Softwaretechnik / Software-Engineering

Lecture 10: Requirements Engineering Wrap-Up

2016-06-13

Prof. Dr. Andreas Podelski, **Dr. Bernd Westphal**

Albert-Ludwigs-Universität Freiburg, Germany

Topic Area Requirements Engineering: Content

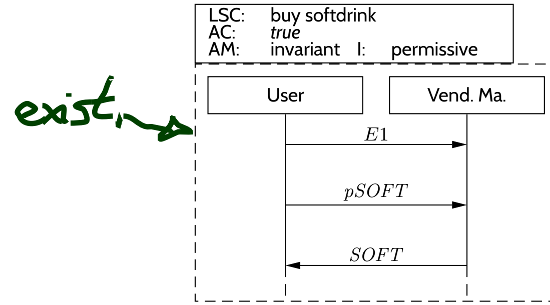
Content

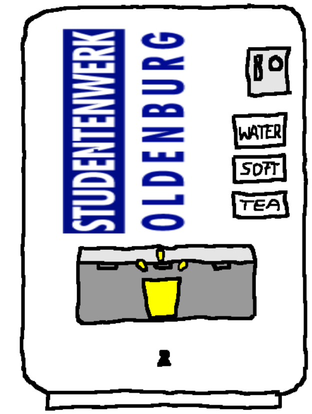
- **Pre-Charts**
  - Semantics, once again
  - Requirements Engineering with scenarios
  - Strengthening scenarions into requirements
- **Software**, formally
  - Software specification
  - Requirements Engineering, formally
  - Software **implements** specification
- **LSCs vs. Software**
  - Software **implements** LSCs
  - **Scenarios and tests**
  - Play In/Play Out
- **Requirements Engineering Wrap-Up**

Pre-Charts (Again)

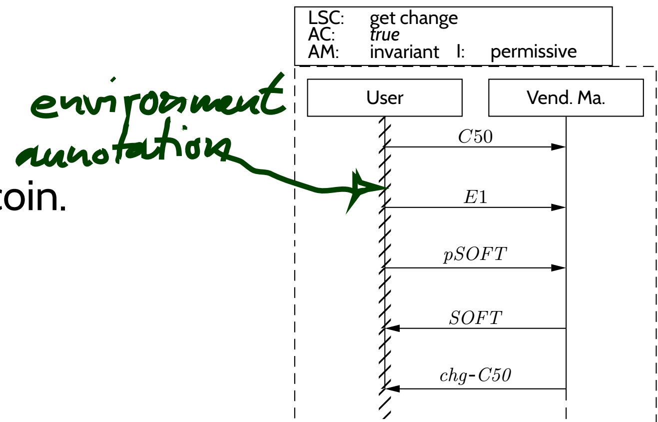
Example: Vending Machine

- **Positive scenario**: Buy a Softdrink
  - (i) Insert one 1 euro coin.
  - (ii) Press the 'softdrink' button.
  - (iii) Get a softdrink.

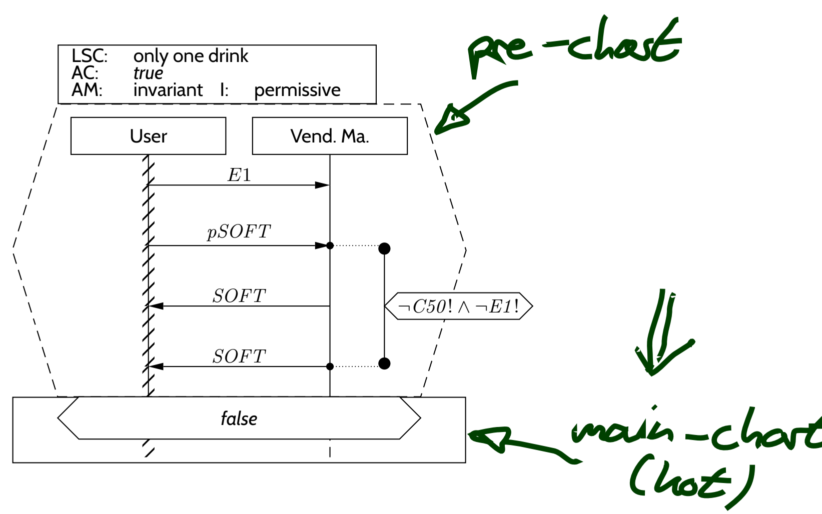




- **Positive scenario**: Get Change
  - (i) Insert one 50 cent and one 1 euro coin.
  - (ii) Press the 'softdrink' button.
  - (iii) Get a softdrink.
  - (iv) Get 50 cent change.



