# [B](#_bookmark5)



## [Tactics-Based Questionnaires](#_bookmark5)

This appendix provides a set of tactics-based questionnaires for the seven most important quality attributes: availability, interoperability, modifiability, **perfor- mance, security, testability, and usability**. How do we know that these are the seven most important ones? This decision was based on an analysis of the qual- ity attributes that were elicited from stakeholders in more than 15 years of SEI ATAM data.

In addition to these “top seven”, we include a tactics-based questionnaire for DevOps, which is a combination of tactics from modifiability, availability, performance, and testability, to illustrate how simple it is to tailor such question- naires for your own use.

### [Using the Questionnaires](#_bookmark5)

These questionnaires coul**d be used by an analyst, who poses each question, in turn, to the architect and records the responses, as a means of conducting a lightweight architecture review. Alternatively, the questionnaires could be employed as a set of reflective questions, that you could, on your own, use to examine your architecture.**

In either case, to use these questionnaires, simply follow these four steps:

**247**

1o.r eaFch tactics question, fill the “Supported” column with Y if the tactic is supported in the architecture and with **N** otherwise. The tactic name in the “Tactics Question” column appears in **bold.**

1. If the answer in the “Supported” column is **Y**, then in the “Design Decisions and Location” column describe the specific design decisions made to support the tactic and enumerate where these decisions are manifested (located) in the architecture. For example, indicate which code modules, frameworks, or packages implement this tactic.
2. In the “Risk” column, indicate the anticipated/experienced difficulty or risk of implementing the tactic using a **(H = high, M = medium, L = low)** scale. For example, a tactic that was of medium difficulty or risk to implement (or which is anticipated to be of medium difficulty, if it has not yet been implemented) would be labeled M.
3. In the “Rationale” column, describe the rationale for the design decisions made (including a decision to *not* use this tactic). Briefly explain the implications of this decision. For example, you might explain the rationale and implications of the decision in terms of the effort on cost, schedule, evolution, and so forth.

### [Availability](#_bookmark5)

**Design Decisions**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Tactics**  **Group** | **Tactics Question** | **Supported?**  **(Y/N)** | **and**  **Risk Location** | **Rationale and**  **Assumptions** |
| 1 | Detect faults | Does the system use **ping/echo** to detect a failure of a component or connection, or network congestion? |  |  |  |
| 2 |  | Does the system use a component to **monitor** the state of health of other parts of the system? A system monitor can de- tect failure or congestion in the network or other shared resources, such as from a denial-of-ser- vice attack. |  |  |  |
| 3 |  | Does the system use a **heartbeat**—a periodic message exchange be- tween a system monitor and a process—to detect a failure of a component or connection, or network congestion? |  |  |  |
| 4 |  | Does the system use a **time stamp** (as in section A.4.1) to detect incorrect sequences of events in distributed systems? |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Tactics Group** | **Tactics Question** | **Supported? (Y/N)** | **Design Decisions and**  **Risk Location** | **Rationale and Assumptions** |
| 5 |  | Does the system do any **sanity checking:** checking the validity or reasonableness of a  component’s operations or outputs? |  |  |  |
| 6 |  | Does the system do **condition monitoring,** checking conditions in a process or device, or vali- dating assumptions made during the design? |  |  |  |
| 7 |  | Does the system use **voting** to check that replicated components are producing the same results? The replicated components may be iden- tical replicas, functionally redundant, or analytically redundant. |  |  |  |
| 8 |  | Do you use **exception de- tection** to detect a system condition that alters the normal flow of execution (e.g., system exception, parameter fence, parame- ter typing, timeout)? |  |  |  |
| 9 |  | Can the system do a **self-test** to test itself for correct operation? |  |  |  |
| 10 | Recover from faults (prepara- tion and repair) | Does the system employ **active redundancy** (hot spare)? In active redun- dancy, all nodes in a pro- tection group (a group of nodes where one or more nodes are “active”, with the remainder serving as redundant spares) receive and process identical inputs in parallel, allowing redundant spares to main- tain synchronous state with the active node(s). |  |  |  |
| 11 |  | Does the system employ **passive redundancy** (warm spare)? In passive redundancy, 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. |  |  |  |

