

Software Requirements and Architecture (SENG404)

Matthias Galster

Lecture 5 – Requirements documentation

March 8, 2023

Schedule 2023

Lecture	Week	Date	Topic
1	1	February 22	Kick-off; Introduction
2	1	February 23	Instead of May 3; Requirements and requirements engineering processes
3	2	March 1	Requirements elicitation (part 1)
4	2	March 2	Instead of May 17; Requirements elicitation (part 1); Requirements elicitation (part 2)
5	3	March 8	Requirements elicitation (part 2); Requirements documentation
6	3	March 9	Backup (Matthias <i>might</i> be away)
7	4	March 15	
8	4	March 16	Backup
9	5	March 22	Assignment 1
10	6	March 29	
Term break			
11	7	April 26	
12	8	May 3	Matthias away
13	9	May 10	
14	10	May 17	Matthias away
15	11	May 24	Assignment 2: presentations + report
16	12	May 31	
		TBD	Final exam

Assignment 1

Student(s)	Topic
Saskia van der Peet	Use of design thinking in requirements engineering
April Clarke	Influence of social factors on requirements engineering
Jonathan Tomlinson + Danish Jahangir	Software requirements elicitation techniques

- Before you start
 - What would a practitioner want to know?
 - Source(s)?
 - Key message(s) – build blog post around it
 - Provide facts and evidence; if you share opinions mark them as such

Previous lecture

1. Elicitation techniques – observe
2. Elicitation techniques – build and play
3. Elicitation techniques – analyze
4. Requirements elicitation – summary

Reading for this session

- T. Mendes, M. de F. Farias, M. Mendonça, H. Frota Soares, M. Kalinowski, and R. Spínola. *Impacts of agile requirements documentation debt on software projects: a retrospective study*. In 31st Annual ACM Symposium on Applied Computing (SAC), 2016, pp. 1290-1295, doi.org/10.1145/2851613.2851761

Questions and lessons



Reading for next session

- R. Britto, E. Mendes, and J. Borstler. *An empirical investigation on effort estimation in agile global software development*. In 10th International Conference on Global Software Engineering (ICGSE), 2015, pp. 38-45, doi: 10.1109/ICGSE.2015.10

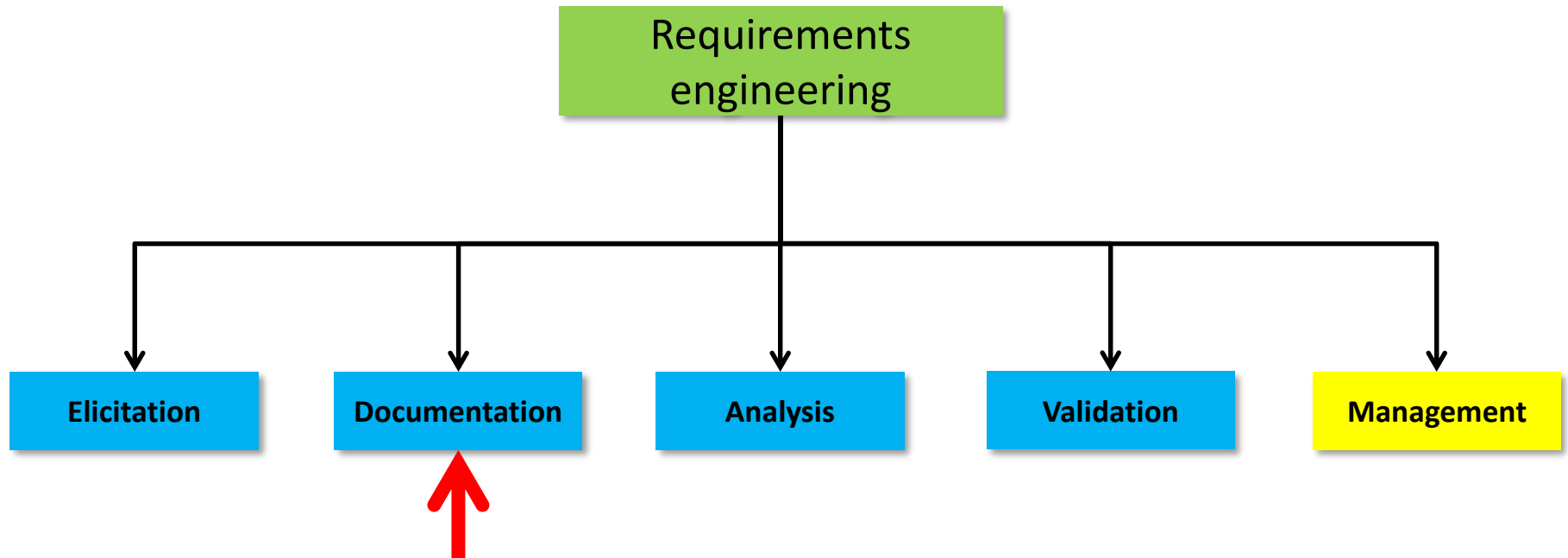
Agenda

1. Requirements documentation – overview
2. Natural language requirements
3. Formal languages
4. Requirements modeling
5. Characteristics of requirements

Agenda

1. Requirements documentation – overview
2. Natural language requirements
3. Formal languages
4. Requirements modeling
5. Characteristics of requirements

Generic RE activities

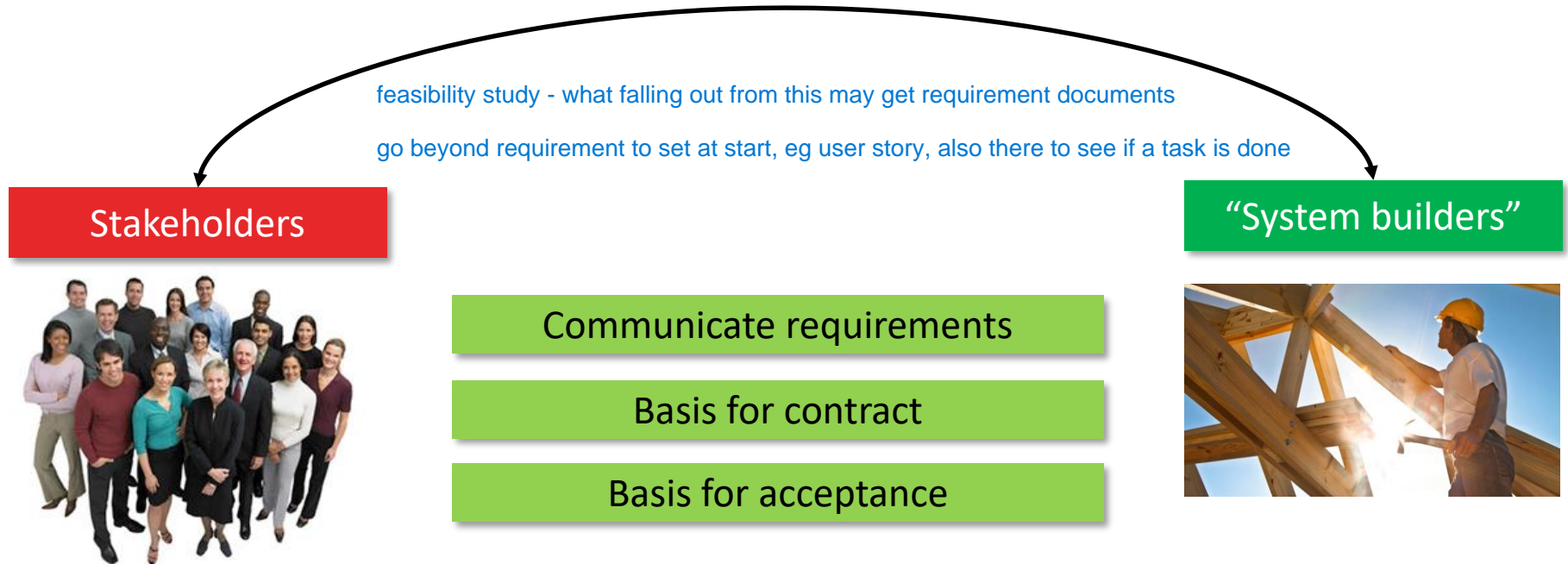


Requirements documentation

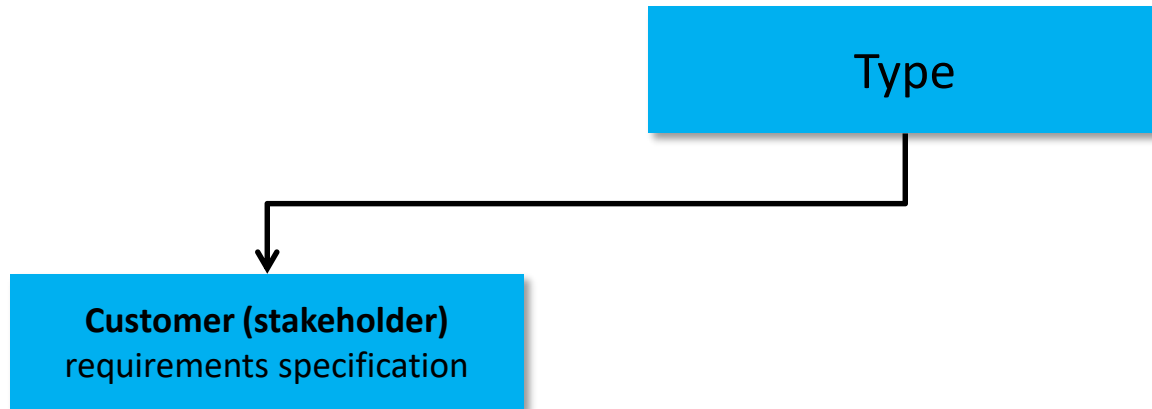
- A **systematically represented collection**
 - of desires and needs,
 - typically for a system or component,
 - clearly and precisely, satisfying given characteristics



