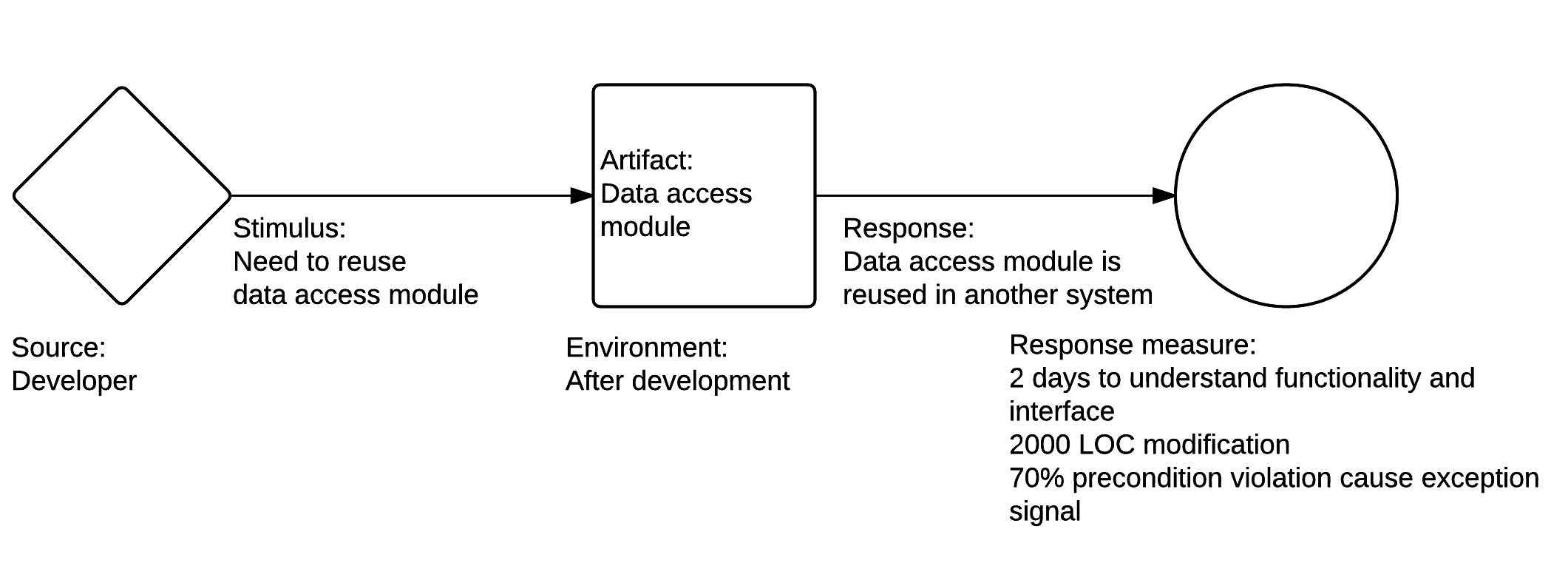
## Task1

**Reusability**

1. General scenario

|  |  |
| --- | --- |
| Portion of Scenario | Possible Values |
| Source | Software, people |
| Stimulus | Have partial of an existing program used in another program |
| Artifact | Framework or component that is going to be reused |
| Environment | After development |
| Response | Component is reused in another system |
| Response Measure | Time to understand the functionality of a component.  Modification needed to adapt one component to the specific functional requirement in a new system.  Proportion of precondition violation get handled by exception signaling. |

1. Concrete scenario



1. Tactics for reusability

Modularization: Modularization is the process of encapsulating complexity of related classes and exposing their abstract interface. After modularization, program is divided into components, which have high intra-component cohesion and low inter-component coupling. Components have relatively simple and independent function. They can be composed to perform complex tasks. So they are quite suitable for code reuse.

Reuse services: Should be properly encapsulated, abstracted, and hosted in a scalable environment to facilitate reuse across multiple business processes.