(*continues*)

**Design Decisions**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Tactics**  **Group** | **Tactics Question** | **Supported?**  **(Y/N)** | **and**  **Risk Location** | **Rationale and**  **Assumptions** |
| 12 |  | Does the system employ **spares** (cold spares)? Here redundant spares of a protection group  remain out of service until a failover occurs, at which point a power-on-reset procedure is initiated  on the redundant spare prior to its being placed in service. |  |  |  |
| 13 |  | Does the system employ **exception handling** to deal with faults? Typically the handling involves either reporting the fault or handling it, potentially masking the fault by cor- recting the cause of the exception and retrying. |  |  |  |
| 14 |  | Does the system employ **rollback**, so that it can re- vert to a previously saved good state (the “rollback line”) in the event of a fault? |  |  |  |
| 15 |  | Can the system perform in-service **software upgrades** to execut- able code images in a non-service-affecting manner? |  |  |  |
| 16 |  | Does the system system- atically **retry** in cases where the component or connection failure may be transient? |  |  |  |
| 17 |  | Can the system simply **ignore faulty behavior** (e.g., ignore messages sent from a source when it is determined that those messages are spurious)? |  |  |  |
| 18 |  | Does the system have a policy of **degradation**  when resources are com- promised, maintaining the most critical system functions in the presence of component failures, and dropping less critical functions? |  |  |  |

**Tactics**

**# Group Tactics Question**

**Supported?**

**(Y/N) Risk**

**Design Decisions and Location**

**Rationale and Assumptions**

1. Does the system have consistent policies and mechanisms for **recon- figuration** after failures, reassigning responsi- bilities to the resources left functioning, while maintaining as much functionality as possible?
2. Recover from faults (reintro- duction)

Can the system operate a previously failed or

in-service upgraded component in a “**shadow** mode” for a predefined time prior to reverting the component back to an active role?

1. If the system uses active or passive redundancy, does it also employ **state resynchronization**, to send state information from active to standby components?
2. Does the system employ **escalating restart**— that is, does it recover from faults by varying the granularity of the component(s) restarted and minimizing the level of service affected?
3. Can message process- ing and routing portions of the system employ **nonstop** (as in section A.4.1) **forwarding**, where functionality is split into supervisory and data planes? In this case, if a supervisor fails, a router continues forwarding packets along known routes while protocol information is recovered and validated.
4. Prevent faults

Can the system **remove components from ser- vice**, temporarily placing a system component in an out-of-service state, for the purpose of miti- gating potential system failures?

(*continues*)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Tactics Group** | **Tactics Question** | **Supported? (Y/N)** | **Design Decisions and**  **Risk Location** | **Rationale and Assumptions** |
| 25 |  | Does the system employ **transactions**—bundling state updates so that asynchronous messages exchanged between distributed components are *atomic*, *consistent*, *isolated*, and *durable*? |  |  |  |
| 26 |  | Does the system use a **predictive model** to monitor the state of  health of a component to ensure that the system is operating within nominal parameters? When condi- tions are detected that are predictive of likely future faults, the model initiates corrective action. |  |  |  |
| 27 |  | Does the system **pre- vent exceptions** from occurring by, for example, masking a fault, using smart pointers, abstract data types, or wrappers? |  |  |  |
| 28 |  | Has the system been designed to **increase its competence set,** for example by designing a  component to handle more cases—faults—as part of its normal operation? |  |  |  |

### [Interoperability](#_bookmark5)

**Tactics**

**# Group Tactics Question**

**Supported?**

**(Y/N) Risk**

**Design Decisions and Location**

**Rationale and Assumptions**

1. Locate Does the system

have

a way to **discover ser- vices** (typically through a directory service)?

1. Manage interfaces

Does the system have a way to **orchestrate** the activities of services? That is, does it have a control mechanism to coordinate, manage, and sequence the invo- cation of services?

**Tactics**

**# Group Tactics Question**

**Supported?**

**(Y/N) Risk**

**Design Decisions and Location**

**Rationale and Assumptions**

1. Does the system have a way to **tailor inter- faces**? For example, can it add or remove capabilities to an inter- face such as transla- tion, buffering, or data smoothing?

