EDUCE:
A Pattern Language of Language Patterns

Nik Boyd

Introduction

Software requirements engineers elicit, analyze, and organize software product requirements. The earliest stages of software development usually include discussions about what the software product will do, whose needs it will serve, what problems it will solve, what business process improvements it will make, what opportunities it will address, and how it will support business decision making.

> EDUCE = Essential Domain + Usage Concept Extraction
>
> **educe**, ... **1:** to bring out (as something latent)
> --Webster's Collegiate Dictionary

Initial ideas for software solutions usually revolve around specific business problems or opportunities, but they will most often concentrate on some business vision or a mission-critical or mission-central theme, i.e., some specific problem domain which has attendant vocabulary, techniques, equipment, and quality concerns. Therefore, the first purpose of EDUCE is to foster clarity and commonality throughout the various stakeholder communities, especially those tasked with establishing software product requirements.[1]

Useful (and even reusable) object discovery is at the heart of the object-oriented approach to software development. There are many opportunities for useful object discovery throughout the development of an object-oriented system. However, one of the greatest opportunities exists early in development during the elicitation and analysis of software product requirements. The metaphors used in object-oriented design center on the elements, techniques, and qualities found in the domain to be considered and modeled, and the attendant problems to be solved. Finding good class candidates in software product requirements can be challenging. Few software development methodologies offer guidance for finding good class candidates. Therefore, the second purpose of EDUCE is to guide the discovery of good class candidates in software requirements.

Additionally, software elements and components need good names, including class names, instance names, method names, state names, value names, relationship names, role names, exception names, and responsibilities. The object-oriented approach to software design takes the names of software elements and components primarily from the domain to be modeled and the problems to be solved. There are several pervasive metaphors used during software design that directly impact the naming of software elements and components.[2] Many of these metaphors are derived from the language, vocabulary, and people discovered in discussions about the problems to be solved. Therefore, the third purpose of EDUCE is to help software developers find the terminology they need to make good software naming decisions.

EDUCE does not (nor intends to) replace other methodologies that serve the object-oriented approach to software development. EDUCE complements other methodologies and can help make them more effective. EDUCE is compatible with several other approaches to object discovery, including problem frames,[3] user stories,[4] use cases,[5] and feature driven development.[6] Also, EDUCE is oriented toward and intends to foster natural naming for software elements and components.[7] Finally, because EDUCE focuses on statements, it can be applied incrementally. Therefore, EDUCE can be used in conjunction with agile requirements processes.

§ § §

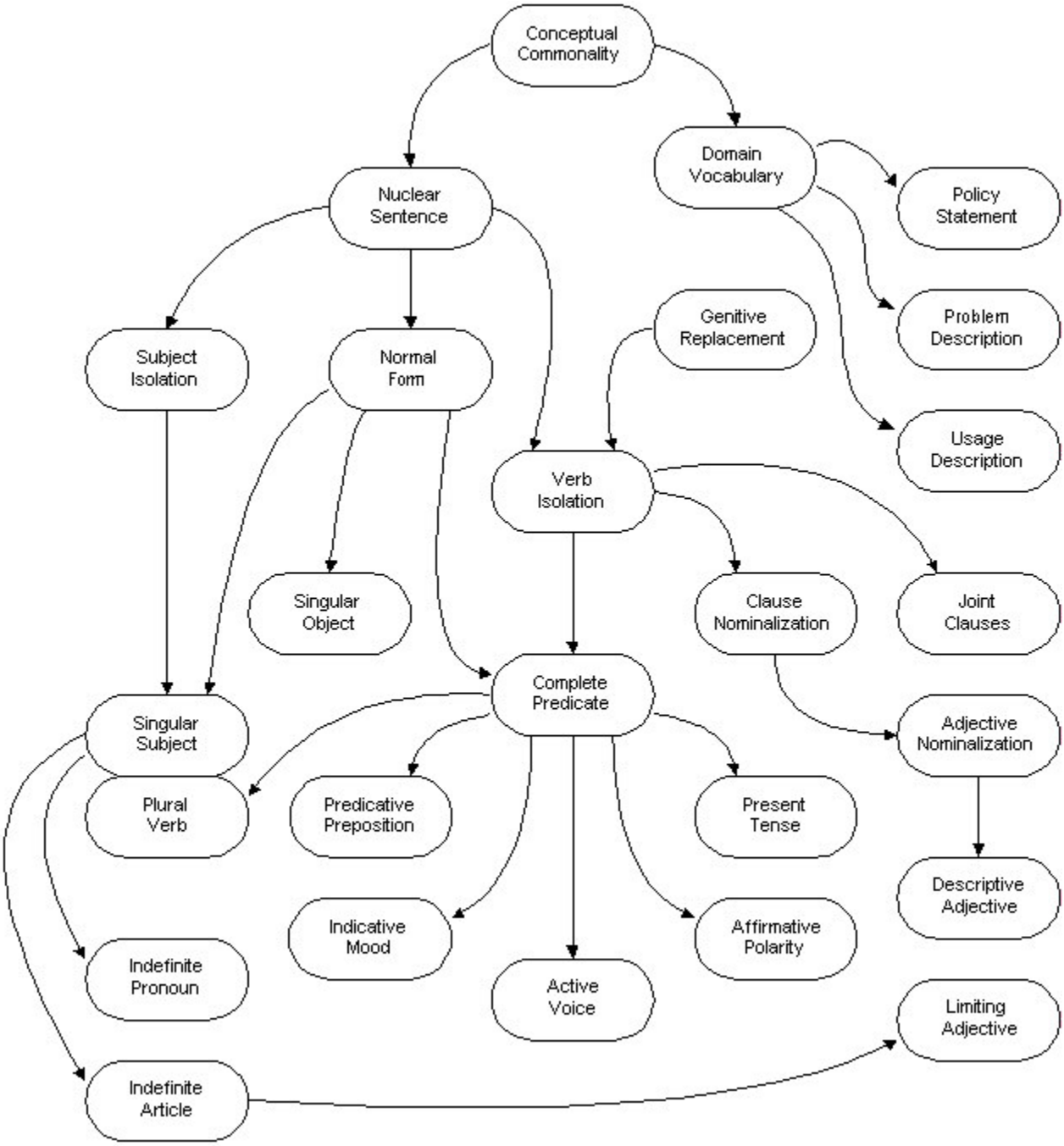


Figure 1. EDUCE Patterns Overview

Conceptual Commonality

Intent

Create and share a common understanding among solution development stakeholders.

Motivation

Discussions of business problems and opportunities often center on the situations that present themselves in a business: information that needs to be collected and shared, business activities that need to be improved, business policies that need to be enforced. Such discussions include statements that express vision, mission, values, viewpoints, interests, wants, needs, goals, conditions, situations, alternatives, options, requirements, roles, relationships, responsibilities, collaborations, qualities, quantities, and measures.

Software solution development involves many kinds of people as stakeholders. These various and diverse stakeholders need to share knowledge about business policies, problems, and practices in order to develop appropriate solutions that address their business needs. Without a common vocabulary and shared conceptual models of a business opportunity and its attendant problem(s), the effectiveness of a software solution development team and the likelihood that the developed solution will achieve its intended ends are substantially reduced.

Applicability

Use conceptual commonality when

- you have a complex business problem that warrants a software solution
- you want the solution development team to share a common understanding of:
    - the problem domain elements
    - the solution usage requirements
    - the business and technical policies that constrain the solution

Considerations

Software solutions intend to improve the quality of our lives, our business activities, and our business decisions. However, creating useful and usable software solutions requires effective communication among the solution stakeholders. So, creating a shared understanding of business policies, problems, and practices is the primary challenge facing a solution development team.

Achieving *commonality* among solution stakeholders requires a simple and consistent form for sharing their knowledge. A comprehensive domain vocabulary serves as a natural repository of business domain and solution requirements knowledge. Maintaining a domain vocabulary in an on-line repository (e.g., a Wiki server) provides an effective mechanism for sharing domain and requirements knowledge. The structure of a domain vocabulary is also important. A domain vocabulary focuses on domain concepts and their relationships. Thus, a domain vocabulary will naturally focus on topical subjects and the predicates that surround those subjects.

