

Chapter

7



Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Chapter 7: Specifying Features

DTU course 02264

Agenda

Abstract

- In this chapter we will look at different techniques and styles of specifying features, also known as functional requirements. Each technique has an individual profile of pros and cons, so that we need to analyze the application conditions and discuss the factors contributing to them.
- Among the many techniques that are known, we will select a few representative examples along the scale from informal prose, controlled languages, Use Cases, Snow Cards (e.g. Volere, XP User Stories), and detailed attribute tables.

Contents

1. Prose Descriptions of Requirements
2. Requirements' Attributes
3. Reinforced Natural Languages
4. Controlled Natural Languages
5. Use Cases Templates
6. Snow Cards



*"The difference between the right word
and the almost right word is the same as
between a lightning and a glow worm."*

Mark Twain

Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Chapter 7.1:

Prose Descriptions of Requirements

Documenting Requirements with Natural Languages

- **The most straightforward way to document requirements for a system is to write them down in some natural language.**
 - Any language could be used, e.g. English, German, or Danish.
- **Using natural language in this way offers a number of valuable benefits.**
 - Natural language is universally well understood and spoken/written, and everybody in the project speaks it fluently. So, no time consuming and expensive trainings are needed.
 - Natural language comes natural, so it is easy to use and everybody has powerful tools for creating texts in natural languages.
 - Natural languages have unlimited expressiveness and flexibility.
- **Therefore, it is easy to start with a prose description of requirements and it is no wonder that many requirements documents are written as a prose text in a natural language.**

Documenting Requirements with Natural Languages: Problems

- **The advantages of natural languages are not always warranted, however.**
 - In international teams, there might not be a universal language equally well used by all (or even most) team members.
 - Even if a team is culturally homogeneous, language skills vary widely. Not all people are aware of their deficiencies.
 - Software Engineers (and IT people in general) are often not very talented in natural languages, including their native language; they often prefer diagrammatic representations or formal languages.

- **But what is more important, natural languages have no fixed formal structure.**
 - They have no formal syntax or semantics.
 - Vocabularies are extremely large, and the domain vocabularies are often quite different from the base language (consider medicine, finance, IT).
 - Additionally, in many languages new words may be created routinely as compounds of other words, making the vocabulary also infinitely large.
 - The vocabulary, syntax, semantics, and grammar keep changing.

- **Therefore, it is very difficult to define or measure the quality of prose texts.**
- **Generally, prose texts can not be reliably processed automatically (beyond spelling and basic grammar).**

Documenting Requirements with Natural Languages: Problems

- As a consequence of their informality, natural language requirements specification have a tendency to be

consistent	Is this set of requirements consistent or contradictory?
incomplete¹	Is this set of requirements complete?
incomplete²	Are all individual requirements complete?
ambiguous	Is there more than one interpretation, and if so, which one is valid?
vague	What does this requirement mean?
viscous	What are the consequences of this change?

These problems are very widespread and have been around for decades.

- They are not specific to software requirements, but many types of software afford only a small penalty of change
- For a natural language text, it is difficult, time-consuming, error-prone, and costly to ensure completeness.

Problems in Expressing Requirements with Natural Language

- To understand the problems with natural language, take a sheet of paper and draw a picture according to the following instructions.

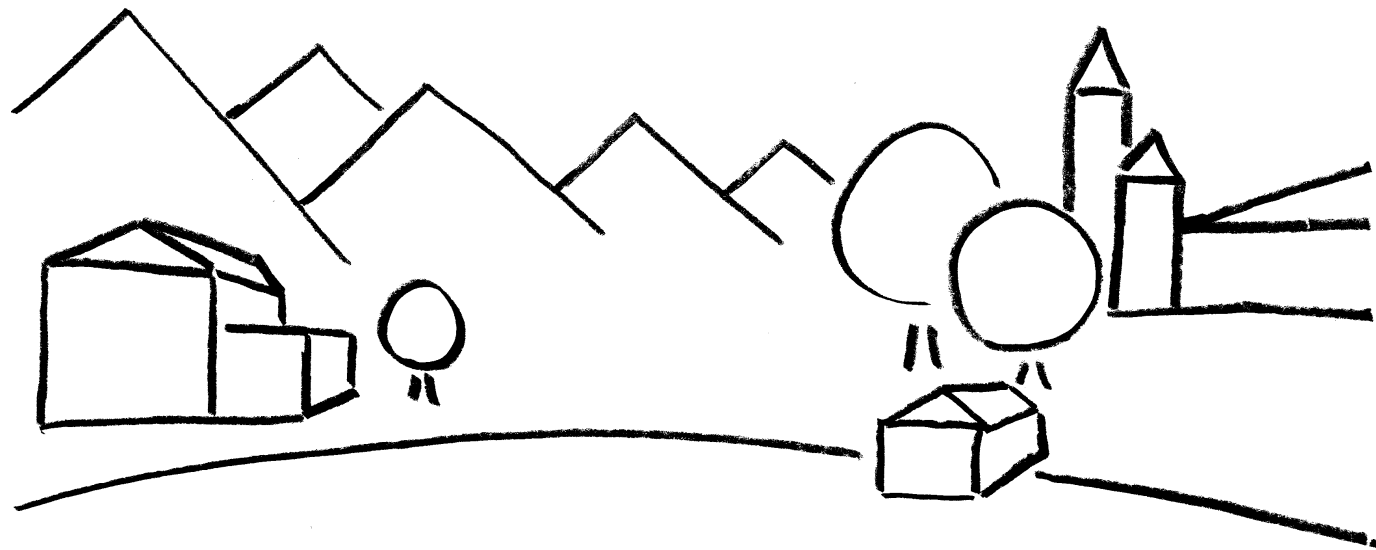
In the foreground, there is a lake shore, in the background, there are snow-covered mountain tops. The sky is bright, almost without clouds.

The lake shore is dotted with trees and bushes.

On the left foreground, there is a hotel with balconies and jutties.

On the right there are a few houses, behind them on a small mound a church with a pointed tower.

In the right foreground there is a landing stage, behind it two party tents.



The closest someone ever got...





Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Chapter 7.2:

Requirement Attributes

DTU course 02264

The Requirements Lifecycle

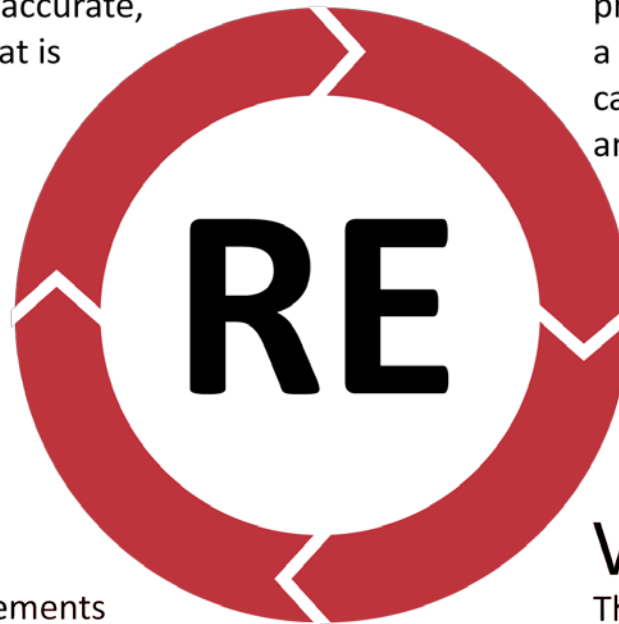
At different stages of the requirements lifecycle, our focus and purposes shift. Our tasks and focus shift accordingly.