Why documentation – bridging the gap

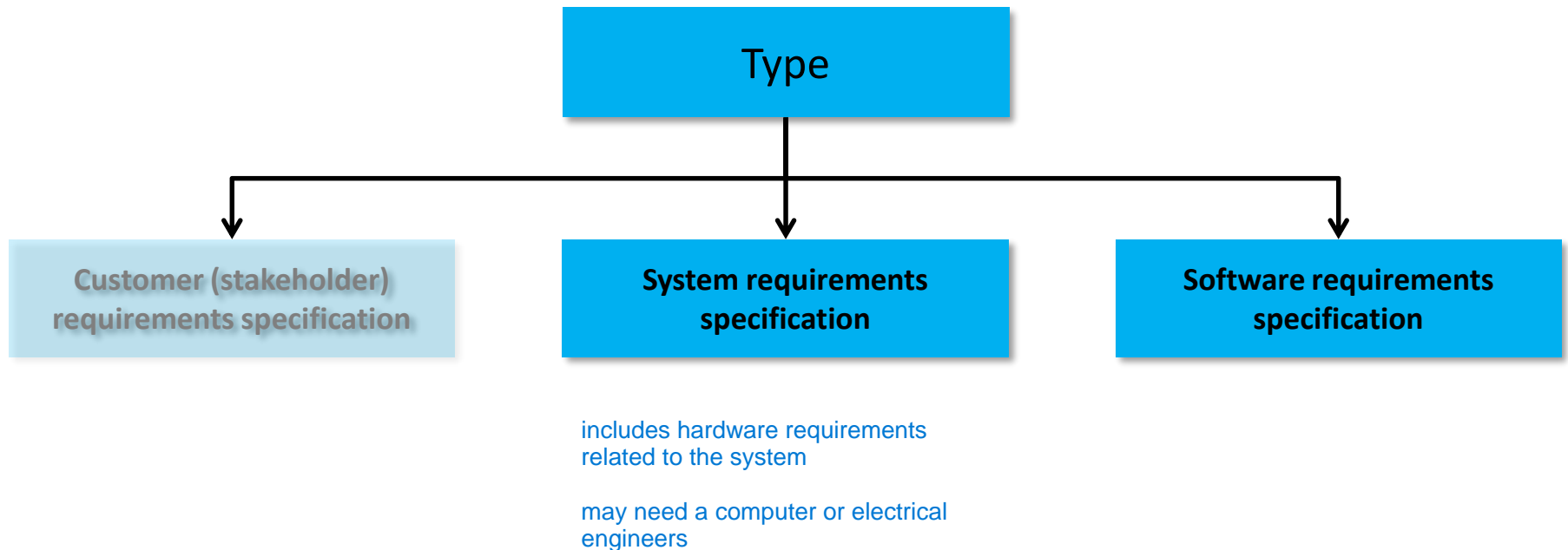


Document types



- **What** stakeholders want, independent of system, **before** development considerations
- **Precedes** and informs system or software requirements specification
- Typically written by **domain experts on customer side**
 - May involve “RE consultants” (business analysts, product owners and other “requirements-related” roles)

Document types



- Classic form of a requirements specification
- No methodological difference between system and software specification
- [Software requirements specification if system pure software]
- Typically written by customer and “vendor” or “producer” and “RE consultants”
 - Could be external providers or vendors, internal development teams, etc.

How to document

- Natural language (narrative text)
 - Can be informal or structured
- Formal languages
 - Typically based on mathematical logic and set theory
- Models and diagrams
 - Often semi-formal or formal
 - Sometimes enhanced with natural language text
- Always keep metadata
 - Author, date created/modified, status, source, rationale, assumptions

Group reading

Status Quo in Requirements Engineering: A Theory and a Global Family of Surveys

STEFAN WAGNER, University of Stuttgart, Germany
DANIEL MÉNDEZ FERNÁNDEZ, Technical University of Munich, Germany
MICHAEL FELDERER, University of Innsbruck and Blekinge Institute of Technology, Sweden
ANTONIO VETRÒ, Nexa Center for Internet & Society, DAUIN, Politecnico di Torino, Italy
MARCOS KALINOWSKI, Pontifical Catholic University of Rio de Janeiro, Brazil
ROEL WIERINGA, University of Twente, The Netherlands
DIETMAR PFAHL, University of Tartu, Estonia
TAYANA CONTE, Universidade Federal do Amazonas, Brazil
MARIE-THERESE CHRISTIANSSON, Karlstad University, Sweden
DESMOND GREER, Queen's University, Belfast, UK
CASPER LASSENIUS, Aalto University and SimulaMet, Norway
TOMI MÄNNISTÖ, University of Helsinki, Finland
MALEKNAZ NAYEBI, University of Calgary, Canada
MARKKU OIVO, University of Oulu, Finland
BIRGIT PENZENSTADLER, California State University, Long Beach, USA

Tayana Conte is supported by CNPq (311494/2017-0). Dietmar Pfahl was supported by the institutional research grant IUT20-55 of the Estonian Research Council. Rafael Prikladnicki is partially funded by Fapergs (process 17/2551-0001205-4) and CNPq. For the work of Dietmar Winkler, the financial support by the Austrian Federal Ministry of Science, Research and Economy and the National Foundation for Research, Technology and Development is gratefully acknowledged.

Authors' addresses: S. Wagner, University of Stuttgart, Stuttgart, Germany; email: stefan.wagner@iste.uni-stuttgart.de; D. M. Fernández, Technical University of Munich, Garching, Germany; email: daniel.mendez@tum.de; M. Felderer, University of Innsbruck, Innsbruck, Austria, Blekinge Institute of Technology, Karlskrona, Sweden; email: michael.felderer@uibk.ac.at; A. Vetrò, Nexa Center for Internet & Society, DAUIN, Politecnico di Torino, Torino, Italy; email: antonio.vetro@polito.it; M. Kalinowski, Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil; email: kalinowski@inf.puc-rio.br; R. Wieringa, University of Twente, Enschede, The Netherlands; email: r.j.wieringa@utwente.nl; D. Pfahl, University of Tartu, Tartu, Estonia; email: dietmar.pfahl@ut.ee; T. Conte, Universidade Federal do Amazonas, Manaus, Brazil; email: tayanaconte@gmail.com; M.-T. Christiansson, Karlstad University, Karlstad, Sweden; email: marie-therese.christiansson@kau.se; D. Greer, Queen's University Belfast, Belfast, UK; email: des.greer@qub.ac.uk; C. Lassenius, Aalto University, Espoo, Finland; email: casper.lassenius@aalto.fi, SimulaMet, Oslo, Norway; email: casper@simula.no; T. Männistö, University of Helsinki, Helsinki, Finland; email: tomi.mannisto@helsinki.fi; M. Nayebi, University of Calgary, Calgary, Canada; email: mnyayebi@ucalgary.ca; M. Oivo, University of Oulu, Oulu, Finland; email: markku.oivo@oulu.fi; B. Penzenstadler, California State University, Long Beach, Long Beach, USA; email: birgit.penzenstadler@csulb.edu; R. Prikladnicki, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, Brazil; email: rafael.prikladnicki@gmail.com; G. Ruhe, University of Calgary, Calgary, Canada; email: Ruhe@ucalgary.ca; A. Scheckelmann, Hochschule Niederrhein, Krefeld, Germany; email: andre.scheckelmann@hs-niederrhein.de; S. Sen, Simula, Fornebu, Norway; email: sagar@simula.no; R. Spinola, Salvador University - UNIFACS, Salvador, Brazil; email: rodrigo.spinola@gmail.com; A. Tuzcu, zeb.rolfes.schierenbeck.associates GmbH, Munich, Germany; email: atuzcu@zeb.de; J. L. de la Vara, Carlos III University of Madrid, Madrid, Spain; email: jvara@inf.uc3m.es; D. Winkler, Technische Universität Wien, CDL-SQL, Vienna, Austria; email: dietmar.winkler@tuwien.ac.at

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.

1049-331X/2019/02-ART9 \$15.00

<https://doi.org/10.1145/3306607>

ACM Transactions on Software Engineering and Methodology, Vol. 28, No. 2, Article 9. Pub. date: February 2019.

