

# Chapter 11



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

---

## Chapter 11: Transition to Design

---

DTU course 02264



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

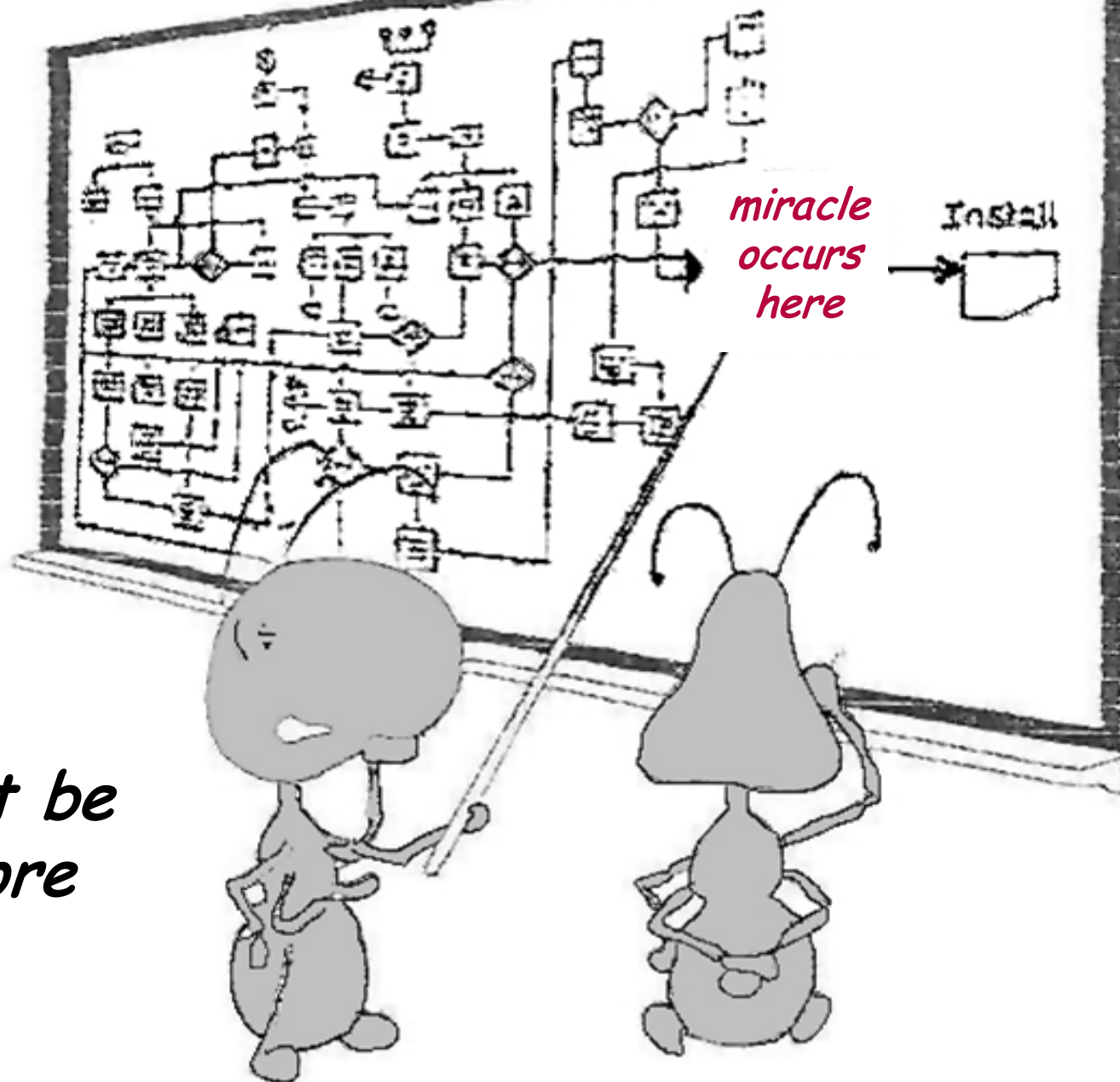
---

# **Chapter 11.1**

## **From Requirements to an Analysis Model**

---

DTU course 02264



*"Great job!*

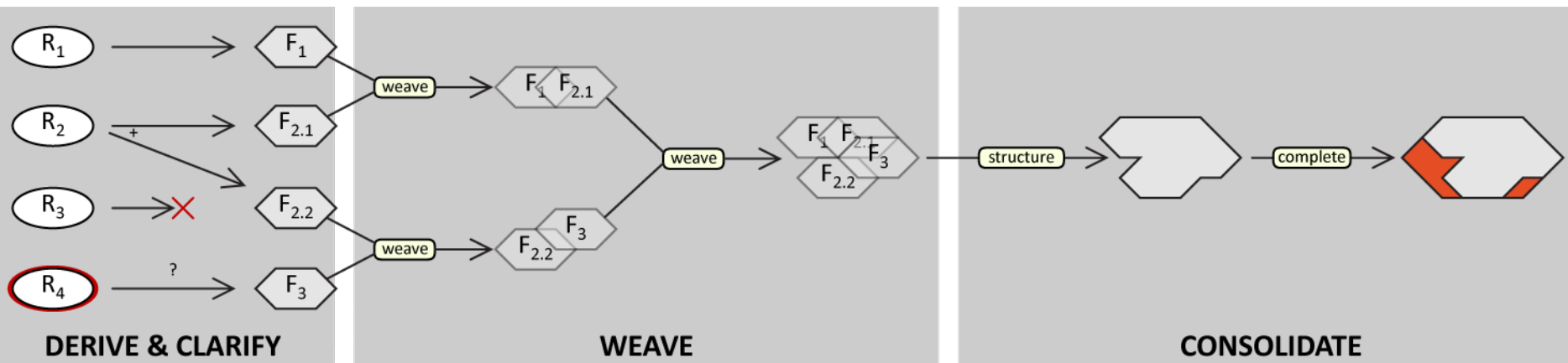
*But shouldn't it be  
a just little more  
detailed over  
here...?"*

# Requirements to Analysis Level Models

- **In many domains, the only viable way of expressing requirements is using unstructured natural language (“prose”).**
  - Prose is the one language universally understood. Also, tool support is widely available at no extra cost for licenses or training (i.e., MS Office).
  - The short term gains of this choice outweigh the long-term gains offered by a more structured approach in the eye of many clients.
  - However, as soon as the design and implementation start, most prose descriptions exhibit a lack of precision, and missing appeal to technical staff.
  - These type of people are more in favor of notations such as UML.
- **If we wish to maintain prose as our initial notation but transition into UML models, we face two problems.**
  - The transitions is difficult, error-prone, invites misunderstanding, implies many design decisions (as well as requirements resolutions), and be done only by skilled modelers.
  - As a consequence, it may be difficult to convince clients that the model is actually a faithful translation of their textual requirements.

# Weaving Models

- Instead of one single complex manual translation, we propose a sequence of simple transformation steps:
  - Translate individual requirements into small model fragments,
  - weave the fragments together into a comprehensive model,
  - consolidate the model manually.
- Each step requires only one type of knowledge (domain or technology), and some steps can be partially or fully automated.
- This procedure also creates tracing links between requirements and design automatically.



# Requirements-to-Model Transformation

**Elaborate Requirements until they appear sufficiently precise.**

## 1: Harvest

**Harvest each requirement into one or more (small) fragments of UML models.**

If the translation results in a large or many different fragments, decompose requirement further or justify “large” requirement.

## 2: Weave

**Weave all fragments in to a raw comprehensive model with RED.**

Improve the layout such that it makes sense to you. If there are unexpected results (errors, omissions, duplicates), change requirements and/or translation to compensate, and repeat weaving.

## 3: Consolidate

**Consolidate the raw model into a final model.**

Model structures may suggest, by symmetry, the addition of other model elements not yet implied by the requirements. The occurrence of design anti-patterns may suggest overlap in requirements, which are potentially inconsistent. The model may need to be reformulated, completed, and restructured.

# Step 1: Translate

## Requirement

### MLC2

Librarians may add, update, and delete corpus items manually.

### MLC4

Librarians and Readers may post and inspect media they think should be acquired by the library to a public “wish list” indicating the status of the wish and the originator.

### MLC6

Librarians may remove or deactivate entries to the wish list.

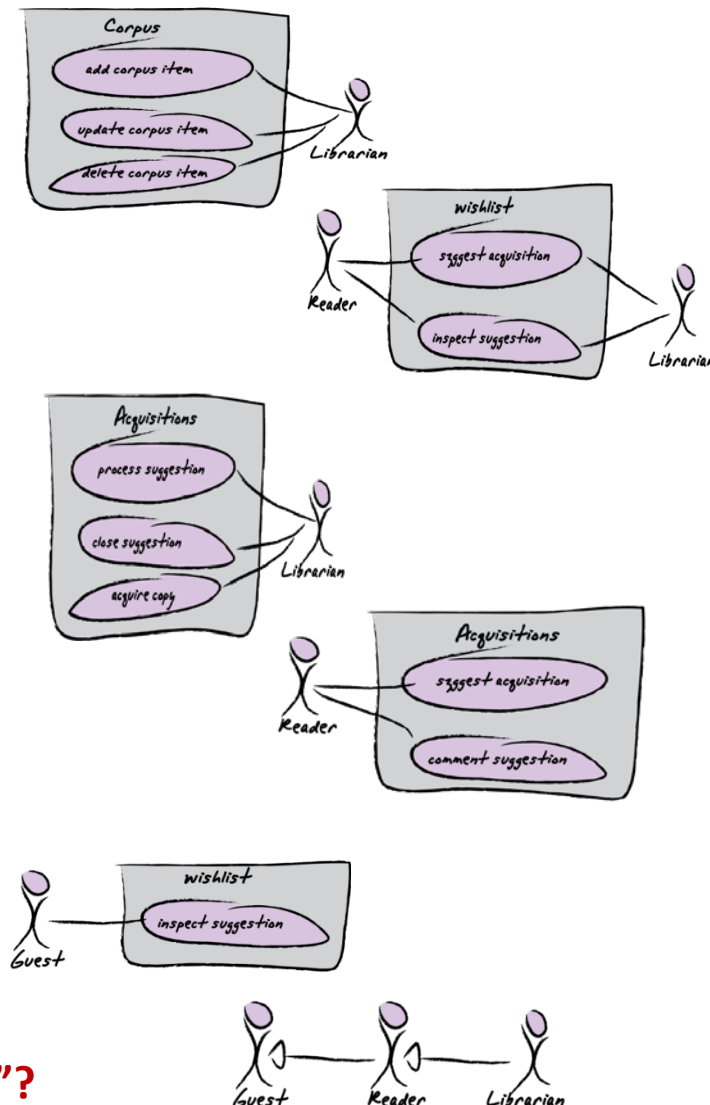
### MLC9

Guest readers may inspect suggestions.

