Software Systems Architecture

Quality Attributes

Poștaru Andrei

Understanding Quality Attributes

Quality Attribute (QA) is a measurable or testable property of a system that is used to indicate how well the system satisfies the needs of its stakeholders beyond the basic function of the system

Quality Attribute Considerations

Problems with most discussions of system quality attributes:

1. The definitions provided for an attribute are not testable. It is meaningless to say that a system will be “modifiable.” Every system will be modifiable with respect to one set of changes and not modifiable with respect to another.
2. Discussion often focuses on which quality a particular issue belongs to. Is a denial-of-service attack on a system an aspect of availability, an aspect of performance, an aspect of security, or an aspect of usability?

Quality Attribute Considerations

3. Each attribute community has developed its own vocabulary. The performance community has “events” arriving at a system, the security community has “attacks” arriving at a system, the availability community has “faults” arriving, and the usability community has “user input.” All of these may actually refer to the same occurrence, but they are described using different terms.

Quality Attribute Considerations

A solution to the first two problems (untestable definitions and overlapping issues) is to use quality attribute scenarios as a means of characterizing quality attributes. A solution to the third problem is to illustrate the concepts that are fundamental to that attribute community in a common form.

Quality Attribute Scenarios

We use a common form to specify all QA requirements as scenarios.

Quality attribute scenarios have six parts:

Quality Attribute Scenarios

- *Stimulus.* We use the term "stimulus" to describe an event arriving at the system or the project. The stimulus can be an *event* to the performance community, a *user operation* to the usability community, or an *attack* to the security community, and so forth. We use the same term to describe a motivating action for developmental qualities. Thus, a stimulus for modifiability is a request for a modification; a stimulus for testability is the completion of a unit of development.
- *Stimulus source.* A stimulus must have a source—it must come from somewhere. Some entity (a human, a computer system, or any other actor) must have generated the stimulus. The source of the stimulus may affect how it is treated by the system. A request from a trusted user will not undergo the same scrutiny as a request by an untrusted user.