Ideation & Elicitation

First, the requirements must be elicited. The focus lies on obtaining valid, accurate, and complete information on what is needed.

Elaboration

Second, the requirements must be detailed, precise, unambiguous, and written down in a way (“specified”) so they may be communicated, passed on, and revisited in the future, and so they are ready to being implemented.



Management

Finally, the quality of the requirements must be assessed/improvements and it must be ensured that they are in fact valid from the stakeholders’ point of view.

Validation

Third, the requirements must be administered throughout their lifecycle, as they are bound to change in various ways (add, abandon, update, split, merge, complete, ...).

Why must we elaborate requirements?

- **Assume our team has made a great job in eliciting all relevant requirements as a list of keywords or short phrases.**
- **We are now faced with two problems:**
 - As we have seen in the introduction, it pays to improve the quality as early on in the software lifecycle as possible.
 - But even if our requirements were perfect, they must be communicated to a team, to programmers, to contractors, to other stakeholders etc. We have to make sure that everybody during the whole process understands the requirements.
 - Also, the requirements may be in use for a long time. Will we ourselves understand our own requirements in just the same way we intended them when we wrote them, half a year ago?
- **So, we should try and remove all defects we can find with reasonable effort, and we should cast our requirements in a generally understood and durable form.**
 - This process is called requirements elaboration.
 - The result is called a “Software Requirements Specification” (SRS).
 - Such specifications may easily be hundreds or thousands of pages long.

Requirements Attributes (Elicitation)

- **Probably the most frequently arising problem is incompleteness.**
 - That is to say, an individual requirement does not contain all the information needed to work with it.
- **Yet, it is very easy to act against this problem by simply defining explicitly all the attributes a requirement should have and collecting them in a table.**
 - This way, any missing items may be spotted directly.
- **The next two pages define the set of attributes with their names, purpose, and sample values.**
 - The table also indicates whether they may be used for sorting the requirements by them which obviously depends on their usage and restricts the attribute values.
 - Also, the table defines whether the attribute is facultative (optional) or mandatory (must be there).
- **The attributes should be filled in more or less in the order defined by the table.**

Requirements Attributes (Elicitation)

Stage	Attribute	Description	Values
Elicitation	Identifier	unique and persistent identifier	project specific, e.g. integers
	Name	descriptive term, possibly phrase	proper name, phrase
	Description	brief text describing what is included in this requirement	3-10 sentences, at most two paragraphs
	Rationale	A justification of this requirement: why is it being selected	1-3 short sentences OR reference to a goal
	Source	reference to origin of requirement	reference to documents, workshops, individuals, existing systems etc.
	Details	A more detailed treatment of this requirement	reference to an external document
	Remarks	any additional remark, e.g. comments or open questions	text in project language

Requirements Attributes (Elicitation)

Identifier

- A unique and persistent identifier. Identifiers never change and may never be reused.
- The simplest ID scheme is to use consecutive integers starting from 1.
- More complex systems might encode additional information like the type, domain, or level into the identifier.

Name

- A short and descriptive term, possibly a phrase.
- Usually, there are no constraints for requirement names, but with practice, people will be aware of what the requirement will be implemented as (e.g. a process), and name them according to the respective naming convention.

Description

- A brief text of 3-10 sentences, at most two paragraphs describing what is included in this requirement.
- If at all possible, use bullet lists or enumerations to structure the contents.

Rationale

- A justification of this requirement: why is it being selected (1-3 brief sentences).
- Even better is a reference to one of the goals from the goal model.
- If a suitable goal is not readily found, adding a goal might solve the problem.

Source

- Describes where the requirement came from.
- Typical values for this field include:
 - “interviews with users”
 - “Observation protocol #42, chief librarian Mads Tofte, 9.9.2009, 10:54:12-10:56:08”
 - “Usage scenario, line 27”

Details

- More lengthy description without constraints, or a references to external documents

Remarks

- Any additional comments that fit nowhere else.

Requirements Attributes (Elaboration)

- **Once the elicitation baseline has been established, the requirements need to be elaborated somewhat further so that they may be validated by the client.**
- **After an initial validation, requirements are ready to be used for project management purposes such as planning and scoping.**
 - Using the requirements as the basis for software development will need a further step (transition).

Requirements Attributes (Elaboration)

Stage	Attribute	Description	Values
Elaboration	Type	classification of requirement	- feature ("functional req.") - quality ("non-functional req.")
	Level	granularity level or scope of the requirement	-market / domain - product - feature - component
	Derived From	(1) Reference to a requirement that has been split up into several smaller requirements, that collectively replace the original requirement. (2) Reference to a requirement	reference to an obsolete requirement
	Acceptance Test	(1) operational procedure to test this requirement (2) quantification of minimum acceptable quality (3) reference to another artifact detailing the acceptance criteria such as a test class or test specification document	(1, 2) text in project language (3) path on project drive, CM system etc.

Aspects of Requirements: Elaboration

Type

- A classification of requirements according to a project specific schema.
- The most generic schema would be to distinguish between the following.
 - feature ("functional requirement")
 - quality ("non-functional requirement")
 - constraint – a project side condition
- Additional classifications might add interface requirements, or may want to differentiate features into crosscutting and self-contained requirements.

Acceptance test

- Any operational procedure to check whether a deliverable satisfies a given requirement.
- Typical examples are tests of any kind, or observable parameters and qualities like response times or presence of artifacts.

Derived From

- Many requirements are split up or significantly changed. In order to keep track of where a requirement came from, the “derived from” attribute links such requirements to their predecessor.

Level

- The level of granularity or scope to which a requirement applies.
- A raw collection of requirements will yield items at very different levels of abstraction.
- For instance, the requirements “identify book”, “return book”, and “media lending & returning” belong to 3 different levels:
 - “media lending & returning” is a domain containing a process “return book” which in turn contains the activity “identify book”.
- All of these may be fine examples of requirements, but they should be treated rather differently.

Crosscutting Features

- **There are different kinds of problems that requirements engineers encounter when dealing with specifying and implementing features.**
 - A feature may rely on other features being implemented before.
 - A feature may contain complex logic making it difficult to understand.
 - A feature may require large effort or advanced technology to implement.
- **While challenging in their own way, all of these cases are relatively benign, if the features are self-contained.**
 - In the malignant case, a feature is closely connected to many other features, either by its logic, or by its implementation.
 - Such features are called crosscutting, typical examples include Undo/Redo, global search, logging and monitoring, high-level business rules (e.g., four-eyes-principle), and legal constraints (e.g., auditable recording of bookings, compliance with capital market regulations).
- **Many required quality attributes (aka. non-functional requirements) are cross-cutting in nature, including integrity, much of usability, and performance.**



Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Chapter 7.3:

Reinforced Natural Languages

DTU course 02264

Reinforced Natural Languages (RNL)

