Rubric for concept design

area	skill	skill description	criterion name	criterion description	evidence	example failing
Concept design	Modularize functionality	Express the functionality of a software application in a modular way	Independence	Concepts are fully independent of each other, and can therefore be understood and used independently of one another.	Concept description limits any references to context of use to notes.	Concept purpose mentions the way in which the concept is intended to be used (eg, a payment concept whose purpose says "enables payment for magazine subscription").
					Concept does not refer to another concept by name.	Concept mentions working in concert with another (eg, session concept says "works with authentication concept to provide authenticated sessions").
					Concept does not rely on any properties of other concepts.	Concept action "calls" an action of another concept or queries the state of another concept.
					All external datatypes are either generic parameters or built-in types (such as String).	Concept treats arguments as objects that have been constructed elsewhere (eg, takes in a user object that is assumed to have a name field).
			Completeness	Each concept provides a complete and coherent unit of functionality that delivers the value described in the purpose without the help of other concepts.	Concept functionality covers entire lifecycle of the purpose.	Concept doesn't include actions for set up (eg, defining available slots for reservations), or for closing down (eg, no deletion for an account).
					Concept embodies real functionality that fulfills a compelling purpose.	Concept is a data structure with CRUD actions when purpose calls for richer behavior (eg, concept holds contact info for a user but doesn't include any notification behaviors).
					Concept state is rich enough to support all the concept actions.	Concept state is expressed as the instance variables of a single object (eg, password auth concept that declares state as username and password, failing to support lookup by username needed to check password).
					Concept actions are sufficient to provide essential functionality to users.	No action to allow users to undo the effects of prior actions (eg, to cancel a reservation).
			Separation of concerns	Concept does not conflate two concerns that could be broken into separate concepts that could be reused independently of one another.	All components of the state work together for a single purpose.	The state admits a factoring into two or more independent parts (eg, a user concept mixes preferences and profile fields).
					No state component can be dropped without compromising essential functionality.	The concept gratuitously includes state that is not needed to support actions (eg, a password authentication concept that stores, in addition to username and password, the date on which the user first joined).
					The concept does not include state components that could be easily expanded into much richer, self-contained structures.	The concept contains references to external objects and stores properties of them that are not needed for this concept (eg, references to users along with their names, which would better be stored in a separate profile concept).
					Concept represents at most one reusable and ideally familiar units of functionality.	The concept does not include a subpart that could easily stand by itself, and may even be familiar in its own right (eg, user concept includes karma points).
					The concept is balanced in the attention to behavioral detail.	The concept does not include a fragment of functionality that would in practice grow into a full and complex concept of its own (eg, a restaurant reservation concept including some details of table sizes, which would in practice belong to a concept that managed table layouts).
	Define behavior	Define the detailed behavior of an application using states and actions	Purpose	Define the purpose of a concept that motivates its inclusion or invention	Purpose is a succinct and compelling description of a need or problem that the concept solves.	The purported purpose is instead a partial description of behavior (eg, purpose of Authentication concept is defined as being able to register and login, rather than as a means of identifying users).
					Purpose expresses a need and not a means by which the need is fulfilled.	Purposes hints at mechanism of concept (eg, purpose for a Reservation concept saying that it enables users to obtain commitments of allocation of a resource in advance).
					Purpose is focused on the concept at hand and not a larger need.	Purpose cannot be fulfilled by the concept itself, but would require other concepts too (eg, purpose of Friend concept is to allow users to connect to and share posts with each other, which cannot be fulfilled without a Post concept in addition).
					Purpose is expressed in an intelligible way that is easy to understand.	Purpose uses technical terms or the name of the concept itself without explanation (eg, saying that the purpose of a Following concept is to allow users to follow each other).