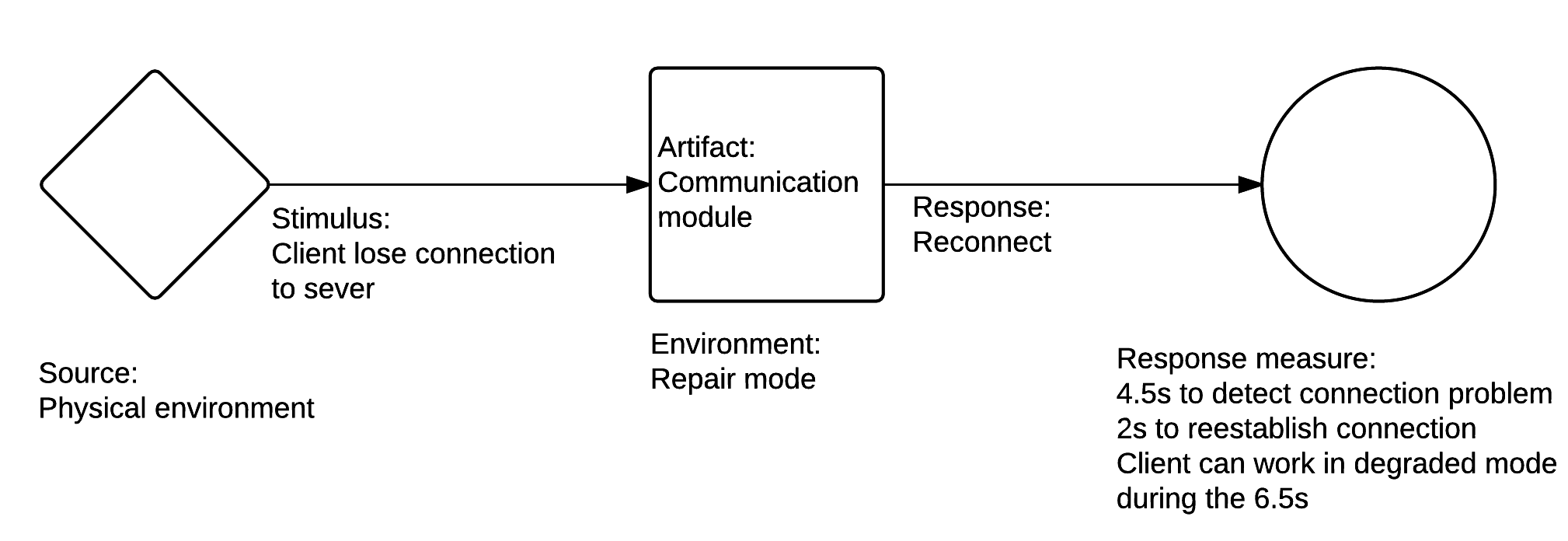
Interface oriented programming: Interface defines the function a class or a module provides and how developer should use it. Developer need not care about implementation details hidden behind the interface. By defining interface, a standard way of code reusing is provided.

**Recoverability**

1. General scenario

|  |  |
| --- | --- |
| Portion of Scenario | Possible Values |
| Source | Software, physical infrastructure, physical environment, people |
| Stimulus | Process crash, power off, hardware failure |
| Artifact | Process, persistent storage |
| Environment | Overloaded operation, degraded operation |
| Response | Log the fault  Restore system to a consistent state |
| Response Measure | Time to detect the fault  Time to recover from the fault  Time in which system can work in degraded state |

1. Concrete scenario



1. Tactics for recoverability

Replication: Replication can be used on server or database, usually with a master/slave relationship between the primary and the copy. Primary responds to request and inform the copy of state updates it must make. When the primary is down, the copy takes over and continues to provide service.

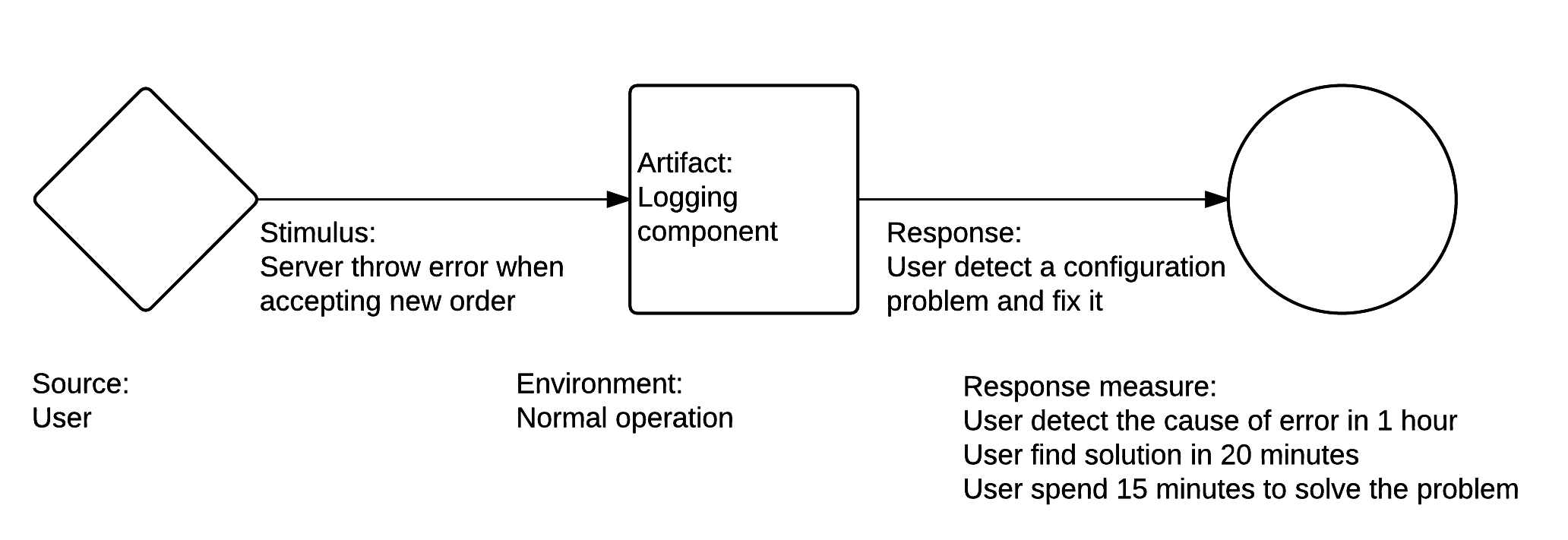
Checkpoint and operation log: Checkpoint is periodical dumping of consistent state. Operation log records important operation performed by system, typically data updating operation. When a failure occurs, system loads latest checkpoint and replays operation log to restore to the state before failure.