Study overview

- Study question
 - How are requirements elicited and **documented**?
- Method
 - Survey research
 - Participants from 228 organizations in 10 countries
- Your task (in pairs)
 - Read Section 5.2, page 20/21/22 of paper
 - Prepare summary (should fit on one slide)

Summary of findings (related to documentation)

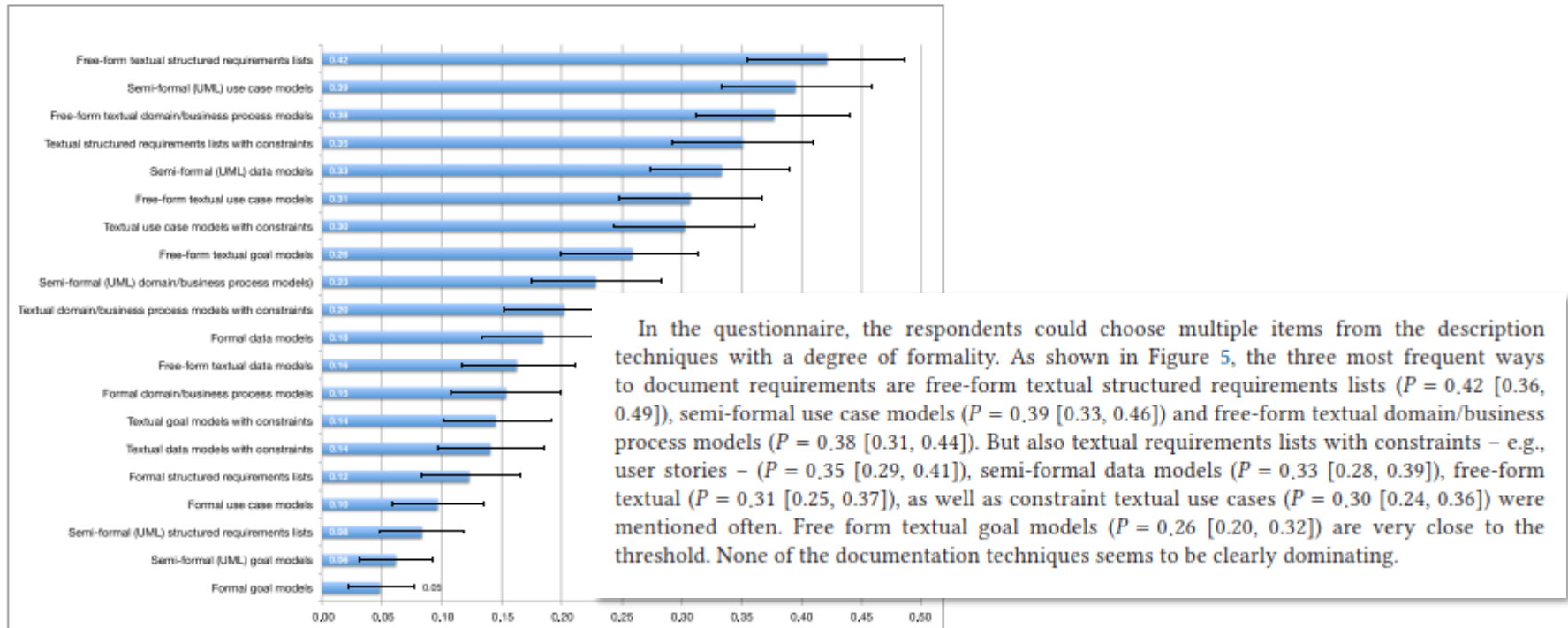



Fig. 5. How do you document functional requirements? ($N = 228$).

All other documentation techniques fall below our threshold by including 0.20 in their CI. Semi-formal domain/business models ($P = 0.23$ [0.17, 0.28]), domain/business model with constraint text ($P = 0.20$ [0.15, 0.25]), formal data models ($P = 0.18$ [0.13, 0.24]), and free-form textual data models ($P = 0.16$ [0.11, 0.21]) all are around 20%. The remaining techniques are then clearly below 20%: Formal business/domain models have $P = 0.15$ [0.11, 0.20], textual goal models with constraints have $P = 0.14$ [0.10, 0.19], textual data models with constraints have $P = 0.14$ [0.10, 0.19], and formal structured requirements lists have $P = 0.12$ [0.08, 0.16]. Rarely used are formal use case models ($P = 0.10$ [0.06, 0.13]), semi-formal structured requirements lists ($P = 0.08$ [0.05, 0.12], as well as semi-formal ($P = 0.06$ [0.03, 0.09]) and formal goal models ($P = 0.05$ [0.02, 0.07]).

Agenda

1. Requirement documentation – overview
2. **Natural language requirements**
3. Formal languages
4. Requirements modeling
5. Requirements characteristics

Natural language requirements

- Specify requirements as text
- Phrase structure or template for **individual requirements** “≡”
eg. user story
- Structure or template for **requirements documents** 
would be a set of individual requirements

Examples

how to reduce documentation debt

ensuring consistency between different projects and employees

- Documents

- Requirements specification document (“traditional”)
- Product and iteration backlogs (“agile”)
- Organization-specific standards, imposed by customer or vendor

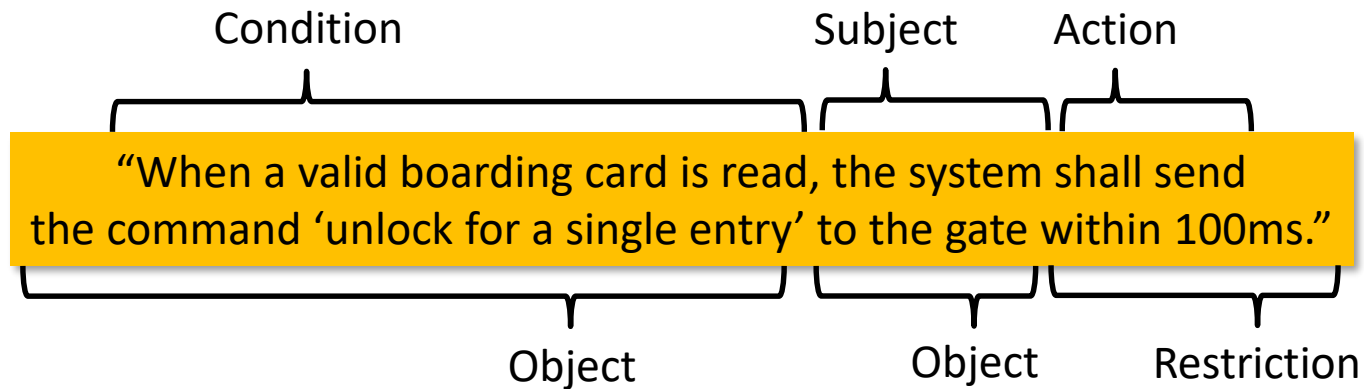
misconception that requirement - dont have time - mis understanding that requirement are not that long

- Individual requirements

- User stories, epics, acceptance tests (“agile” practices)
- Issues, tickets, change requests (“change-based”)
- Code comments (“code-driven”)

Requirements template: basic

- Typical template for software requirements
 - <Condition> <Subject> <Action> <Objects> [<Restriction>]



Requirements template: SAFe (Scaled Agile)

- Forward-looking statement

For *<customers>*
who *<do something>*
the *<solution>*
is a *<something – the “how”>*
that *<provides this value>*
unlike *<competitor, current solution, or non-existing solution>*
our solution *<does something better – the “why”>*
 including rationale for having this requirement in the first place

- Scope

- Success criteria
- What is in and out of scope
- “Non-functional” requirements

Requirements template: story cards

As a user, I shall be able/want to [...], which helps [with goals]

Tasks:

Verify:

Requirements template: user stories

- Potentially with DoD and acceptance criteria, acceptance tests
 - “As <role> in order to <goal> I want to be able to <activity> then <result>/so that <value>”
 - Potentially with Behavior-driven Development : Given-when-then

As a store owner

In order to keep track of stock

I want to add items back to stock when they're returned

Scenario 1: Refunded items should be returned to stock

Given that a customer previously bought a black sweater from me

And I have three black sweaters in stock

When she returns the black sweater for a refund

Then I should have four black sweaters in stock

Scenario 2: Replaced items should be returned to stock

Given that a customer previously bought a blue garment from me

And I have two blue garments in stock

And three black garments in stock

When she returns the blue garment for a replacement in black

Then I should have three blue garments in stock

And two black garments in stock

Example follow-up questions to refine user stories

validate how good this user story is, anything to refine for this mystery user

- “As a user, I want to _ so that I can _”
 - What sort of user? Power, occasional, new, internal, external? Are there behaviours or demographics that make them different from other users?
 - Is this the group of users whose needs we want to address right now? Why?
 - Why do these users want to do the thing you claim they want to do? What is their actual intent or end goal?
 - How do we know they want to do this? Is this based on research or optimism or vibes?
 - Are the action and goal actually the same? For example, is it “I want to log in so that I can see my logged in information?” If so, revise

Requirements template: Atlassian Confluence

(1)

Normal text **B** *I* U ... **Publish** **Close** ...

Requirements

Fill in project details in the table below. Type `/date` to quickly add the target release date, `/jira` to link to Jira epics or issues, and `@mention` functional owners to keep everyone on the same page.

Target release	
Epic	
Document status	DRAFT
Document owner	
Designer	
Tech lead	
Technical writers	
QA	

(2)

Objective

Provide context on this project and explain how it fits into your organization's strategic goals

Success metrics

List project goals and the metrics you'll use to judge its success

Goal	Metric
e.g., Simplify the user experience	e.g., Customer satisfaction score increases

Assumptions

List any assumptions you have about your users, technical constraints, or business goals (e.g., Most users will access this feature from a tablet)

-

Milestones

Use the roadmap planner (`/roadmap`) to help your team stay on track. To edit workstreams or dates, select the placeholder below and tap the pencil icon.

(3)

Milestones

Use the roadmap planner (`/roadmap`) to help your team stay on track. To edit workstreams or dates, select the placeholder below and tap the pencil icon.

Type `/trellio` to add a card or board to this page or `/jira` to include a Jira issue, chart, or project.

Options considered

	Requirement	User story	Importance	Jira Issues	Notes
1					
2	e.g., Must be mobile responsive	e.g., John is a PM who wants to check on his team's progress	HIGH		

(4)

	Requirement	User story	Importance	Jira Issues	Notes
2	e.g., Must be mobile responsive	e.g., John is a PM who wants to check on his team's progress from the train station	HIGH		
3					

User interaction and design

Add mockups, diagrams, or visual designs related to these requirements. Type `/image` to upload a file.

