conditions for a stated period of time; the characteristic of an item expressed by the probability that it will perform a required function under stated conditions for a stated period of time [IEEE Trans. On Reliability, V.19, N.4, Nov., 1970]
  - Availability: reliability + recovery (when the system breaks, it repairs itself)
  - Dependability: the ability to avoid failures that are more frequent and more severe than is acceptable

Availability

- Availability refers to the ability of a system to mask or repair faults such that the cumulative service outage period does not exceed a required value over a specified time interval.
  - These definitions make the concept of failure subject to the judgment of an external agent, possibly a human.
  - They also subsume concepts of reliability, confidentiality, integrity, and any other quality attribute that involves a concept of unacceptable failure.

Availability

- Availability is closely related to security:
  - A denial-of-service attack is explicitly designed to make a system fail – that is, to make it unavailable.
- Availability is also closely related to performance:
  - because it may be difficult to tell when a system has failed and when it is simply being outrageously slow to respond.
- Availability is closely allied with safety:
  - which is concerned with keeping the system from entering a hazardous state and recovering or limiting the damage when it does.

Availability

- A fault (failure's cause) can be either internal or external to the system under consideration.
- Faults can be prevented, tolerated, removed, or forecast. (In this way a system becomes "resilient" to faults.)
- We are concerned:
  - how system faults are detected
  - how frequently system faults may occur
  - what happens when a fault occurs
  - how long a system is allowed to be out of operation
  - when faults or failures may occur safely
  - how faults or failures can be prevented
  - what kinds of notifications are required when a failure occurs

Availability

- When referring to hardware, there is a well-known expression used to derive steady-state availability:
  - MTBF / (MTBF + MTTR)
    - where MTBF refers to the mean time between failures and MTTR refers to the mean time to repair.
  - In the software world, you should think about: what will make your system fail, how likely that is to occur, and that there will be some time required to repair it.
    - The scheduled downtime is not counted against any availability requirements: service-level agreements (SLAs).

Availability

- In operational systems, faults are detected and correlated prior to being reported and repaired.
  - Fault correlation logic will categorize a fault according to its severity (critical, major, or minor) and service impact (service-affecting or non-service-affecting) in order to provide the system operator with timely and accurate system status and allow for the appropriate repair strategy to be employed.
    - The repair strategy may be automated or may require manual intervention.

Availability

Table 5.1. System Availability Requirements

Availability	Downtime/90 Days	Downtime/Year
99.0%	21 hours, 36 minutes	3 days, 15.6 hours
99.9%	2 hours, 10 minutes	8 hours, 0 minutes, 46 seconds
99.99%	12 minutes, 58 seconds	52 minutes, 34 seconds
99.999%	1 minute, 18 seconds	5 minutes, 15 seconds
99.9999%	8 seconds	32 seconds

- Examples of system availability requirements and associated threshold values for acceptable system downtime, measured over observation periods of 90 days and one year
- The term high availability typically refers to designs targeting availability of 99.999 percent (“5 nines”) or greater.
- By definition or convention, only unscheduled outages contribute to system downtime.

Availability
Planning for Failure

- In fact, failure is not only an option, it's almost inevitable.
- What will make your system safe and available is planning for the occurrence of failure or (more likely) failures, and handling them with aplomb.
  - The first step is to understand what kinds of failures your system is prone to, and what the consequences of each will be.
  - Here are three well-known techniques for getting a handle on this.