### MLC10

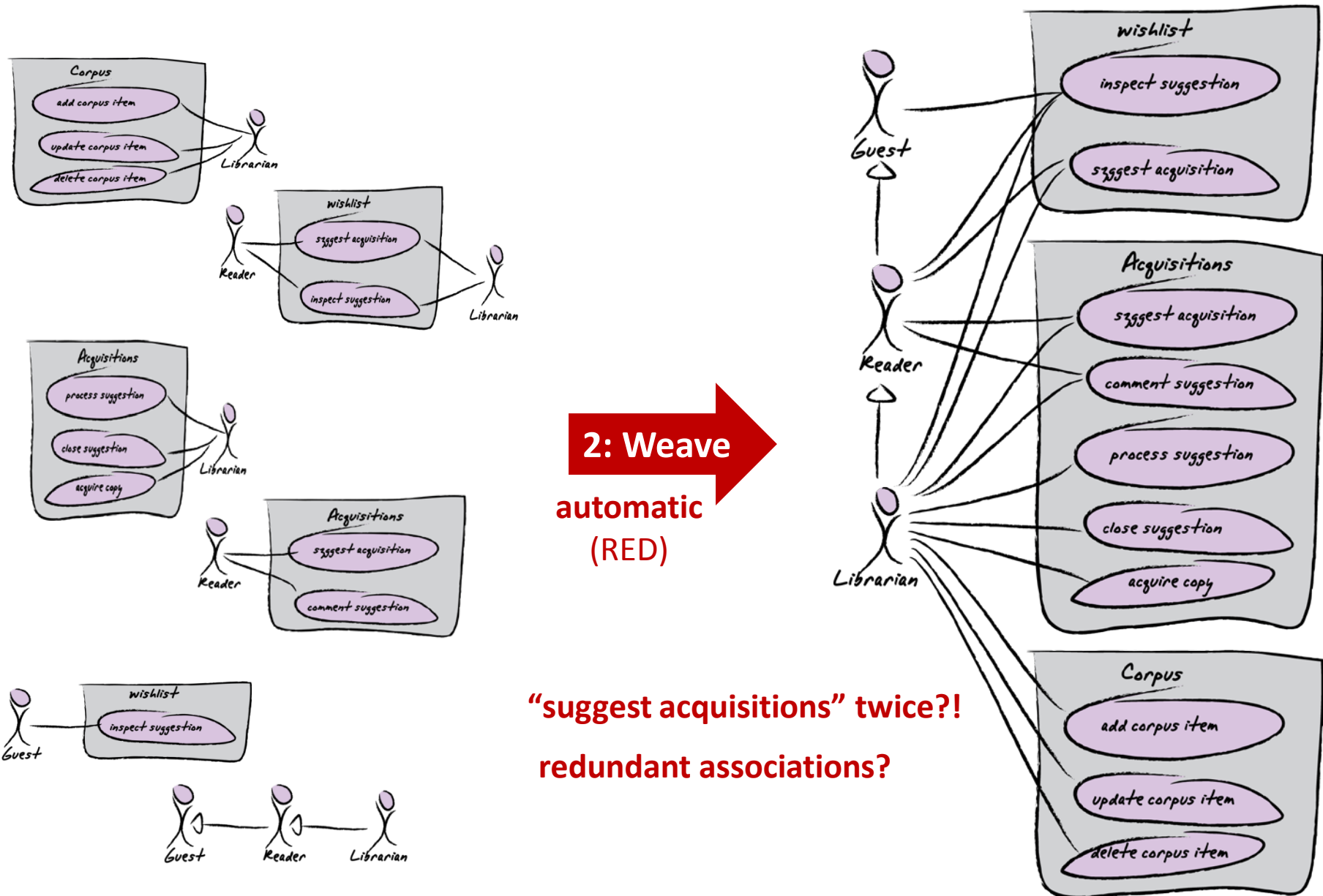
A librarian can do all a reader can do; a reader can do all a guest reader can do.

**1: Harvest**  
manual  
(domain expert)



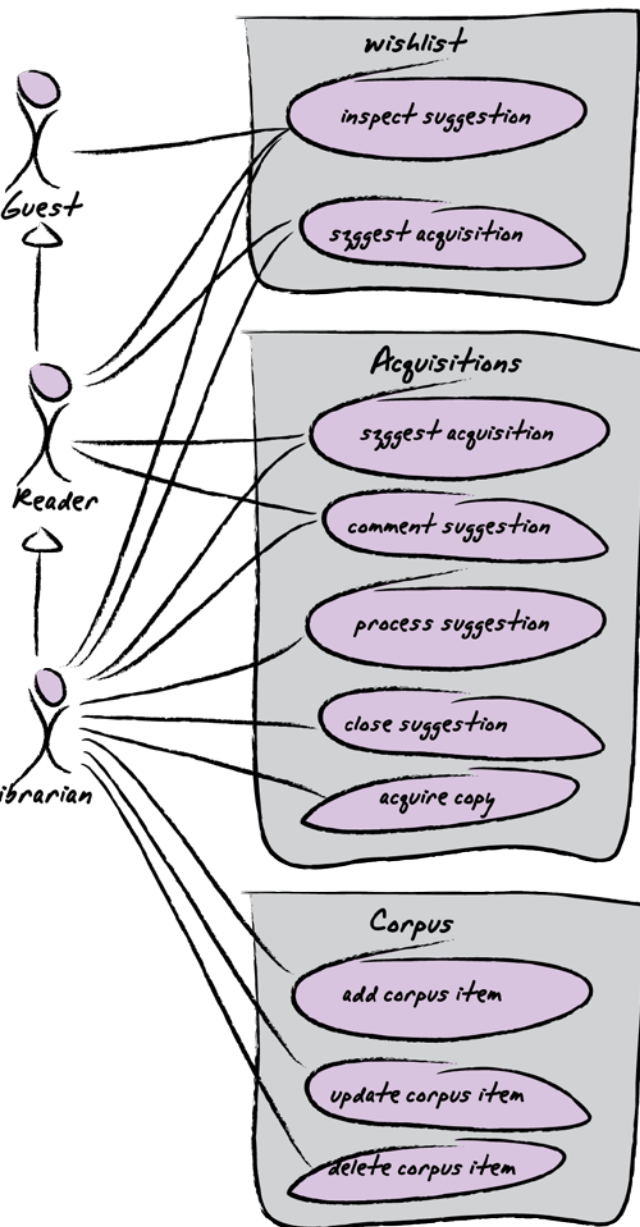
**split & elaborate MLC4?**  
**what about “deactivate”?**  
**Terminology?**

# Step 2: Weave



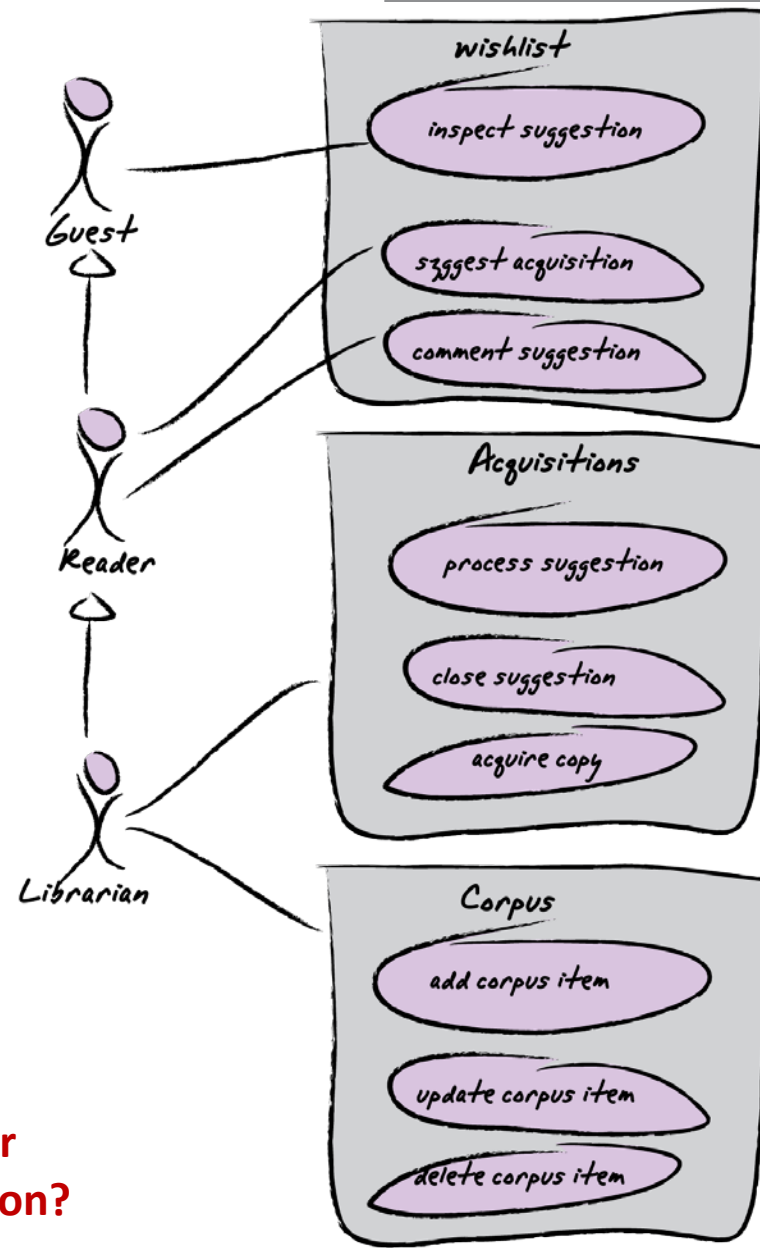


# Step 3: Consolidate



## 3: Consolidate

manual  
(Sw. Engineer)



VMTL-support for  
anti-pattern detection?



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

---

## **Chapter 11.2**

# **Harvesting Model Fragments from Requirements**

---

DTU course 02264

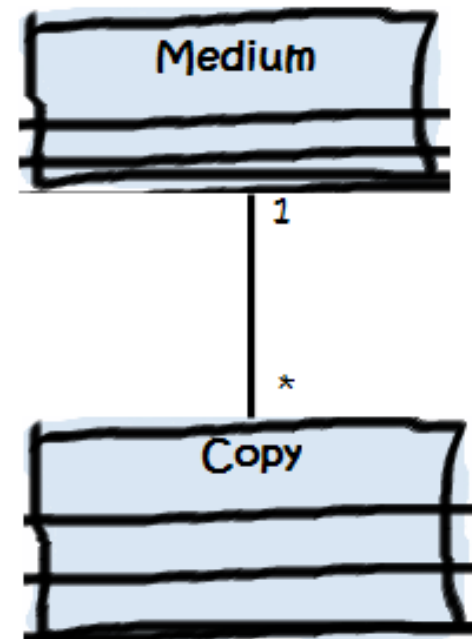
# Harvesting Example 1:

## Successful extraction

- Consider the example on the right. Translating it gives rise to the small class diagram at the bottom.
- The detail description suggests more model elements than are apparent in the title alone, e.g., methods
  - `search()`
  - `is_available_for_lending()`.
- Also, additions like return types for the methods may arise. Here, `search()` will have to return `Medium` rather than `Copy`.
- Details such as multiplicities may also arise from requirements.
- This step needs so little knowledge of UML that it can be done by domain experts with little or no help.

**MLC11: The corpus may contain several copies to a medium.**

- The catalog shall provide readers with access to media rather than individual copies.
- Readers shall be able to find out, whether there is a copy of a given medium available for lending.



# Harvesting Example 2: Incomplete Fragment

- Extracting a fragment often results in a deficient model at first
  - Classic requirements flaws like ambiguity and incompleteness often become very obvious in fragments.
  - Creating alternative fragments helps finding unresolved design decision lies.
- This may result in changes to the requirements, and appropriate updates in the fragment.

**Requirement MLC9c**  
Book suggestions may be inspected.

*who triggers this?*

*inspect suggestion*

**Requirement MLC9b**  
Guest readers may inspect suggestions.

*Guest*

*inspect suggestion*

*what system executes this?*

**Requirement MLC9a**  
Guest readers may inspect suggestions in the wishlist system.

*Guest*

*wishlist*

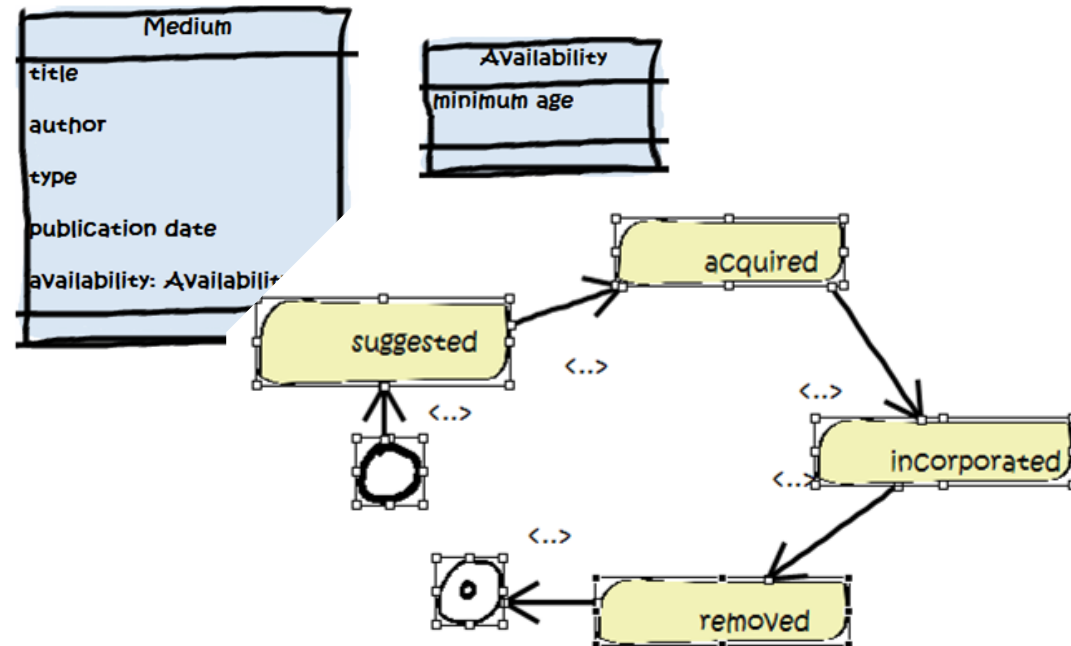
*inspect suggestion*

# Harvesting Example 3: Large Fragment

- Some requirements result in more than one or overly large fragments.
  - As a rule of thumb, one requirement should not yield more than ten model elements.
- This is ok if we think the domain is well enough understood and the receiver of the model can handle it.
- Sometimes, however, it indicates an insufficiently refined requirement that needs another iteration.