> to the extent that people share similar cognitive mappings they share similar psychological processes and similar subjective experience
> -- NLP Axioms, Practical Magic

Consequences

Natural conceptual models offer an effective means for producing domain vocabularies. Natural conceptual models are based on natural language, but with a restricted, simplified grammar, which is in turn based on a nuclear sentence. Each nuclear sentence focuses on the clear expression of a single idea using a normal form for clauses. The normal form gives sentences a consistently simple structure. This consistency and simplicity make ideas more readily sharable and comparable.

You can simplify the statements of stakeholders using the normal form and thereby produce statements that have a consistent structure. Participating analyst(s) can then assemble the normalized statements and publish them in a comprehensive domain vocabulary. The various stakeholders then review, validate, and assimilate the published domain vocabulary. Participating analyst(s) can also categorize the statements in a domain vocabulary, identifying which are policy statements, problem descriptions, and usage descriptions.

§ § §

Nuclear Sentence

(aka Kernel Sentence)

Intent

Simplify sentences and their clausal extensions.

Motivation

People often express their ideas using complex sentences. But, such sentence complexity, while ordinary and natural enough in conversations, sometimes hinders effective duplication and interpretation. Conversely, simplicity promotes clarity and subsequent duplication and interpretation.

Applicability

Use nuclear sentences when

- you want to simplify statements made by policymakers, domain experts, and usage stakeholders
- you want their policy, problem, and usage statements to conform to a consistent format

Considerations

Simple sentences help people communicate more effectively. Simple sentences increase the likelihood that people will actually duplicate and construe them similarly. The essential nucleus of a simple English sentence resembles the following structural form:

Nuclear Sentence	NS = S Pred
Subject	S = NP
Predicate	Pred = V O (PP)*
Verb	V
Object	O = NP
Prepositional Phrase	PP = Prep NP
Preposition	Prep
Noun Phrase	NP = A* N+
Adjective	A
Noun	N

Examples:

a predator *captures* a prey
a giver *gives* a gift *to* a recipient
a sender *sends* a message *to* a recipient *through* a medium

The nuclear sentence structure provides readers with a consistent way to parse and construe sentences. Such simplicity reduces ambiguity (or makes it more obvious) and contributes to a better understanding of a subject while retaining (sometimes even revealing) essential meaning.

Certain grammatical transformations preserve meaning while simplifying and converging on the nuclear sentence structure. Many of the EDUCE patterns focus on these grammatical transformations.

Consequences

Each simple sentence and each *clause* (in a complex sentence) has a normal form, which is based on a nuclear sentence. Achieving sentence normal form entails an initial examination of the action or relationship expressed by a verb using verb isolation and subsequent determination of a single subject using subject isolation.

The sentence normal form offers people a consistent way to construct, parse, and construe sentences. Such consistency helps people share their understanding of a subject matter, thereby fostering commonality.

§ § §

Normal Form

Intent

Conform sentences and clauses to a simple consistent structure.

Motivation

English grammar supports a rich variety of complex expressions. However, with this richness comes the potential for ambiguity and misinterpretation. Even the nuclear sentence structure offers a great deal of variety with respect to the various grammatical aspects of nouns and verbs used in subjects, objects, and predicates.

Applicability

Use sentence normal form when

- you want to standardize the format of nuclear sentences and their clausal extensions

Considerations

The normal form offers a simple and consistent structure with which to construct and construe English clauses and sentences. Simplicity and consistency contribute to better understanding of a subject while retaining (and sometimes revealing) essential meaning. The following table specifies the normal form for English clauses:

clause part		normal form
verb	uses	complete predicate with third person, active voice, present tense, plural number, affirmative polarity, and indicative mood
subject	uses	third person and singular number
	prefers	an indefinite article (**a, an**) to reference a single member of a class an indefinite pronoun (**each**) to reference a single member of a collection the definite article (**the**) only to reference a domain singleton
object	uses	third person and singular number whenever possible
	prefers	an indefinite article (**a, an**) to reference a single member of a class an indefinite pronoun (**some**) to reference multiple instances (of a collection or a class) the definite article (**the**) only to reference a domain singleton
preposition	used	only to complete predicates but replace possessive prepositions with appropriate verb phrases
conjunction	used	only for joint clauses *correlative conjunctions* may require clause nominalization but *coordinate conjunctions* should be separated with verb isolation or subject isolation

The following statements provides a typical example:

> a sender *sends* a message *to* a recipient *through* a medium

Consequences

You can organize simplified (normalized) statements by subject into a comprehensive domain vocabulary. You can also identify predicates shared by multiple subjects, and thereby discover common object responsibilities and protocols.

§ § §

Verb Isolation

Intent

Break complex and compound sentences down into simpler sentences.

Motivation

Complex and compound sentences often contain several verbs arranged in phrases and dependent clauses (like this one). Complex sentences indicate that some conditions, actions, or events are related. Sometimes verbs are used to relate and tie together other verbs. Compounded verbs indicate that two or more ideas relate to each other, or relate to a single subject. Compounded verbs condense these expressions into a single sentence. While more efficient, such condensation can obscure important relationships.

Applicability

Use verb isolation when

- you want a nuclear sentence expressing a single idea, but
- you discover a particular sentence contains several related ideas (several verbs)

Considerations

You can bring sentences towards normal form by initially focusing on verbs. Ideally, each sentence should contain only one verb. Consider the following statement and its subsequent decomposition to isolate the verbs it contains:

> a cat *chases, captures,* and *eats* a mouse
>
> a cat *chases* a mouse
> a cat *captures* a mouse
> a cat *eats* a mouse

Compounded verbs can easily be separated. While this increases the number of sentences, it allows each verb to contribute its distinctive meaning to a conceptual model. Object-oriented software designs often identify object ***responsibilities*** and interaction ***protocols*** using distinct verbs. Also, object ***method*** names often start with distinct verbs.

Some complex sentences include both independent and dependent *clauses.* Isolating the verbs contained in a complex sentence can be far more challenging. Consider the following statement and its subsequent decomposition to isolate some of the (many) verbs it contains:

> "The depot management introduced a company regulation that requires the depot manager be able to monitor the depot and to always be able to check if the depot is in a vulnerable state."
>
> the depot management *introduced* a company regulation
> a company regulation *requires* depot vulnerability monitoring,
> where depot vulnerability monitoring =
> a depot manager *monitors* the depot *for* a vulnerability

Consequences

To achieve the sentence normal form, each isolated verb will also need active voice, plural number, affirmative polarity, indicative mood, an appropriate tense, and a complete predicate. Once you've separated each verb into its own sentence, you may also discover that a resulting sentence has multiple (compounded) subjects. You can separate multiple subjects using subject isolation.

Sometimes a complex predicate will use multiple verbs to express its meaning. As you can see in the more complex example above, you may need to use clause nominalization to break down a complex sentence. Clause nominalization separates out a clause and converts it into a noun phrase (a name), with an attendant definition. The definition ties the separated statements together so that the paired statements retain the meaning of the original sentence.

§ § §

Active Voice

Intent

Convert a passive verb to an active verb.

Motivation

People often express their ideas using passive voice, especially when speaking of past events with past tense verbs. So, passive expressions are a normal part of ordinary conversation and written discourse. However, a sentence that uses passive voice may be missing its true subject, or the true subject may be displaced from its normal location.

> a mouse *was eaten.* (by what?)
> a mouse *was eaten by* a cat

Notice that even when a missing subject has been discovered, passive voice displaces the subject from its normal location at the beginning of a sentence. You can correct both of these deficiencies using active voice for each verb.

Applicability

Use active voice when

- you've extracted a sentence with an isolated verb, but
- you discover that the verb uses passive voice

Considerations

Every passive expression has a corresponding active expression. An active verb strengthens an expression and forces the discovery of its true subject. An active verb forces the true subject to move to its proper place at the beginning of a sentence.

> a cat *ate* a mouse
> a cat *eats* a mouse

Putting the true subject at the beginning of a sentence is especially useful for *transitive verbs,* which indicate action or a relationship. The subject is the *initiator* of an action or a *principal* participant in a relationship.

Object-oriented software designers often use present tense verbs with active voice to describe the *responsibilities* of collaborators in object systems.

Consequences

To achieve the sentence normal form, each active verb will also need plural number, affirmative polarity, indicative mood, an appropriate tense, and a complete predicate.