Availability
Planning for Failure

- Three well-known techniques for analyzing failures
  - Hazard analysis
    - Catastrophic
    - Hazardous
    - Major
    - Minor
    - No effect
  - Fault tree analysis
  - Failure Mode, Effects, and Criticality Analysis (FMECA)

Availability
Hazard Analysis

- Hazard analysis is a technique that attempts to catalog the hazards that can occur during the operation of a system.
  - It categorizes each hazard according to its severity.
  - For example, the DO-178B standard used in the aeronautics industry defines these failure condition levels in terms of their effects on the aircraft, crew, and passengers:
    - Catastrophic
    - Hazardous
    - Major
    - Minor
    - No effect

Availability
Hazard Analysis

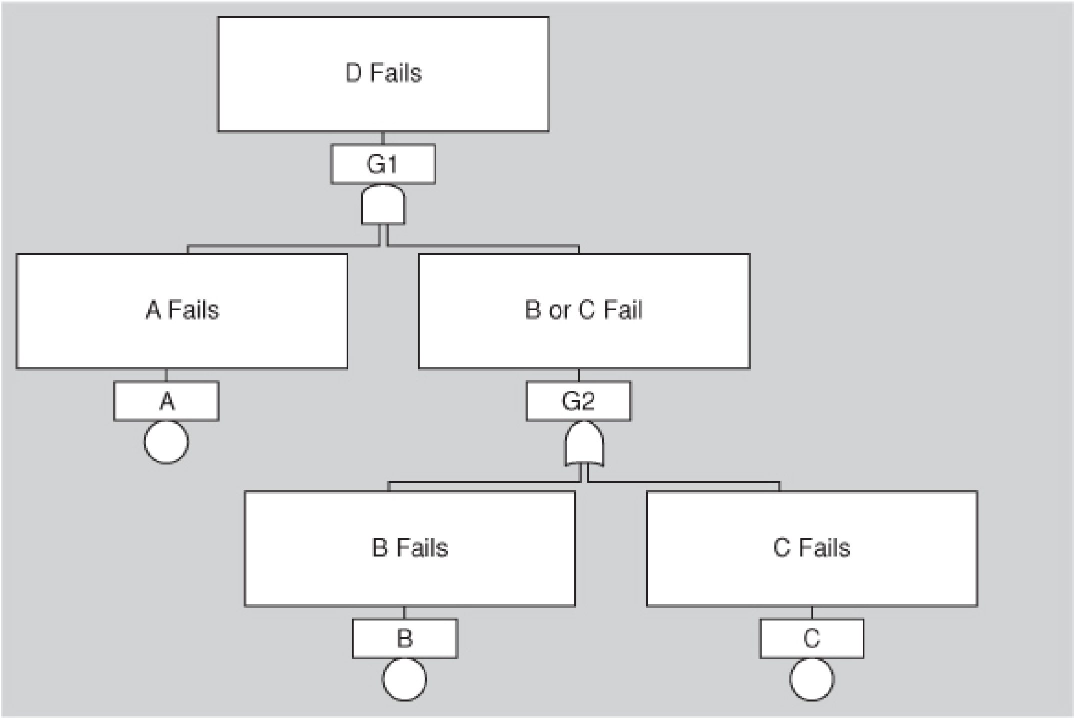
- Other domains have their own categories and definitions.
- Hazard analysis also assesses the probability of each hazard occurring.
  - Hazards for which the product of cost and probability exceed some threshold are then made the subject of mitigation activities.

Availability
Fault Tree Analysis

- Fault tree analysis is an analytical technique that specifies a state of the system that negatively impacts safety or reliability, and then analyzes the system's context and operation to find all the ways that the undesired state could occur.