**MLC1: Media follow a defined lifecycle from suggested, via acquired, incorporated, to removed.**

- The availability of incorporated media may be restricted, e.g. in terms of age restrictions, access restrictions for valuable copies and highly demanded media and so on.
- The status of incorporated media is regularly updated to reflect damages and lending status.
- Creating new media requires information such as title, author, type, publication date, etc.



# Harvesting Example 4:

## No Fragment

- **The translation process might not yield any fragments at all.**
- **There are several possible reasons for such an event.**
  - A. The translation is possible in principle, but the a domain expert may not know UML well enough.
  - B. No translation is possible because there is a language impedance mismatch between the prose used and UML.
  - C. No translation is possible because the intent of the requirement is simply not expressible in UML.
- **Many high-level requirements (e.g., crosscutting requirements, qualities) provoke a language level impedance mismatch.**
  - “The system should be highly usable”.
- **Such requirements need to be further refined until they are concrete enough to amount to a (set of) specific element(s) in the design.**
- **The UML does not (directly) speak about user interface elements, so requirements involving that topic may be difficult to express.**
  - “There should be a tab sequence for all input fields”.
- **Other areas the UML is not well suited for are resource constraints, physical entities like location, and processes & threads.**

## 2) Multiple resulting fragments

- **Once elicited and defined, the requirements need to be elaborated to the degree needed to implement the system.**
  - Expect a major increase in the number of requirements, and thus, the administrative problems associated to them.
  - Robert Glass says: *"Explicit requirements explode by a factor of 50 or more into implicit (design) requirements as a software solution proceeds."*
- **If translating a requirement yields several fragments, they must have different types.**
  - The same type would just increase the existing diagram.
- **Sometimes, it is unavoidable that a requirement results in several model fragments, e.g., to cover**



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

---

## **Chapter 11.3**

### **Weaving Model Fragments**

---

DTU course 02264



# Weaving

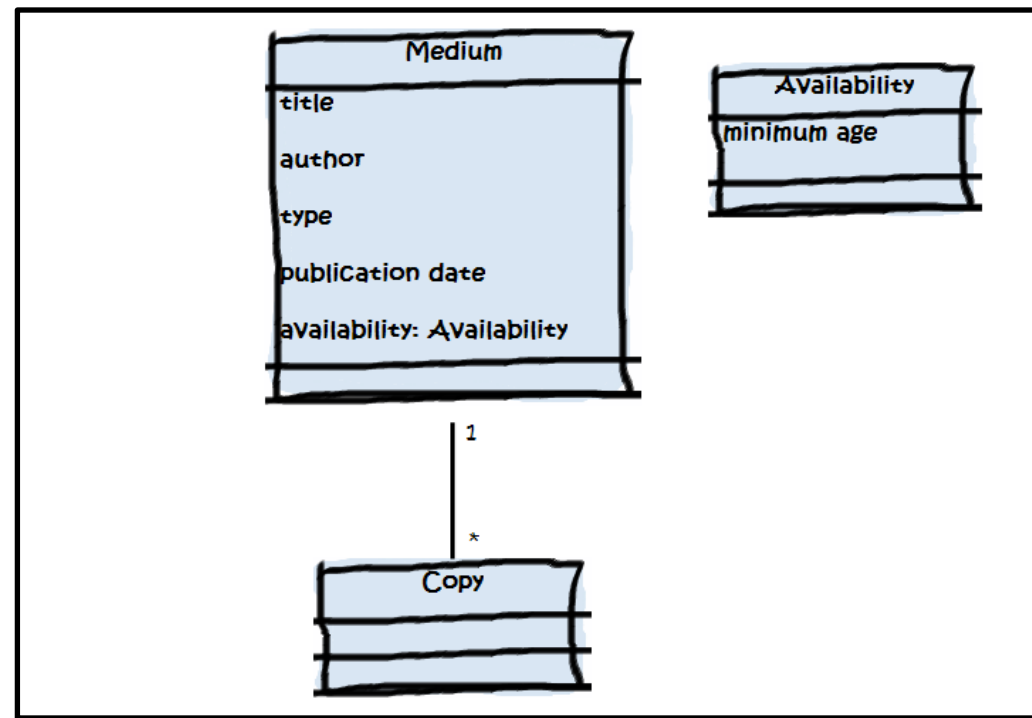
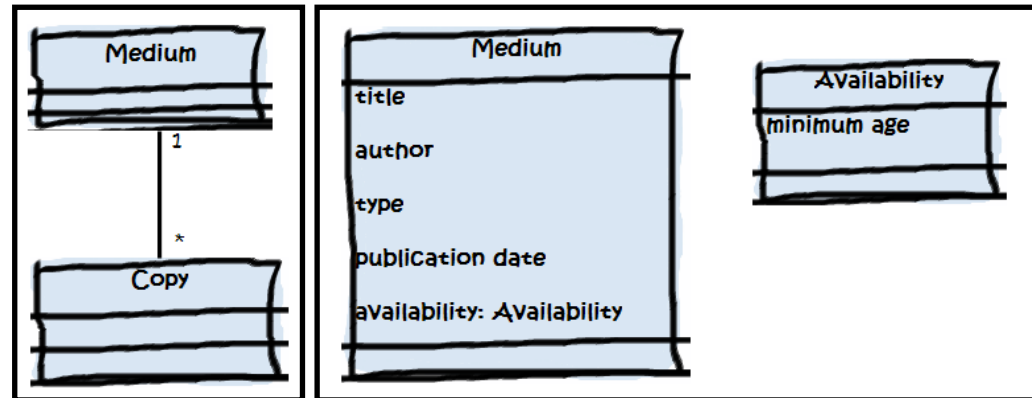
- **Once requirements are translated into model fragment, we need to combine (“weave”) them into one coherent model.**
- **Weaving two fragments will look for elements with the same type and matching names from the two fragments, and merge them.**
  - If elements are intended to represent the same thing, this is just what the user wants.
  - If not, then the result will be nonsensical to the user, and should be fairly easy to discover.
  - Tightening the name comparison from “matching” to “identical” name will exhibit typos, because what ought to be the same element ends up as two elements.
- **However, problems may occur in the weaving process.**
  - Fragments may overlap or contradict each other, or
  - their combination leads to poor or wrong models, or
  - there may be gaps between the fragments.

# Weaving Example 1:

## Simple overlap

- **Consider a simple example first.**

- Here, both fragments contain a class “Medium”.
- Merging them results in a class that has both the association to Copy and the properties (title, ...).
- Everything else stays unchanged.

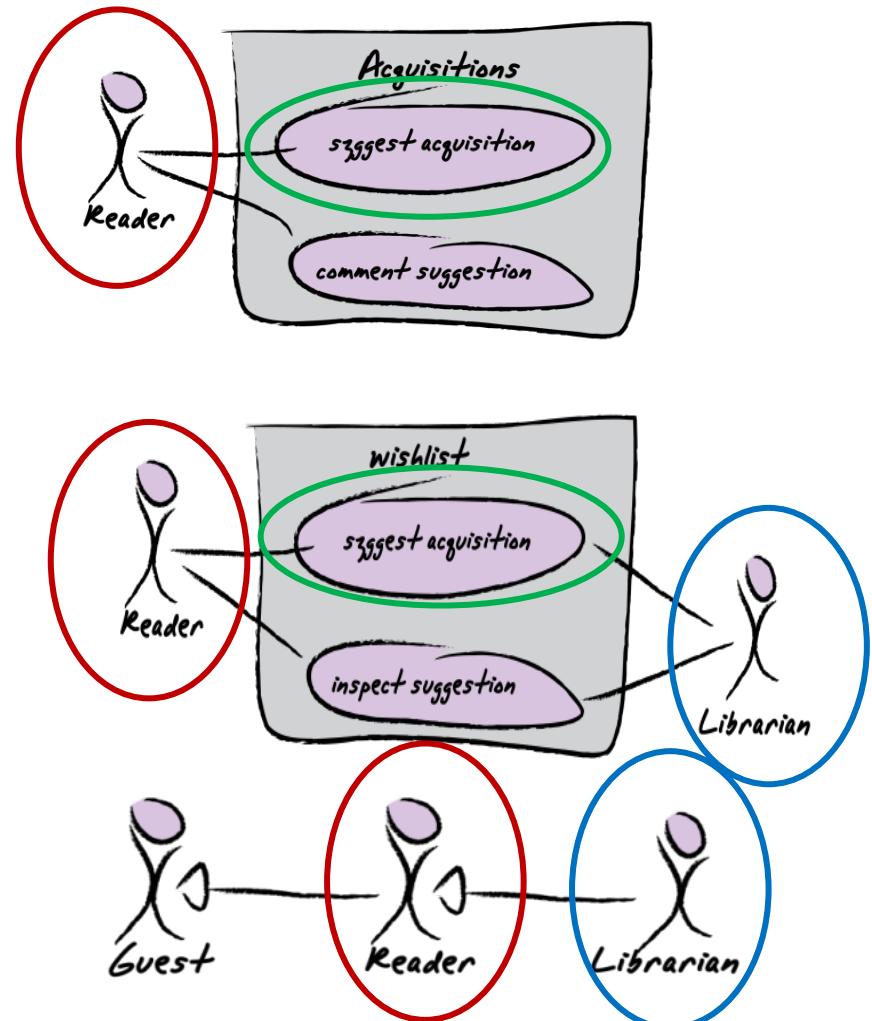


# Weaving Example 2:

## Multiple/consecutive overlaps

- Here is an excerpt from the LMS requirements specification, and how the features described may be captured as models.