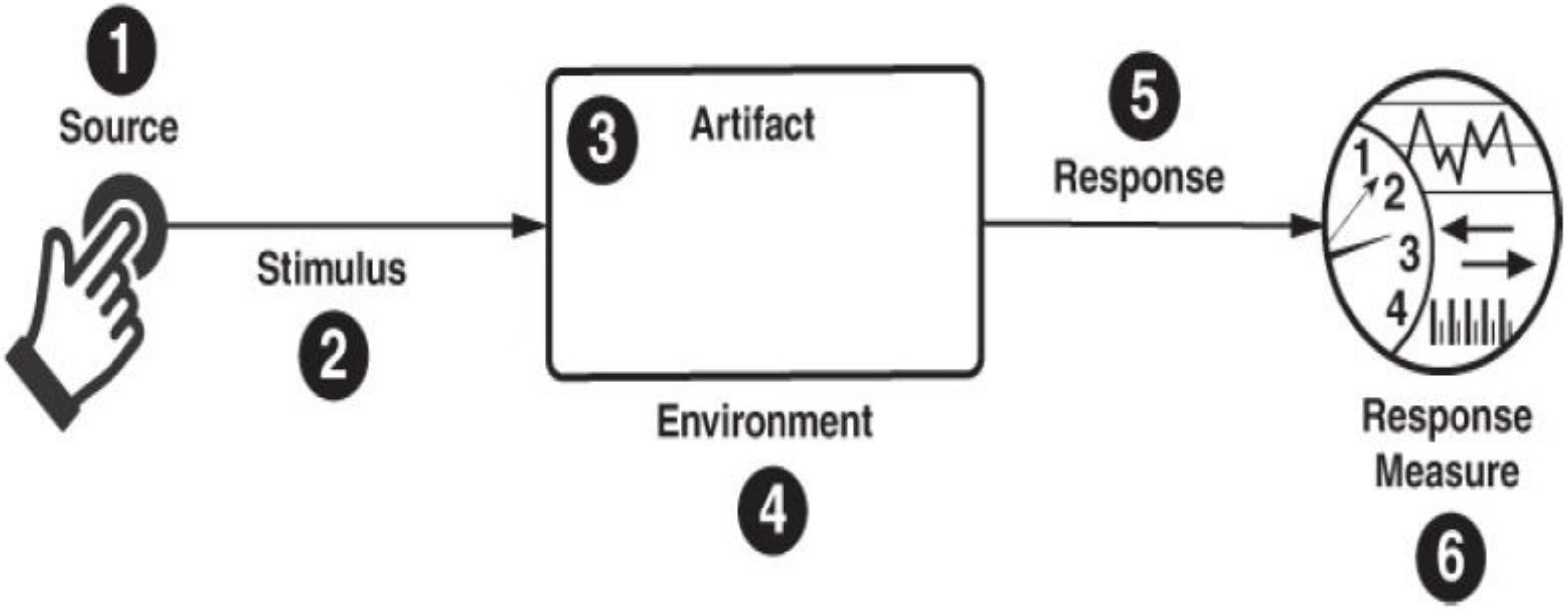
Quality Attribute Scenarios

- *Response.* The response is the activity that occurs as the result of the arrival of the stimulus. The response is something the architect undertakes to satisfy. It consists of the responsibilities that the system (for runtime qualities) or the developers (for development-time qualities) should perform in response to the stimulus. For example, in a performance scenario, an event arrives (the stimulus) and the system should process that event and generate a response. In a modifiability scenario, a request for a modification arrives (the stimulus) and the developers should implement the modification—without side effects—and then test and deploy the modification.
- *Response measure.* When the response occurs, it should be measurable in some fashion so that the scenario can be tested—that is, so that we can determine if the architect achieved it. For performance, this could be a measure of latency or throughput; for modifiability, it could be the labor or wall clock time required to make, test, and deploy the modification.