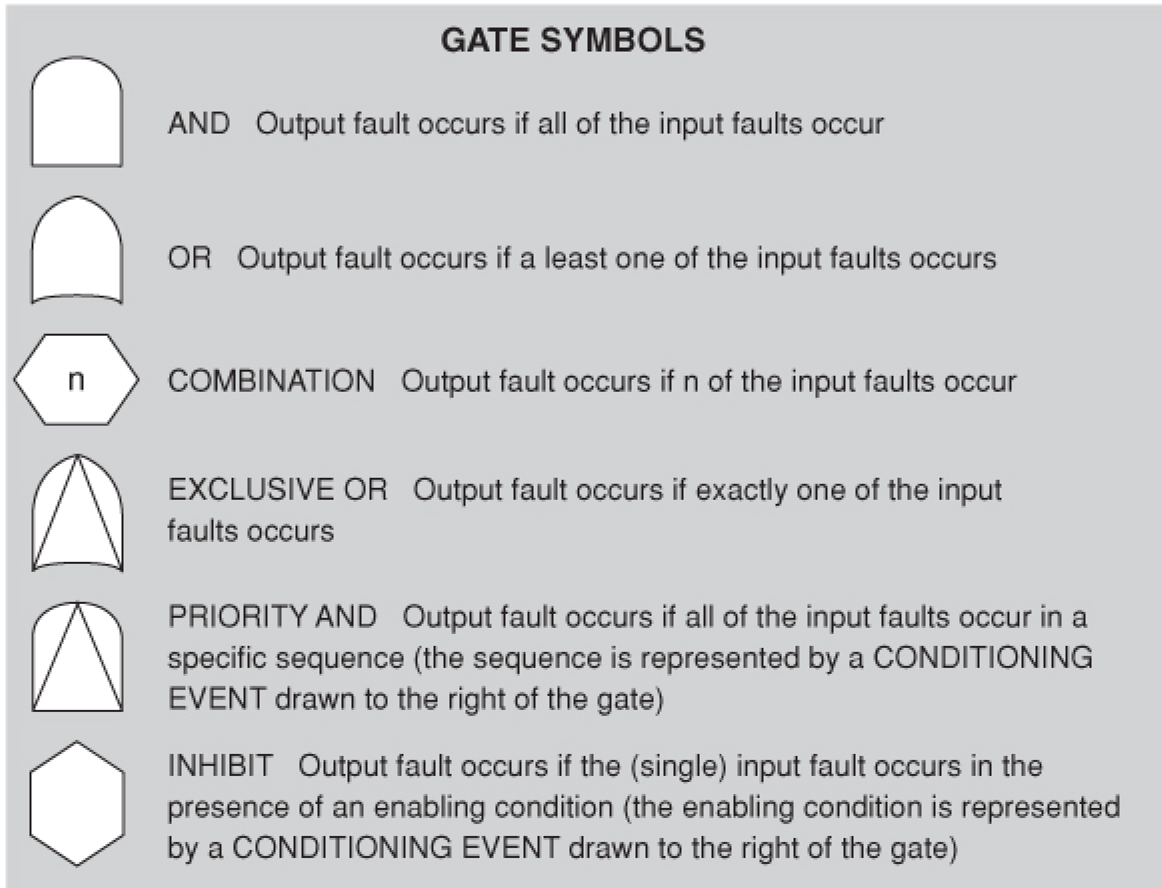
Availability
Fault Tree Analysis
Figure 5.1. A simple fault tree. D fails if A fails and either B or C fails.

- The technique uses a graphic construct (the fault tree) that helps identify all sequential and parallel sequences of contributing faults that will result in the occurrence of the undesired state, which is listed at the top of the tree (the "top event").
- The contributing faults might be hardware failures, human errors, software errors, or any other pertinent events that can lead to the undesired state.



Availability
Fault Tree Analysis
Figure 5.2. Fault tree gate symbols

- The symbols that connect the events in a fault tree are called gate symbols, and are taken from Boolean logic diagrams.



Availability
Fault Tree Analysis

- Fault trees aid in system design, but they can also be used to diagnose failures at runtime.
  - If the top event has occurred, then (assuming the fault tree model is complete) one or more of the contributing failures has occurred, and the fault tree can be used to track it down and initiate repairs.

Availability
Failure Mode, Effects, and Criticality Analysis (FMECA)

- Failure Mode, Effects, and Criticality Analysis (FMECA) catalogs the kinds of failures that systems of a given type are prone to, along with how severe the effects of each one can be.
- FMECA relies on the history of failure of similar systems in the past.

Availability

Failure Mode, Effects, and Criticality Analysis (FMECA)

Table 5.2. Failure Probabilities and Effects

- This figure shows the data for a system of redundant amplifiers. Historical data shows that amplifiers fail most often when there is a short circuit or the circuit is left open, but there are several other failure modes as well (lumped together as “Other”).