### [Modifiability](#_bookmark5)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Tactics Group** | **Tactics Question** | **Supported? (Y/N)** | **Design Decisions and**  **Risk Location** | **Rationale and Assumptions** |
| 1 | Reduce size of a module | Do you make modules simpler by **splitting the module**? For example, if you have a large, complex module, can you split it into two (or more) smaller, simpler modules? |  |  |  |
| 2 | Increase cohesion | Does the system consistently support **increasing semantic coherence**? For exam- ple, if responsibilities in a module do not serve the same purpose, they should be placed in different modules. This may involve creating a new module or moving a responsibility to an existing module. |  |  |  |
| 3 | Reduce coupling | Does the system con- sistently **encapsulate** functionality? This typ- ically involves isolating the functionality under scrutiny and introducing an explicit interface to it. |  |  |  |

(*continues*)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Tactics Group** | **Tactics Question** | **Supported? (Y/N)** | **Design Decisions and**  **Risk Location** | **Rationale and Assumptions** |
| 4 |  | Does the system consis- tently **use an interme- diary** to keep modules from being too tightly coupled? For example, if A calls concrete func- tionality C, you might in- troduce an abstraction B that mediates between A and C. |  |  |  |
| 5 |  | Do you **restrict de- pendencies** between modules in a systematic way? Or is any system module free to interact with any other module? |  |  |  |
| 6 |  | When two or more unrelated modules change together—that is, when they are regularly affected by the same changes—do you regularly **refactor** the functionality to isolate the shared functionality as common code in a distinct module? |  |  |  |
| 7 |  | Does the system **abstract common services**, in cases where you are providing several similar services? For example, this technique is often used when you want your system to be portable across operating sys- tems, hardware, or other environment variations. |  |  |  |
| 8 | Defer binding | Does the system regularly **defer binding** of important function- ality so that it can be replaced later in the life cycle, perhaps even  by end users? For example, do you use plug-ins, add-ons, or user scripting to extend the functionality of the system? |  |  |  |

### [Performance](#_bookmark5)

**Tactics**

**Supported?**

**Risk**

**Design Decisions and**

**Rationale and**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Group** | **Tactics Question** | **(Y/N)** | **Location** | **Assumptions** |
| 1 | Control resource demand | If your inputs are a continuous stream of data, does the system **manage the sampling rate**? That is, is it pos- sible to sample the data at varying rates (with concomitant changes in accuracy/fidelity)? |  |  |  |
| 2 |  | Does the system mon- itor and **limit its event response**? Does the system limit the number of events it responds  to in a time period, to ensure predictable responses for the events that are actually serviced? |  |  |  |
| 3 |  | Given that you may have more requests for service than available resources, does the sys- tem **prioritize events**? |  |  |  |
| 4 |  | Does the system **reduce the overhead** of responding to service requests by, for exam- ple, removing interme- diaries or co-locating resources? |  |  |  |
| 5 |  | Does the system monitor and **bound execution time**? More generally, do you bound the amount of any resource (e.g.,  memory, CPU, storage, bandwidth, connections, locks) expended in response to requests for services? |  |  |  |
| 6 |  | Do you **increase resource efficiency**? For example, do you regularly improve the efficiency of algorithms in critical areas, to decrease latency and improve throughput? |  |  |  |

(*continues*)

**Tactics**

**# Group Tactics Question**

**Supported? (Y/N)**

**Risk**

**Design Decisions and Location**

**Rationale and Assumptions**

1. Manage resources

Can the system seamlessly **increase resources** (e.g., CPU, memory, network band- width)?

1. Can the system **intro- duce concurrency**? For example, does it support the seamless addition of parallel pro- cessing streams so that more requests for ser- vices can be processed concurrently?
2. Does the system **main- tain multiple copies of data** (e.g., by replicating databases or using caches) to decrease contention for frequently accessed data?
3. Does the system **main- tain multiple copies of computations** (e.g., by keeping a pool of serv- ers in a server farm) to decrease contention

for frequently ac- cessed computational resources?