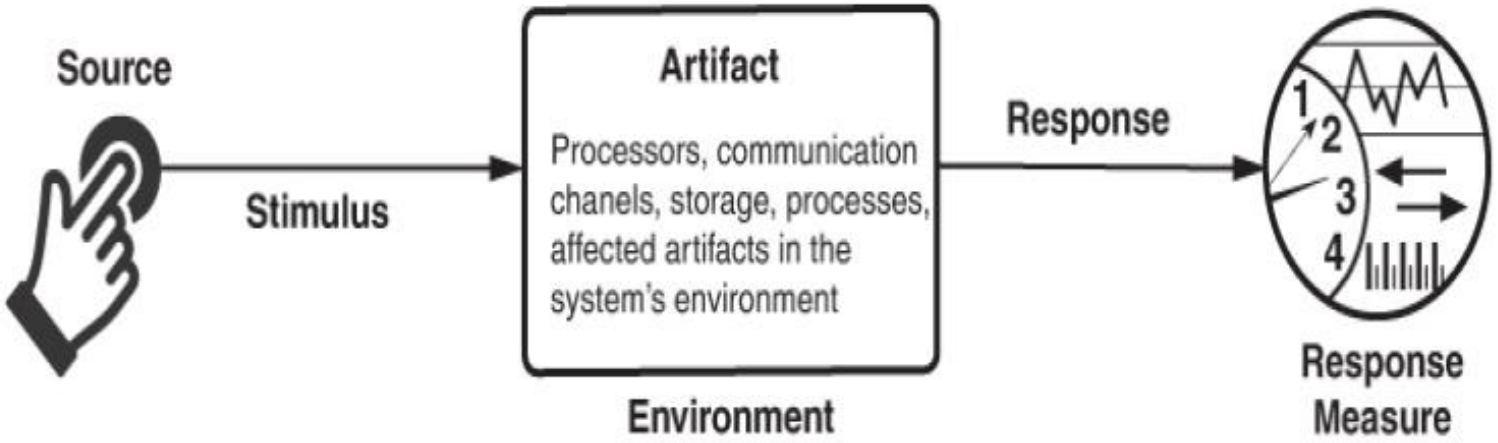
Quality Attribute Scenarios

- *Environment.* The environment is the set of circumstances in which the scenario takes place. Often this refers to a runtime state: The system may be in an overload condition or in normal operation, or some other relevant state. For many systems, "normal" operation can refer to one of a number of modes. For these kinds of systems, the environment should specify in which mode the system is executing. But the environment can also refer to states in which the system is not running at all: when it is in development, or testing, or refreshing its data, or recharging its battery between runs. The environment sets the context for the rest of the scenario. For example, a request for a modification that arrives after the code has been frozen for a release may be treated differently than one that arrives before the freeze. The fifth successive failure of a component may be treated differently than the first failure of that component.
- *Artifact.* The stimulus arrives at some target. This is often captured as just the system or project itself, but it's helpful to be more precise if possible. The artifact may be a collection of systems, the whole system, or one or more pieces of the system. A failure or a change request may affect just a small portion of the system. A failure in a data store may be treated differently than a failure in the metadata store. Modifications to the user interface may have faster response times than modifications to the middleware.

The parts of a quality attribute scenario



A general scenario for availability



Source	Stimulus	Environment	Response	Response Measure
Internal/external: people, hardware, software, physical infrastructure, physical environment	Fault: omission, crash, incorrect timing, incorrect response	Normal operation, startup, shutdown, repair mode, degraded operation, overloaded operation	Prevent the fault from becoming a failure Detect the fault Recover from the fault	Time or time interval when the system must be available Availability perentage Time to detect the fault

Quality Attributes

1. Availability
2. Deployability
3. Energy Eficiency
4. Integrability
5. Modifiability
6. Performance
7. Safety
8. Security
9. Testability
10. Usability

Availability

Availability refers to a property of software—namely, that it is there and ready to carry out its task when you need it to be. This is a broad perspective and encompasses what is normally called reliability (although it may encompass additional considerations such as downtime due to periodic maintenance). Availability builds on the concept of reliability by adding the notion of recovery—that is, when the system breaks, it repairs itself. Repair may be accomplished by various means

Availability

A **failure** is the deviation of the system from its specification, where that deviation is externally visible. Determining that a failure has occurred requires some external observer in the environment.

A failure's cause is called a **fault**. A fault can be either internal or external to the system under consideration. Intermediate states between the occurrence of a fault and the occurrence of a failure are called errors. Faults can be prevented, tolerated, removed, or forecast.

Availability

Distinguishing between faults and failures allows us to discuss repair strategies. If code containing a fault is executed but the system is able to recover from the fault without any observable deviation from the otherwise specified behavior, we say that no failure has occurred.

System Availability Requirements

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

Availability General Scenario

Portion of Scenario	Description	Possible Values
Source	This specifies where the fault comes from.	Internal/external: people, hardware, software, physical infrastructure, physical environment
Stimulus	The stimulus to an availability scenario is a fault.	Fault: omission, crash, incorrect timing, incorrect response
Artifact	This specifies which portions of the system are responsible for and affected by the fault.	Processors, communication channels, storage, processes, affected artifacts in the system's environment
Environment	We may be interested in not only how a system behaves in its "normal" environment, but also how it behaves in situations such as when it is already recovering from a fault.	Normal operation, startup, shutdown, repair mode, degraded operation, overloaded operation