§ § §

Present Tense

Intent

Convert verbs to present tense if possible (and appropriate).

Motivation

Verbs can express the relative timing of an event in the *past, present,* or *future.* Verbs with *past tense* indicate an event occurred (or the state resulting from the event). Verbs with *present tense* indicate the unfolding of a current event. Verbs with *future tense* indicate an event can (potential) or will (imperative) occur.

> a predator *captures* its prey (action)
> an employer *employs* an employee (relationship)
> a wife *married* her husband (event, state)
> a recipient *will receive* a message *from* its sender (event)

Applicability

Use present tense when

- you've isolated an active verb,
- especially a *transitive verb* that describes
    - an action of an actor,
    - a responsibility of a role, or
    - a relationship between its participants

Use *past tense* only as a descriptive adjective when

- you've identified a state that follows directly from an event (an action or activity)

Use *future tense* only when

- you're formulating software requirements that use an imperative mood

Considerations

Object-oriented software designs tend toward use of the present tense for actions and relationships, especially for expressing the *responsibilities* of collaborators. Therefore, the normal form recommends, but does not require, the use of present tense for verbs. While present tense may be preferred for most statements, it's important to find the appropriate tense.

Many requirements specifications adopt the use of *future tense* with imperative mood to express individual requirements, especially using the auxiliaries *will* or *shall.* You can easily insert these auxiliaries into sentences that use present tense if needed (e.g., in a proposal or statement of work). But, they are not generally useful in the context of conceptual models of problem domains.

Verbs that indicate an event or a state will often be expressed in the *past tense* because they indicate an action that occurred to bring about the indicated state. In this case, the *past tense* verb will be retained as a descriptive adjective (describing an object state), while the present tense verb will be used to indicate how an object arrives at that state.

Consequences

To achieve the sentence normal form, each present tense verb will also need an active voice with plural number, affirmative polarity, indicative mood, and a complete predicate.

In English, a relationship exists between the tense of a verb and its mood. The following table explores this relationship.

Mood	**Tense**
indicative	present
indicative	past
imperative	future
potential	future

§ § §

Indicative Mood

Intent

Convert verbs to the indicative mood, unless the verb describes an optional (potential) behavior.

Motivation

Verbs can express an action or state as an *actual* fact, or the *potential* for such. The moods include *indicative, subjunctive, potential, imperative, infinitive.* Some examples include:

> *strive* lest you fail (subjunctive)
> to *sympathize* is to understand (infinitive)
> a cat *may eat* a mouse (potential)
> a cat *will eat* a mouse (imperative)
> a cat *eats* a mouse (indicative)

Applicability

Use indicative mood when

- you've extracted a sentence with an isolated verb, but
- you discover that the verb uses a mood other than the indicative

Considerations

A predicate with *subjunctive* mood indicates a state (or event) supposed or imagined, or contingent or contrary to the fact. The *indicative* mood serves equally well to express these modes. So, usage of the *subjunctive* mood has virtually disappeared from English. If you find a statement with the *subjunctive* mood, rephrase it using the *indicative* mood or the *imperative* mood with affirmative polarity. Consider the following rephrasing of the former *subjunctive* example.

> you *will succeed* if you *strive*

A predicate with *potential* mood indicates a possible or probable (or necessary) state (or event). Some consider these to be variations of contingency, and therefore *subjunctive.* However, they differ in the explicit use of auxiliaries such as **can, may, could, might, must, should, would.** If you find a problem or solution has an optional feature, retain the *potential* mood. However, for clarity sake, you should also indicate this point of variation or optionality explicitly in a requirements specification.

A predicate with *infinitive* mood has no subject, and so has no limitations of person or number. A verb with *infinitive* mood is often used to express a purpose, an objective, or a goal. The *infinitive* mood is especially useful for expressing the quality concerns of business stakeholders. Consider the following examples:

> to *ensure* safety compliance
> to *minimize* depot vulnerability

A predicate with *imperative* mood issues a command or permission, or indicates a wish, thereby describing an event to bring about a desired state. A predicate with *indicative* mood indicates a real or factual state (or event). The *imperative* and *indicative* moods complement each other, and they are both useful in relation to object-oriented designs. A verb with *indicative* mood is often used to describe object *responsibilities,* while a verb with *imperative* mood is often used as a prefix in object *method* names.

It's interesting to note that many programming languages are considered *imperative* (as opposed to declarative). These *imperative* programming languages usually include reserved words such as **if, then, else, do, goto.** The *imperative* mood is the traditional way in which software requirements are phrased. However, for the purpose of conceptual models and object-oriented models, which are fact-oriented, the *indicative* mood is preferable.

Consequences

To achieve the sentence normal form, each indicative verb will also need an active voice with plural number, affirmative polarity, an appropriate tense, and a complete predicate.

§ § §

Affirmative Polarity

Intent

Convert a verb with negative polarity to affirmative polarity.

Motivation

A verb can be expressed with an *affirmative* or a *negative* polarity. For example:

> a storage building's drum count *must not exceed* its drum storage limit (negative)
> a storage building's drum storage limit *must exceed* its drum count (affirmative)

Applicability

Use affirmative polarity when

- you've extracted a sentence with an isolated verb, but
- you discover that the verb uses negative polarity

Considerations

Negative polarity may obscure intent and can lead to misunderstandings, especially when used in a complex sentence or if there are double negatives. Most negative assertions are oriented toward prevention. For example, policies sometimes dictate that a situation or state must be avoided.

> "Don't hit me with those negative waves, baby."
> -- Sgt. Oddball in *Kelly's Heroes*
> played by Donald Sutherland

You can rephrase most negative assertions in the affirmative. As reflected in the example above, simply removing the word ***not*** and swapping the order of the subject and object in a clause will often suffice. However, sometimes achieving affirmative polarity requires choosing a related replacement verb. Sometimes an equivalent positive verb must replace a negative verb.

> *must not allow => prevents*

Consequences

To achieve the sentence normal form, each affirmative verb will also need an active voice with plural number, indicative mood, an appropriate tense, and a complete predicate.

§ § §

Plural Verb

Intent

Convert a singular present tense verb to plural.

Motivation

In English grammar, a finite verb must agree with the *number* and *person* of its subject (*subject-verb agreement*). However, for a third person singular subject with a present tense verb, the verb becomes inflected with plural.

Applicability

Use plural number for a verb when

- you've extracted a sentence with an isolated verb, and
- you've converted an isolated subject from plural to singular number

Considerations

Consider the following statement and its subsequent rephrasing:

> cats *eat* mice (plural subject + singular verb)
> a cat *eats* mice (singular subject + plural verb)

Consequences

To achieve the sentence normal form, each plural verb will also need an active voice with affirmative polarity, indicative mood, an appropriate tense, and a complete predicate.

To retain *subject-verb agreement,* a plural verb requires a singular subject.

§ § §

Subject Isolation

Intent

Break compounded subjects out into separate sentences.

Motivation

Compounded subjects indicate that two or more subjects participate in a common relationship or activity. Compounded subjects condense these ideas into a single sentence.

> a bat and an insect *fly* through the air

While more efficient, such condensation can hide or obscure important relationships.

Applicability

Use subject isolation when

- you want a nuclear sentence expressing a single idea, but
- you discover a particular sentence contains several related ideas (several subjects)

Considerations

Subject isolation separates compounded nouns or noun phrases into distinct sentences. Consider the example above and how its subjects may be separated into distinct sentences.

> a bat *flies* through the air
> an insect *flies* through the air

Object-oriented software designs focus on relationships and collaborations between individuals. Object-oriented software designs often use a basic *metonymy* as a source of design components: a single instance represents an entire category or *class* of individuals. So, a statement about a single individual is applied to all the instances of a *class.* Thus, object-oriented software designers often use sentence subjects and objects as candidates for *class* names. So, while subject isolation increases the number of sentences, it allows each subject to contribute its own distinctiveness to a conceptual model.

Consequences

Having isolated subjects into separate sentences, you may discover that some isolated subject has plural number, which needs to be converted to a singular subject.

§ § §

Singular Subject

Intent

Convert a sentence subject from plural to singular.