**Supportability**

1. General scenario

|  |  |
| --- | --- |
| Portion of Scenario | Possible Values |
| Source | End user, technical support staff |
| Stimulus | Maintainer also need to install, configure and upgrade the program. Maintainer needs to identify and resolve issues when the program works incorrectly. |
| Artifact | monitoring component, logging component |
| Environment | installation, upgrading, normal operation |
| Response | Log the fault, together with global variables and execution path |
| Response Measure | Time to find cause of a problem  Time to find solution of the problem in document  Time to actually solve the problem |

1. Concrete scenario



1. Tactics for supportability

Graceful degradation: Maintain limited function when part of the software failed to work properly due to hardware failure or exhausted resource. Degradation allows recovery from exception without intervention by technical support staff.

Event logging: Event logging provides system administrators with information useful for diagnostics and auditing. Log high-level information, especially failure information.

## Task2

|  |  |  |
| --- | --- | --- |
| Pattern | Strength | Weakness |
| Layered | Reusability: Each layer can be reused separately.  Flexibility: Layer can be modified in isolation from other layers.  Maintainability: The maintenance of program is easier because of the low coupling between layers. | Testability  Performance  Extensibility |
| Broker | Flexibility: Encapsulating functions in a layer allows you to swap this layer with a different implementation  Portability: Only client need to be designed portable  Scalability: Server location is transparent, new server can be dynamically added.  Reusability: Services are independent and can be reused. | Performance  Testability  Security |
| MVC | Extensibility: Adding a view or business model just need a little modification to controller  Reusability: Domain model focus on business logic and can be reused among programs.  Testability: Domain model can be tested using the interface it exposes to view. | Performance |
| Pipe-filter | Configurability: Filter configuration can be changed dynamically.  Reusability: Filters that implement simple transformations typically encapsulate fewer assumptions about the problem they are solving. The simpler filters can be reused in other solutions that need similar transformations.  Performance: A pipe and filter pattern process messages as soon as they arrived. | Reliability  Maintainability |
| Client-server | Security: All data is stored on server. Data access requires authentication.  Maintainability: Roles of a computing system is distributed among several servers. Client remains unaware of server repair and upgrade. | Performance  Reliability  Extensibility  Scalability |
| P2P | Availability: In a P2P-based distributed storage system, data is available only if one node has requested data.  Scalability: Clients both provide and use resources. Thus, the content serving capacity of peer-to-peer networks can actually *increase* as more users begin to access the content. | Performance  Security |
| Share data | Flexibility: Changes are easier to implements because data consumers can be modified in isolation from their producers. | Performance  Reliability |
| SOA | Reusability: Services can be reused by other business program.  Portability: Client can be deployed on multiple platforms as long as client use the same communication protocol.  Because the protocols and data formats are based on industry standards, the provider and consumer of the service can be built and deployed on different platforms. | Performance |
| Publish-subscribe | Reusability: Pub/sub better scalability than traditional client server pattern through message caching and load balancing.  Scalability: Publishers are loosely coupled with subscribers. New publishers or subscribers can be easily added because they are ignorant of the existence of system topology.  Testability: Topics reduce the number of messages that are required for testing.  Security: The communication infrastructure transports the published messages only to the applications that are subscribed to the corresponding topic. Specific applications can exchange messages directly, excluding other applications from the message exchange. | Reliability  Performance  Flexibility  Maintainability |
| Map-reduce | Reliability: Commodity computers are not reliable. In map reduce system, each node is expected to report back periodically. If a node falls silent for longer than that interval, the master node records the node as dead and sends out the node's assigned work to other nodes.  Adaptability:  Testability: Complex business work is decomposed into multiple map-reduce jobs. Only need to test whether map-reduce job can be executed correctly.  Scalability: It is extremely easy to add new node. After proper configuration, new node will be assigned tasks in future map-reduce job. | Configurability  Manageability |