Component	Failure Probability	Failure Mode	% Failures by Mode	Effects: Critical	Effects: Noncritical
		Open	90		X
A	1×10^{-3}	Short	5	X (5×10^{-5})	
		Other	5	X (5×10^{-5})	
B	1×10^{-3}	Open	90		X
		Short	5	X (5×10^{-5})	
		Other	5	X (5×10^{-5})	

- Don’t let safety engineering become a matter of just filling out the tables. Instead, keep pressing to find out what else can go wrong, and then plan for it.

AVAILABILITY GENERAL SCENARIO

Availability General Scenario

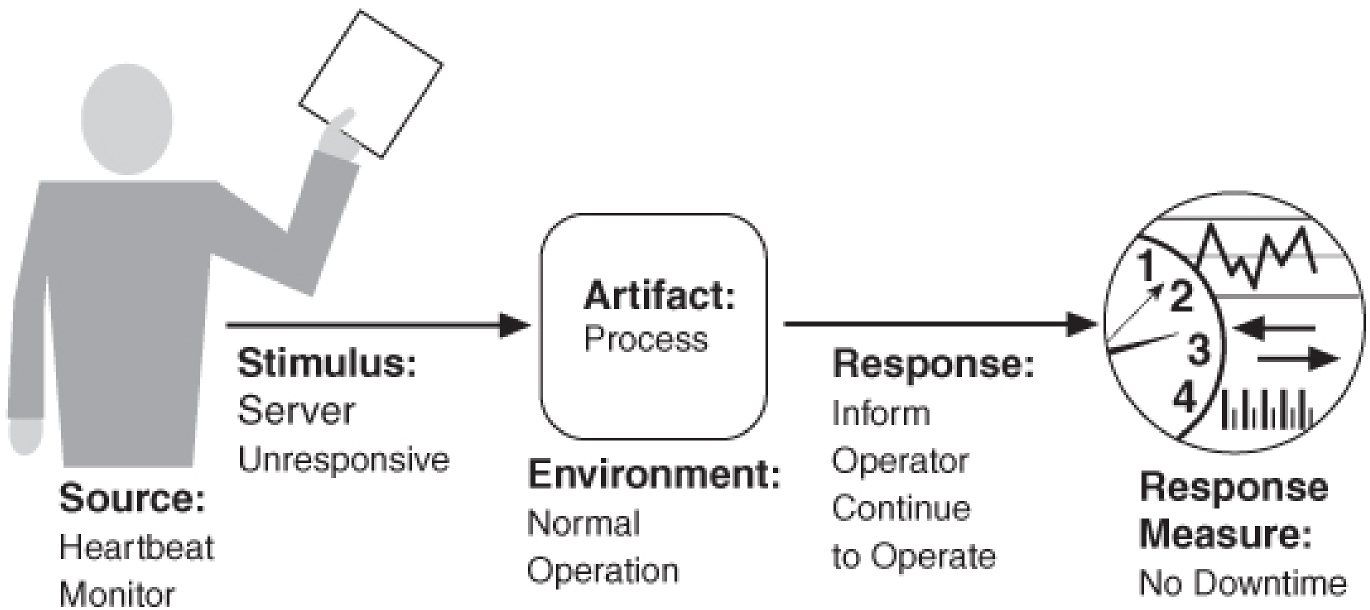
Table 5.3. Availability General Scenario

Portion of Scenario	Possible Values
Source	Internal/external: people, hardware, software, physical infrastructure, physical environment
Stimulus	Fault: omission, crash, incorrect timing, incorrect response
Artifact	Processors, communication channels, persistent storage, processes
Environment	Normal operation, startup, shutdown, repair mode, degraded operation, overloaded operation
Response	Prevent the fault from becoming a failure Detect the fault: ▪ Log the fault ▪ Notify appropriate entities (people or systems) Recover from the fault: ▪ Disable source of events causing the fault ▪ Be temporarily unavailable while repair is being effected ▪ Fix or mask the fault/failure or contain the damage it causes ▪ Operate in a degraded mode while repair is being effected
Response Measure	Time or time interval when the system must be available Availability percentage (e.g., 99.999%) Time to detect the fault Time to repair the fault Time or time interval in which system can be in degraded mode Proportion (e.g., 99%) or rate (e.g., up to 100 per second) of a certain class of faults that the system prevents, or handles without failing