- **There are several means of reinforcing natural languages.**
 - 1) **Glossary:** Standardizing the vocabulary in a glossary helps reducing ambiguity.
 - 2) **Tables/Outlines/Graphs:** With predefined document outlines, templates, and fixed tables, omissions become easier to detect, and the context supports disambiguation.
 - 3) **Writing Rules and Style Guides:** Providing grammatical and stylistic rules helps to avoid potentially dangerous constructions and words. Writing style rules are language specific.
 - 4) **Checkers:** Modern word processors include checkers for spelling and grammar.
 - 5) **Controlled Natural Languages (CNL):** CNLs drastically constrain the grammar and vocabulary to the point where automated tools parse and check CNL texts.
- **Each RNL text is a text in the natural language it is based on, but not vice versa. Some of these means can be combined (1..4, 1+5).**

Benefits and Problems with RNL

- **Obviously, glossaries, tables/outlines, and style guidelines are easy to implement, but hard to automate.**
 - Manual effort is expensive, slow, and error prone (→ regression).
- **Checkers and CNLs may be automated, but are expensive and difficult to create and use.**
- **Reinforcing natural language is no silver bullet.**
 - Reinforcing natural language will reduce the risk of mistakes originating from erroneous interpretation of RNL texts (e.g. a requirements specification), but there is no guarantee.
 - Also, avoiding mistakes does not automatically lead to a better end product, it just reduces the probability of delivering a bad one.
 - Finally, creating an error-free system does not necessarily lead to the right system (the one addressing the client's needs).
- **The general rule is to reduce complexity wherever possible.**

R1: Verbs and Tenses

- **Usage of present tense.**
 - Using the present tense generally makes for livelier and clearer prose.
 - Use IF-THEN, or PRE-ACTION-POST structures to highlight the parts of your
- **Usage of future tense.**
 - Sometimes, present tense is also used to describe preconditions.
 - Avoiding IF-THEN, or PRE-ACTION-POST structures requires to use future tense for the requirements.
- **Use key verbs consistently (see e.g. RFC 2119 guidelines).**
 - The verbs “shall,” “will” or “must” express a definite need.
 - The verbs “should”, “should not” or “may” express only a wish or desire.
 - The verbs “shall not” and “must not” express a definite needs.
 - Avoid “may not” and “will not”.
 - Use the qualifiers “required” or “optional” to refer to required and optional features or behavior, respectively.

R2: Prefer Active Voice

- **Passive allows to construct grammatical sentences without actor.**
 - They “sound ok”, i.e., it is hard to spot the omission when just reading the sentence.
 - If no actor is specified, confusion may arise concerning who is doing what (e.g. the user, the system, some component of the system, a neighbor system).
 - In active voice, such omissions are not possible without breaking the sentences (which is easy to detect).

No.	Wrong	Risky	Safe
a	If the query is refused, the reader issuing the query shall be referred to the front desk service.	If the query is refused by the remote catalog, the reader issuing the query shall be referred to the front desk service.	If the remote catalog refuses the query, the LMS shall refer the reader issuing the query to the front desk service.
b	If the catalog search is closed, the LMS shall delete the search history.	If the catalog search is closed by the user, the LMS shall delete the search history.	If the user closes the catalog search, the LMS shall delete the search history.
c	Reports of all lent and reserved items may be created.	Reports of all lent and reserved items may be created by librarians.	Librarians may create reports of all lent and reserved items.
d	To register a guest, the login data is entered on a mobile device or terminal.	To register a guest, the login data must be entered on a mobile device or terminal by the guest	To register, a guest must enter login data on a mobile device or terminal.

R3: Avoid Empty Verbs

- Empty verbs function as a predicate of a clause together with a noun, usually expressing a state or change of state.
 - Using empty verbs makes a requirement unnecessarily complex and may lead to misinterpretations.
 - Expressions using empty verbs can usually be simplified.

No.	Risky	Empty Verb	Safe
a	User registration is performed before <X>.	perform	A user must register before <X>.
b	<X> exerts influence on <Y>	exert	<X> influences <Y>
c	<X> gets access to <Y>	get	<X> accesses <Y>
d	<X> gives a reply to <Y>	give	<X> replies to <Y>
e	<X> makes a contribution to <Y>	make	<X> contributes to <Y>
f	<X> does increase <Y>	do	<X> increases <Y>

Before the guest can log into the system, registration must have been **performed**.

Who or what is actually doing the registration? Just when is it completed? What happens if it is only half-finished?

Before the guest can log into the system, the guest must have completed registration.

R4: Avoid Incomplete Verbforms

- **Many verbs have implicit references to various objects involved in the activity described by the verb.**
 - Common examples are: prepare, communicate, report, and notify.
- **Often, not all of these objects are present. They can be detected by asking: who, what, when, how, ...**

No.	Risky	Questions	Safe
a	LMS processes and saves the data.	Which data? What processing?	LMS processes and saves the registration data. <i>+ specification of "registration data" (e.g. glossary entry, data model entity, bullet list)</i>
b	When a reserved medium is returned, the librarian is notified.	Where/How is it returned? When is which librarian notified? With which message? How is he notified?	When a medium is returned at the front desk, the librarian present there is notified within 1s if the medium is currently reserved.
c	Each reader is provided with a unique identifier.	When is he provided with the id and by whom?	The subsystem "ReaderAccount" provides a new unique reader identifier when a reader is registered for the first time.

R5: Avoid Negation

- **Reduce negations whenever possible.**
 - Express requirements positively.
- **Absolutely avoid double negations.**
 - Double negation is difficult to process cognitively and invites errors and misinterpretations.
- **Be explicit about different cases.**
 - If necessary, split requirements up into several individual cases.

No.	Risky	Safe
a	Nobody except chief librarians may edit the transaction log.	Only chief librarians may edit the transaction log.
b	The input length is not unrestricted.	The input length is restricted.
c	All but the most wanted media have an unrestricted loan period.	The most wanted media have a loan period restriction. Media other than the most wanted media have no loan period restriction.

R6: Complete Comparisons and Attributes

- **If your requirement involves system properties with a degree, quantify them.**
 - Many attributes (grammatically speaking) imply a quantity or measure, that is, they all answer the question “how (much)” (e.g., easy, fast, large, etc.).
 - This also applies to more technical notions: modularity, coupling/binging, reusability.
- *We will elaborate this topic in the next Chapter “Quality Attributes”.*

No.	Risky	Questions	Safe
a	Reserving a medium from a search result list must be easy.	How easy? In comparison to what? Under all circumstances?	If the reader is logged in and has conducted a catalog search, any of the search results may be reserved by a single mouse click, the status of the reader and the medium permitting.
b	The most common queries are cached to speed them up.	How many queries? When is a query considered „common“? How fast shall they be?	The 100 most common queries of the last 3000 library opening hours are cached. Cached queries shall be executed with a latency of less than 1s.

Rule 4: Incomplete Comparisons

- **Comparisons and all kinds of quantifications should be made complete and precise.**

R4a: “All other readers will receive their monthly lending status by email.”

- To what does the comparative “all other” refer?

R4b: “All readers that have selected the “lending status by email” will receive their monthly lending status by email.”

- **Adjectives may also hide a comparison.**

R5a: “It must be simple for readers to access their lending status online.”

- Simple to access... in comparison to what?
- How is accessing the lending status made simple for readers?
- How can we measure simplicity of access?

R5b: “For 95% of the Readers obtaining access to their lending status online for the first time must not take them longer than 30s from login to status display”.

R7: Use Specific Conditions

- **Specify all branches of a condition, if necessary with a catch-all.**
 - This commonly leads to splitting up requirements.
 - Rules similar to refactoring of code apply.