Motivation

People often express ideas as generalizations. Generalizations often use plural subjects. Plural subjects indicate *collections, categories* or *classes.* However, a plural subject often hides the true *cardinality* of the verb in a sentence. How many objects does the verb relate to the subject? The answer to this question helps determine the *cardinality* of the relationship expressed by the verb.

Applicability

Use singular subject when

- you've extracted a sentence with an isolated verb, but
- you discover that the subject is plural

Considerations

Object-oriented software designs focus on relationships and collaborations between individuals. Object-oriented software designs often use a basic *metonymy* as a source of design components: a single instance represents an entire category or *class* of individuals. So, a statement about a single individual is applied to all the instances of a *class.* Thus, object-oriented software designers often use sentence subjects and objects as candidates for *class* names.

Plural subjects often obscure the *cardinality* of the relationship expressed by a verb. Consider the following sentence and examples of its rephrasing:

> generalizations *use* plural subjects
>
> a generalization *uses* a plural subject
> each generalization *has* a plural subject

Consequences

After conversion, a sentence subject refers to a single instance using a *singular* noun or noun phrase. Converting the subject to singular often exposes whether the verb relates the subject to a single or to multiple instances of the direct and indirect objects in a sentence.

Notice that you can use an indefinite article **(a** or **an)** to indicate that the subject is an instance of a *class.* Also, notice that an indefinite pronoun **(each)** can be used to retain a stronger indication that the subject is a member of a *collection.*

Singular subjects support the identification of *stable concepts.* When several nuclear sentences contain the same subject, they collectively reveal how the subject relates to other objects and concepts. These relational connections to other concepts help to stabilize the conception of the subject.

§ § §

Singular Object

Intent

Convert a plural (direct or indirect) object to singular when possible.

Motivation

People often express ideas as generalizations. Generalizations often use plural objects. Plural objects indicate *collections, categories* or *classes.* However, a plural object often hides the true *cardinality* of the verb in a sentence. How many objects does the verb relate to the subject? The answer to this question helps determine the *cardinality* of the relationship expressed by the verb.

Applicability

Use singular object when

- you've extracted a sentence with an isolated verb, and
- you've reduced the sentence to a singular subject, but
- you discover that the sentence contains some plural (direct or indirect) object(s)

Considerations

Object-oriented software designs focus on relationships and collaborations between individuals. Object-oriented software designs often use a basic *metonymy* as a source of design components: a single instance represents an entire category or *class* of individuals. So, a statement about a single individual is applied to all the instances of a *class.* Thus, object-oriented software designers often use sentence subjects and objects as candidates for *class* names.

Plural subjects and objects often obscure the *cardinality* of the relationship expressed by a verb. If the *cardinality* between a subject and an object is singular, make both the subject and the object singular.

> drum identifiers *identify* drums (obscured cardinality)
> a drum identifier *identifies* a drum (revealed cardinality)

If the *cardinality* between a subject and an object is singular, but the relationship is plural, make the clausal object singular with the indefinite pronoun **each.**

> the system *assigns* identifiers *to* drums (obscured cardinality)
> the system *assigns* an identifier *to* each drum (revealed cardinality)

If the *cardinality* between a subject and a direct object is one to (possibly) many, make the subject singular, but keep the object plural.

> a drum load *contains* drums

If the *cardinality* of a relationship is indeterminate (one or more), especially in the context of a dependent clause, you may keep the clausal object plural with the indefinite pronoun **any.**

> . . . *if* a storage building *contains* any drums

However, you probably ought to replace this phrasing with the equivalent *cardinality* revealing condition (for clarity sake).

> . . . *if* a storage building drum count > 0

Consequences

Other opportunities for exploring and revealing *cardinality* include examination of the articles used in noun phrases and the use of a limiting adjective to explicitly indicate the *cardinality* of a relationship.

§ § §

Indefinite Article

Intent

Convert a definite article to an indefinite article, unless the definite article identifies a true domain singleton.

Motivation

Casual conversational English frequently (and appropriately) uses an indefinite article to indicate one, or any one, or some one, or a certain one, (usually) of many. **An** is original Anglo-Saxon meaning one; **a** is an abbreviation of **an.**

Casual conversational English frequently (and appropriately) uses the definite article (**the**) as a simple referential indicator. However, it can also be used to emphasize, distinguish, or indicate uniqueness. It is this final usage which raises an issue for conceptual modeling.

Applicability

Use an indefinite article (**a** or **an**) when

- you want to indicate that an object is an instance of a *class*

Use a definite article (**the**) only when

- you've identified a subject or an object that is a true domain singleton

Considerations

Object-oriented software designs focus on relationships and collaborations between individuals. Object-oriented software designs often use a basic *metonymy* as a source of design components: a single instance represents an entire category or *class* of individuals. So, a statement about a single individual is applied to all the instances of a *class.* Thus, object-oriented software designers often use sentence subjects and objects as candidates for *class* names.

Conceptual models and object-oriented design models derived from them must respect assertions of uniqueness within a discussion domain. So, conceptual models need some precision with respect to the *cardinality* of instances and relationships, especially uniqueness versus multiplicity.

True domain singletons are rare. If problem statements explicitly mention the business enterprise under consideration, the enterprise may be a true singleton.

> the ECO depot ...

However, business enterprises often have some complex internal structure that will also need to be modeled. This may call into question whether it's appropriate to model the business enterprise as a whole or as a collection of parts.

Statements about the problem domain, especially problem frames and requirements, may mention a solution system. So, a solution system may appear as a true singleton in such statements.

> the system will ...

However, some solutions implemented as distributed computing systems replicate system instances on multiple hosts in order to achieve reliability, availability, and/or scalability. In this kind of solution, clearly "the system" is no longer a true singleton.

Consequences

You can use an indefinite article (**a** or **an**) to indicate that an object is an instance of a *class.* You can use an indefinite pronoun (**each**) retain a strong indication that an object is a member of a *collection.* Reserve the definite article (**the**) to indicate that only one instance of a subject (or a clausal object) exists in the discussion domain. You can also use a limiting adjective to explicitly indicate the *cardinality* of a relationship.

§ § §

Indefinite Pronoun

Intent

Use an appropriate indefinite pronoun to indicate that an object (or a subject) is a member of a *collection.*

Motivation

Casual conversational English uses indefinite pronouns with both singular and plural significance when referring to collections. However, indefinite pronouns with singular significance are preferable for conceptual models.

Applicability

Use an indefinite pronoun with singular significance when

- you want to indicate that a sentence subject or an object is a member of a *collection*

Considerations

Object-oriented software designs focus on relationships and collaborations between individuals. Object-oriented software designs often use a basic *metonymy* as a source of design components: a single instance represents an entire category or *class* of individuals. So, a statement about a single individual is applied to all the instances of a *class.* Thus, object-oriented software designers often use sentence subjects and objects as candidates for *class* names.

Because object-oriented designs pay special attention to instances, conceptual models also focus on individuals, especially those established as a singular subject or a singular object. So, while some indefinite pronouns may have plural significance (all, both, many, several, ...), convert them to the corresponding pronouns that have singular significance (every, either, each, ...). The following table offers suggestions for such correspondences and conversions:

plural	**singular**	**plural**	**singular**
all	every, no, none	most	each
any	every, no, none	several	each
both	either, neither	many	each
other	another	certain	each
few	each	some	each

Consequences

Note that **some** serves as the preferred pronoun for indicating indeterminate *cardinality,* i.e., for zero or more instances. You can use an indefinite article (**a** or **an**) to indicate an instance of a *class.* Reserve the definite article (**the**) to indicate that the subject truly has only one instance within the discussion domain.

§ § §

Complete Predicate

Intent

Verify that all thematic semantic roles of a *transitive verb* have actants.

Motivation

A verb relates a subject to some clausal object(s), or it expresses the action of a subject in relation to some clausal object(s). A *transitive verb* often relates multiple objects to the sentence subject. However, when discussing problems and requirements informally, stakeholders often use incomplete sentences. They leave out important elements through simplification or oversight.

Applicability

Use complete predicate when

- you've extracted a sentence with an isolated verb, but
- you discover it has a *transitive verb* with missing actants

Considerations

Typically, *transitive verbs* have actants that play thematic semantic roles.