Availability General Scenario

Figure 5.3. Sample concrete availability scenario

- The heartbeat monitor determines that the server is nonresponsive during normal operations. The system informs the operator and continues to operate with no downtime.

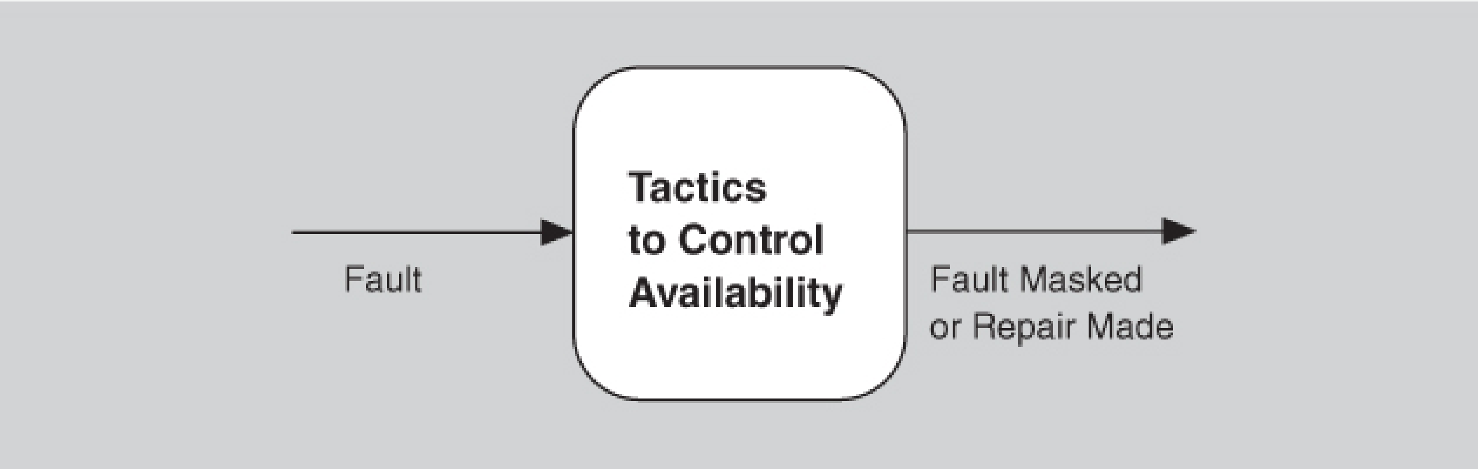


TACTICS FOR AVAILABILITY

Tactics for Availability

- A failure occurs when the system no longer delivers a service that is consistent with its specification; this failure is observable by the system's actors.
  - A fault (or combination of faults) has the potential to cause a failure.
- Availability tactics, therefore, are designed to enable a system to endure system faults so that a service being delivered by the system remains compliant with its specification.
- The tactics we discuss in this section will keep faults from becoming failures or at least bound the effects of the fault and make repair possible. (Figure 5.4)