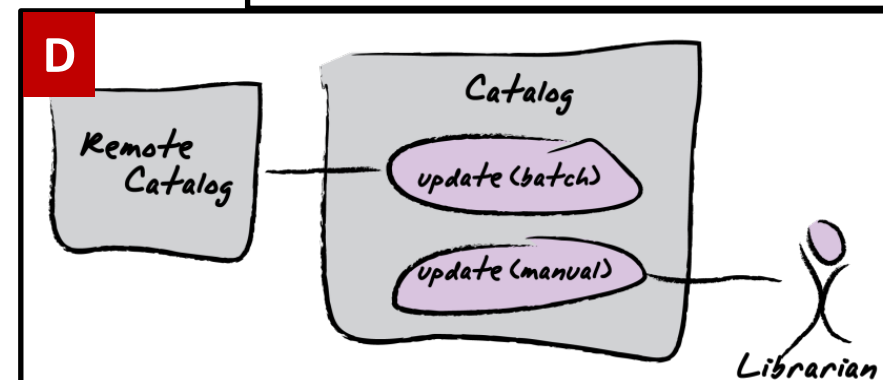
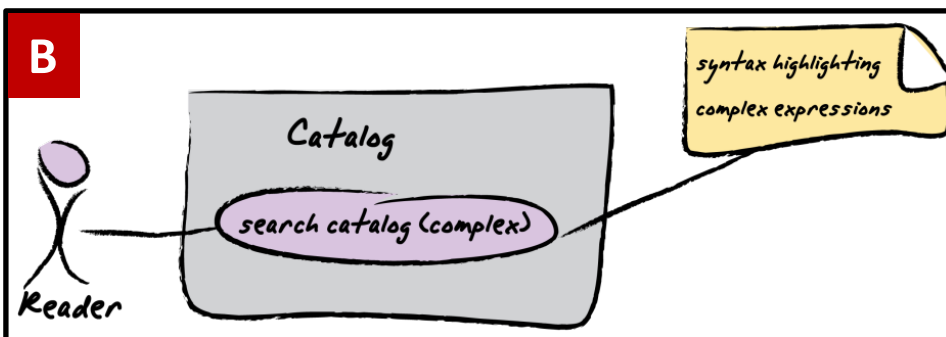
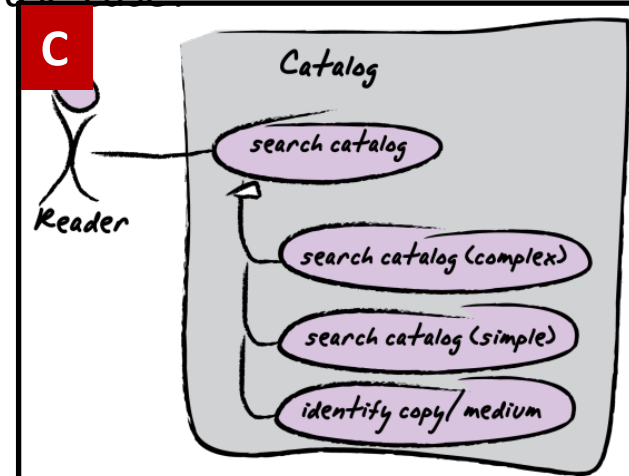
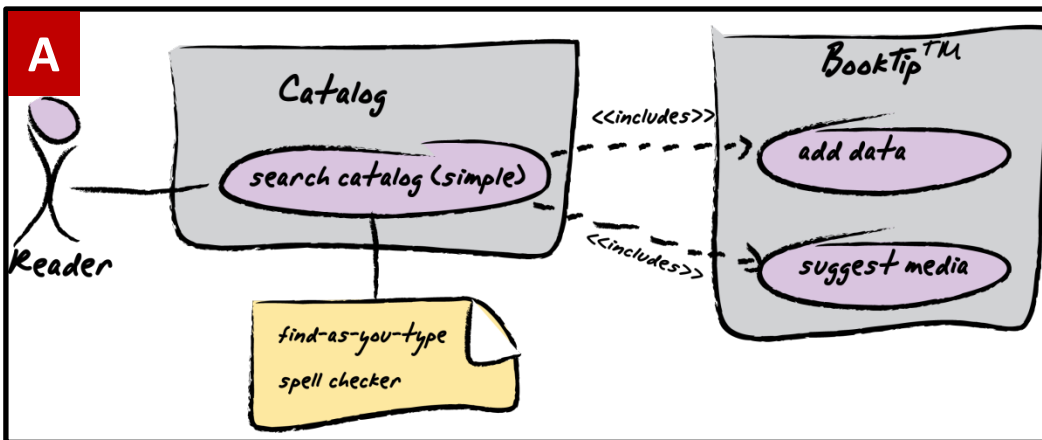
ID	Requirement
MLC4	Librarians and Readers may post and inspect media they think should be acquired by the library to a public "wish list" indicating the status of the wish and the originator.
MLC10	A librarian can do all a reader can do; a reader can do all a guest reader can do.



# Weaving Example 3:

## Semantic Implications

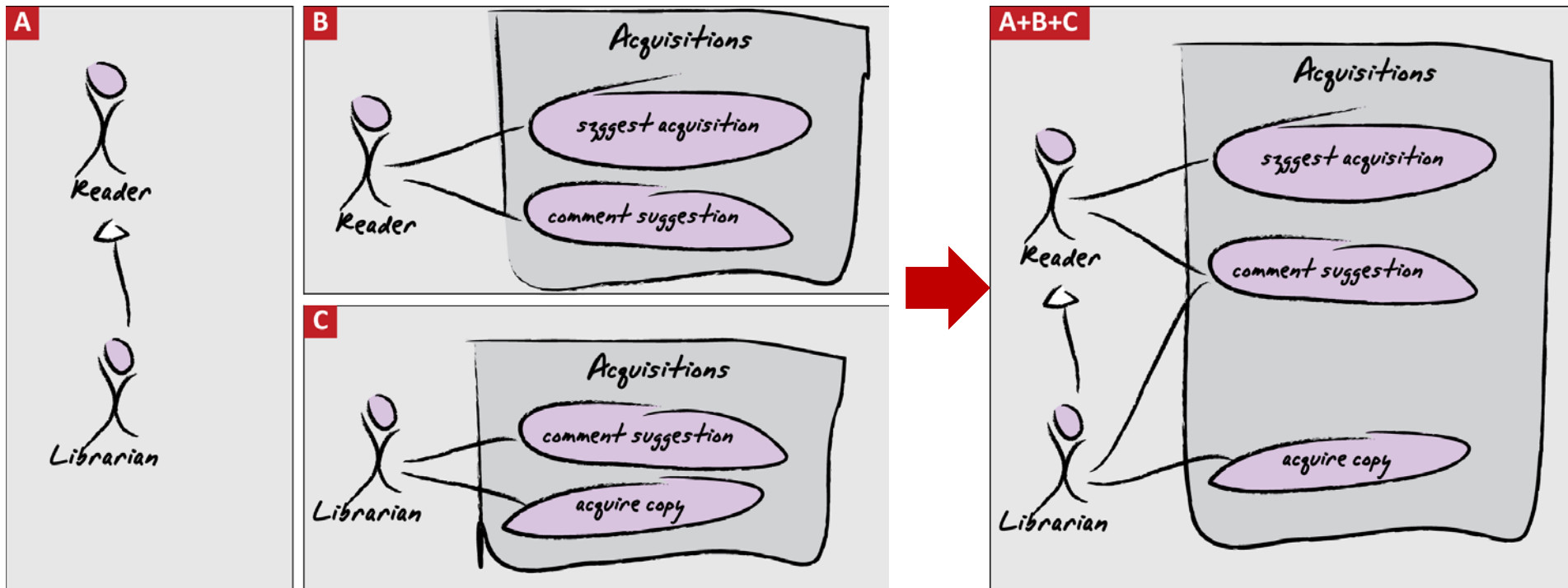
- Weaving these Use Case Fragments raises conceptual questions.
  - Should book tips also be given to complex searches? Then, the includes (A) should be going out from “search catalog” (C).
  - Are the main use cases of A and B really that different? If sometimes use cases are included (A) and sometimes not (B), shouldn't they be extensions instead?
  - Do the inclusions/extensions maybe belong to a sub-use case?



# Weaving Example 4:

## Weaving exhibits duplication or flaw

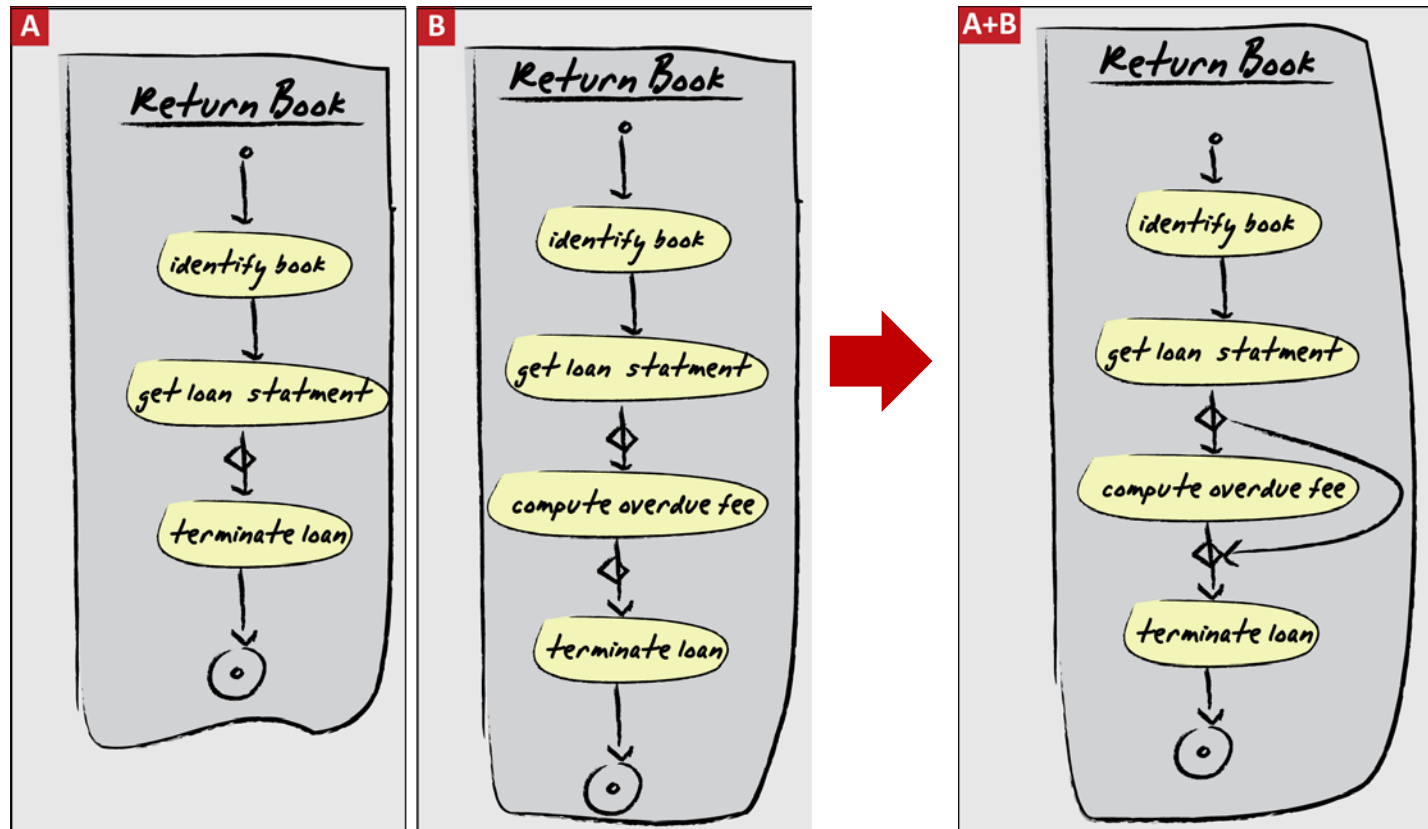
- The weaving may produce models that are considered flawed, e.g., by containing anti-patterns.
  - Pulling up attributes and associations
  - Avoiding circular inheritance
  - Avoiding implicit synchronization



# Weaving Example 5:

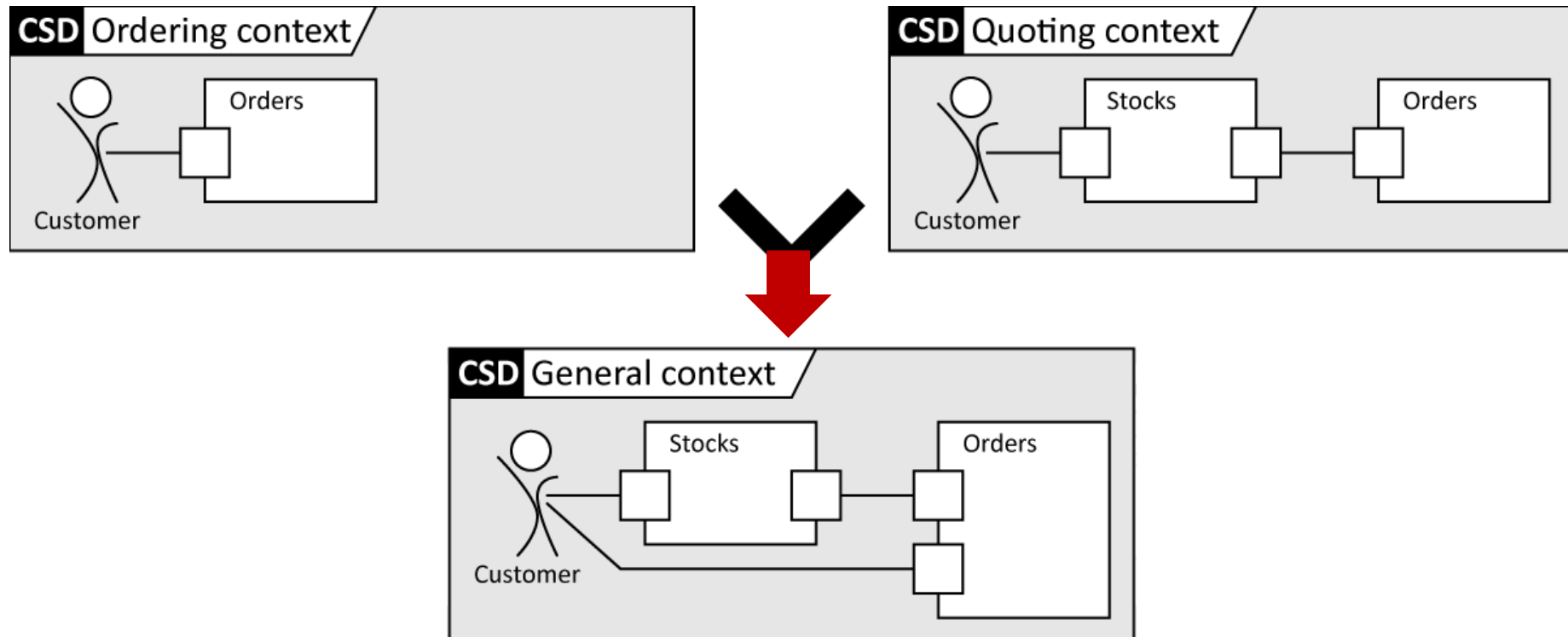
## Weaving yields unexpected results

- The weaving result may deviate from the modeler's expectation.
  - This is always an instructive starting point, and may lead to either changed requirements, fragments, or better understanding of UML.