1. Does the system **bound queue sizes**? That is, do you limit the number of events placed in

a queue, waiting for services?

1. Does the system **sched- ule resources**, particu- larly scarce resources, so that they may be allocated according to an explicit scheduling policy?

### [Security](#_bookmark5)

**Tactics**

**Supported?**

**Design Decisions and**

**Rationale and**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Group** | **Tactics Question** | **(Y/N)** | **Risk Location** | **Assumptions** |
| 1 | Detecting attacks | Does the system sup- port the **detection of intrusions**? An example is comparing network traffic or service request patterns within a system to a set of signatures or known patterns of mali- cious behavior stored in a database. |  |  |  |
| 2 |  | Does the system support the **detection of denial-of-service attacks**? An example is the comparison of the pattern or signature of network traffic coming into a system to historic profiles of known deni- al-of-service attacks. |  |  |  |
| 3 |  | Does the system support the **verification of message integrity**? An example is the use of techniques such as checksums or hash val- ues to verify the integrity of messages, resource files, deployment files, and configuration files. |  |  |  |
| 4 |  | Does the system support the **detection of message delays**? An example is checking the time that it takes to deliver a message. |  |  |  |
| 5 | Resisting attacks | Does the system sup- port the **identification of actors**? An example is identifying the source of any external input to the system. |  |  |  |
| 6 |  | Does the system sup- port the **authentication of actors**? An example is ensuring that an actor (a user or a remote com- puter) is actually who or what it purports to be. |  |  |  |

(*continues*)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Tactics Group** | **Tactics Question** | **Supported? (Y/N)** | **Design Decisions and**  **Risk Location** | **Rationale and Assumptions** |
| 7 |  | Does the system support the **authorization of actors**? An example  is ensuring that an authenticated actor has the rights to access and modify either data or services. |  |  |  |
| 8 |  | Does the system support **limiting access**? An ex- ample is controlling what and who may access which parts of a system, such as processors, memory, and network connections. |  |  |  |
| 9 |  | Does the system support **limiting exposure**? An example is reducing the probability of a success- ful attack, or restricting the amount of potential damage, by concealing facts about a system (“security by obscurity”) or dividing and distrib- uting critical resources (“don’t put all your eggs in one basket”). |  |  |  |
| 10 |  | Does the system sup- port **data encryption**? An example is to apply some form of encryption to data and to commu- nication. |  |  |  |
| 11 |  | Does the system **validate input** in a consistent, system-wide way? An example is the use of a security frame- work or validation class to perform actions such as filtering, canonical- ization, and escaping of external input. |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Tactics Group** | **Tactics Question** | **Supported? (Y/N)** | **Design Decisions and**  **Risk Location** | **Rationale and Assumptions** |
| 12 |  | Does the system design consider the **separation of entities**? An example is the physical separa- tion of different servers attached to different networks, the use of virtual machines, or an “air gap”. |  |  |  |
| 13 |  | Does the system support **changes in the default settings**? An example is forcing the user to change settings assigned by default. |  |  |  |
| 14 | Reacting to attacks | Does the system sup- port **revoking access**? An example is limiting access to sensitive resources, even for nor- mally legitimate users and uses, if an attack is suspected. |  |  |  |
| 15 |  | Does the system sup- port **locking access**? An example is limiting access to a resource if there are repeated failed attempts to access it. |  |  |  |
| 16 |  | Does the system sup- port **informing actors**? An example is notifying operators, other person- nel, or cooperating sys- tems when an attack is suspected or detected. |  |  |  |
| 17 | Recover- ing from attacks | Does the system support **maintaining an audit trail**? An example is keeping a record of user and system actions and their effects, to help trace the actions of, and to identify, an attacker |  |  |  |