No.	Risky	Questions	Safe
a	Readers may reserve media if they are leased.	Do you mean to say “Any Reader” or does it take two to reserve? How many media are there in a reservation?	The reader may reserve any leased medium.
b	Readers have unique identifiers in the LMS.	How many readers? How many identifiers?	Each reader has exactly one unique identifier.
c	-	-	When the reader returns several books at once, ...

R8: Use Specific Quantities

- **Be specific about numbers and quantities.**
 - use: all, exactly one, one or more, at least one
 - avoid: one, several, many
- **Prefer definite articles over indefinite articles**
 - use: he/she/it, the, this, that, these, any, all
 - avoid: no article, a/an

No.	Risky	Questions	Safe
a	If two readers with „Proust“ status have reserved the same medium, ...	What if there are three such readers? What happens when the readers have another status?	If two or more readers with „Proust“ status have reserved the same medium, ... If two or more readers with different status ... In any other combination of two or more conflicting reservations, ...
b	If the reader asks for mail delivery of the leased media, he will be notified of the costs.	What about other forms of delivery?	-

R9: Make Universal Quantifiers Explicit

- **Universal quantification is a powerful concept – make sure you really intend what you say!**
 - Look for exceptions to the rule, check boundary cases.
 - Universal quantification hides behind: all, no, none, always, never, every, etc.
- **Reduce ambiguity by using only a small number of approved quantifiers.**
 - use: no/none, any/each, always, every, either/neither
 - avoid: some (→ one or more),

No.	Risky	Questions	Better
a	Librarians may enter new readers to the system with a reader dialog.	ALL Librarians – really? Is that what we want?	-
b	The 100 most common queries of the last 3000 library opening hours are cached.	All queries, including those that come via the web?	The 100 most common queries issued from any library terminal within the last 3000 library opening hours are cached.
c	Cached queries shall be executed with a latency of less than 1s.	Under all circumstances, even when configuring the system?	Cached queries shall be executed with a latency of less than 1s under normal operation.

R10: Move conditions to the front

- **Many requirements are restricted to a specific operational mode, a time when they may be applied, or some system state.**
 - If such conditions are only mentioned at the end, it amounts to an implicit negation making such a requirement more difficult and time consuming to interpret.
 - Move conditions to the front to avoid negation and to allow readers to abort reading a requirement earlier if it does not specify their case.

No.	Difficult	Easier
a	The reader may not create another reservation, initiate a new lease, or prolong an existing lease while the LMS is updating the reservation log.	If LMS is updating the reservation log, a reader may not create another reservation, initiate a new lease, or prolong an existing lease.
b	Cached queries shall be executed with a latency of less than 1s under normal operation.	Under normal operation, cached queries shall be executed with a latency of less than 1s.

R10: Reduce complexity

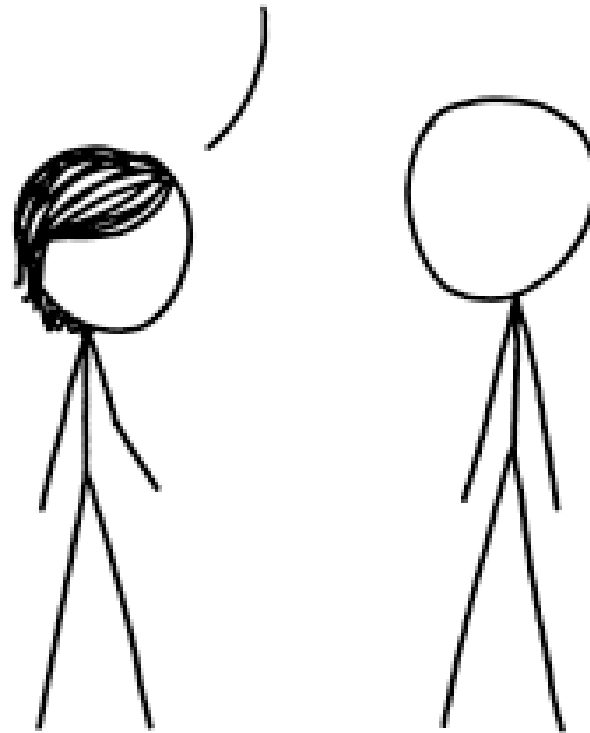
- **Obscure and complex terms and expressions make your requirements difficult to interpret.**
 - Remove all redundant expressions!
 - Replace all complex expressions and obscure terms by simpler ones!

Complex	Simple	Redundant
bring to an end...	complete, end	along the lines of
at all times	always	as a matter of fact
at the present moment, at this point in time	now, currently	actual, actually
by means of	by	as it were
grace to, by virtue of	because of	basically, essentially
as to whether	whether	completely
being that	since	extremely, totally
so as to, in order to	to	in terms of
utilize	use	moreover
is capable of	can	very
all of	all	quite
on a daily basis	daily	the fact that

Rules for Glossary Entries

- A glossary definition could be an analytical definition, an enumeration of characteristics, or a behavioral description.
- Referring to other glossary entry in the definition is fine (including circular references), if the reference is made explicit, e.g. by preceeding it with →.
 - A reader is a natural person possessing exactly one set of person-specific data (including address and media preferences) who leases →media.
- Always follow the schema <Subject> <Verb> <Object>.
- Split up different aspects into separate sentences.
 - A reader is a natural person.
 - A reader possesses exactly one set of person-specific data (including address and media preferences).
 - A reader leases →media.

I DON'T MEAN TO GO ALL LANGUAGE
NERD ON YOU, BUT I JUST LEGIT
ADVERBED "LEGIT," VERBED "ADVERB,"
AND ADJECTIVED "LANGUAGE NERD."





Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Chapter 7.4:

Simplified and Controlled Natural Languages

DTU course 02264

Simplified Languages

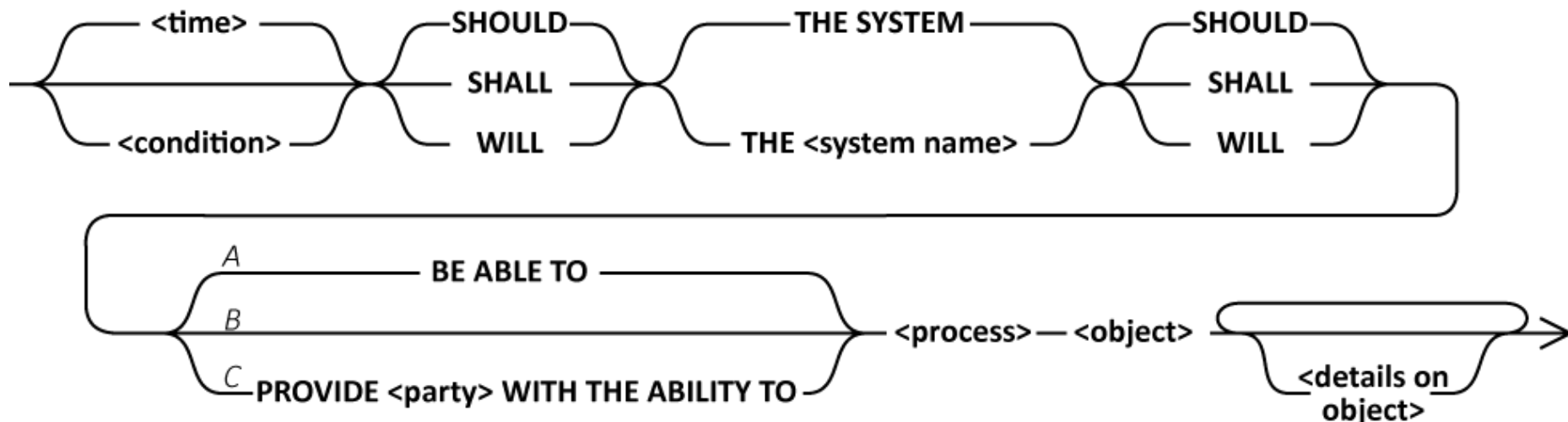
- **Many major languages have simplified forms.**
 - Plain Language http://en.wikipedia.org/wiki/Plain_English
 - Leichte Sprache http://de.wikipedia.org/wiki/Leichte_Sprache

- **Purpose & Audience**
 - These languages are primarily directed towards people with language-related challenges, such as non-native speakers, semi-illiterate people, and people with cognitive or perceptual deficits.
 - The United Nations support plain language as one of the *"modes, means and formats of communication."*
 - Requirements engineering may also benefit from simplified languages.