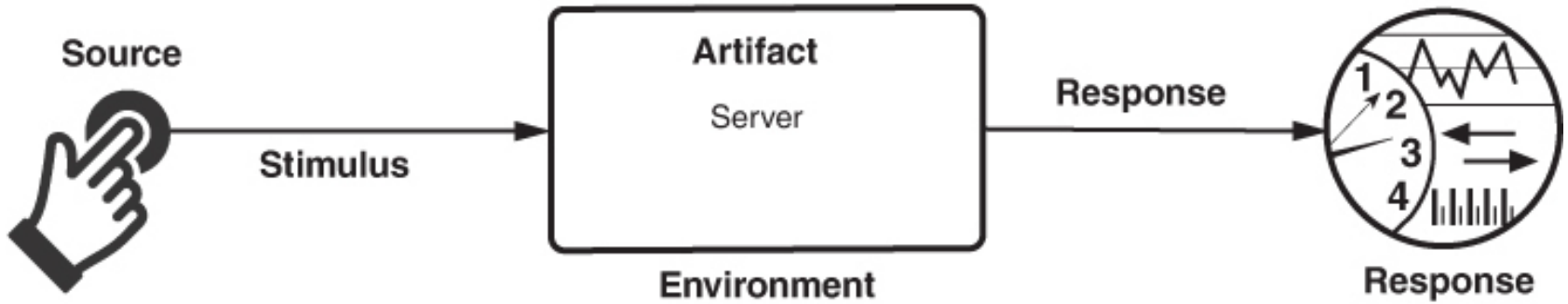
Availability General Scenario

Response	The most commonly desired response is to prevent the fault from becoming a failure, but other responses may also be important, such as notifying people or logging the fault for later analysis. This section specifies the desired system response.	Prevent the fault from becoming a failure Detect the fault: ▪ Log the fault ▪ Notify the appropriate entities (people or systems) ▪ Recover from the fault ▪ Disable the source of events causing the fault ▪ Be temporarily unavailable while a repair is being effected ▪ Fix or mask the fault/failure