### [Testability](#_bookmark5)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Tactics Group** | **Tactics Question** | **Supported? (Y/N)** | **Design Decisions and**  **Risk Location** | **Rationale and Assumptions** |
| 1 | Control and observe system state | Does the system or the system components provide **specialized interfaces** to facilitate testing and monitoring? |  |  |  |
| 2 |  | Does the system provide mechanisms that allow information that crosses an interface to be recorded so that  it can be used later for testing purposes (**record/playback**)? |  |  |  |
| 3 |  | Is the state of the system, subsystem, or modules stored in a  single place to facilitate testing (**localized state storage**)? |  |  |  |
| 4 |  | Can you **abstract data sources**—for example, by abstracting inter- faces? Abstracting  the interfaces lets you substitute test data more easily. |  |  |  |
| 5 |  | Can the system be executed in isolation (a **sandbox**) to experiment or test it without worry- ing about having to undo the consequences of the experiment? |  |  |  |
| 6 |  | Are **executable asser- tions** used in the system code to indicate when and where a program is in a faulty state? |  |  |  |
| 7 | Limit complexity | Is the system designed in such a way that **structural complexity is limited**? Examples include avoiding cyclic dependencies, reducing dependencies, and us- ing techniques such as dependency injection. |  |  |  |

**Tactics**

**# Group Tactics Question**

**Supported?**

**(Y/N) Risk**

**Design Decisions and Location**

**Rationale and Assumptions**

8 Does the system include few or no (i.e., **limited) sources of nondeter- minism**? This helps

to limit the behavioral complexity that comes with unconstrained par- allelism, which in turn simplifies testing.

### [Usability](#_bookmark5)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Tactics Group** | **Tactics Question** | **Supported? (Y/N)** | **Design Decisions and**  **Risk Location** | **Rationale and Assumptions** |
| 1 | Support- ing user initiative | Does the system support operation **can- celing**? |  |  |  |
| 2 |  | Does the system support operation **undoing**? |  |  |  |
| 3 |  | Does the system support operations to be **paused** and later  resumed? Examples are pausing the download of a file in a web browser and allowing the user to retry an incomplete (and failed) download. |  |  |  |
| 4 |  | Does the system support operations to be applied to groups of objects (**aggregation**)? For example, does it allow you to see the cumulative size of a number of files that  are selected in a file  browser window? |  |  |  |

(*continues*)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Tactics Group** | **Tactics Question** | **Supported? (Y/N)** | **Design Decisions and**  **Risk Location** | **Rationale and Assumptions** |
| 5 | Support system initiative | Does the system provide assistance to the user based on the tasks that he or she is performing (by **main- taining a task model**)? Examples include: |  |  |  |
|  |  | * Validation of input data * Drawing user attention to changes in the UI * Maintaining UI consistency * Adding toolbars and menus to help users find functionality pro- vided by the UI * Using wizards or other techniques to guide users in performing key user scenarios |  |  |  |
| 6 |  | Does the system support adjustments to the UI with respect to the class of users (by **maintaining a user model**)? Examples include supporting UI customization (including localization) and sup- porting accessibility. |  |  |  |
| 7 |  | Does the system provide appropriate feedback to the user based on the system characteristics (by **maintaining a system model**)? Examples include: |  |  |  |
|  |  | * Avoiding blocking the user while handling long-running requests * Providing feedback on action progress (i.e., progress bars) * Displaying user- friendly errors without exposing sensitive data by managing exceptions * Adjusting the UI with respect to screen size and resolution |  |  |  |