- The subject serves as *agent.*
- The direct object serves as *patient.*
- Indirect objects serve as *recipient, companion,* or *instrument.*

Once you've identified the thematic roles associated with a *transitive verb,* you can identify whether any clausal objects are missing from a sentence and which roles they play in the action expressed by the verb.

Sentences with missing objects reduce opportunities for object discovery. However, you can recover the missing objects by requesting them from a stakeholder, or by inferring them directly from the thematic roles entailed by a verb.

Consequences

For those *transitive verbs* with more than two actants, the associated predicative prepositions are considered part of the verb for the purposes of conceptual modeling, object-oriented analysis, and *method* signature design.

For example, consider the following archetypal *transitive verbs* and their thematic roles:

a predator *captures* a prey
a giver *gives* a gift *to* a recipient
a sender *sends* a message *to* a recipient *through* a medium
a surgeon *cuts* a portion *from* a body *using* a knife
a carpenter *fastens* a plank *to* a wall *with* a fastener (nail)

Completing a predicate with its appropriate predicative prepositions will provide a more complete conceptual model. Complete predicates also support the identification of *stable concepts.*

§ § §

Predicative Prepositions

Intent

Discover any prepositional phrases needed to complete the predicate indicated by a verb, especially a *transitive verb.*

Motivation

Predicative prepositions associate clausal objects with the verb of a sentence, especially sentences with *transitive verbs* whose completion require more than one clausal object. Each *transitive verb* requires some number of clausal objects as arguments. Thus, each verb can be characterized in terms of its *valence.*

Applicability

Use a predicative preposition when

- you've extracted a sentence with an isolated verb, but
- you discover it has a *transitive verb* with missing actants

Considerations

In his pioneering work on catastrophe theory,[10] René Thom developed a theory for the spatial origin of syntactical structures. Thom suggested that the interactions indicated by *transitive verbs* can be graphed as spatiotemporal processes. Thom proposed that the interaction graphs of these processes belong to one of sixteen archetypal morphologies, which serve as semantic bases for these essential *transitive verbs.* Thom suggested that the syntactic structures emerged naturally based on the semantic structure of these forms.

Additional support for this idea can be found in the works of several cognitive scientists, including George Lakoff, Mark Johnson,[11] and Peter Gärdenfors.[12] Certain kinds of *image schemas* have the same kind of semantic implications as those developed by Thom. Image schemas also depict transitive events graphically.

Consequences

Completing a predicate with its appropriate predicative prepositions will provide a more complete conceptual model. Most conceptual modeling languages eliminate these predicative prepositions, retaining only the verbs. Retaining the prepositions associated with a verb provides better traceability from a conceptual model back to the source statements from which it was derived. It is also advantageous for a programming language to retain these linguistic elements. For example, the keyword *method* naming conventions found in Smalltalk[8, 9] support the retention of prepositions.

§ § §

Genitive Replacement

Intent

Replace each genitive case with an appropriate verb to form a complete predicate.

Motivation

Exposing the (sometimes hidden) relationships between domain elements increases the value of a domain analysis, offering the opportunity for further modeling and steps toward a more natural and comprehensive solution design. The genitive case often hides a substantive verb that can be revealed through appropriate analysis or inference.

Applicability

Use genitive replacement when

- you discover a noun phrase with a possessive form (**'s**), or
- you discover a noun phrase that uses a possessive preposition (**of**)

Considerations

The *genitive* (possessive) case relates two nouns or noun phrases. The genitive case uses the preposition **of** or the possessive form (**'s**) to indicate ownership, possession, property, containment, or aggregation.

Identifying the nature of the relationship hidden by use of the genitive case helps to determine whether to model the elements of the relationship as separate objects or whether the object of the relationship should become an attribute of the subject. The importance of these distinctions to domain modeling has been raised by others.[13]

Category	**Example** (genitive => *verb*)
possession	the hair of the dog => a dog *has* hair (as a physical attribute)
property	the color of a car => a car body *has* a color (as a property)
ownership	a library's book => a libray *owns* a book
containment	drums of chemicals => a drum *contains* a chemical
aggregation	a bicycle's pedals => a bicycle *has* pedals (as parts)

Consequences

Once you've discovered the kind of relationship hidden by the genitive case, you'll want to extract it using verb isolation. However, if the genitive phrase is embedded in another clause, you'll need to use clause nominalization to extract it. Fortunately, the names for genitive replacements are easily derived from the genitive phrase as descriptive noun phrases. Consider the following examples:

dog hair = a dog *has* hair
car body color = a car body *has* a color
library book = a library *owns* a book
chemical drum = a drum *contains* a chemical
bicycle pedals = a bicycle *has* pedals

§ § §

Joint Clauses

Intent

Retain conjunctions that join clauses together.

Motivation

Conjunctions can be categorized as *coordinate, subordinate,* or *correlative.* The *coordinate* conjunctions connect elements of equal rank - words, phrases, clauses. The *subordinate* conjunctions connect dependent elements to independent ones. The *correlative* conjunctions are used in pairs - one complementing the other. The following table provides a representative sample of the various kinds of conjunctions.

Category	Subcategory	Examples
coordinate	additive	also, **and,** besides, both, likewise, moreover, then
	disjunctive	but, either, else, **or,** neither, nor, other, otherwise
	final	consequently, for, hence, so, thus, therefore
subordinate	reason	as, because, for, hence, inasmuch, since, whereas, wherefore
	degree	as, else, other, otherwise, rather, than
	condition	**if,** provided, since, unless
	place	after, before, whence, whereat, wherever
	time	as, before, ere, since, still, till, **until, when,** whenever
correlative	both-and	**both** boys **and** girls are going
	either-or	**either** boys **or** girls will go
	neither-nor	**neither** boys **nor** girls will go
	if-then	**if** you fail, **then** you must try again

Applicability

Use joint clauses when

- you've simplified some statements with verb isolation, but
- you've discovered that some clauses only make sense when paired together

Considerations

Many independent clauses describe *conditions* or *situations* that can occur in the world (and thereby in a problem domain). *Situations, conditions,* and *constraints* are generally *testable* and *verifiable.* These kinds of statements hold special significance throughout software development, from requirements through code, because they usually describe some qualities that need to be established and maintained (enforced). Thus, the primary object *responsibility* categories include ensuring quality.

Many complete nuclear sentences stand on their own (as *independent* clauses), but some statements only make sense in relation to others, especially when one of them (a *dependent clause*) describes a possible *situation,* a *condition,* or a *constraint.*

While simple sentences can stand alone, you can also compose them to describe *situations.*

> a building *stores* type 1 hazardous chemicals **and**
> that building also *stores* type 2 hazardous chemicals

While comparative relations can stand alone, you can also compose them to describe *conditions* or *constraints.*

> a building's drum storage count *equals* its drum storage limit **and**
> a neighboring building's drum storage count *equals* its drum storage limit

Consequences

You can use adjective nominalization to simplify some *conditions.* The previous *condition* can be simplified as follows:

> a ***full*** storage building = a storage building whose drum count *equals* its drum storage limit
> a ***full*** storage building *neighbors* another ***full*** storage building

You can pair clauses to produce a *condition - action pair* that describes a *behavior rule* or a *business rule.* Consider the following example of a *condition - action pair:*

> the EPA *closes* a chemical storage facility ***if*** ...

the chemical storage facility *violates* a safety regulation

You can use clause nominalization to simplify some rules. Consider the following simplification of the foregoing rule.

storage facility closure = the EPA *closes* a chemical storage facility
safety violation = a chemical storage facility *violates* a safety regulation

a safety violation *triggers* a storage facility closure

Some *correlative* conjunctions are *coordinate* and some are *subordinate.* Thus, you can treat them as described above based on their usage.

§ § §

Clause Nominalization

Intent

Replace an entire clause with an appropriate summary name (noun phrase) derived from a verb.

Motivation

Simplifying complex sentences increases the likelihood that they will be understood by more people. However, sometimes multiple verbs (esp. infinitive verbs) participate in a complex statement. When analyzed, each verb can be separated out into its own predicate. However, splitting a complex sentence into several simpler ones often requires including references between them so that they collectively preserve the meaning of the original statement.