Tactics for Availability
Figure 5.4. Goal of availability tactics

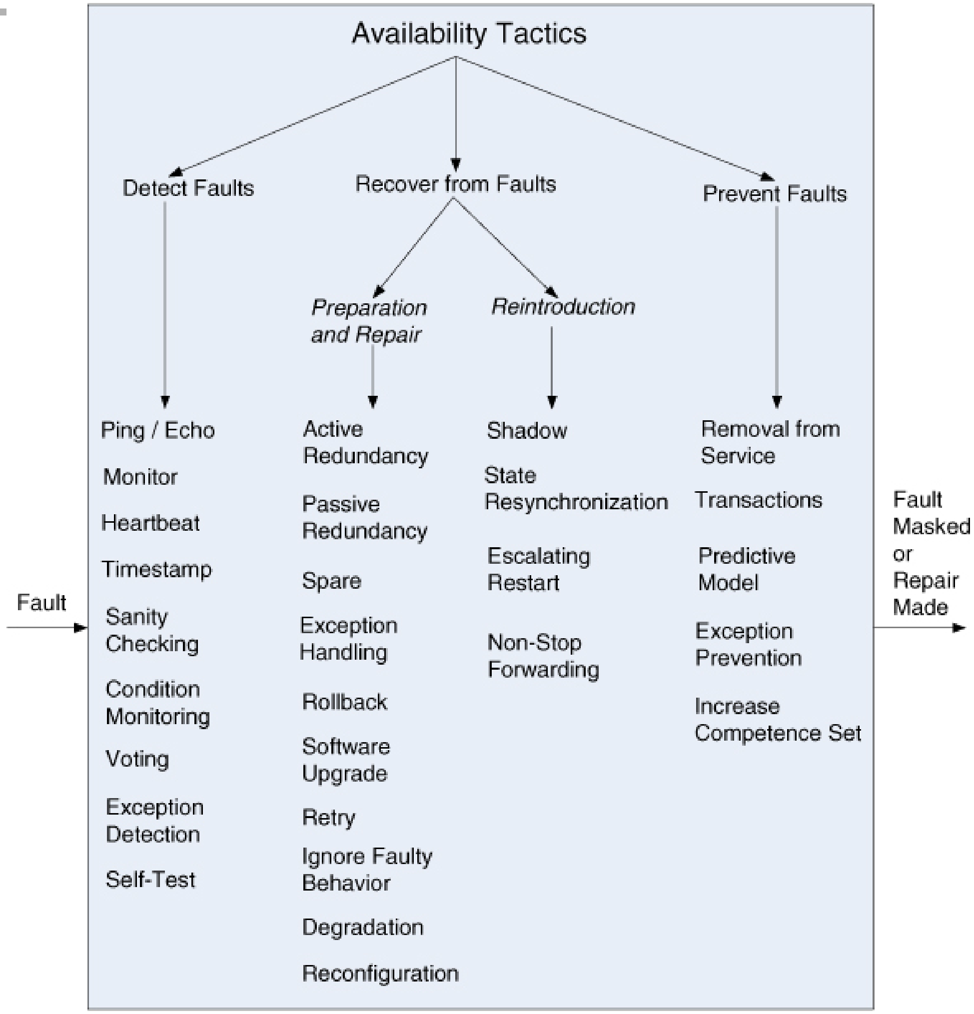


Tactics for Availability

- Availability tactics may be categorized as addressing one of three categories:
  - Fault detection
  - Fault recovery
  - Fault prevention
- The tactics categorization for availability is shown in Figure 5.5.

Tactics for Availability Figure 5.5. Availability tactics

- Note that it is often the case that these tactics will be provided for you by a software infrastructure, such as a middleware package, so your job as an architect is often one of choosing and assessing (rather than implementing) the right availability tactics and the right combination of tactics.



Tactics for Availability
Detect Faults

- Before any system can take action regarding a fault, the presence of the fault must be detected or anticipated.
- Tactics in this category include the following:
  - Ping/echo
  - Monitor
  - Heartbeat
  - Time stamp
  - Sanity checking
  - Condition monitoring
  - Voting (Replication, Functional redundancy, Analytic redundancy)
  - Exception detection (System exceptions, Parameter fence, Parameter typing, Timeout)
  - Self-test

Tactics for Availability
Recover from Faults

- <u>Recover-from-faults tactics</u> are refined into <u>preparation-and-repair tactics</u> and <u>reintroduction tactics</u>.
  - <u>Preparation-and-repair tactics</u> are based on a variety of combinations of <u>retrying a computation or introducing redundancy</u>.
  - <u>Reintroduction tactics</u> are concerned with <u>reintroducing a failed (but rehabilitated) component</u> <u>back into normal operation</u>.