- **Simplified languages disallow certain ambiguous constructions, and impose rules to make the language easier to understand.**
 - Sentences may not exceed 8 words, no passive voice, defined vocabulary, ...
 - One statement per sentence, one sentence per line, ...
 - In German: keep verbs together, hyphenate concatenated words, ...

Requirement Templates

- A common way to reinforce prose for requirements specification is to apply language patterns or templates.
 - These can be expressed in EBNF-like structures, such as railroad diagrams.
- Below, there is a template that allows to express three kinds of requirements pertaining to reactive/proactive system behavior, and user interactions, respectively.



A - reactive behavior

B - proactive behavior

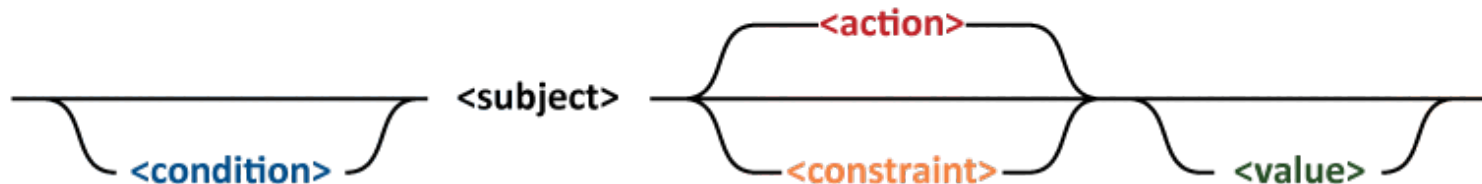
C - user interaction

Requirements Templates

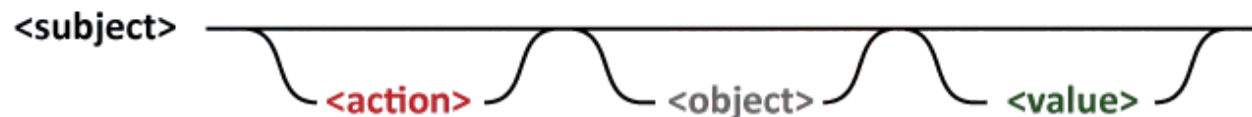
- Another set of templates is proposed by ISO 29148:2011, which is presented here in an abridged and extended version.



When signal x is received, the system shall set the signal x received bit within 2 seconds.



At sea state 1, the Radar System shall detect targets at ranges out to 100 nautical miles.



The Invoice system shall display pending customer invoices in ascending order in which invoices are to be paid.

Controlled Languages

- **The strongest kind of reinforcement possible for natural languages are Controlled Natural Languages (CNL).**
- **A CNL is a strict subsets of some proper Natural Language (e.g. English, German, or Chinese), but with a tightly restricted vocabulary and grammar.**
 - Only a relatively small subset of the original natural language is used.
 - This allows for fully automatic processing of texts written in these languages, like error checking, translation, and summarizing.
 - For some CNLs, there is even a direct translation into some formal logic (cf. ACE \rightarrow DRT-E)*.
 - Reading CNL texts is somewhat boring to humans (like most technical texts), but requires only limited knowledge of the base language.
- **Writing CNL is hard, when no interactive checkers with good advice are available, but feasible nevertheless.**

*See paper by Fuchs, Schwertel, and Schwitter.

Simplified Technical English (ASD-STE100)

- **One example of a Controlled Natural Language is the Simplified Technical English (STE).**
 - STE consists of 61 rules and approximately 2800 words and word forms.
 - Texts conforming to STE are regular simple English, although they do sound somewhat mechanical and inelegant at times.
 - However, everyone who speaks a fair amount of English can understand it.
 - Sentences conforming to STE are rather safe from ambiguity and misunderstandings.
- **As for most controlled languages, STE is geared towards mechanical engineering/maintenance application domains**
 - A typical example is a technical handbook for the maintenance of planes.
 - The vocabulary is not particularly well suited for describing IT purposes like typical commercial business processes.
 - Applying STE to such applications will require to modify the vocabulary substantially.

- **The STE Dictionary defines the admissible vocabulary by:**
 1. Acceptable words with part of speech, meaning and examples.
 2. Unacceptable words with part of speech, meaning, acceptable replacement, examples and counter examples.
- Generally, ACCEPTABLE TERMS ARE WRITTEN IN UPPER CASE, while unacceptable terms are written in lower case.

ASD100

<i>Word</i>	<i>Meaning Alternative term</i>	<i>approved example</i>	<i>non-approved example</i>
abnormality (n)	DEFECT (TN)	EXAMINE THE CANOPY SEAL FOR DEFECTS.	Inspect the canopy seal for abnormalities
ACCESS (n)	The “ability” to go into or near	GET ACCESS TO THE ACCUMULATOR.	-
evenly (adv)	GRADUALLY, EQUALLY	INCREASE THE TEMPERATURE GRADUALLY.	Increase the temperature evenly.
		APPLY THE LOAD EQUALLY ON THE AREA.	Apply the load evenly on the area.
exactly (adv)	CORRECT (adj), FULLY	THE SEAL MUST BE OF THE CORRECT DIMENSION FOR THE GROOVE.	The seal must fit the groove exactly.
		OBEY THE PROCEDURE FULLY.	Obey the procedure exactly.

STE Rules

Rule 1.2: Use approved words from the Dictionary only as the part of speech given.

Each approved word in the Dictionary has a part of speech.

Do not use it as another part of speech for which it is not approved.

For example, if a word is given only as a noun, do not use it as a verb.

- Example: “Test” is approved as a noun but not as a verb.

Non-STE: Test the system for leaks.

STE: Do the leak test for the system.

STE: Do a test for leaks in the system.

- Example: “Close” is a verb and not an adverb.

Non-STE: Do not go close to the landing gear if ...

STE: Do not go near the landing gear if ...

Rule 1.3: Keep to the approved meaning of a word in the Dictionary.

Do not use the word with any other meaning.

- Example: “Follow” means “come after”. It does not mean “obey”.

Non-STE: Follow the safety instructions.

STE: Obey the safety instructions.

Natural Language

Removal of the Overspeed Governor.