### [DevOps](#_bookmark5)

**Tactics**

**Supported?**

**Design Decisions and**

**Rationale and**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Group** | **Tactics Question** | **(Y/N)** | **Risk Location** | **Assumptions** |
| 1 | Testability: control and observe system state | Does the system or the system components provide **specialized interfaces** to facilitate testing and monitoring? |  |  |  |
| 2 |  | Does the system provide mechanisms that allow information that crosses an interface to be recorded so that  it can be used later for testing purposes (**record/playback**)? |  |  |  |
| 3 |  | Can the system be executed in isolation (a **sandbox**) to experiment or test it without worry- ing about having to undo the consequences of the experiment? |  |  |  |
| 4 | Perfor- mance: manage resources | Can the system seamlessly **increase resources** (e.g., CPU, memory, network band- width)? |  |  |  |
| 5 |  | Can the system **intro- duce concurrency**? For example, does it support the seamless addition of parallel pro- cessing streams so that more requests for ser- vices can be processed concurrently? |  |  |  |
| 6 |  | Does the system **main- tain multiple copies of data** (e.g., by replicating databases or using caches) to decrease contention for frequently accessed data? |  |  |  |
|  |  |  |  |  | (*continues*) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Tactics Group** | **Tactics Question** | **Supported? (Y/N)** | **Design Decisions and**  **Risk Location** | **Rationale and Assumptions** |
| 7 |  | Does the system **main- tain multiple copies of computations** (e.g., by keeping a pool of servers in a server farm) to decrease  contention for frequently accessed computational resources? |  |  |  |
| 8 |  | Does the system **sched- ule resources**, particu- larly scarce resources, so that they may be allocated according to an explicit scheduling policy? |  |  |  |
| 9 | Perfor- mance: control resource demand | Does the system **reduce overhead** of responding to service requests by, for exam- ple, removing interme- diaries or co-locating resources? |  |  |  |
| 10 |  | If your inputs are a continuous stream of data, does the system **manage the sampling rate**? |  |  |  |
|  |  | That is, is it possible for you to sample the data at varying rates (with concomitant changes in accuracy/fidelity)? |  |  |  |
| 11 |  | Does the system monitor and **limit its event response**? That is, does the system limit the number of events  it responds to in a time period, to ensure predictable responses for the events that are actually serviced? |  |  |  |
| 12 |  | Given that you may have more requests for service than available resources, does the sys- tem **prioritize events**? |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Tactics Group** | **Tactics Question** | **Supported? (Y/N)** | **Design Decisions and**  **Risk Location** | **Rationale and Assumptions** |
| 13 | Modifiabil- ity: reduce coupling | Does the system con- sistently **encapsulate** functionality? This typ- ically involves isolating the functionality under scrutiny and introducing an explicit interface to it. |  |  |  |
| 14 |  | Does the system **abstract common services**, in cases where you are providing several similar services? For example, this technique is often used when you want your system to be portable across operating sys- tems, hardware, or other environment variations. |  |  |  |
| 15 | Modifiabil- ity: defer binding | Does the system regularly **defer binding** of important function- ality so that it can be replaced later in the life cycle, perhaps even  by end users? For example, do you use plug-ins, add-ons, or user scripting to extend the functionality of the system? |  |  |  |
| 16 | Availability: detect faults | Does the system use a component to **moni- tor** the state of health of other parts of the  system? A system mon- itor can detect failure  or congestion in the network or other shared resources, such as from a denial-of-service attack. |  |  |  |
| 17 |  | Do you use **exception detection** to detect a system condition that alters the normal flow of execution (e.g., system exception, parameter fence, parameter typing, timeout)? |  |  |  |
|  |  |  |  |  | (*continues*) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Tactics Group** | **Tactics Question** | **Supported? (Y/N)** | **Design Decisions and**  **Risk Location** | **Rationale and Assumptions** |
| 18 |  | Does the system use **voting** to check that replicated components are producing the same results? The replicated components may be identical replicas, func- tionally redundant, or analytically redundant. |  |  |  |
| 19 | Availability: recover from faults (prepara- tion and repair) | Does the system employ **rollback**, so that it can revert to a previously saved good state (the “rollback line”) in the event of a fault? |  |  |  |
| 20 |  | Does the system employ  **active redundancy** (hot spare)? In active redundancy, all nodes in a protection group (a group of nodes where one or more nodes  are “active”, with the remainder serving as re- dundant spares) receive and process identical inputs in parallel, allow- ing redundant spares to maintain synchronous state with the active node(s). |  |  |  |
| 21 |  | Does the system have consistent policies and mechanisms for **reconfiguration** after failures, reassigning responsibilities to the  resources left function- ing, while maintaining as much functionality as possible? |  |  |  |
| 22 |  | Does the system employ  **exception handling**  to deal with faults? Typically, the handling involves either reporting the fault or handling it, potentially masking the fault by correcting the cause of the exception and retrying. |  |  |  |

### [Bth.e1r0ReFaudring](#_bookmark5)