Tactics for Availability Recover from Faults

- Preparation-and-repair tactics:
  - Active redundancy (hot spare)
  - Passive redundancy (warm spare)
  - Spare (cold spare)
  - Exception handling
  - Rollback
  - Software upgrade
  - Retry
  - Ignore faulty behavior
  - degradation
  - Reconfiguration
- Reintroduction tactics:
  - Shadow
  - State resynchronization
  - Escalating restart
  - Non-stop forwarding (NSF)

Tactics for Availability Recover from Faults

- Preparation-and-repair tactics are based on a variety of combinations of retrying a computation or introducing redundancy.
  - Active redundancy (hot spare)
  - Passive redundancy (warm spare)
  - Spare (cold spare)
  - Exception handling
  - Rollback
  - Software upgrade
  - Retry
  - Ignore faulty behavior
  - degradation
  - Reconfiguration

Tactics for Availability
Recover from Faults

- Reintroduction is where a failed component is reintroduced after it has been corrected.
- Reintroduction tactics include the following:
  - Shadow
  - State resynchronization
  - Escalating restart
  - Non-stop forwarding (NSF)

Tactics for Availability
Prevent Faults

- Instead of detecting faults and then trying to recover from them, what if your system could prevent them from occurring in the first place?
- These tactics deal with runtime means to prevent faults from occurring.
- This can be done by means of code inspections, pair programming, solid requirements reviews, and a host of other good engineering practices.

Tactics for Availability
Prevent Faults

- Tactics for Fault Prevention
  - Removal from service
  - Transactions
  - Predictive model
  - Exception prevention
  - Increase competence set

A DESIGN CHECKLIST FOR AVAILABILITY

A Design Checklist for Availability

- A design checklist is to support the design and analysis process for quality attributes.
  - Table 5.4 is a checklist to support the design and analysis process for availability.

A Design Checklist for Availability

Table 5.4. Checklist to Support the Design and Analysis Process for Availability

Category	Checklist
Allocation of Responsibilities	Determine the system responsibilities that need to be highly available. Within those responsibilities, ensure that additional responsibilities have been allocated to detect an omission, crash, incorrect timing, or incorrect response. Additionally, ensure that there are responsibilities to do the following: ▪ Log the fault ▪ Notify appropriate entities (people or systems) ▪ Disable the source of events causing the fault ▪ Be temporarily unavailable ▪ Fix or mask the fault/failure ▪ Operate in a degraded mode

A Design Checklist for Availability

Table 5.4. Checklist to Support the Design and Analysis Process for Availability

Category	Checklist
Coordination Model	Determine the system responsibilities that need to be highly available. With respect to those responsibilities, do the following: ▪ Ensure that coordination mechanisms can detect an omission, crash, incorrect timing, or incorrect response. Consider, for example, whether guaranteed delivery is necessary. Will the coordination work under conditions of degraded communication? ▪ Ensure that coordination mechanisms enable the logging of the fault, notification of appropriate entities, disabling of the source of the events causing the fault, fixing or masking the fault, or operating in a degraded mode. ▪ Ensure that the coordination model supports the replacement of the artifacts used (processors, communications channels, persistent storage, and processes). For example, does replacement of a server allow the system to continue to operate? ▪ Determine if the coordination will work under conditions of degraded communication, at startup/shutdown, in repair mode, or under overloaded operation. For example, how much lost information can the coordination model withstand and with what consequences?

A Design Checklist for Availability

Table 5.4. Checklist to Support the Design and Analysis Process for Availability

Category	Checklist
Data Model	Determine which portions of the system need to be highly available. Within those portions, determine which data abstractions, along with their operations or their properties, could cause a fault of omission, a crash, incorrect timing behavior, or an incorrect response. For those data abstractions, operations, and properties, ensure that they can be disabled, be temporarily unavailable, or be fixed or masked in the event of a fault. For example, ensure that write requests are cached if a server is temporarily unavailable and performed when the server is returned to service.