Open questions

Question	Answer	Date answered
e.g., How might we make users more aware of this feature?	e.g., We'll announce the feature with a blog post and a Summit presentation	

Out of scope

List anything that's out of scope for this feature or release

-

Document template: IEEE Std 830-1998

a document template based on a standard.

1. Introduction

1.1 Purpose

1.2 Scope

1.3 Definitions, acronyms and abbreviations

1.4 References reference thing that are relevant to explain background information and motivation

1.5 Overview

2. Overall description

2.1 Product perspective

2.2 Product functions

can put reference in overall description to add more context too.

2.3 User characteristics

2.4 Constraints

2.5 Assumptions and dependencies

3. Specific requirements

Appendices

Index

Document template: Volere

Project Drivers

1. The Purpose of the Project
2. Client, Customer & other Stakeholders
3. Users of the Product

Project Constraints

4. Mandated Constraints
5. Naming Conventions and Definitions
6. Relevant Facts and Assumptions

Functional Requirements

7. The Scope of the Work
8. The Scope of the Product
9. Functional and Data Requirements

Non-Functional Requirements

10. Look and Feel Requirements
11. Usability and Humanity Requirements
12. Performance Requirements
13. Operational Requirements

14. Maintainability and Support Requirements

15. Security Requirements
16. Cultural and Political Requirements
17. Legal Requirements

Project Issues

18. Open Issues
19. Off-the-Shelf Solutions
20. New Problems
22. Tasks
22. Cutover
23. Risks
24. Costs
25. User Documentation and Training
26. Waiting Room
27. Ideas for Solutions

Rules for natural language

- Define precise meanings and semantics (glossary if needed)
 - Auxiliary verbs
 - “Shall”, “must”: must be implemented
 - “Should”, “may”: can be implemented
 - “Will”: future requirement
 - Process verbs (“produce”, “generate”, “create”, “the data”, “the display”)
- When comparing, specify reference: “faster” → “faster than”
- Scrutinize absolute quantifiers
 - “every”, “always”, “never”, etc. rarely hold without exceptions
- Avoid redundancy where possible: “never ever” → “never”

Example requirements

- Do you see any problem(s) with the requirements?
 - If so, what are problems?
 - What would be the negative impact of using the requirement as proposed?
- What would be a better way to write requirement(s)?
 - If any?

Example (1)

“The software will not be installed from unknown sources on the system without having the software tested and approved.”

- **Problem**
 - Two constraints are specified simultaneously
 - Unclear relationship between constraints

TODO 12:09

- **Better**

“3.2.5.2 The software shall be installed on the operational system only after it has been tested and approved.”

Example (2)

“3.4.6.3 The system shall prevent the processing of duplicate electronic files by checking a SDATE record. An e-mail message shall be sent.”

- **Problem**

- Two “shall” under one requirements identifier
- What is the e-mail message? For what purpose?
- What is SDATE?

- **Better**

“3.4.6.3 The system shall: a. Prevent processing of duplicate electronic files by checking the date and time of the submission. b. Send the following e-mail message: b.1 Request updated submission of date and time, if necessary, or b.2. That the processing was successful, if successful.”

Example (3)

“3.2.6.3 The system shall receive and process state data from the state processing subsystem.”

- **Problem**
 - Vague terms, what are “process”, “state data”
- **Better**

“3.2.6.3 The system shall receive a. Production data that contain data from multiple states b. Financial state data for one or more states, extracted by the state processing subsystem.”

“3.2.6.4 The system shall parse multi-state data to respective state files.”

Example (4)

“3.2.8.2 The enrolment process shall take 1 to 10 calendar days to complete all enrolment types.” (*system administrator*)

“3.2.8.3 The enrolment process shall take no more that 3 days to complete for credit enrolments.” (*financial administrator*)

- **Problem**
 - Inconsistent requirements
- **Better**

“3.2.8.2 The enrolment process shall take a. 1 to 3 calendar days to complete for credit enrolment b. 1 to 10 calendar days to complete for all other enrolment types.

Example (5)

what/who is doing the calculations

what even is correct

“3.2.8.6 When doing calculations, the software shall produce correct results.”

- **Problem**
 - This is not a requirement
 - Something “obvious”
- **Better**
 - Remove it

User stories examples

- Do you see any problem(s) with the user stories?
 - If so, what are problems?
 - What would be the negative impact of using the requirement as proposed?
- What are follow-up questions?
 - If any?

User story example (1)

As a customer, I can add multiple books to my shopping card which is shown.

- **Problem**
 - “multiple”: how many books?
 - “shown”: where, what detail?
- **Follow-up questions**
 - How many items may be kept in the shopping cart?
 - Must the content of the shopping cart always shown?
 - If so, how? With how much details?

User story example (2)

As a customer, I can search for a product and add it to my shopping list.

- **Problematic terms**
 - “and”: two use cases/stories?
 - “search for”: based on which criteria?
- **Follow-up questions**
 - What are the search criteria?
 - How can I search for a product, is there a search bar, where, when?
 - If so, are there advanced search facilities or filters?

User story example (3)

As a customer, I can add a product to my cart.

- **Problematic terms**

- “customer”: any type of customer?
- “my cart”: retention policy?

whats the value, goal, outcome

a non log in customer can add up to 5 product from the catalogue to the cart that will remain in there for 2 days.

- **Follow-up questions**

- Do I have to be logged in to add products to my cart or not?
- How long is my cart's content retained?
- What if the product is sold out?

User story example (4)

As a customer, I can view the content of my shopping cart.

what type of content to display and is it interactive

- **Problematic term**
 - “view”: what type of devices are considered?
 - “view”: what type of format is considered?
- **Follow-up questions**
 - Do we have to deal with responsiveness?
 - Do we have to build a dedicated mobile app?
 - Do we have to deal with exports to separate views or files?

User story example (5)

As a customer, I can log into my account so that I can view my cart.

- **Problematic term**
 - “log into”: what are the needed details to log in?
- **Follow-up questions**
 - What happens after multiple login attempts failed?
 - How many failed logins are allowed?
 - Is there a “lost password” procedure?
 - May users be blocked by administrators?

User story example (6)

As a customer, I can search for products so that I can find types of products I am interested in.

- **Problematic terms**
 - “search”: based on what criteria?
 - “find”: how are results displayed?
- **Follow-up questions**
 - How are books defined in the system?
 - How are results shown?
 - Are there additional filtering criteria?

User story example (7)

As a customer, I can review the items in my shopping cart and pay for them.

- **Problematic terms**
 - “review”: what does this mean exactly, may I delete or update books?
 - “pay”: what kind of payment procedure?
- **Follow-up questions**
 - What is the complete flow between the review and payment?
 - What are third parties required for payment?
 - Am I allowed to cancel the process at any time?

Natural language: pro and con

- Pro

- The oldest and most widely used way
- Expressive
- Comes “natural” suitable for non technical, natural to lots of types of stakeholders

- Con

- Error-prone: contradictions
- Often ambiguous, imprecise, redundancies
- Noise and silence, over-specification templates can help
- Flow: forward and backward references story that refer to each other, how to identify and respond to dependencies
- Verification only by careful reading

silent, may be difficult to know when we have captured all the details, as we may not be aware of what we don't know. a hidden requirement very hard to know

Agenda

1. Requirements documentation – overview
2. Natural language requirements
3. **Formal languages** opposite end of spectrum to natural language
formal language inspired by concepts from math, specify and validate (model checker)
requirements in a formal way, resulting in very high-quality requirements as it, compared to
natural language especially, have less ambiguity and interpretation errors.
4. Requirements modeling
5. Characteristics of requirements

Formal languages

- Languages with formally defined **syntax** and **semantics**
 - Mathematical model of system state and state change
 - Often based on sets, relations, and logical expressions
 - Examples
 - Algebraic, Petri nets, automata, hybrid (e.g., Alloy, OCL, VDM, Z), LTL, CTL
 - **Very close to implementation level, pseudo-code**
- Vision
 - Specify requirements formally
 - Prove that requirements meet properties, e.g.,
 - Theorem provers, model checkers (properties inferred from logic, states)
 - Implement by correctness-preserving transformations
 - Maintain specification, no longer the code

Example (properties): stack (last-in-first-out)

AXIOMS

$\forall s \in \text{Stack}, e \in \text{elem}$

- (1) $\neg \text{full}(s) \rightarrow \text{pop}(\text{push}(s,e)) = s$ -- pop reverses effect of push
- (2) $\neg \text{full}(s) \rightarrow \text{top}(\text{push}(s,e)) = e$ -- top retrieves most recently stored element
- (3) $\text{empty}(\text{new}) = \text{true}$ -- a new stack is always empty
- (4) $\neg \text{full}(s) \rightarrow \text{empty}(\text{push}(s,e)) = \text{false}$ -- after push, a stack is not empty
- (5) $\text{full}(\text{new}) = \text{false}$ -- a new stack is not full
- (6) $\neg \text{empty}(s) \rightarrow \text{full}(\text{pop}(s)) = \text{false}$ -- after pop, a stack is not full

Example: Z

- Theatre: Tickets for the first night are only sold to friends

$Status ::= standard \mid firstNight$

$Friends$
$friends : \mathbb{P} Person$
$status : Status$
$sold : Seat \rightarrow Person$
$status = firstNight \Rightarrow \mathbf{ran} sold \subseteq friends$

$TicketsForPerformance1 \hat{=} TicketsForPerformance0 \wedge Friends$

and

$TicketsForPerformance1$
$Friends$
$TicketsForPerformance0$

Formal specifications in practice

- Intensive research
 - Dates back to 1977
- Situation today
 - **Safety**-critical applications (e.g., railway switches)
 - **Security**-critical applications (e.g., access control, electronic banking)
 - **Regulations** (e.g., e-signature, EAL6/7* common criteria for security)
 - However
 - Broad use often not possible/feasible (validation problems)
 - Broad use often not reasonable (cost exceeds benefit)
- Semi-formal models where critical parts are fully formalized

The role of formalism in system requirements

JEAN-MICHEL BRUEL, University of Toulouse, IRIT

SOPHIE EBERSOLD, University of Toulouse, IRIT

FLORIAN GALINIER, University of Toulouse, IRIT

MANUEL MAZZARA, Innopolis University

ALEXANDR NAUMCHEV, Innopolis University

BERTRAND MEYER, Schaffhausen Institute of Technology and Innopolis University

A major determinant of the quality of software systems is the quality of their requirements, which should be both understandable and precise. Most requirements are written in natural language, good for understandability but lacking in precision.