1. Any leaking oil can be collected by placing the cleaning cloth under the overspeed governor (3).
2. Disconnect electrical connector (2) from electrical receptacle (1).
3. Note the position of bracket (6), it must be installed in the same position. Remove nut (4) and washer (5), which attaches bracket (6) and overspeed governor (3) to the gearbox. Nut (4) is to be discarded.
4. The bracket (6) is to be safetied to the aircraft structure with a temporary tie, away from work area.
5. Holding the overspeed governor (3), remove remaining three nuts (9) and three washers (8), which attach overspeed governor (3) to gearbox. Three nuts (9) are to be discarded.
6. Remove overspeed governor (3) from gearbox.
7. Protect electrical connector (2), electrical receptacle (1), and mounting pad of gearbox by installing an applicable blanking cap.
8. Remove packings (10 and 11) from overspeed governor (3). Packings (10 and 11) are to be discarded.
9. Remove, and safely discard cleaning cloth.

ASD-STE100

Remove the Overspeed Governor.

- A. Put the cleaning cloth below the overspeed governor (3) to collect the oil leakage.
- B. Disconnect the electrical connector (2) from the electrical receptacle (1).
- C. Record the position of the bracket (6), you must subsequently install it in the same position.
- D. Remove the nut (4) and the washer (5) that attach the bracket (6) and the overspeed governor (3) to the gearbox.
- E. Discard the nut (4).
- F. Use a temporary tie to safety the bracket (6) to the aircraft structure, away from the work area.
- G. Hold the overspeed governor (3). Remove the remaining three nuts (9) and the three washers (8) that attach the overspeed governor (3) to the gearbox.
- H. Discard the three nuts (9).
- I. Remove the overspeed governor (3) from the gearbox.
- J. Install an applicable blanking cap to prevent damage to:
 - The electrical connector (2)
 - The electrical receptacle (1)
 - The mounting pad of the gearbox.
- K. Remove the packings (10 and 11) from the overspeed governor (3).
- L. Discard the packings (10 and 11).
- M. Remove, and safely discard the cleaning cloth.



Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Chapter 7.5: Use Case Templates

DTU course 02264

Use Cases

- **A Use Case is a scenario of a user interaction with a system.**
 - The "User" may be another system, and the Use Case may comprise more than just one scenario.
- **Use Cases have been made popular with the Objectory Method.**
 - Their popularity is to a large degree explained by the fact that they are the only way to abstractly specify functionality, i.e., doing top-down analysis in an OO setting: Use Cases are to Object-Oriented methods what Data-Flow-Diagrams are to structured methods.
- **Use cases can be used for prioritizing and project/release planning.**
 - The processes summarized in the domain architecture are basically (bundles of) use cases.
- **Use cases are not well-suited for expressing cross-cutting concerns, including quality attributes.**

Describing Use Cases

- **Use cases may be described in different degrees of formality:**
 - Basic: a sequence of steps described in prose.
 - Simple: a predefined tabular schema with a couple of properties, one of which is a basic use case description.
 - Complete: a simple use case description embedded in prose text defining the purpose and side conditions of the use case, and complemented by one UML use case model for each individual use case to illustrate its embedding, and one or more overview use case diagrams to show the relationships between the whole set of use cases.
- **It is helpful to describe all use cases in the basic format before going on to elaborate individual use cases.**

Simple Use Cases (Tabular Description)

- **Tables are common to help refining basic use cases.**
 - There are many different templates like this.
 - The user (i.e., a person or a system) is called Actor here to be in line with the UML terminology.

- **Remarks to table**
 - The slots marked with ¹⁾ and ²⁾ are optional, but at least one of them must be filled, respectively; the slots marked with * are optional.
 - Slots that are empty on purpose must be filled with a dash.
 - The text in angle brackets are placeholders, the italic texts describe the slots.

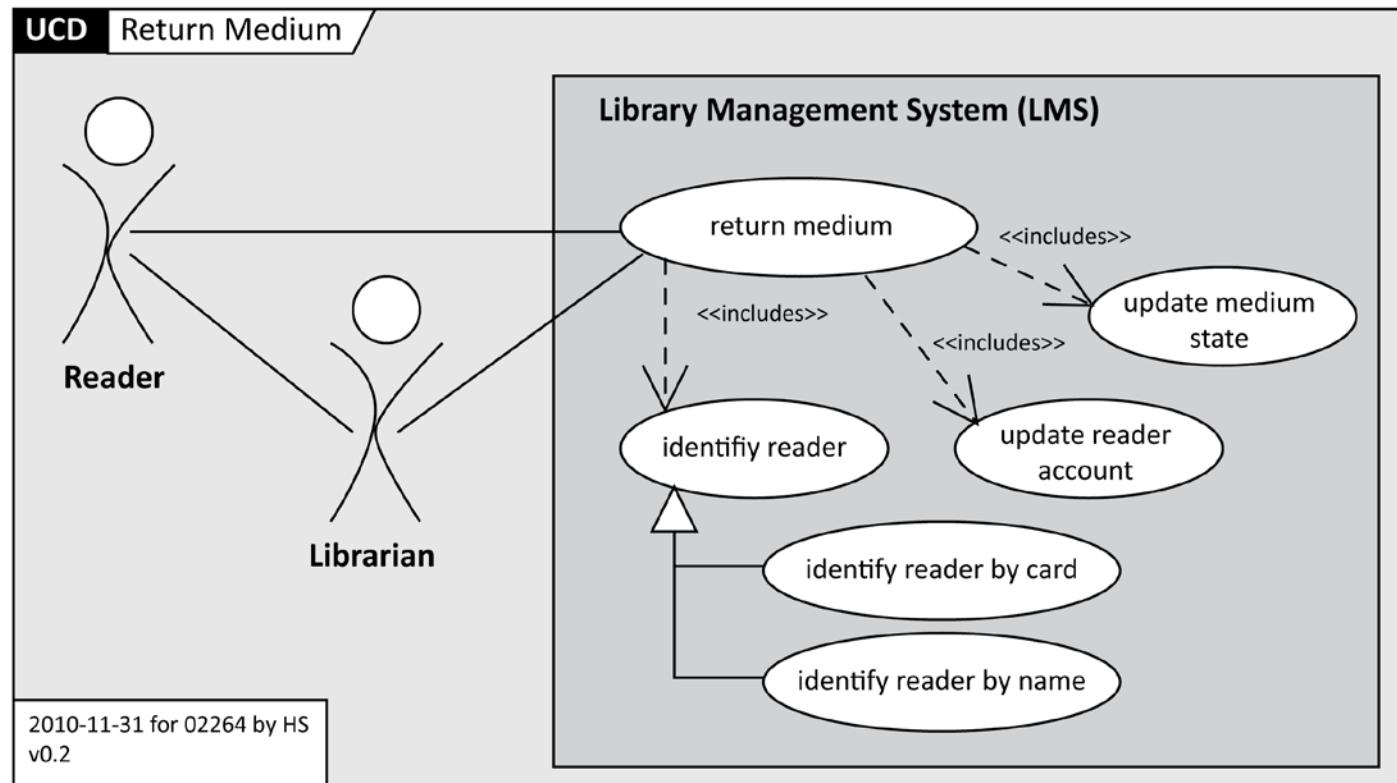
<ID>	<Name>		
Description	<i>a few sentences at most</i>		
Actors	<i>primary actor triggers this UC secondary actors may participate</i>		
Trigger¹⁾	<i>an event or command</i>		
Parameter¹⁾	<i>data needed to trigger this UC</i>		
Preconditions¹⁾	<i>system states required to trigger this UC</i>		
<table> <tr> <td>Regular Scenario <i>the most common or least problematic course of action ("sunshine scenario")</i></td><td>Variants <i>any other courses of action with exceptions, problems, etc.</i></td></tr> </table>		Regular Scenario <i>the most common or least problematic course of action ("sunshine scenario")</i>	Variants <i>any other courses of action with exceptions, problems, etc.</i>
Regular Scenario <i>the most common or least problematic course of action ("sunshine scenario")</i>	Variants <i>any other courses of action with exceptions, problems, etc.</i>		
Postconditions²⁾	<i>system states guaranteed after completing this UC</i>		
Result²⁾	<i>data provided or computed by this UC</i>		
Incidence*	<i>frequency of this UC</i>		
Duration*	<i>duaration of this UC (average, best case, worst case)</i>		
References*	<i>references to other documents</i>		
Remarks	<i>open questions, comments</i>		