A Design Checklist for Availability

Table 5.4. Checklist to Support the Design and Analysis Process for Availability

Category	Checklist
Mapping among Architectural Elements	Determine which artifacts (processors, communication channels, persistent storage, or processes) may produce a fault: omission, crash, incorrect timing, or incorrect response. Ensure that the mapping (or remapping) of architectural elements is flexible enough to permit the recovery from the fault. This may involve a consideration of the following: ▪ Which processes on failed processors need to be reassigned at runtime ▪ Which processors, data stores, or communication channels can be activated or reassigned at runtime ▪ How data on failed processors or storage can be served by replacement units ▪ How quickly the system can be reinstalled based on the units of delivery provided ▪ How to (re)assign runtime elements to processors, communication channels, and data stores ▪ When employing tactics that depend on redundancy of functionality, the mapping from modules to redundant components is important. For example, it is possible to write one module that contains code appropriate for both the active component and backup components in a protection group.

A Design Checklist for Availability

Table 5.4. Checklist to Support the Design and Analysis Process for Availability

Category	Checklist
Resource Management	Determine what critical resources are necessary to continue operating in the presence of a fault: omission, crash, incorrect timing, or incorrect response. Ensure there are sufficient remaining resources in the event of a fault to log the fault; notify appropriate entities (people or systems); disable the source of events causing the fault; be temporarily unavailable; fix or mask the fault/failure; operate normally, in startup, shutdown, repair mode, degraded operation, and overloaded operation. Determine the availability time for critical resources, what critical resources must be available during specified time intervals, time intervals during which the critical resources may be in a degraded mode, and repair time for critical resources. Ensure that the critical resources are available during these time intervals. For example, ensure that input queues are large enough to buffer anticipated messages if a server fails so that the messages are not permanently lost.

A Design Checklist for Availability

Table 5.4. Checklist to Support the Design and Analysis Process for Availability

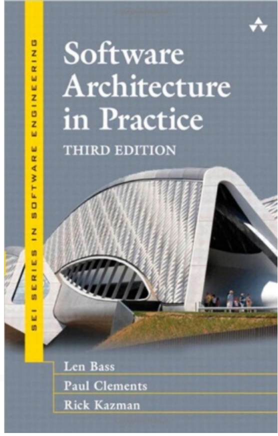
Category	Checklist
Binding Time	Determine how and when architectural elements are bound. If late binding is used to alternate between components that can themselves be sources of faults (e.g., processes, processors, communication channels), ensure the chosen availability strategy is sufficient to cover faults introduced by all sources. For example: ▪ If late binding is used to switch between artifacts such as processors that will receive or be the subject of faults, will the chosen fault detection and recovery mechanisms work for all possible bindings? ▪ If late binding is used to change the definition or tolerance of what constitutes a fault (e.g., how long a process can go without responding before a fault is assumed), is the recovery strategy chosen sufficient to handle all cases? For example, if a fault is flagged after 0.1 milliseconds, but the recovery mechanism takes 1.5 seconds to work, that might be an unacceptable mismatch. ▪ What are the availability characteristics of the late binding mechanism itself? Can it fail?

A Design Checklist for Availability

Table 5.4. Checklist to Support the Design and Analysis Process for Availability

Category	Checklist
Choice of Technology	Determine the available technologies that can (help) detect faults, recover from faults, or reintroduce failed components.
	Determine what technologies are available that help the response to a fault (e.g., event loggers).
	Determine the availability characteristics of chosen technologies themselves: What faults can they recover from? What faults might they introduce into the system?

References



- Bass, Len, Clements, Paul, and Kazman, Rick. *Software Architecture in Practice*, (3rd Edition). Addison-Wesley Professional, 2012. (ISBN-13: 978-0-321-81573-6) (ISBN-10: 0-321-81573-4)