To make requirements precise, researchers have for years advocated the use of mathematics-based notations and methods, known as “formal”. Many exist, differing in their style, scope and applicability. The present survey discusses some of the main formal approaches and compares them to informal methods.

The analysis uses a set of 9 complementary criteria, such as level of abstraction, tool availability, traceability support. It classifies the approaches into five categories based on their principal style for specifying requirements: natural-language, semi-formal, automata/graph, mathematical, seamless (programming-language-based). It includes examples from all of these categories, altogether 21 different approaches, including for example SysML, Relax, Eiffel, Event-B, Alloy.

The review discusses a number of open questions, including seamlessness, the role of tools and education, and how to make industrial applications benefit more from the contributions of formal approaches.

ACM Reference Format:

Jean-Michel Bruel, Sophie Ebersold, Florian Galinier, Manuel Mazzara, Alexandr Naumchev, and Bertrand Meyer. 2021. The role of formalism in system requirements. *ACM Comput. Surv.* 1, 1, Article 1 (January 2021), 35 pages. <https://doi.org/10.1145/3448975>

Study overview

- Study question
 - What degree of formalism is required
 - How does it impact different aspects of software development
- Method
 - Literature review
- Your task (in pairs)
 - Read paper, focus on one approach
 - Prepare summary, including example (should fit on one slide)

Summary of findings

Name	Category	References	Section
Requirements Grammar*	Natural language	[111]	4.1.1
Relax		[130]	4.1.2
Stimulus		[59]	4.1.3
NL to OWL*		[65]	
NL to OCL*		[45]	
NL to STD*		[4]	
Reqtify	Semi-formal	[118]	
KAOS		[127]	
SysML		[94]	
URN		[6]	
URML		[11]	
Statecharts	Automata- or graph-based	[47]	
Problem Frames		[58]	
FSP/LTSA		[69]	
Petri Nets		[99]	
Event-B	Mathematical notation	[1]	
Alloy		[57]	
FORM-L		[91]	
VDM		[12]	
Tabular Relations		[98]	
Multirequirements	Seamless, PL based	[75]	
SOOR		[87]	

Table 1. Requirements approaches survey

	Scope	Audience prerequisites	Level of abstraction	Associated method	Traceability support	Non-functional requirements support	Semantic definition	Tool support	Verifiability
Requirements Grammar	B	N	H	✗	✗	✓	✗	✓	✗
Relax	B	N	H	✗	✓	✓	✓	✓	✓
Stimulus	B	N	H	✗	✗	✗	✓	✓	✓
NL to OWL	B	S	H	✓	✓	✓	✓	✗	✗
NL to OCL	S	S	B	✓	✗	✗	✓	✓	✓
NL to STD	B	S	H	✓	✗	✓	✓	✗	✓
Reqtify	B	N	L	✗	✓	✓	✗	✓	✗
KAOS	B	S	B	✓	✓	✓	✓	✓	✗
SysML	S	S	H	✗	✓	✓	✗	✓	✗
URN	S	S	H	✓	✓	✓	✗	✓	✗
URML	B	S	H	✗	✓	✓	✓	✓	✗
Statecharts	S	S	H	✓	✗	✗	✓	✓	✓
Problem Frames	B	S	H	✓	✗	✗	✗	✓	✓
FSP/LTSA	B	S	H	✗	✗	✓	✓	✓	✓
Petri Nets	S	S	H	✗	✗	✗	✓	✓	✓
Event-B	B	F	B	✓	✗	✗	✓	✓	✓
Alloy	S	F	H	✗	✗	✗	✓	✓	✓
FORM-L	B	S	H	✗	✗	✗	✓	✓	✓
VDM	B	F	H	✓	✗	✗	✓	✓	✓
Tabular Relations	B	M	L	✗	✓	✗	✓	✗	✓
Multirequirements	B	S	B	✓	✓	✗	✓	✓	✓
SOOR	B	S	B	✓	✓	✗	✓	✓	✓

Formal languages: pro and con

- Pro

- Purely descriptive, mathematically elegant, unambiguous by definition
- Fully verifiable
- Important properties can be proven and tested automatically

- Con

need training to get the certain skill to get use of cryptic notations

- Cost versus value pokemon go would be hard
- Over- and under-specification difficult to spot
- Stakeholders cannot read specification: how to validate?

Agenda

1. Requirements documentation – overview
2. Natural language requirements
3. Formal languages
4. **Requirements modeling** middle of spectrum between natural language, and formal language
5. Characteristics of requirements

What can be modeled?

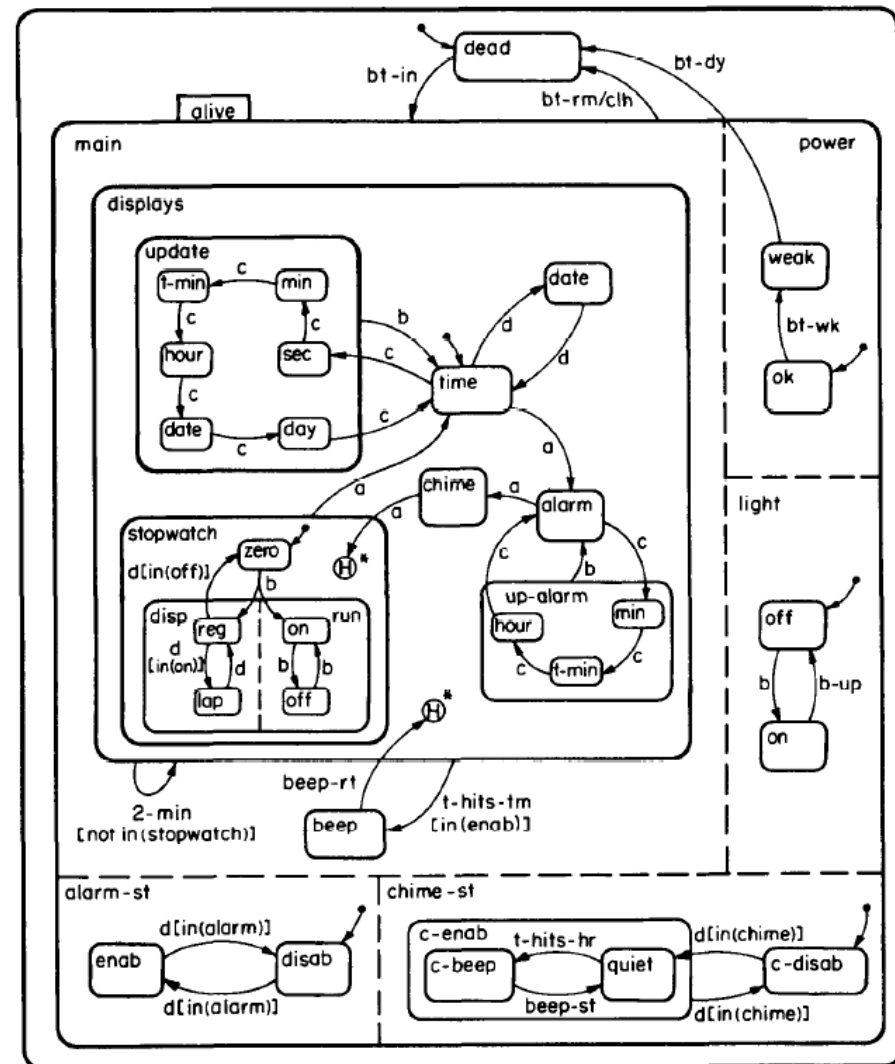
- System, behavior, functions and flows
- Goal view can study and decompose the high level business goal into sub goals and how they work towards achieving the high goal.

What can be modeled?