# Folding Assemblies (Composite Structures)

- Weaving can be applied to all diagram types, including goal and context diagrams.





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

---

## **Chapter 11.4**

# **Tracing between Requirements and Models**

---

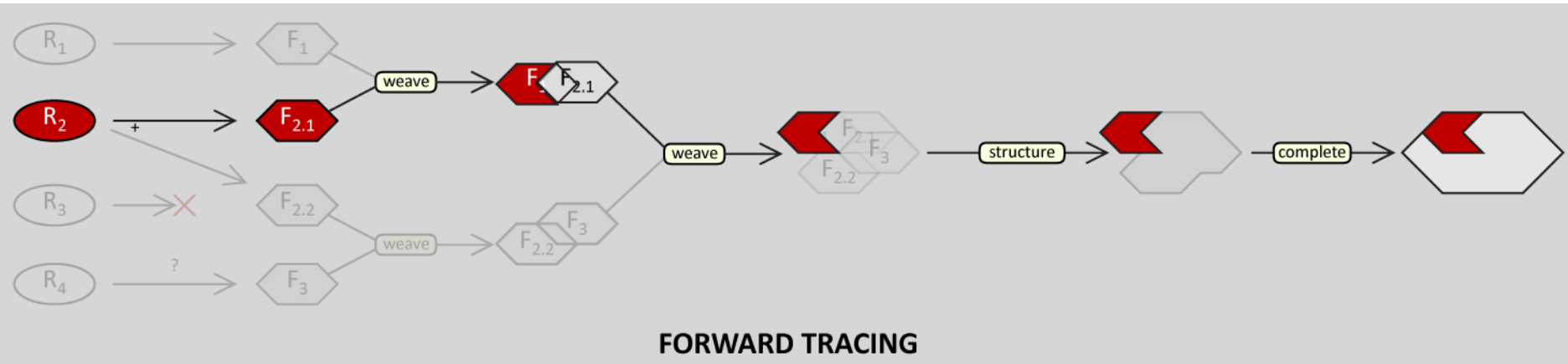
DTU course 02264



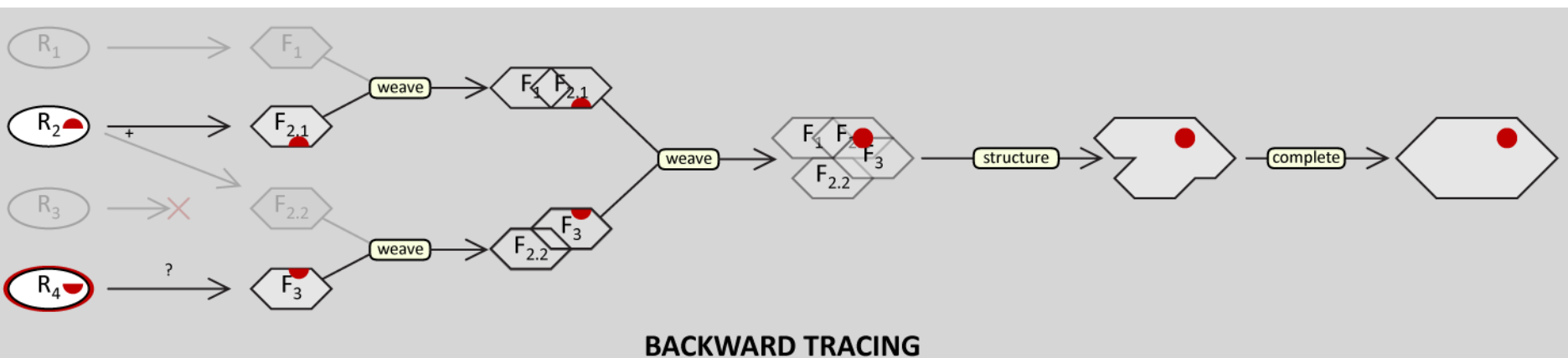
- **The trace connections established by the folding process can be used to relate changes in the requirements and the analysis model, both ways.**
- **Typically, the analysis model is translated manually into a design level model, incorporating design decisions and technology choices.**
  - If both ends of this transformation are expressed in the same language (e.g., UML), it is feasible to establish trace links manually.
  - The similarity between typical design and implementation languages is sufficient to also allow simple linking from the design model to the implementation.
  - That way, a complete chain of trace links from requirements to code via analysis and design models can be established
- **This kind of linkage is required in high-assurance software, e.g., aerospace applications (cf. DO-178A/B/C).**

# Forward/Backward Tracing

- **Question: Given a requirement, how is it realized in the model?**
  - If code is generated from the model: how is the requirement implemented?



- **Question: Given a model element, which requirement justifies it?**
  - If code is generated from the model: why is a given line of code where it is?



# Tracing in RED

- **During weaving, RED embeds trace links into diagrams that are woven together.**
  - On the tab “Management & Tracing”, the two input diagrams are linked in the set of “Sources”, and can be navigated from there.
  - These sources are also reflected in the title of the diagram.
  - The details of the merge procedure are documented in the “Merge Log” field on the tab “Diagram Editor”.
- **A visualization of the tracing is currently missing, as are more advanced facilities to query the trace relationships.**



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

---

## **Chapter 11.5:**

# **Requirements Tools**

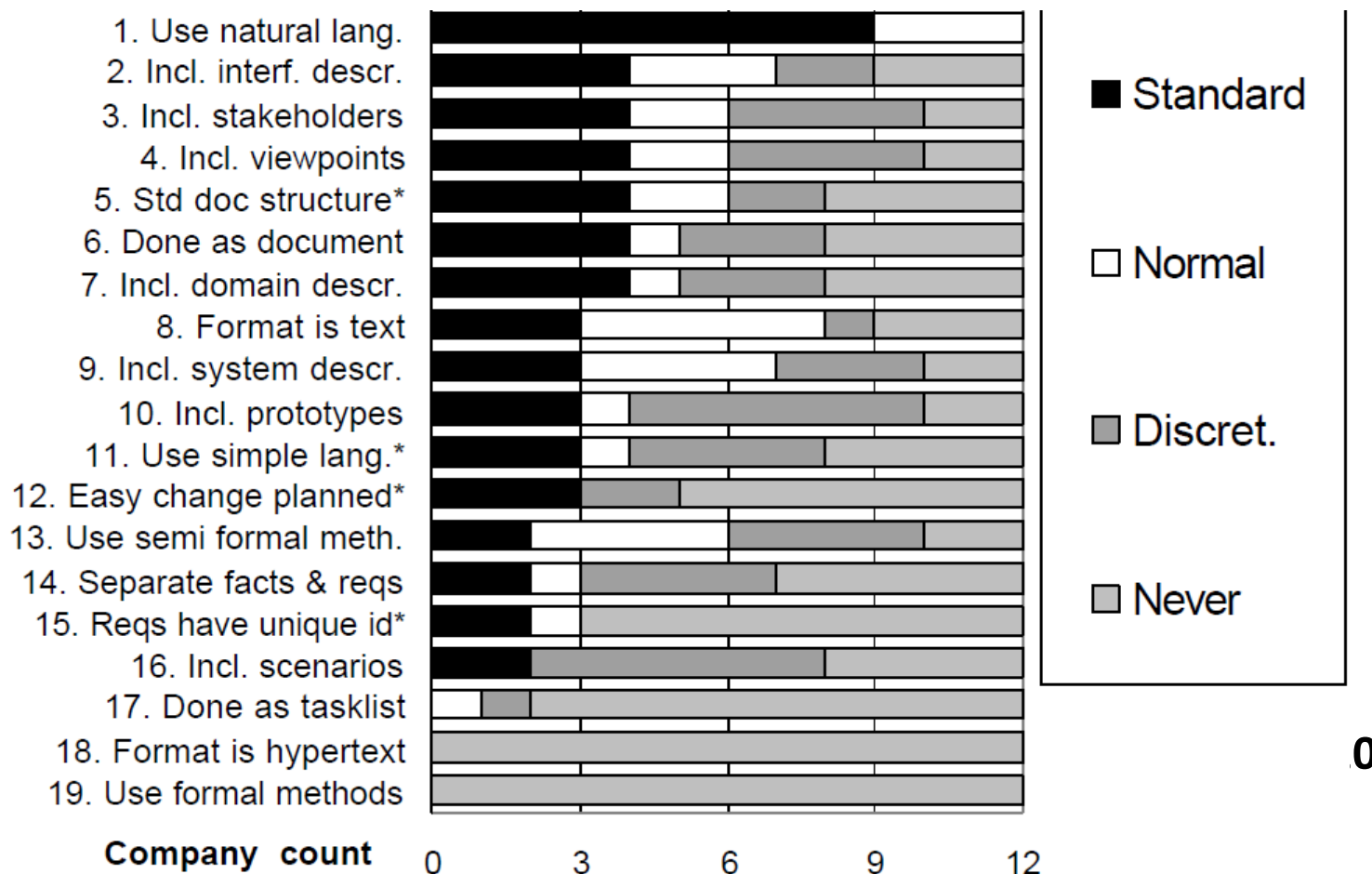
---

DTU course 02264

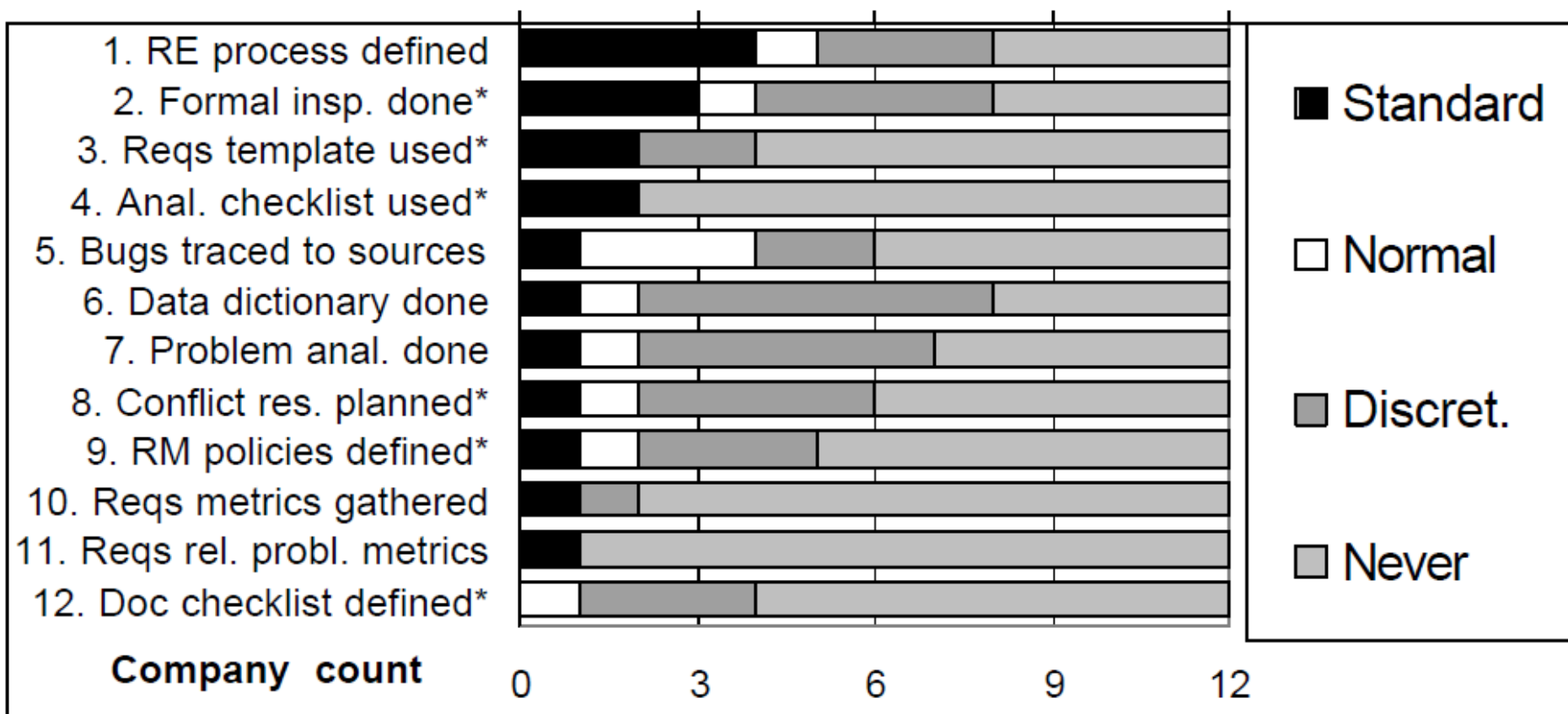
# State of RE in Practice is Poor

- “There is a lot of information available on solid RE practices but anecdotal evidence still indicates poor practices.”