Simple Use Case (Example)

UC_1	Return Medium	
Description	<i>A medium is scanned by a user (either a reader at a self-service terminal or a librarian at a front desk terminal) in order to identify the medium and take the obvious action (i.e. lending or returning)</i>	
Actors	<i>1: Librarian / Reader</i>	
Trigger	<i>identification of reader OR selecting action from menu</i>	
Parameter	-	
Preconditions	-	
Regular Scenario <i>1. identify reader and medium</i> <i>2. determine lease, reader account and medium state</i> <i>3. terminate lease, update reader account</i> <i>4. acknowledge medium being returned</i>		Variants <i>a) lease is overdrawn → computer due fees</i> <i>b) allow to settle outstanding fees</i>
Postconditions	<i>regular scenario: lease terminated, reader account updated:</i> <i>b) outstanding fee decreased by payment amount</i>	
Result		
Incidence	<i>500 times a day</i>	
Duration	<i><0.3 seconds</i>	
References	-	
Remarks	-	

Complete Use Cases

- Complete Use Cases provide illustrations to the basic and simple use case descriptions. These illustrations (and only these) are UML's use case diagrams.
- The purpose of Use Case Diagrams is just to collect information in a visual way, provide overview, and facilitate the communication between the stakeholders.



Use Cases vs. Business Processes

- **A Use Case is a sheaf of similar scenarios or interaction of users with a system**
 - similar pre-/post conditions
 - similar primary Actor
 - same user goal/rationale
 - similar individual steps
- **Use Case Models may be used**
 - either to describe a (business) process (“Business Use Case”)
 - or to describe individual activities inside a process (“System UC” or “Product UC”).
- **A Business Process (BP) is a value-/cost-producing process.**
 - It may span more than one organization, usually involving several people and systems, and often lasts for days to months, possibly even longer. BPs may often be interrupted and suspended, they frequently have multiple courses and outcomes and require a substantial amount of interaction.
- **A System Use Case (SUC) is a usage scenario of a system.**
 - It spans only one system and its users (sometimes also systems contributing to the SUC), and lasts up to minutes, possibly hours. Use cases may not be suspended or interrupted and often run automatically or require very little interaction outside the system.
- **Most of the time, the term Use Case is used for either of these...**
...and, unsurprisingly, many people confuse the two.



Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Chapter 7.6: Snow Cards

DTU course 02264

Snow Cards

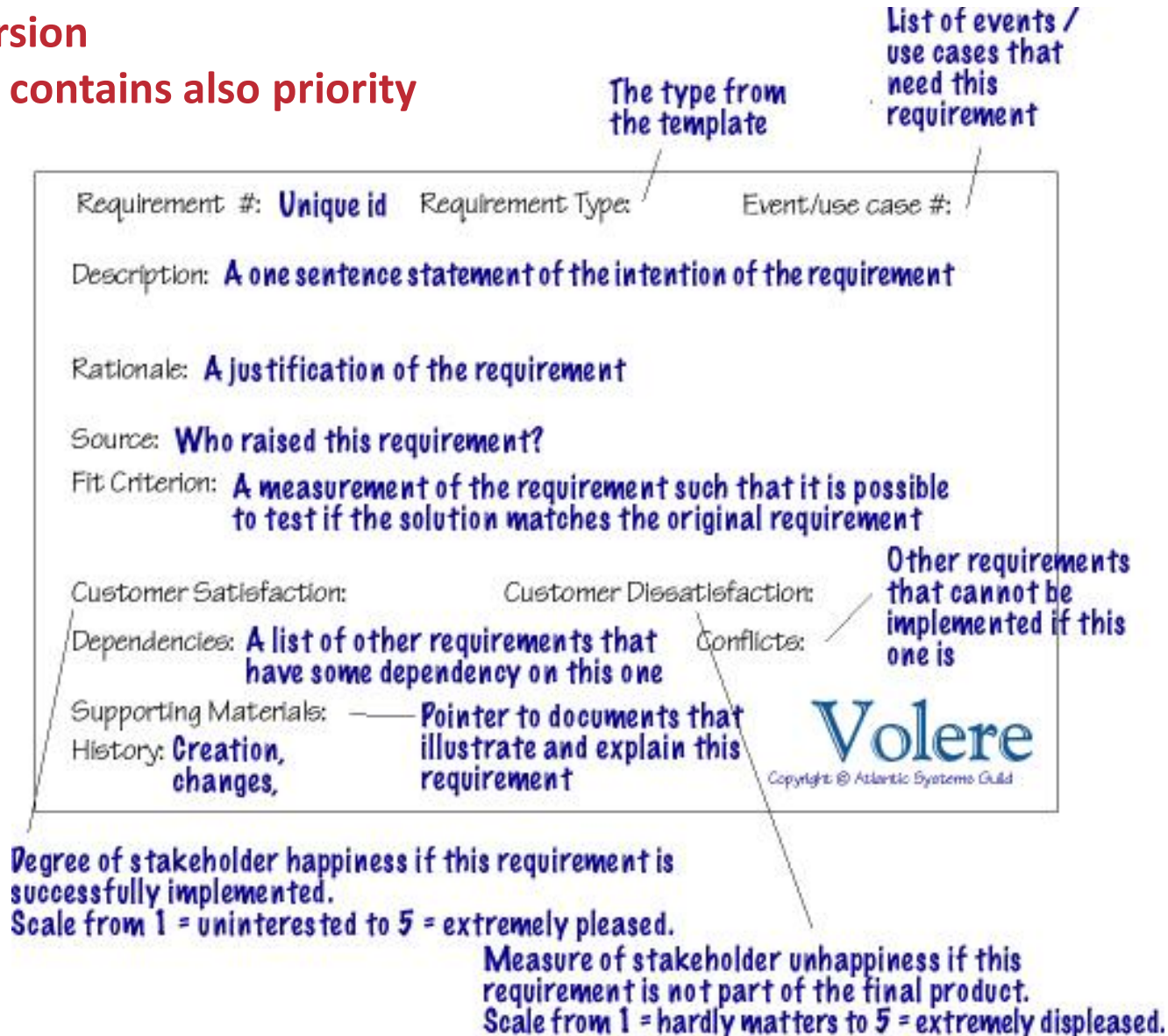
- **A popular technique for gathering and elaborating requirements are “Snow Cards”.**
 - A Snow Card is an index card with a set of attributes any requirement needs.
 - Capturing them as index cards allows to use them in group exercises and visualizations, but may also be useful for sorting and clustering them.
 - The predefined structure makes it easy to check a requirement for formal completeness.
 - Another big advantage is the size limit imposed by the medium.
 - Sometimes, however, the size limit is perceived more as a disadvantage.