area	skill	skill description	criterion name	criterion description	evidence	example failing
					Purpose captures an end-to-end need that brings real value and does not focus on an aspect of behavior that brings no value in itself.	Purpose specifies ability to enter data without indicating why the data is useful (eg, saying that the purpose of a school Attendance concept is to record whether students are present or absent, without saying how the concept makes use of this information, say for generating end-of-term reports).
					Purpose is expressed in an application-independent way that would make sense for any context of use.	Purpose includes application-specific details that are not relevant to the design of the concept (eg, saying that the purpose of an Authentication concept in a concert ticketing app is to ensure that tickets are used by the people who bought them, which goes beyond authentication and includes irrelevant details of who is authenticated and why).
			Operational principle	Summarize archetypal concept behavior as a scenario	OP is a scenario that involves a sequence of steps.	OP is instead a restatement or elaboration of the purpose (eg, an OP for Authentication that says that users are identified so that decisions can be made about what actions they may perform, rather than enumerating the key steps of registration, login, etc).
					OP covers the full lifecycle of the concept.	OP covers only a user interaction that requires a prior setup (eg, an OP for Authentication that starts with login and neglects registration).
					OP includes actions by all stakeholders that modify the state.	OP neglects the setting up the conditions by administrators or company employees that make consumer interactions possible (eg, an OP for a ProductCatalog concept that includes a customer searching the catalog but does not include the actions that create the catalog).
					OP only includes actions of the concept at hand, and not the actions of other concepts.	OP describes a user journey that involves multiple concepts (eg, an OP that says that a user registers for an account, logs in and then posts a message, mixing Authentication and Post).
			State	Design the abstract state of a concept, aka the data model	State clearly defines distinct components.	Not clear exactly what is being stored in the state (eg, state for a UserProfile concept mentions only "information about users", rather than saying that it stores a display name and bio for each user, for example).
					State covers all the objects needed to support the actions.	State lists the properties of a single object, as if the concept were an object-oriented class (eg, state for an Authentication concept listing "username and password" and not recognizing that a username and password is required for each user, so that the login action can match against all possible users).
					State indexes components appropriately by object.	State includes some components that are either not indexed, or are indexed incorrectly (eg, state for a Friend concept defines friends as a set of users, failing to say that there is a set of users for each user; or says that each user has a set of friends and a date the friendship started, when instead there should be a set of friendships per user, each with the friend and the date started).
					State includes components that belong to other concepts and are not needed for the actions.	State includes properties of an object when just the identity would be sufficient (eg, the state of an Article concept for a blogging app represents the author of an article as a username and thumbnail image, when only the identity of the author is required and these additional fields should appear only in a UserProfile concept).
					State references external objects by properties when identities would suffice.	State references objects by user-friendly names (eg, in an Article concept, the author is represented by their username rather than by a user type that represents the identity of a user).
					State is sufficiently rich to support all actions.	A precondition or postcondition cannot be expressed fully because of a missing state component (eg, the state of a Token concept fails to include an expiry date needed to determine whether a validation action succeeds).
					State is abstract and not tainted by implementation concerns that are irrelevant to the behavior.	A collection of objects is declared as a list (or worse a tree) when a simple set would suffice (eg, in a Friend concept, the friends of a user are given as a list rather than a set).

area	skill	skill description	criterion name	criterion description	evidence	example failing
					State does not include needless redundancies (except those introduced to enable easier querying).	State includes a set of objects that are implicitly ordered by some property, and needlessly declares the set as an ordered list (eg, in a GroupChat concept, declaring the collection of messages in a chat as an ordered list when the messages are already implicitly ordered by send time).
			Actions	Design the actions of a concept that update the state	Actions required to set up the state are included.	State includes components that have to be set up in advance of typical user interaction but actions are not provided to do so (eg, a ProductCatalog concept that does not include actions for populating a catalog in the first place).
					Set of actions is sufficient to reach all states.	The state includes a special case that is not handled by an action (eg, a Style concept that allows styles to be organized into a hierarchy, and includes an “as is” value for a format associated with a style, but offers no way to specify this through an action).
					Set of actions is sufficient to update state components as needed.	A state component is assumed to be mutable, but no action allows its mutation (eg, in a concept for handling user authentication by passwords, no action is provided to allow a user to change their password).
					For objects managed by the state, actions are provided to create, update and delete the objects as needed.	Objects can only be created but never deleted (eg, a user account concept permits the creation of user accounts but no way to delete them).
					Undo or compensating actions are present when needed.	A user can submit a request for a resource but has no action to cancel the request (eg, in a restaurant reservation concept, there is no action for canceling a reservation).
					Actions should not include getter methods.	Actions are included that correspond to queries that a user would not typically be aware of and that might be performed repeatedly internally (eg, for a Post concept, an action that gets the author of a post).
					Actions should specify all necessary preconditions. (For a code-level specification, a valid alternative is to specify an error return.)	An action depends on a resource being allocated, and will fail if the resource is not available (eg, for an action to reserve a table in a restaurant reservation concept, a slot at the given time must be available).
					Actions should only refer to state components of this concept.	A common mistake is to refer to a component associated with an object that belongs to another concept (eg, an action in a Post concept takes a user object as an argument, and its specification refers to the name of the user, which belongs to a separate UserProfile concept).
					Set of actions should be minimal and not include actions that are easily expressed in terms of other actions.	An action is included that performs another existing action over all the elements of some set, wrongly included because the designer imagines this might be useful (eg, a room reservation concept has an action for reserving a room at multiple times, but which does not create any kind of repeat reservation which might justify the action).