Applicability

Use clause nominalization when

- you discover a sentence with multiple complex clauses (multiple verbs)

Considerations

You can give a simple statement a name (as a noun phrase) derived from the verb used in the statement, thereby creating a definition for the name. The general process of creating names in this manner is called *nominalization.* Consider the following simple example:

> employment = employer *employs* employee
> (*employs* - *s* + ment = employment)

Once you've derived an appropriate term, you can then substitute it into another sentence in place of the original clause it replaces. *Nominalization* allows you to tie separated statements together and retain the meaning of the original complex sentence. Consider the following statements, their decomposition, and the results of clause nominalization:

> "The depot management introduced a company regulation that requires the depot manager be able to monitor the depot and to always be able to check if the depot is in a vulnerable state."
>
> a company regulation *requires* that the depot manager *monitor* the depot
> the depot manager *monitors* whether the depot is in a vulnerable state
>
> a company regulation *requires* depot monitoring, where
> depot monitoring = a depot manager *monitors* the depot *for* vulnerability, where
> depot vulnerability = the depot *has* two (or more) ***full*** neighboring storage buildings

Consequences

Note that the last statement in the example above includes an adjective nominalization. You will often find these two forms of *nominalization* very powerful when used together to dissect and decompose complex statements.

Note that clause nominalizations are reversible. The reversibility of *nominalizations* can play an important part in translating requirements into working code, especially when extracting the detailed meaning from problem descriptions and usage requirements that contain descriptive phrases.

§ § §

Adjective Nominalization

Intent

Replace a comparative clause with an appropriate summary name (a descriptive noun phrase).

Motivation

Some predicates describe conditions, constraints, or situations (esp. results). These predicates compare objects or their attributes and values. Conditions often appear as dependent clauses in business rules, or as business policy statements, or quality requirements. In code, conditions will often become class constraints or guarantees, method pre-conditions or post-conditions, tests, or guards.

Applicability

Use adjective nominalization when

- you discover a complex sentence that contains a condition (a value relation), esp. as a dependent clause

Considerations

You can give a comparative clause a descriptive name (as a descriptive noun phrase), thereby creating a definition for the descriptive name. The general process of creating names in this manner is called *nominalization.* More specifically, you can summarize a value relation using a noun phrase with a descriptive adjective. The adjective describes the *condition* or *situation* contained in the clause. Consider the following simple example:

> a ***full*** storage building = a storage building whose drum count *equals* its drum storage limit

Once you've derived an appropriate term, you can then substitute it into another sentence in place of the original clause it replaces. Such *nominalization* and substitution allow you to tie the separated statements together and retain the meaning of the original complex sentence.

This kind of transformation can be especially powerful when used to simplify joint clauses. For example, consider the following pair of coordinated clauses and the simplification that results from adjective nominalization and substitution:

> a building's drum storage count *equals* its drum storage limit **and**
> a neighboring building's drum storage count *equals* its drum storage limit
>
> a ***full*** storage building *neighbors* another ***full*** storage building
>
> depot vulnerability = the depot *has* two (or more) ***full*** neighboring storage buildings

Consequences

Note that the last statement in the example above also uses clause nominalization. You will often find these two forms of *nominalization* very powerful when used together to dissect and decompose complex statements.

Note that adjective nominalizations are reversible. The reversibility of *nominalizations* can play an important part in translating requirements into working code, especially when translating problem descriptions and usage requirements that contain descriptive phrases into comparative relations between objects and their attributes.

§ § §

Descriptive Adjective

Intent

Discover and name conditions, which may be the states of an object in its lifecycle, usage constraints, or consequential results.

Motivation

Descriptive adjectives indicate states (of being). Business policy statements intend to foster quality operations and relationships, but are best formulated when they focus on specific ends (end results). Objects often transition through specific states in their respective lifecycles. Object behaviors are often contingent on or triggered by conditions. Proper service usage is also usually contingent on some condition or state. Completion results are often described as conditions or states. All of these cases warrant consideration of descriptive adjectives.

Applicability

Use a descriptive adjective when

- you need to describe an intended end result (as from a policy objective), or
- you need to describe a state in the lifecycle of an object, or
- you need to describe (name) a condition, constraint, or result (as from a comparative relation)

Considerations (Policy Statements)

Business policy statements (esp. vision and mission statements) focus on qualities that businesses value and intend to foster in their operations and relationships. But, considerations and discussions regarding quality are often abstract. Many business mission and vision statements are typical in this regard. Some common mission themes include loyalty, honesty, integrity, diversity, innovation, leadership, growth, etc. However, the targets of these quality themes are often unclear or unspecified, which in turn obscures the intended ends. So, while quality abstractions may be useful in framing and prioritization discussions, they diminish the impact and clarity of mission and vision statements. Focusing on ends makes mission and vision statements much more powerful.

Quality abstractions expressed as nouns often hide more concrete concepts. Quality concerns can be expressed as abstractions using nouns, or they can be instantiated and made concrete using descriptive adjectives. This subtle linguistic shift can have a dramatic impact on the consideration and formulation of quality objectives. However, the value of shifting quality expressions in this way may not be obvious or may be easily overlooked.

Hence, ends are best expressed as effects that make qualitative (valuable) changes in the world. So, ends are best expressed using adjectives that describe the intended effects, i.e., the state (of being) to be achieved. Descriptive adjectives indicate such states of being. So, descriptive adjectives are generally more useful for formulating ends statements, e.g., compare: customer loyalty vs. *loyal* customers. Making this simple linguistic shift can have a profound impact on how we conceive of and discuss business quality concerns. This shift can be especially helpful in the discussion and formulation of ends statements and business objectives.

Considerations (Object Lifecycles)

Dispositions derived from verbs play an important role in object-oriented software designs. Dispositions indicate the states through which an object passes during its lifecycle, and the related verbs (and corresponding nouns) often indicate how an object transitions from state to state - i.e., the events that trigger state changes. For example, the states in the lifecycle of a piece of electronic mail could include the following: created, drafted, addressed, queued, sent, received, stored, trashed, destroyed.

However, merely enumerating object states does not convey any information about the transitions between states. The events that trigger transitions between states must also be enumerated in order to complete a lifecycle model. For this reason, we typically model lifecycle states graphically. Consider the lifecycle model for a chemical storage drum depicted in Figure 2 below.

The analysis of dispositions and transitions can be important even in the case of a relatively stable relationship. For example, with respect to the employment relationship, there are several states preceding and following employment that may be relevant to a full lifecycle model. Consider the following states: applied, interviewed, hired, ***employed,*** (retired, discontinued, fired).

Considerations (Conditions)

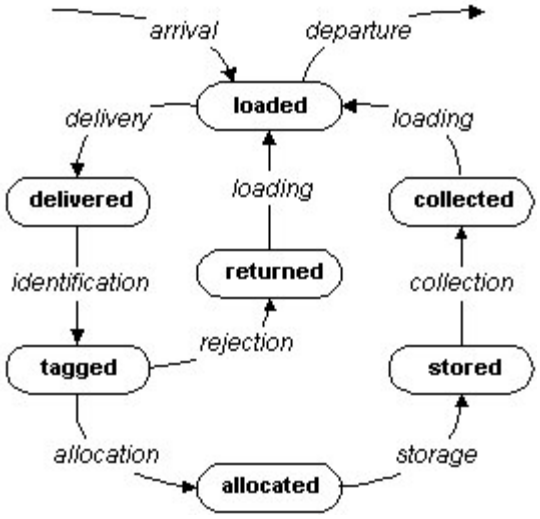
A noun phrase with a descriptive adjective will often identify a condition, a constraint, or a result formulated as a comparative relation between values. Consider the examples from a representative *intelligence card* with quantitative analysis of the quality **vulnerability** for a hazmat storage facility depicted in Figure 3 below.

Consequences

The consideration of descriptive adjectives complements the consideration of adjective nominalization. Both are the (generally) reversible complements of each other.

Opportunities for exploring and revealing *cardinality* include the use of a limiting adjective to explicitly indicate the *cardinality* of a relationship.