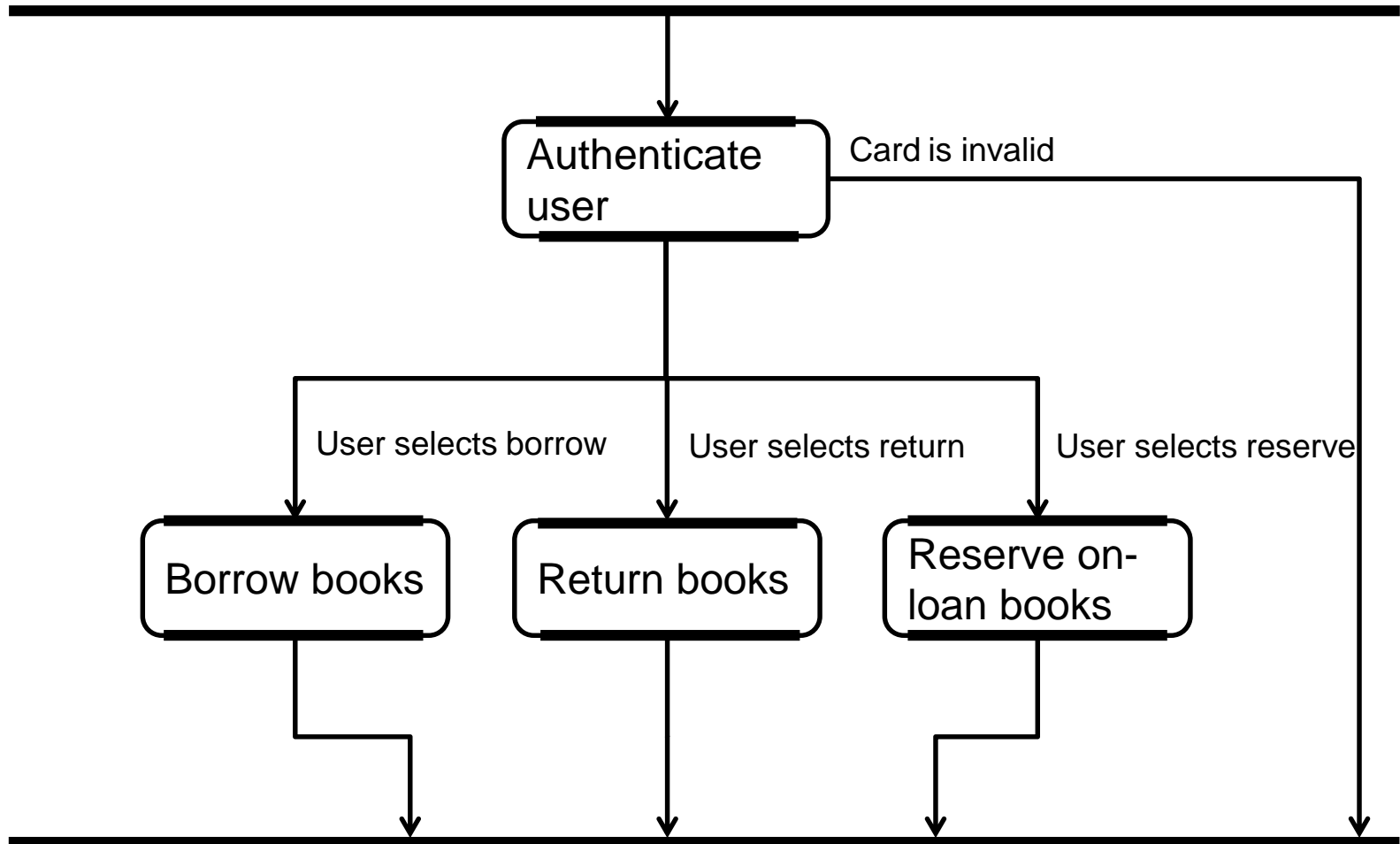
- System, behavior, functions and flows
 - Describes dynamic system behavior
 - How system reacts to a sequence of external events
 - Example techniques
 - [Sequence diagrams]
 - [Use case diagrams]
 - State charts, dependency charts
 - Data and information flow
 - Process and workflow
- Goal view

State charts

- Reaction to external events
 - Dynamic behavior
 - How components coordinate work
 - States may be nested / parallel

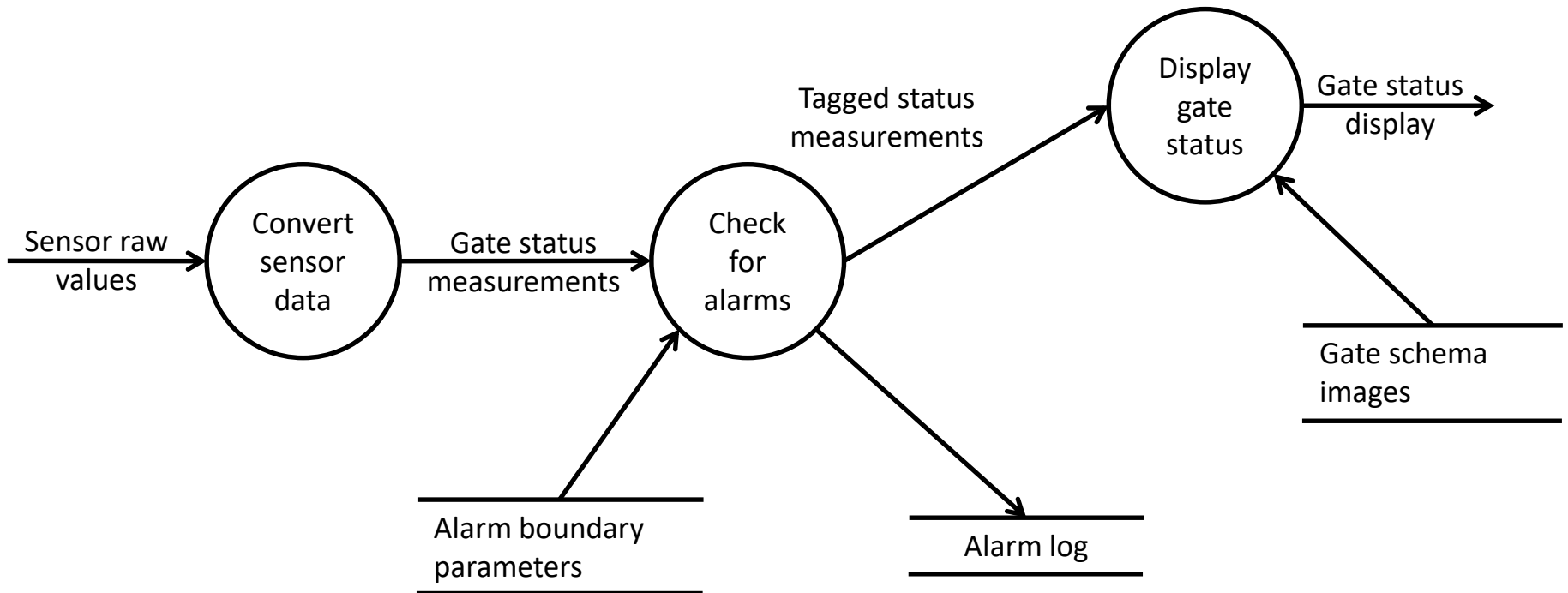


Dependency charts



Data and information flow models

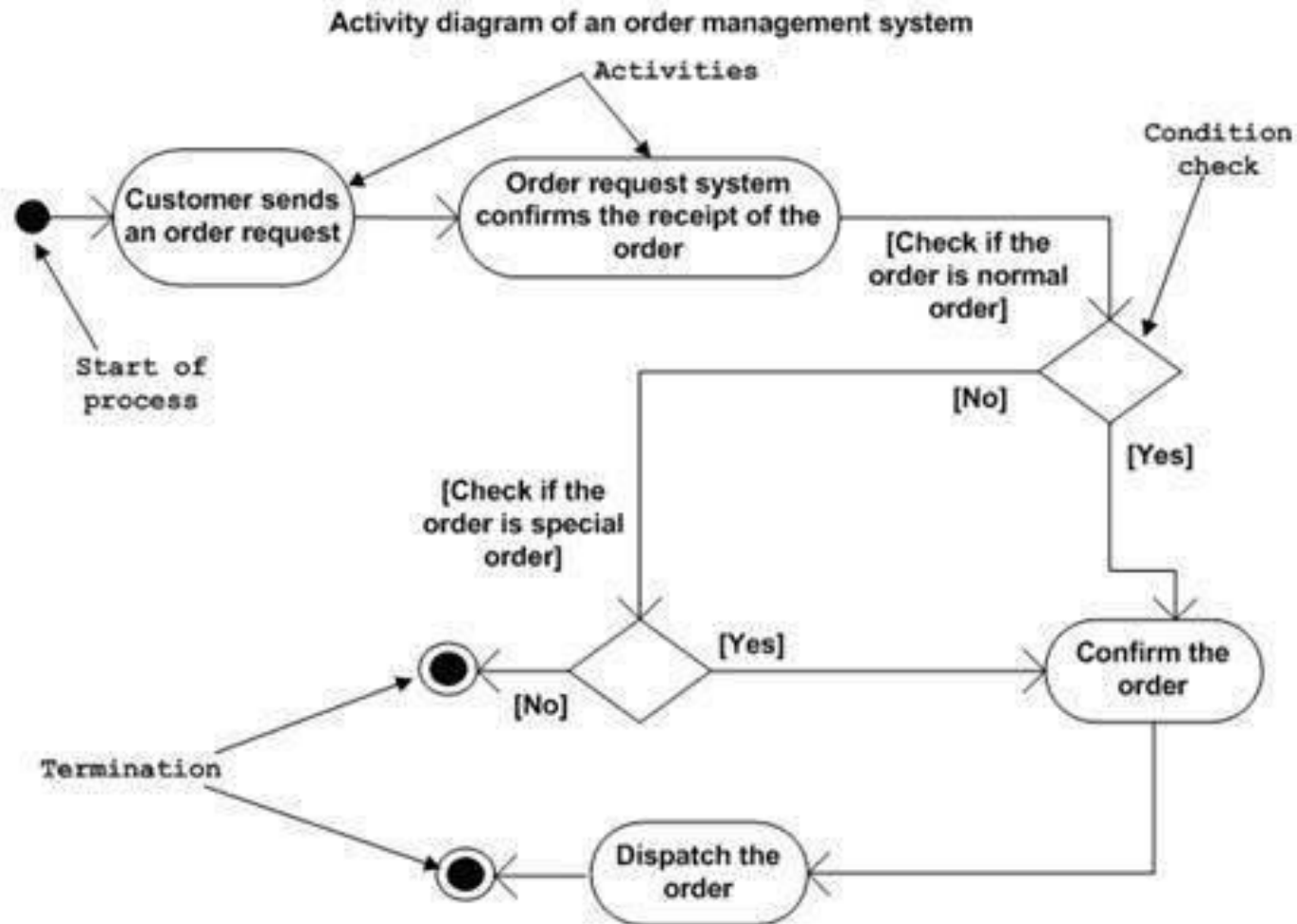
- Model system functionality with data flow diagrams