U. Nikula, J. Sajaniemi, H. Kälviäinen: A State-of-the-Practice Survey on Requirements Engineering in Small- and Medium-Sized Enterprises. Telecom Business Research Center Lappeenranta, Research Report 1, 2000



# RE Process Maturity in Practice is Poor

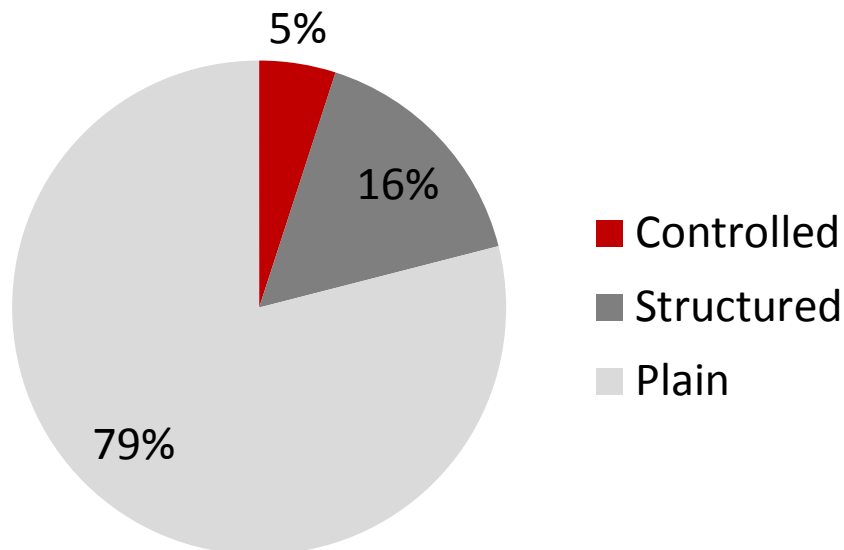


\*REAIMS top 10

# Prose for Requirements Engineering

- Alternatives to Natural Languages (NL) exist.
- Various case studies have demonstrated that they can largely replace NL.

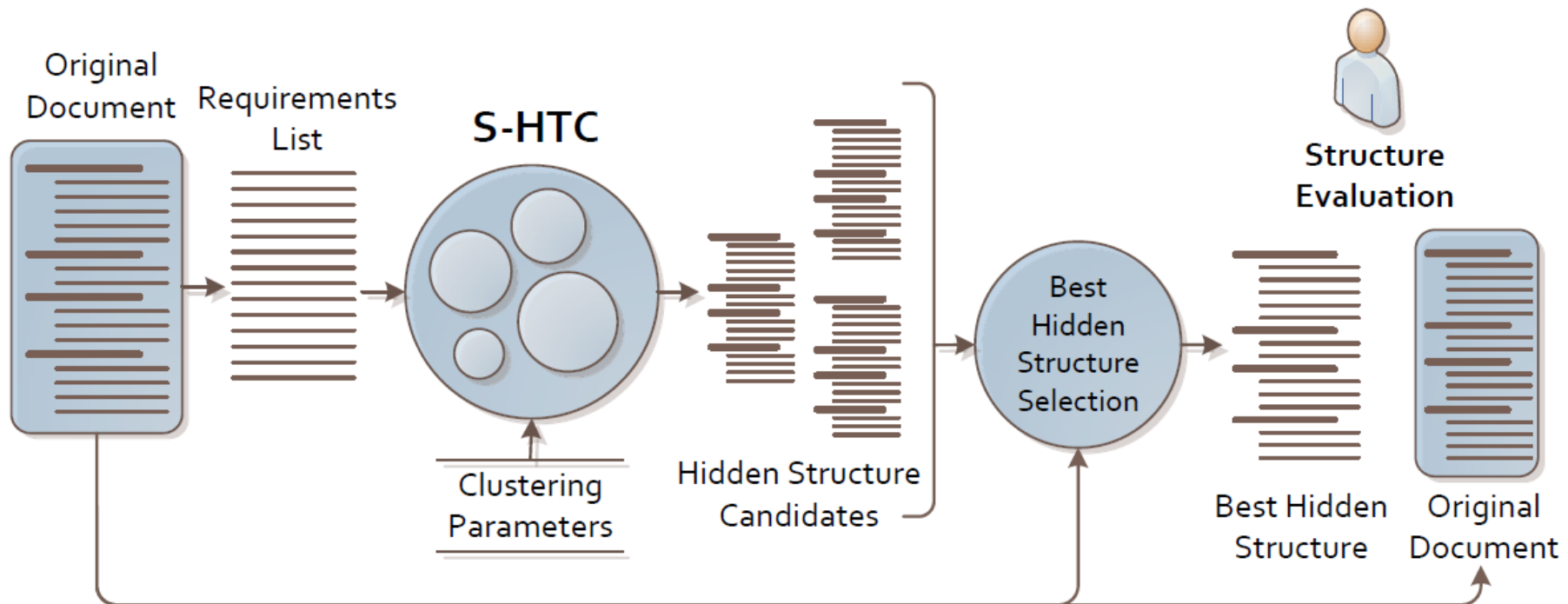
Language Type Used



- Natural Language Processing (NLP) and Information Retrieval (IR) technology can do amazing things:
  - generating sequence diagrams from natural language use case descriptions;
  - generating class diagrams from NL requirements specifications.
- However, if the performance is less than perfect, using tools is often worse than not using them.

# NL-Analysis of document outlines

- There are algorithms to identify parts of natural language requirements documents with poor structuring, sections that ought to be re-arranged, and requirements that are placed in conceptually unconnected sections.



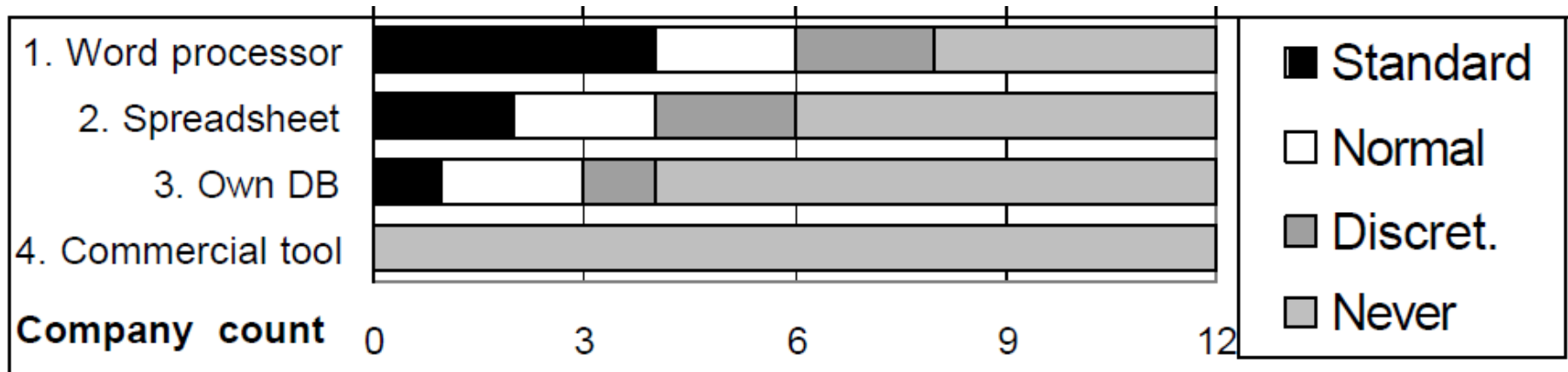


Do you use any tool supporting requirements analysis and top-level design?

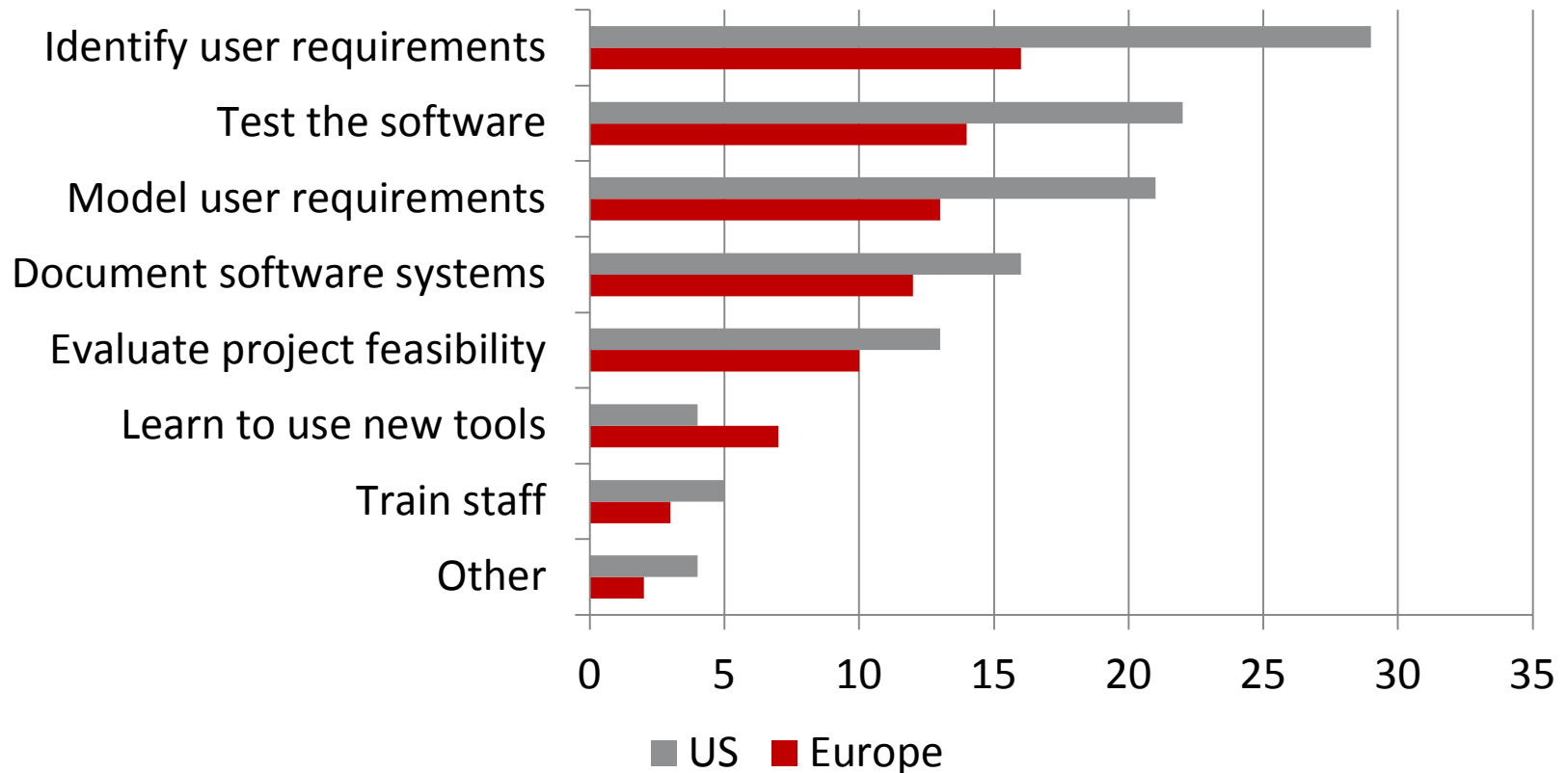
How many employees and consultants are there in your company?