§ § §



dispositions: delivered, tagged, allocated, stored, collected, returned, loaded
transitions: arrival, delivery, identification, rejection, allocation, storage, collection, loading, departure

Figure 2. Chemical Storage Drum Lifecycle

vulnerability	**proximity, drum counts**
vulnerable depot	= *vulnerable* storage building pair
vulnerable buildings	= neighboring pair of *full* storage buildings
neighboring buildings	= building walls within 5 meters proximity
full storage building	= (drum storage capacity = 0) per hazard type
building drum storage capacity	= (drum storage limit - drum count) per hazard type
building drum storage limit	from: building drum storage license
building drum count	from: building drum inventory

Figure 3. Storage Depot Vulnerability

Limiting Adjective

Intent

Explicitly describe the cardinality of a relationship or an event.

Motivation

Software engineers care about measurements, both measurements captured by software solutions and measurements of the solutions themselves. As a consequence, software solution designs need to include information about instance populations, including constraints, limits, thresholds, order, storage sizing, etc. So, the description of relationships between instances must consider their *cardinality:* how many instances of each type participate in a relationship? Definite articles, indefinite articles, and indefinite pronouns offer hints about *cardinality.* However, where practical, explicit *cardinality* should be elicited for problem descriptions and usage requirements.

Applicability

Use a limiting adjective when

- you discover a statement describing a measurable state,
- you discover a statement describing a *transitive event,* or
- you discover a statement expressing a relationship between instances

Considerations

Limiting adjectives supply (or indicate the need for) quantitative information about subjects and objects, including number, order, degree, etc. Their presence indicates quantitative areas of a conceptual model that should be explored and specified more concretely during object-oriented analysis. Specific numbers that characterize the limitations on relationships should be obtained from the domain experts and solution users. For example, are the relationships

1,
0..1,
0..n,
1..n, or
some other interval?

The following table provides a schema for the kinds of limiting adjectives frequently found in human discourse and the questions that elicit them.

Question	**Quantity**	**Examples**
What Portion?	Partition	1 / 3
Which?	Ordinality	1st, 2nd, 3rd, ...
How Many?	Cardinality	1, 2, 3, ...
How Many Times?	Iteration	20 times
	Multiplication	20 fold
How Much?	Degree	close - closer - closest, ...
Whether?	Affirmation	always, every, ...
	Negation	none, never, ...
	Doubt	maybe, ...

Consequences

When a limiting adjective appears in statement about an activity or a relationship, it may be summarized using clause nominalization. When a limiting adjective appears in a condition, it may be summarized using adjective nominalization.

§ § §

Domain Vocabulary

Intent

Capture business elements, facts, relationships, policies, and procedures in a comprehensive domain vocabulary.

Motivation

Software solution development involves many kinds of people as stakeholders. These various and diverse stakeholders need to express and share many kinds of ideas, including vision, mission, values, viewpoints, interests, wants, needs, goals, conditions, situations, alternatives, options, requirements, roles, relationships, responsibilities, collaborations, qualities, quantities, and measures.

Without a common vocabulary and shared conceptual models of a business opportunity and its attendant problem(s), the effectiveness of a solution development team and the likelihood that the developed solution will achieve its intended ends are substantially reduced.

Applicability

Use a domain vocabulary when

- you want to promote better communication between all the stakeholders participating in a software development project

Considerations

A domain vocabulary can be used to integrate viewpoints from multiple stakeholders. They may offer complementary views of interactions with domain elements or relationships between domain elements. These complementary views help complete and stabilize the conception of those domain elements.

While other narrative and structured descriptions will often be generated to serve solution development, the essential form of a domain vocabulary will include a list of domain elements (as nouns or noun phrases) and their attendant predicates (as verb phrases). The verb phrases are assembled from the nuclear sentences extracted from a problem description and its solution usage requirements. Consider the following list excerpted from a sample problem domain:

drum
    contains a hazardous chemical

drum contents
    is a hazardous chemical
    has a category (hazard type)
    has a measurement (quantity + unit of measure)

drum identifier
    identifies a drum *in* the system

drum identification label
    identifies a drum *within* the depot premises
    indicates the drum identifier
    indicates the drum contents, measurement(s) and hazard type

Consequences

A domain vocabulary may highlight differences in stakeholder viewpoints and biases. It may help resolve what might otherwise become conflicts through differences of opinion (esp. those based on differences in viewpoint). Fortunately, software solutions are **soft.** Thus, such differences can often be accommodated simply by offering different solution users different *views* of the same underlying system.

§ § §

Policy Statement

Intent

Describe the business goals to improve quality, and the measurements and instrumentation needed to achieve and detect success.

Motivation

Business vision and mission are often expressed through official statements that become part of the corporate chants. These incantations can become powerful speech acts that influence every aspect of business operations, including those improved by software solutions.

Applicability

Use a policy statement when

- you discover a statement made by a business governor that expresses or addresses a quality concern

Considerations

Enterprise software solution development and maintenance is expensive, being both knowledge intensive and labor intensive. Without deliberate and consistent alignment to business policies and technical standards, waste will inevitably result, wasted money, time, effort, resources, and opportunities. To prevent and minimize waste, quality alignment[14, 15] must remain a foremost concern throughout software solution development.

Policy statements intend to answer the following questions:

- what opportunities - what ends are intended? what qualitative results and effects (as changes in the world) are to be achieved?
- what limits - what means are acceptable for achieving the stated ends? what are the boundaries and regulations imposed on those means?

Consider the business policy objectives and quality concerns captured on a *policy card* in Figure 4 below for an example business problem (a hazmat storage facility).

Consequences

The quality concerns expressed in business policy statements often describe enduring business themes.[16 - 19] Business activities aligned with these themes are the most durable business processes. Business elements related by these business activities often have stable surfaces, but may be replaceable. Technical solution components often have both changing surfaces and frequent replacements. Business architectures informed by these principles can maximize stability where the opportunities for stability are greatest.

Governance guides business through policy making (direction) and decision making (execution). Governance requires feedback. The following *business information food chain* summarizes the business feedback loop linking policies and decision making:

- business policies *identify*
- quality concerns, which *determine*
- quality metrics and measures, which *determine*
- business instrumentation, which *collects*
- business measurements, which when analyzed *offer*
- business intelligence, which *supports*
- business decisions

Business policy statements express and address quality concerns. Quality can be measured. Measurement requires appropriate instrumentation. Also, measurement has costs, especially when humans participate. Software can improve business activities, but software also offers the opportunity to reduce measurement costs through automatic data collection, analysis, and presentation as business intelligence for decision support.

§ § §

safety	**regulatory compliance**
safe hazmat storage	ensure safety compliance
	minimize depot vulnerability
	prevent depot closure
	prevent litigation
	accept all drums that can be stored safely
	allocate drum storage space within licensed limits
	allocate drum storage space to minimize vulnerability

Figure 4. Storage Depot Safety Compliance

Problem Description

Intent

Describe and frame the business problems that need solutions.

Motivation

The world has problems that need solutions, some of which may benefit from software solutions. A problem in the world usually has a structure that will fit into one of several problem frames.[3]

Applicability

Use a problem description when

- you need to describe a business (improvement) opportunity that may benefit from a software solution

Considerations

Problems exist in the world outside of a computer. Jackson[3] has noted this challenge regarding problems: that we must remain focused on the world when describing problems.

> "Not only does everyone agree that you should focus on the problem before the solution; almost everyone agrees that you should focus on *what the system will do* before you focus on *how it will do it. What* before *how* is the motto."
>
> "But it's not a very helpful motto. It's often hard to distinguish a problem from its solution, and it's no easier to distinguish *what* from *how*. It's more helpful to distinguish *where.* That is, to recognize that the solution is located in the computer and its software, and the problem is in the world outside."

So, problem descriptions should focus on describing the entities, phenomena, events, sequences, and facts that exist and/or occur *in the world.* It is also useful to determine *why* a business requires a problem solution. Problem descriptions intend to answer the following questions:

- where - where does the problem exist? what is the surrounding context?
- why - why does the problem need to be solved? what are the business policies to which a solution must be aligned?
- what - what are the problem elements: entities, values, relations, states, transitions, events, sequences?

The answers to these questions will often fit into one of the following problem frames:[3]

- required behavior - what conditions must be maintained by controlling the behavior of some part of the world?
- commanded behavior - what commands may an operator use to control the behavior of some part of the world?
- information display - what information must be displayed to observe the states and behavior of some part of the world?
- simple workpieces - what information must be captured and displayed regarding some part(s) of the world?
- transformation - what are the formats and the rules that determine how to transform data from one format to another?