or contain the damage it causes ▪ Operate in a degraded mode while a repair is being effected

Availability General Scenario

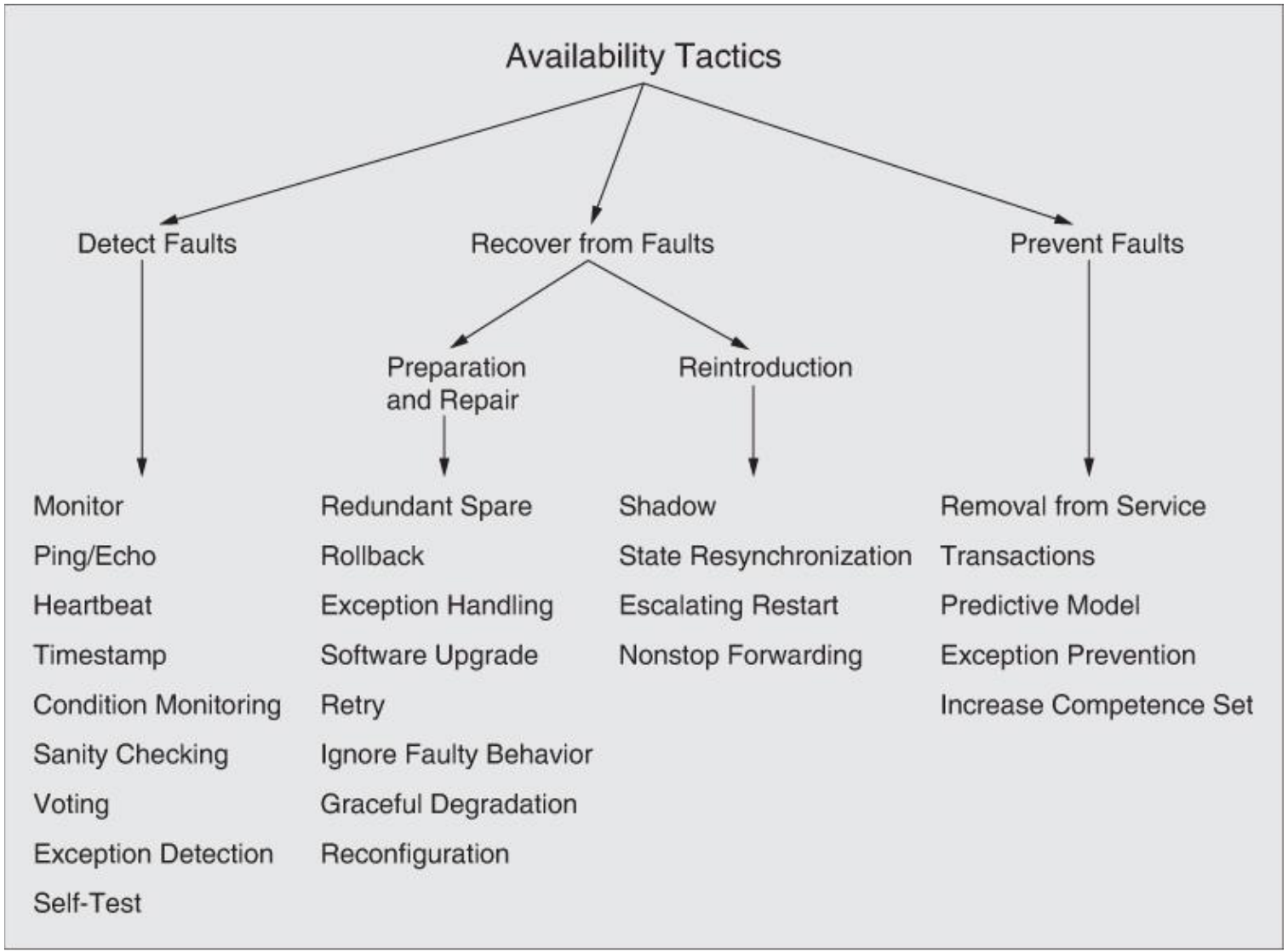
Response measure	We may focus on a number of measures of availability, depending on the criticality of the service being provided.	▪ Time or time interval when the system must be available ▪ Availability percentage (e.g., 99.999 per cent)
		▪ Time to detect the fault ▪ Time to repair the fault ▪ Time or time interval in which system can be in degraded mode ▪ Proportion (e.g., 99 percent) 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