1–5      6–20      21–50      51–100      More than 100

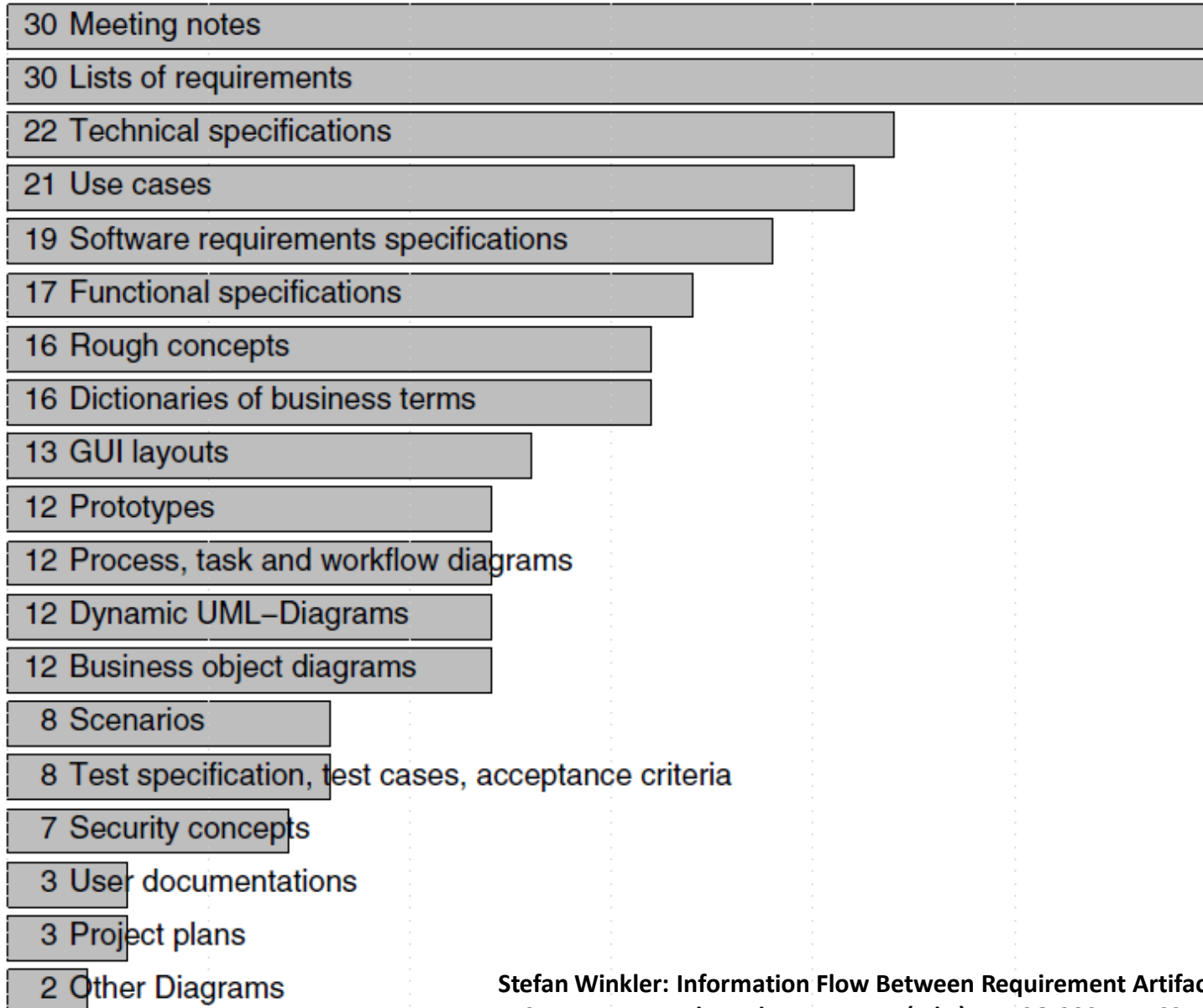
Yes	16%	18%	33%	33%	51%
No	84%	82%	67%	67%	49%



- *“I hate to be a cynic, but there are hardly any worthwhile tools. The overhead in learning to use them is too great for the payoff.”*

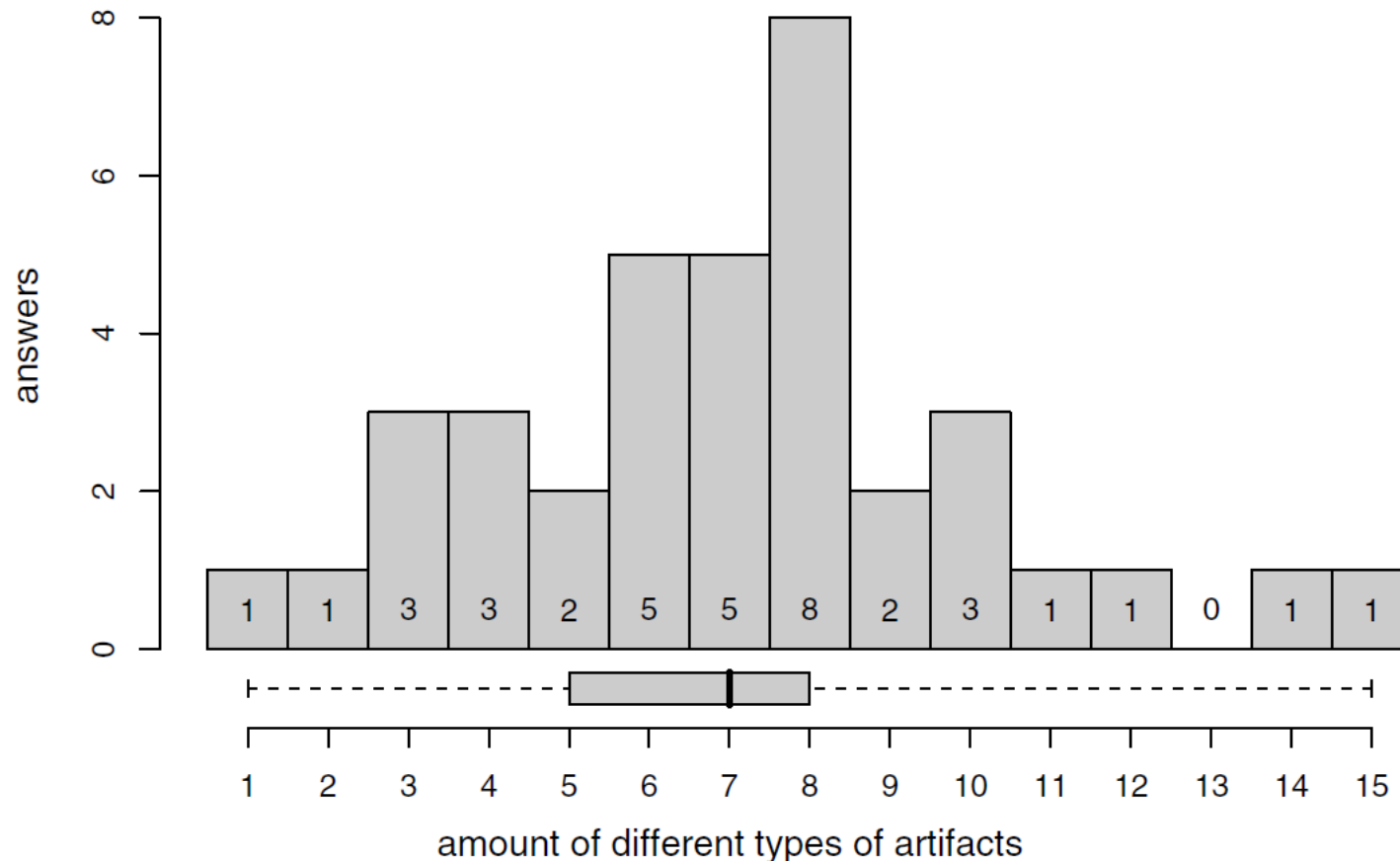


# Diverse Requirements Stores in use

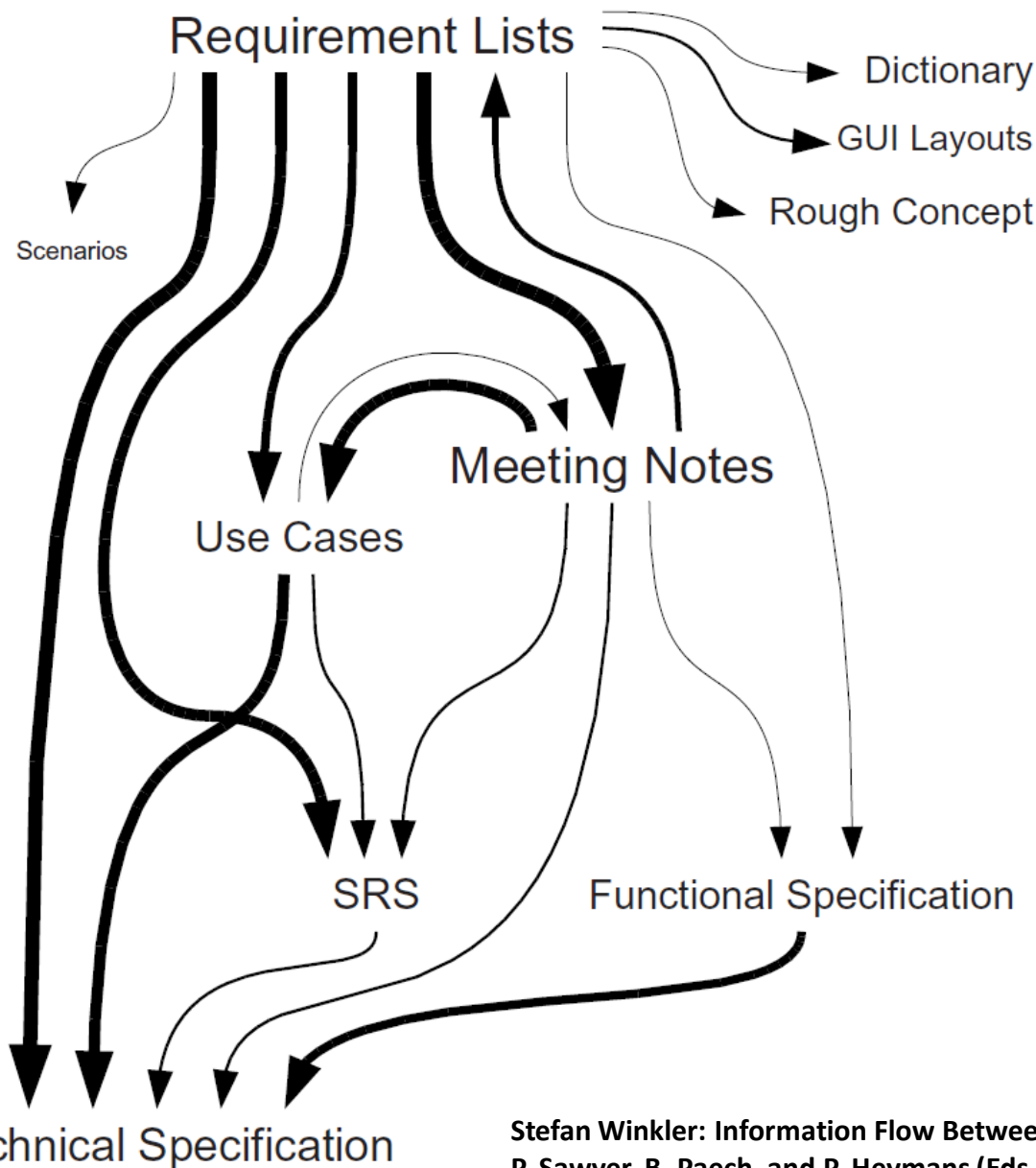


# Concurrent Requirements Stores

- In typical industrial settings, five to eight different media are used to store requirements.