Process and workflow modeling

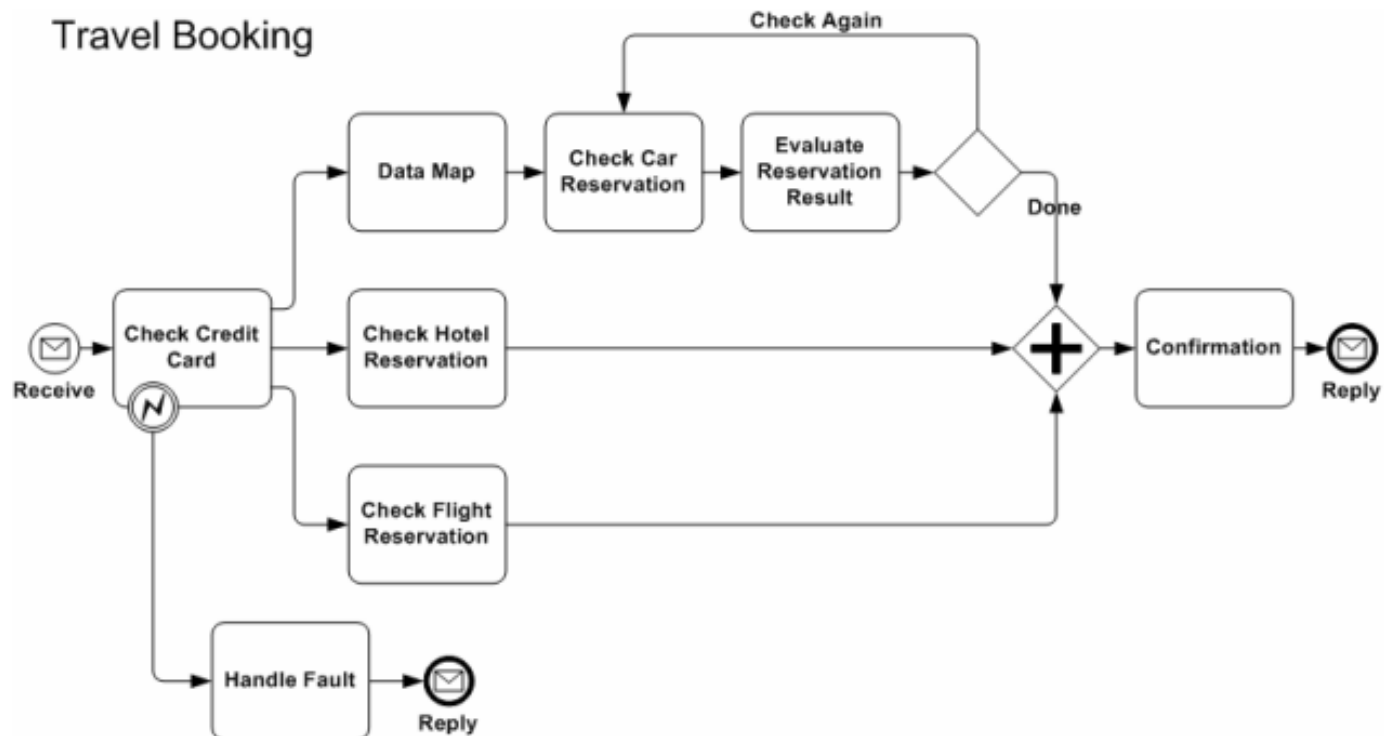
- Elements
 - Process steps
 - Events
 - Control flow
 - Maybe data / information access and responsibilities
- Typical languages
 - UML activity diagrams
 - BPMN (Business Process Model and Notation)

Example: UML activity diagram



Example: BPMN

- Business Process Model and Notation
 - Object Management Group (OMG) standard
 - Rich language for describing business processes



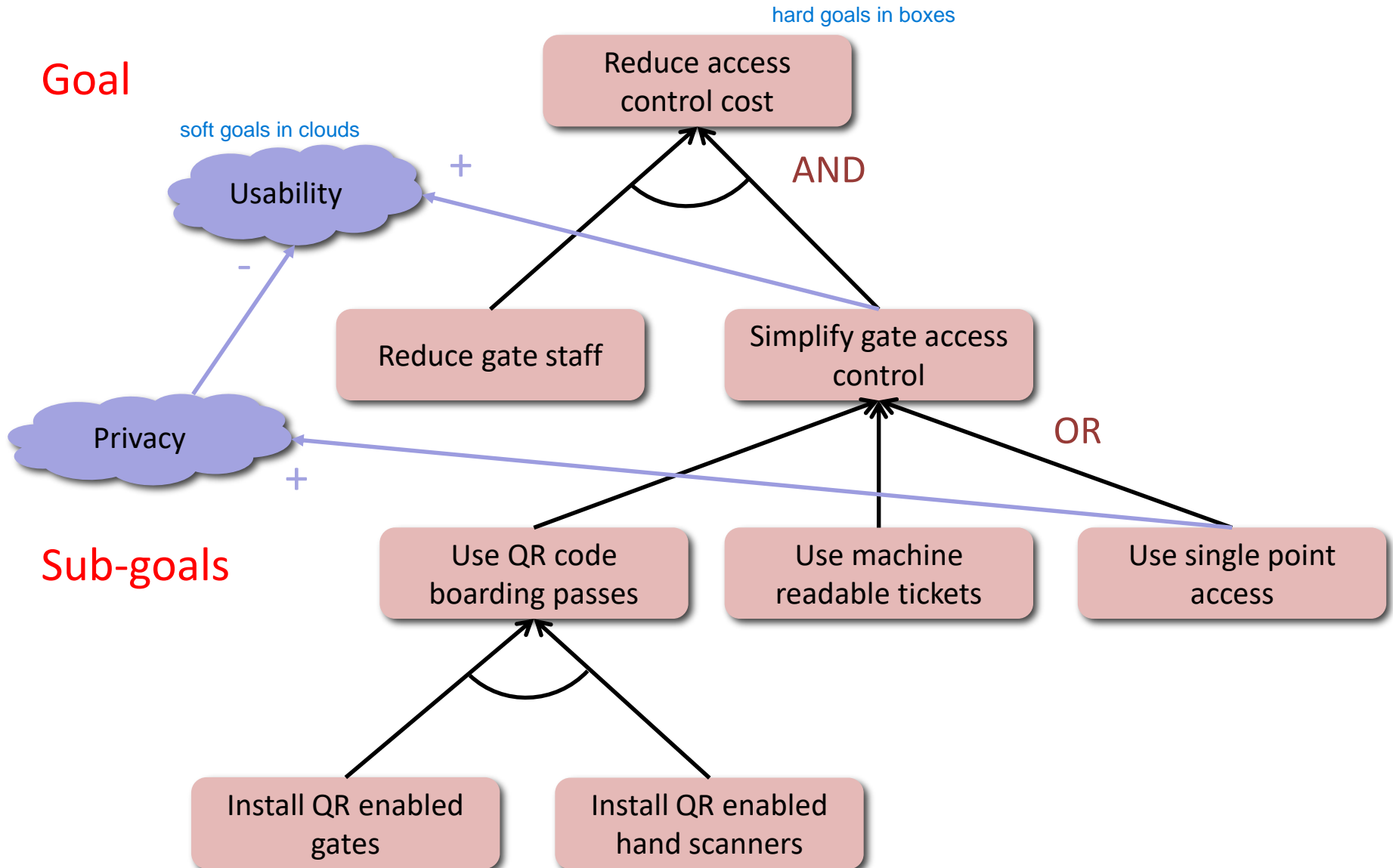
What can be modeled?

- System, behavior, functions and flows
- Goal view

What can be modeled?