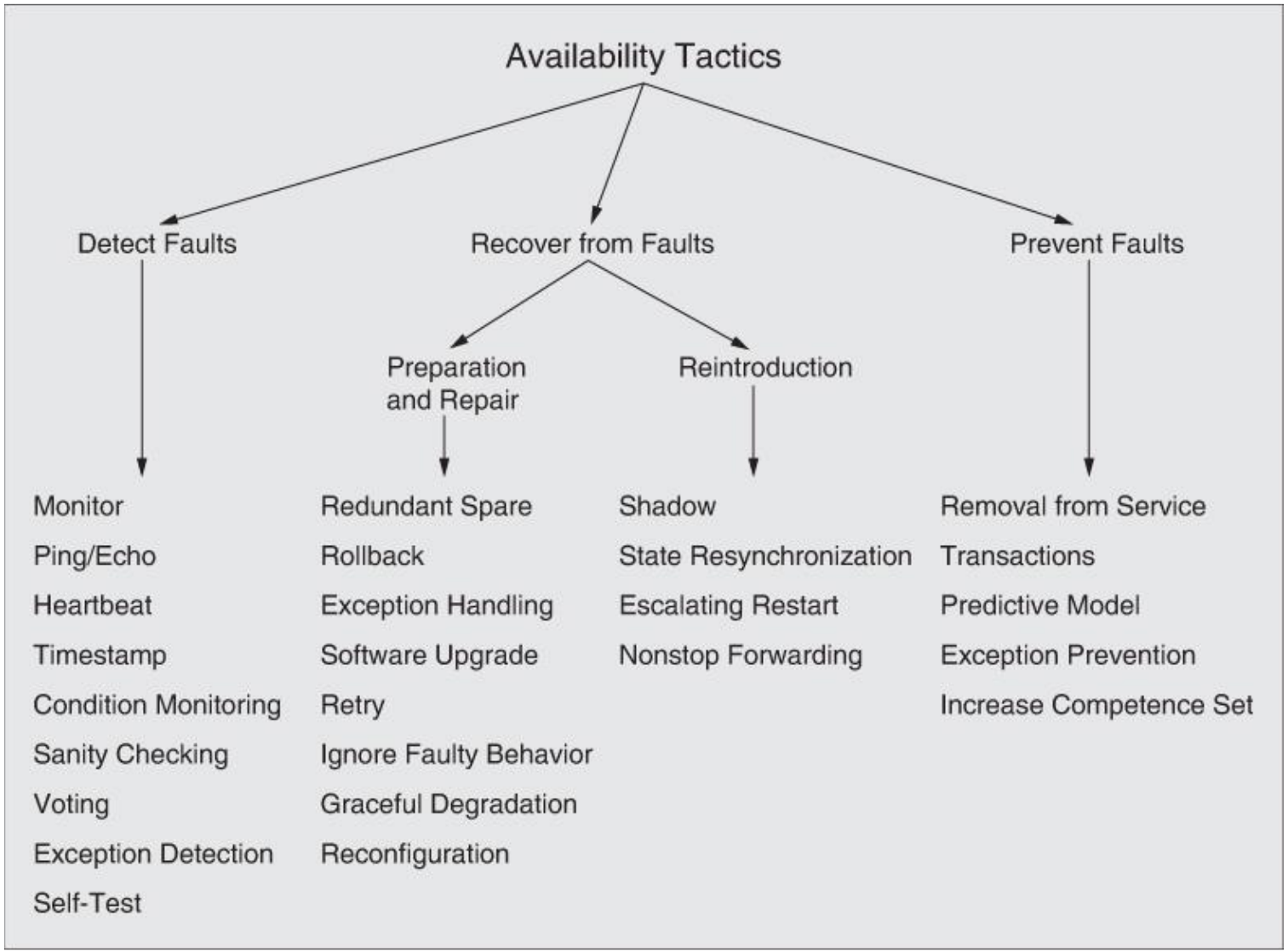


Source	Stimulus	Environment	Response	Response Measure
Server in a server farm	Server fails	Normal operation	System informs operator System continues to operate	No downtime

Tactics for Availability



Tactics for Availability



Patterns for Availability

- *Active redundancy (hot spare).* For stateful components, this refers to a configuration in which all of the nodes (active or redundant spare) in a protection group[4] receive and process identical inputs in parallel, allowing the redundant spare(s) to maintain a synchronous state with the active node(s). Because the redundant spare possesses an identical state to the active processor, it can take over from a failed component in a matter of milliseconds. The simple case of one active node and one redundant spare node is commonly referred to as one-plus-one redundancy. Active redundancy can also be used for facilities protection, where active and standby network links are used to ensure highly available network connectivity.

[4] A protection group is a group of processing nodes in which one or more nodes are "active," with the remaining nodes serving as redundant spares.

Patterns for Availability

- *Passive redundancy (warm spare)*. For stateful components, this refers to a configuration in which only the active members of the protection group process input traffic. One of their duties is to provide the redundant spare(s) with periodic state updates. Because the state maintained by the redundant spares is only loosely coupled with that of the active node(s) in the protection group (with the looseness of the coupling being a function of the period of the state updates), the redundant nodes are referred to as warm spares. Passive redundancy provides a solution that achieves a balance between the more highly available but more compute-intensive (and expensive) active redundancy pattern and the less available but significantly less complex cold spare pattern (which is also significantly cheaper).
- *Spare (cold spare)*. Cold sparing refers to a configuration in which redundant spares remain out of service until a failover occurs, at which point a power-on-reset[5] procedure is initiated on the redundant spare prior to its being placed in service. Due to its poor recovery performance, and hence its high mean time to repair, this pattern is poorly suited to systems having high-availability requirements.

[5] A power-on-reset ensures that a device starts operating in a known state.

Patterns for Availability

Benefits:

- The benefit of a redundant spare is a system that continues to function correctly after only a brief delay in the presence of a failure. The alternative is a system that stops functioning correctly, or stops functioning altogether, until the failed component is repaired. This repair could take hours or days.

Tradeoffs:

- The tradeoff with any of these patterns is the additional cost and complexity incurred in providing a spare.
- The tradeoff among the three alternatives is the time to recover from a failure versus the runtime cost incurred to keep a spare up-to-date. A hot spare carries the highest cost but leads to the fastest recovery time, for example.