# Requirements Flow





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

---

## **Chapter 11.6:**

# **Tailoring the Requirements Toolbox**

---

DTU course 02264

# The necessity of Tailoring

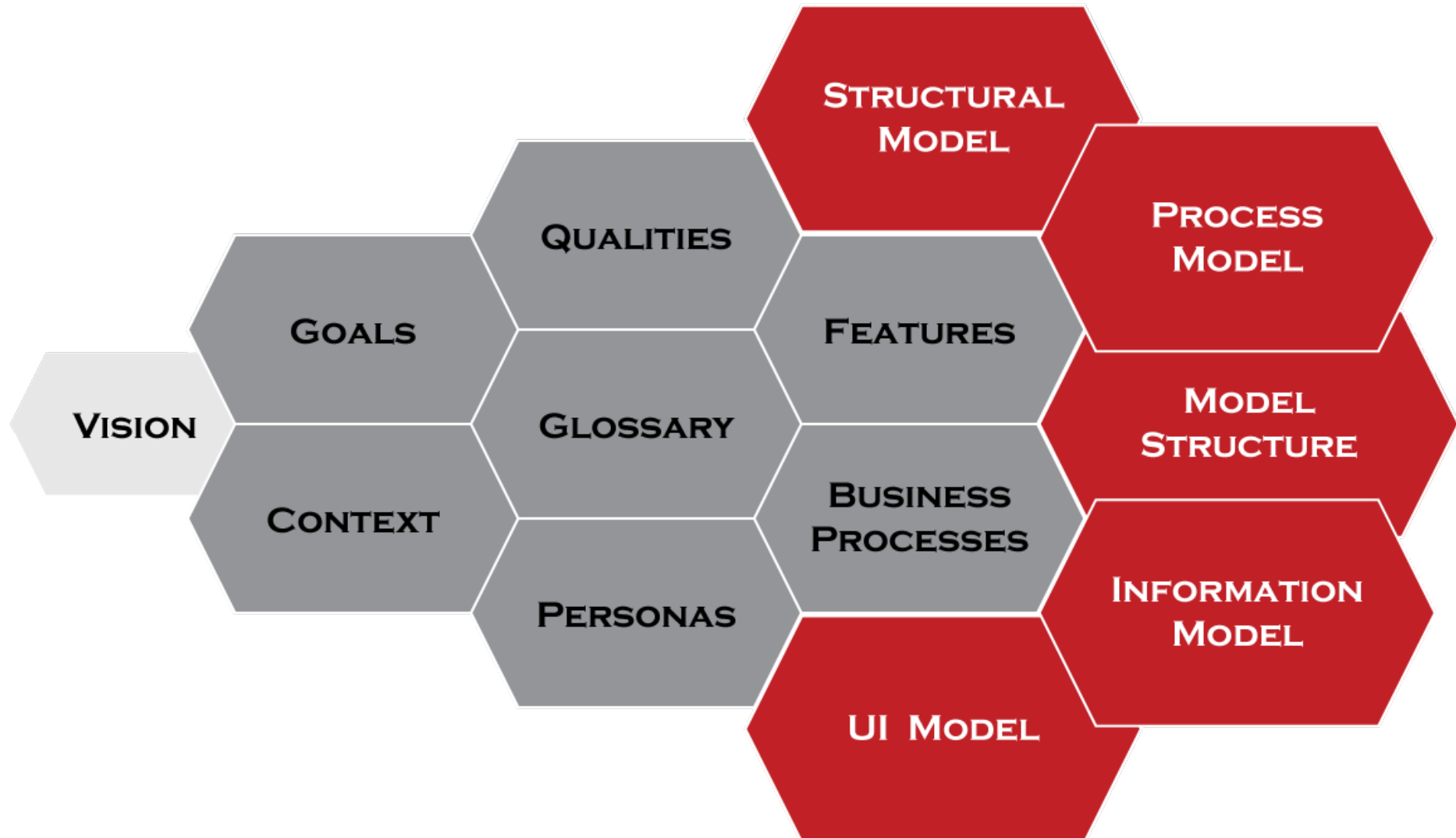
- **There are many different techniques for specifying and managing requirements – which ones should we use?**
  - One size does not fit all.
  - Each method has their specific profile of strengths and weaknesses.
  - For many techniques, we do not have adequate evidence to assert usage conditions: common sense and experience will have to do.
- **Using an inappropriate technique might be worse than using no technique at all for several reasons.**
  - Disagreement about the approach can be distracting (“method wars”) and disrupt the team’s group dynamics.
  - Using methods typically comes with increased effort and/or cost.
  - The techniques may lead to properties of the system document that may not just be wasteful, but actually negative.
- **Imagine a scenario where using User Stories is demanded, while some team members prefer Use Cases.**
  - Convincing and training them requires effort and time.
  - Focusing on features may lead to neglecting qualities.

# The Toolbox

- It is recommended practice to select and fix a set of techniques for a project, based on an initial estimate of the project's needs.
- We call this the „project toolbox“, and the process „tailoring“.
- During tailoring, a (brief) description of the „toolbox“ should be created.
  - The toolbox should be a project specific selection of existing proven practices, possibly with one or two additions of new “experimental” methods.
  - A justification of the decision must be provided.
  - The toolbox must be easily available, e.g., as a printed poster on the wall next to the coffee machine.
  - One team member should be appointed as responsible for maintaining the toolbox (the “tool smith”).
  - After the project, a post-mortem should be conducted to, among other things, assess the toolbox and the tailoring process.



# RE Techniques in Synopsis



# Methods ~ Qualities (1/2)

- Using the ISBSG data set on project outcomes, we can see some interesting correlations:
  - Some methods/techniques have positive influence on many quality metrics.
  - Other actions seem to have little to no practical impact.
  - Some quality metrics are influenced positively by more or less any action.

Variable	MSO	MBR	QF	QD	EU	SDS	SPS
CASE Tool Used		$\rho U \phi C$	$\rho U \phi C$		$\phi C$	$\phi Cu$	$\phi C$
Used Methodology	$\rho U \phi Cu$		$\rho \phi C$			$\rho U$	$\rho U \phi C$
Upper CASE Used	U					$\rho$	
Lower CASE (with code gen)	$\rho U \phi Cu$	$\rho U \phi Cu$	$\rho U \phi Cu$	$\rho U \phi Cu$	$\rho U \phi Cu$	$\rho U \phi Cu$	$\rho U \phi Cu$
Project user involvement	$\rho U \phi Cu$		$\rho$	$\rho$		$\rho$	
Portability requirements			$\phi Cu$		$\phi C$	$\phi Cu$	
Metrics Program	$\rho U u$	$\rho U \phi Cu$	$\rho U \phi Cu$	$\rho U \phi Cu$	$\rho U \phi Cu$	$\phi Cu$	$\rho u$
User satisfaction survey	$\rho U \phi Cu$		$\rho \phi C$		$\rho U \phi C$	$\rho U \phi C$	$\rho U \phi C$
Training given	$\rho H \phi CV$	$\rho H \phi CV u$	$\rho H \phi CV u$	$\rho H \phi CV u$	$\rho H \phi CV u$	$\phi CV$	$\rho H$
Process improvement pgm						$\rho U \phi Cu$	$\rho U \phi Cu$

MSO: meet stated objectives

EU: Ease of use

MBR: meet bus. Reqs.

SDS: speed of def. solution

QF: Quality of functionality

SPS: speed of providing sol.

# Methods ~ Qualities (2/2)

## Development Technique

Business area modeling

Data modelling

OO Analysis

OO Design

## Project Planning

Budget

Schedule

## Specification

Functional specification

Logical data ER model

System concept document

Data flow model

Variable	MSO	MBR	QF	QD	EU	SDS	SPS
Project objective: all functionality		$\rho H \phi C V u$			H	$\phi C V u$	$\rho u$
Project objective: min. defects			H	$\underline{H}$		$\rho \underline{H} u$	$\rho \underline{H} u$
Project objective: min. cost		$\phi C V u$		$\rho$		$\rho H \phi C V u$	$\rho$
Project objective: shortest time			u	$\underline{H}$			
Dev. tech.: Business area modeling			$\rho U \phi C u$	$\rho U \phi C u$	$\rho U \phi C u$	$\rho U \phi C u$	$\rho U \phi C u$
Dev. tech.: Data modelling		$\rho$	$\phi C u$	$\rho U \phi C$	$\phi C$		
Dev. tech.: Event modelling	$\rho U \phi C u$		$\phi C$			$\rho U$	
Dev. tech.: Multifunct. teams				$\rho \phi C$			
Dev. tech.: OO analysis			$\phi C u$	$\rho \phi C$	$\rho \phi C u$	$\phi C u$	$\phi C u$
Dev. tech.: OO design	$\rho U \phi C$		$\phi C u$	$\phi C$	$\phi C$	$\phi C u$	
Dev. tech.: OO			$\phi C u$		$\phi C$	$\phi C u$	$\phi C$
Dev. tech.: Process modelling			$\phi C$		$\phi C$	$\phi C u$	$\rho$
Dev. tech.: Prototyping	$\rho$						
Dev. tech.: Timeboxing	$\rho$	$\rho$	$\rho$				
Dev. tech.: Waterfall				$\rho \phi C$		$\rho U$	$\rho U$
Plan docs: Budget		$\rho \phi C u$	$\rho \phi C u$	$\rho U \phi C$	$\rho \phi C$	$\phi C u$	$\rho \phi C$
Plan docs: Business case		$\rho \phi C$				$\phi C u$	$\phi C$
Plan docs: Feasibility study						$\rho U \phi C$	$\rho$
Plan docs: Project schedule			$\phi C u$	$\rho U \phi C u$	$\rho U \phi C$		$\phi C$
Plan docs: Proposal/tender	$\phi C u$		$\rho \phi C u$	$\rho U \phi C u$		$\rho$	
Plan docs: Quality plan						$\rho U \phi C u$	$\rho U \phi C u$
Plan docs: Resource plan				$\rho U$			
Plan docs: Risk analysis						$\rho U \phi C u$	$\rho U \phi C u$
Plan docs: Software dev. plan				$\rho U \phi C u$		$\phi C$	
Spec. docs: None			$\rho \phi C$				$\rho$
Spec. docs: Functional spec.	$\phi C u$		$\phi C u$	$\rho \phi C u$		$\rho U \phi C u$	$\rho U \phi C u$
Spec. docs: Graph. look & feel						$\rho U \phi C u$	$\rho U$
Spec. docs: Log. data ER model		$\rho$	$\rho u$	$\rho U \phi C u$	$\rho U$	$\rho U \phi C u$	$\rho U \phi C u$
Spec. docs: Requirements spec.						$\rho U \phi C$	$\rho U \phi C$
Spec. docs: System concept doc.				$\rho U \phi C u$	$\phi C$	$\rho U \phi C u$	$\rho U \phi C u$
Spec. docs: Use case model						$\rho U \phi C$	$\rho$
Spec. docs: User interface prototype		$\phi C$	$\rho$				
Spec. docs: Ext. syst. interface spec.							$\rho$
Spec. docs: User manual				$\rho \phi C$	$\rho U$	$\rho U \phi C u$	$\rho U$
Spec. docs: Data flow model			$\phi C u$	$\rho U \phi C u$		$\rho U \phi C u$	$\rho U \phi C u$
Spec. tech. Activity diagram			$\rho \phi C u$	$\rho \phi C u$	$\rho$		
Spec. tech. JAD		$\rho \phi C u$	$\rho$	$\rho$	$\rho U \phi C u$	$\rho U \phi C u$	$\rho \phi C u$
Spec. tech. Timeboxing							$\rho$
Proportion of effort on plan				$\rho$			
Proportion of effort on spec.				$\rho$			
Activity planning			$\phi C$			$\rho U \phi C u$	$\rho U \phi C$
Activity specification	$\phi C$					$\rho U$	

**MSO:** meet stated objectives

**MBR:** meet bus. reqs.

**QF:** Quality of functionality

**EU:** Ease of use

**SDS:** speed of defining solution

**SPS:** speed of providing solution