- **Probably the most well known example of snow cards in RE is the “Volere-Template” (also popular: CRC-cards).**
 - In XP, User Stories are supposed to be captured on index cards, too, mainly for practical reasons like sorting them in the cards game.
 - Since User Stories are plain text and not table-like structures, completeness is not easily checked. Here, the plain-text rules should be used.

Volere-Template

Obsolete version

new version contains also priority



XP User Stories

- **Developer and User write a scenario together.**
 - Using A5 index cards to write them down restricts the size.
 - Later on, the index cards may be used conveniently for planning and prioritizing, as well as distributing and tracking requirements.
 - Involving the customer ensures commitment and validity (theoretically).
- **A User Story consists of five elements:**
 1. administrative data (author, date, version);
 2. planning data (effort estimate, dependencies/prerequisites);
 3. a unique identifier and a descriptive name;
 4. the story proper (necessary);
 5. an acceptance test (optional, use flip side, reference to code base).
- **The story proper is a very concrete account of how the system shall behave from the perspective of the user.**
- **Can contain functional as well as non-functional requirements e.g. "As a library user, I want to search for books and see the results within a second"**
- **acceptance tests are the formal documentation of the requirements**

XP User Story Example for LMS

Using a template can help you getting started writing user stories.

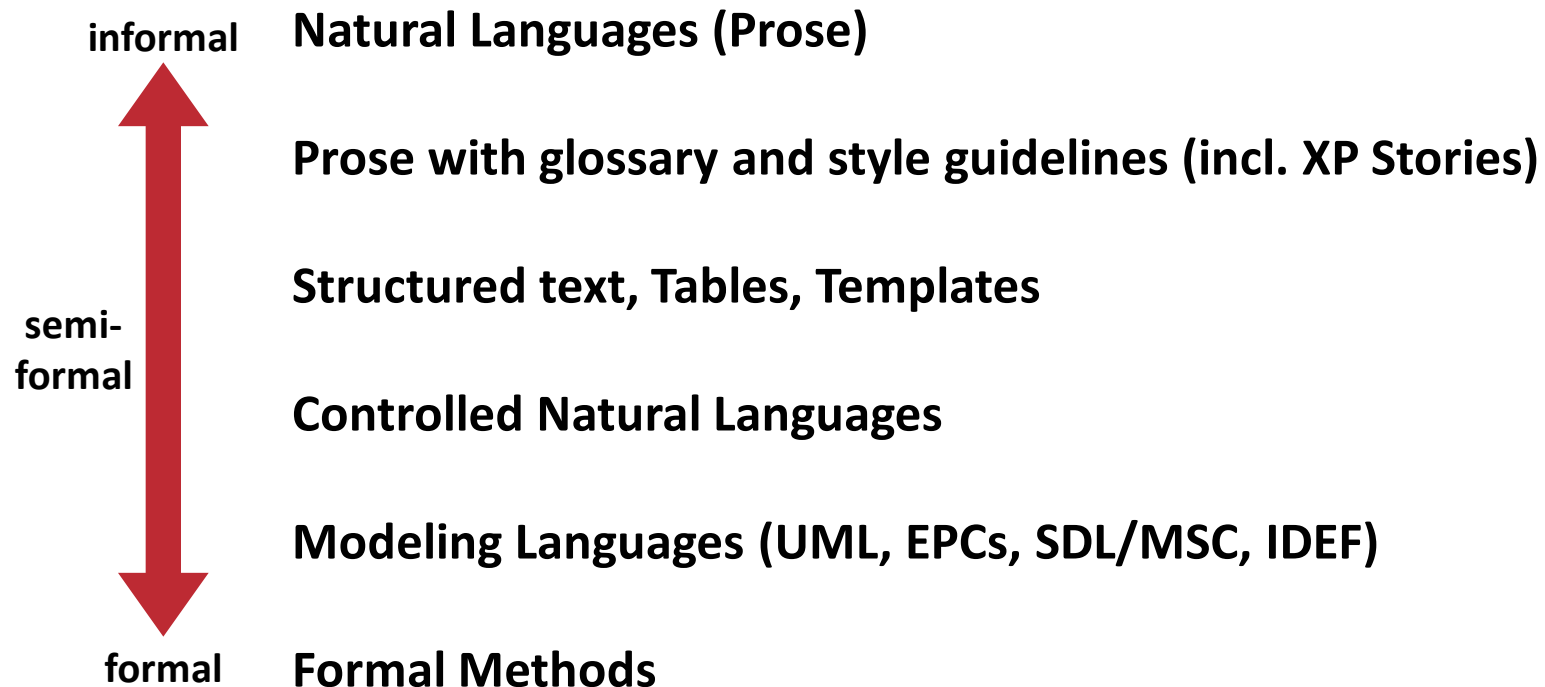
As a <role>, I want <goal>
[so that <benefit>].

- 1. For the first sentence of a user story, fill in values for the variables. Then, ...**
- 2. ...describe the current state,**
- 3. ...the actions and reactions that occur, ...**
- 5. ... and a final state.**

- 1. As a **reader** I want **to return a book to the library**.*
- 2. I have a small amount of outstanding fees, the lease for the book I want to return has not yet expired.*
- 3. I scan my ID card and then the ID badge of the book I want to return.*
- 4. If the book is not damaged, the system terminates the lease and records this fact. The book is made “available” again, and the reader holding the first reservation (if there is any) is informed of this state change.*
- 5. After log out or time out, a receipt is printed for me that states the amount of fees I owe, and the session is ended.*

Techniques in contrast

- There is a large number of techniques to specify requirements ranging from very informal techniques to completely formal techniques.
 - Most practically relevant techniques are semi-formal.
 - The higher the formality, the higher is the degree of automatable tasks.



Techniques in contrast

Technique	Strengths	Weaknesses	automatable tasks
Prose	<ul style="list-style-type: none"> generally available no learning effort understandable 	<ul style="list-style-type: none"> no formal structure ambiguous and vague little automation 	<ul style="list-style-type: none"> spell check grammar check
Prose (+glossary, guidelines)	<ul style="list-style-type: none"> flexible reduces ambiguity understandable 	<ul style="list-style-type: none"> no formal structure little automation checks mostly manual 	<ul style="list-style-type: none"> conformance to guideline review support
Structured text Tables	<ul style="list-style-type: none"> easy to implement omissions are obvious comparison is easy 	<ul style="list-style-type: none"> weak formal structure designing good tables is difficult 	<ul style="list-style-type: none"> conformance to outline simple transformations
Controlled Languages	<ul style="list-style-type: none"> reduces ambiguity understandable 	<ul style="list-style-type: none"> not visual inelegant language needs tailoring to domain learning effort 	<ul style="list-style-type: none"> strict conformance testing translation to other lang.
Modeling Languages	<ul style="list-style-type: none"> visual languages support understanding and presentation powerful tools available 	<ul style="list-style-type: none"> learning effort needs tool support not yet fully standardized 	<ul style="list-style-type: none"> (partial) code generation model transformation simple verification tasks
Formal Methods	<ul style="list-style-type: none"> many mathematical methods may be applied 	<ul style="list-style-type: none"> enormous learning effort difficult to apply 	<ul style="list-style-type: none"> full scale verification