Consequences

Benjamin Kovitz [20] offers an excellent introduction to the usage of problem descriptions and problem frames in the context of software requirements. Problem descriptions and problem frames can be used in conjunction with usage descriptions to capture software requirements.

The dominant metaphors[2] used in the object-oriented approach to software design intentionally blur the distinction between a problem and its solution. Object-oriented designs structure and name design entities (objects) so that they resemble the structure and vocabulary of the entities and activities found in the world (more often, a description of a problem found in the world). Therefore, when using the object-oriented approach to software design, it becomes even more important to distinguish a problem from its solution, and to describe a problem in terms of its place in the world, and the opportunities offered by and the improvements intended by a software solution.

§ § §

Usage Description

Intent

Describe solution users, their business interests and objectives (ends), and their activities (means) to achieve those objectives.

Motivation

Usage descriptions focus on the interactions between participants engaged in an exchange of information, goods, services, or some other form of value (e.g., money). There are several levels at which such interactions may be described, including:

- interactions between a business and its customers, partners, or suppliers,
- interactions between business managers, their peers and subordinates,
- interactions between a software solution and its users, or
- interactions between the components of a software solution.

Solution usage descriptions tend to focus on the interactions between a software solution *user* and the solution (product or service) being used.

Applicability

Use a usage description when

- you discover a statement describing the usage of a software solution feature, or
- you discover a statement describing a business activity that needs improvement or automation

Considerations

Descriptions of business activities and software solution usage tend to focus on predicates and their *transitive verbs* (with active voice). Descriptions of solution usage may take several forms with lesser or greater levels of detail, including: feature descriptions,[6] user stories,[4] and use cases.[5] For example, consider the business interactions between a loading bay clerk and a system for allocating storage space at a hazmat storage facility described as a use case, and depicted on the *face card* in Figure 5 below.

Usage descriptions intend to answer the following questions:

- what - what will a solution do? what is the goal of an activity? what results are achieved?
- who - who are the participants engaged in the interactions needed to achieve the goal?
- how - how will interactions between participants (esp. with a solution) achieve the stated activity goals?
- when - when will a sequence of interactions be triggered, and what are the pre-conditions that allow it to be triggered?

Consequences

Usage descriptions often determine the overall structure (order) of the interactions between participants. Often these interaction sequences will imply a lifecycle whose states can be modeled and identified with descriptive adjectives.

§ § §

depot	**accepts a delivered drum load**
loading bay clerk	wants to accept a delivered drum load
depot governors	want to ensure depot safety compliance
loading bay	must be empty
delivery truck	has arrived at the loading bay
drum handler	unloads drums from a truck into the loading bay
loading bay clerk	*selects drum load delivery*
system	accepts a delivery manifest for a drum load
loading bay clerk	notifies the system after accepting each drum
system	assigns drum ID and prints label for each drum
loading bay clerk	notifies the system when the load is finished
system	prints delivery reports
drum handler	reloads any returned drums onto the truck

Figure 5. Drum Load Delivery

Glossary

cardinality
The number of elements in a given mathematical set.

class
A group of instances in an object-oriented design that all have similar shape and behavior, i.e., they have similar *responsibilities.*

clause
A clause is a group of words having a subject and a predicate. An *independent* clause carries the main predication of a sentence and may consequently stand alone as an independent sentence. A *dependent* clause enters into the construction of a sentence with the force of a noun, an adjective, or an adverb. Consequently, a dependent clause cannot stand alone, but must be expressed and understood in the context of an associated independent clause.

collection
A group of instances in an object-oriented design that may the same or different shapes and behaviors. Collections of various kinds themselves exhibit different behaviors, including sets (uniqueness), maps (key-value pairs), lists (sequencing), etc.

commonality
The degree to which people share a common understanding of a subject.

method
Methods encapsulate the knowledge held by objects in object systems. Methods are the means by which collaborators gain access to each others' services and knowledge. Method names often start with a verb. However, depending on the naming conventions used, some method names may also be nouns or noun phrases.

metonymy
A figure of speech consisting of the use of the name of one thing for that of another of which it is an attribute or with which it is associated.

nominalization
The creation of a noun phrase from some other form, especially a clause (verb), which can then be used as its replacement.

protocol
Object *responsibilities* can often be further detailed in terms of the services each collaborator offers. Protocols are usually used to group object services and methods into meaningful categories. Protocols are often gerunds derived from the verb phrases used to describe object behaviors, with or without the remaining components of a verb phrase. The following are representative examples from Smalltalk: accessing, creating, enumerating, initializing, testing, triggering events.

responsibility
Object-oriented designs are most often articulated in terms of collaborations between objects. Each object in such a collaboration serves a specific role with attendant responsibilities. Object responsibilities can be broken down into three primary categories. What an object **knows** (information + structure), **does** (behavior), **ensures** (quality constraints + guarantees).

stable concept
The relations and actions of a subject are expressed through verbs and verb phrases (predicates) that relate the subject to other domain elements. The predicates that surround a domain element help establish the domain element as a concept.

subject-verb agreement
A principal of English grammar that requires agreement between the number and person of a verb and its subject.

transitive verb
A verb that describes a transition event, whether permanent or reversible.

valence
The number of positions that need to be filled to complete a predicate.

References

1. Nik Boyd. *A Conceptual Model for Software Requirements.* http://www.educery.com/papers/models/requirements.htm
2. Nik Boyd. *Software Metaphors.* http://www.educery.com/papers/rhetoric/metaphors/
3. Michael Jackson. *Problem Frames: Analyzing and Structuring Software Development Problems.* Addison-Wesley Publishing, Inc., 2000. ISBN 0-201-59627-X.
4. Mike Cohn. *User Stories Applied: For Agile Software Development.* Addison-Wesley Publishing, Inc., 2004. ISBN 0-321-20568-5.
5. Alistair Cockburn. *Writing Effective Use Cases.* Addison-Wesley Publishing, Inc., 2000. ISBN 0-201-70225-8.
6. Feature Driven Development, http://www.featuredrivendevelopment.com/
7. Kari Laitinen. *Natural Naming in Software Development and Maintenance.* VTT Publications no. 243. Technical Research Centre of Finland, Espoo, 1995.
8. Adele Goldberg, David Robson. *Smalltalk-80: The Language.* Addison-Wesley Publishing, Inc., 1989. ISBN 0-201-13688-0.
9. Kent Beck. *Smalltalk Best Practice Patterns.* Prentice Hall, 1997. ISBN 0-134-76904-X.
10. René Thom. *Structural Stability and Morphogenesis: An Outline of a General Theory of Models.* W. A. Benjamin Advanced Book Program, 1975. ISBN 0-805-39277-7.
11. George Lakoff, Mark Johnson. *Philosophy in the Flesh.* Basic Books, 1999. ISBN 0-465-05674-1.
12. Peter Gärdenfors. *Conceptual Spaces: The Geometry of Thought.* MIT Press, 2000. ISBN 0-262-07199-1.
13. Donald Firesmith, Brian Henderson-Sellers. Clarifying Specialized Forms of Association in UML and OML. Report on Object Analysis and Design in the *Journal of Object-Oriented Programming,* JOOP (11)2:47-50, May 1998.
14. Nik Boyd. *Quality Alignment and Quality Inventories.* http://www.educery.com/papers/rhetoric/quality/alignment.htm
15. Faisal Hoque. *The Alignment Effect: How to Get Real Business Value Out of Technology.* Financial Times Prentice Hall, 2002. ISBN 0-13-044939-3.
16. Marshall Cline, Mike Girou. Enduring Business Themes. *Communications of the ACM* 43(5), May 2000.
17. Mohamed E. Fayad, Adam Altman. An Introduction to Software Stability. *Communications of the ACM* 44(9), Sep. 2001.
18. Mohamed E. Fayad. Accomplishing Software Stability. *Communications of the ACM* 45(1), Jan. 2002.
19. Mohamed E. Fayad. How to Deal with Software Stability. *Communications of the ACM* 45(4), Apr. 2002.
20. Benjamin Kovitz. *Practical Software Requirements: A Manual of Content and Style.* Manning Publications, 1998. ISBN 1-88-477759-7.