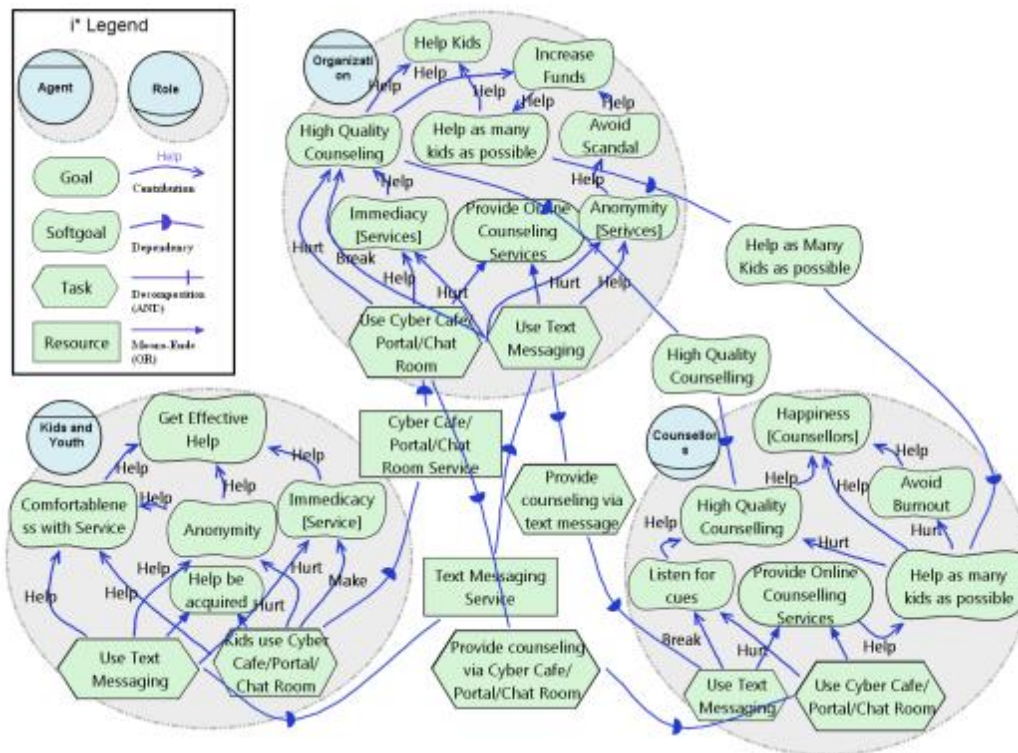
- System, behavior, functions and flows
- Goal view
 - Focus on **why** the system is developed
 - **Why** as a set of stakeholder goals
 - Goals and their dependencies (including “obstacle relationships”)
 - Goal statement
 - *To <active verb phrase> <target object>*
 - E.g., “to maintain market share”
 - Goal hierarchies
 - What are the main goals, how do they relate to each other
 - Use goal refinement to arrive at specific requirements

Trees for goal modeling (soft and hard goals)



Extension: goal-agent networks

- Models
 - Agents (stakeholders) and their goals
 - Tasks that achieve goals and resources
 - Dependencies between these items



Requirements modeling: pro and con

- Pro

- Global view of system behavior
- Supports system decomposition
- Easy to understand than formal models
more appealing to discuss with stakeholders

- Con

- Might be difficult to maintain a large model and changes need to understand its effects
- Might be difficult to comprehend (size of models) may need training

Agenda

1. Documenting requirements
2. Natural language requirements
3. Formal languages
4. Requirements modeling
5. Characteristics of requirements

Typical best practices

- SMART
 - **S**pecific, **M**easurable, **A**chievable, **R**elevant, **T**ime-bound
 - Other definitions exist
 - SMARTER: SMART + **E**valuated, **R**eviewed
- INVEST
 - **I**ndependent, **N**egotiable, **V**aluable, **E**stimable, **S**calable, **T**estable
- CUTFIT
 - **C**onsistent, **U**nambiguous, **T**estable, **F**easible, **I**ndependent, **T**raceable

Characteristics of requirements

- Applies to any form of requirements documentation

Characteristic	Explanation
Cohesive	The requirement addresses one and only one thing.
Complete	The requirement is fully stated in one place with no missing information.
Consistent	The requirement does not contradict any other requirement and is fully consistent with all authoritative external documentation.
Correct	The requirement meets all or part of a business need as authoritatively stated by stakeholders.
Current	The requirement has not been made obsolete by the passage of time.
Externally Observable	The requirement specifies a characteristic of the product that is externally observable or experienced by the user. "Requirements" that specify internal architecture, design, implementation, or testing decisions are properly constraints, and should be clearly articulated in the Constraints section of the Requirements document.
Feasible	The requirement can be implemented within the constraints of the project.
Unambiguous	The requirement is concisely stated without recourse to technical jargon, acronyms (unless defined elsewhere in the Requirements document), or other esoteric verbiage. It expresses objective facts, not subjective opinions. It is subject to one and only one interpretation. Vague subjects, adjectives, prepositions, verbs and subjective phrases are avoided. Negative statements and compound statements are prohibited.
Mandatory	The requirement represents a stakeholder-defined characteristic the absence of which will result in a deficiency that cannot be ameliorated.
Verifiable	The implementation of the requirement can be determined through one of four possible methods: inspection, analysis, demonstration, or test.

Group reading

Challenges of working with artifacts in requirements engineering and software engineering

Parisa Ghazi¹ · Martin Glinz¹

© Springer-Verlag London 2017

Abstract When developing or evolving software systems of non-trivial size, having the requirements properly documented is a crucial success factor. The time and effort required for creating and maintaining non-code artifacts are significantly influenced by the tools with which practitioners view, navigate and edit these artifacts. This is not only true for requirements, but for any artifacts used when developing or evolving systems. However, there is not much evidence about how practitioners actually work with artifacts and how well software tools support them. Therefore, we conducted an exploratory study based on 29 interviews with software practitioners to understand the current practice of presenting and manipulating artifacts in tools, how practitioners deal with the challenges encountered, and how these challenges affect the usability of the tools used. We found that practitioners typically work with several interrelated artifacts concurrently, less than half of these artifacts can be displayed entirely on a large screen, the artifact interrelationship information is often missing, and practitioners work collaboratively on artifacts without sufficient support. We identify the existing challenges of working with artifacts and discuss existing solutions proposed addressing them. Our results contribute to the body of knowledge about how practitioners work with artifacts when developing or evolving software, the challenges they are faced with, and the attempts to address these challenges.

✉ Parisa Ghazi
ghazi@ifi.uzh.ch
Martin Glinz
glinz@ifi.uzh.ch

¹ Department of Informatics, University of Zurich,
Binzmühlestrasse 14, 8050 Zurich, Switzerland

Keywords Documentation · Artifacts · Requirements engineering · Software engineering tools · Collaboration · Interview · Exploratory study

1 Introduction

Adequate documentation is essential for successful software development [54]. This is particularly true for requirements, where missing or deficient documentation may lead to developing the wrong product. As documentation consumes time and effort, it is often neglected, leading to documents which are poor or outdated [3, 56]. However, minimizing the effort for documentation does not necessarily result in time savings and less effort, since proper documentation reduces the duration of tasks, while inadequate documentation increases the risk of making mistakes and causes late discovery of the mistakes which in turn lead to a large amount of rework [64].

In requirements engineering (RE), documenting requirements is of crucial importance, regardless of the RE process or methods being used [46]. Requirements can be documented in a variety of artifacts, from comprehensive requirements specifications to user stories, use cases, glossaries or diagrams. If multiple items (textual requirements, user stories, diagrams, etc.) are stored together in one document, we consider this as one single artifact.

Requirements engineers typically use tools for manipulating (creating, editing or viewing) artifacts, which range from general-purpose tools, such as text or diagram editors, to tools specifically built for RE purposes [12]. The usability of such tools, in particular, the effectiveness, efficiency, and satisfaction with which requirements engineers can use them for viewing and editing artifacts has a

Study overview

- Study question
 - How to work with large and interconnected requirements artifacts
 - RQ1: How does interaction affect efficiency of work with artifacts?
 - RQ2: What interaction challenges do practitioners encounter?
 - RQ3: What methods do practitioners use to handle interaction challenges?
 - RQ4: Which existing solutions address the challenges of working with artifacts?
- Method
 - Interview study with 29 practitioners from 29 companies
- Your task (in pairs)
 - Read paper (focus on Table 8 “Key findings” and surrounding text)
 - Prepare summary (should fit on one slide)

Summary of findings

Table 8 Key findings

RQ1: Properties of artifacts, screens, and tools

- FA1 Only about one-third of the graphical artifacts used by the interviewed practitioners fit on their screens
- FA2 About forty percent of the graphical artifacts do not fit on the largest screen reported in this study
- FA3 More than half of the interviewed practitioners use four or more artifacts at the same time
- FA4 More than two-thirds of the interviewed participants use customized notations for their artifacts
- FA5 On average, practitioners use more than five applications concurrently to create and manage their artifacts
- FA6 Although almost all participants work on their artifacts collaboratively, less than one-third of the collaboration is done with dedicated software development tools





RQ2: Challenges practitioners encounter when working with artifacts

- FB1 “Relying on memory”, “Searching for information”, and “Maintaining the overview” are the most important challenges in handling large artifacts
- FB2 “Switching between windows” and “Working in too small windows” are the most important challenges when working with multiple artifacts
- FB3 Storing insufficient artifact relationship information provokes creating larger artifacts and makes searching and understanding artifacts more demanding

RQ3: Dealing with the challenges

- FC1 Practitioners use their memory extensively
 - FC2 Traditional zooming and scrolling are the dominating techniques for handling large artifacts
 - FC3 Almost all practitioners need to have an overview of the artifacts
 - FC4 Non-software approaches are mostly used for handling the challenges of working with multiple artifacts at the same time
 - FC5 Paper is used by all and whiteboards are used by two-thirds of the participants for creating artifacts
 - FC6 The reasons for using papers and whiteboards for artifact creation include seeking more speed, flexibility and space to work
 - FC7 To store artifact interrelationship information, practitioners extensively use inefficient, time-consuming and error-prone methods
-

Summary

1. Requirements documentation – overview 
2. Natural language requirements 
3. Formal languages 
4. Requirements modeling 
5. Characteristics of requirements 