- **Negative scenario**: A Drink for Free
  - (i) Insert one 1 euro coin.
  - (ii) Press the 'softdrink' button.
  - (iii) Do not insert any more money.
  - (iv) Get **two** softdrinks.

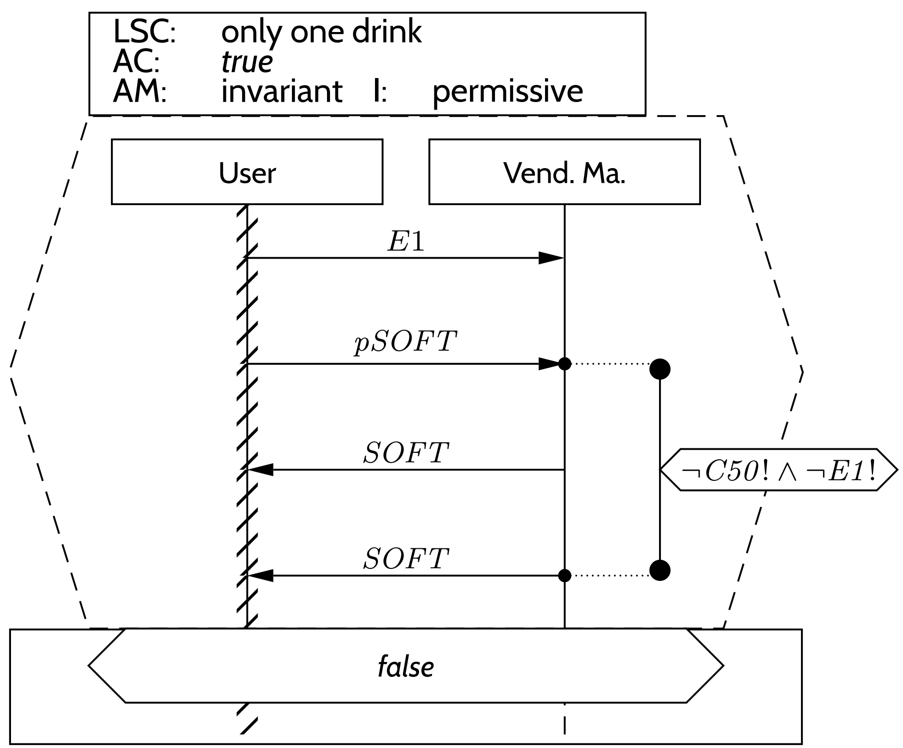


Pre-Charts

A **full LSC** $\mathscr{L} = (PC, MC, ac, am, \Theta_{\mathscr{L}})$ **actually** consists of

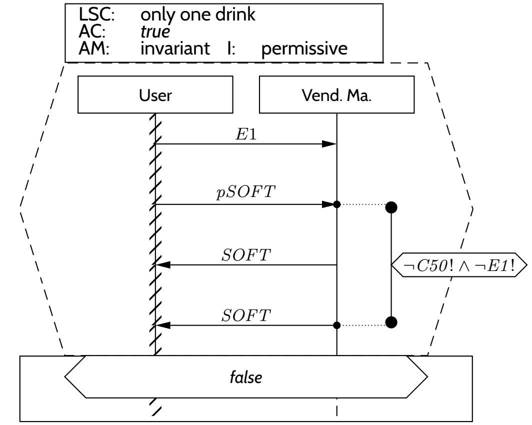
- **pre-chart** $PC = ((\mathcal{L}_P, \preceq_P, \sim_P), \mathcal{I}_P, \mathsf{Msg}_P, \mathsf{Cond}_P, \mathsf{LocInv}_P, \Theta_P)$ (poss. empty),
- **main-chart** $MC = ((\mathcal{L}_M, \preceq_M, \sim_M), \mathcal{I}_M, \mathsf{Msg}_M, \mathsf{Cond}_M, \mathsf{LocInv}_M, \Theta_M)$,
- **activation condition** $ac \in \Phi(\mathcal{C})$, and **mode** *am* $\in \{\text{initial}, \text{invariant}\}$,
- **strictness flag** $strict$, **chart mode existential** ($\Theta_{\mathscr{L}} = $ cold) or **universal** ($\Theta_{\mathscr{L}} = $ hot).

Concrete syntax:



Pre-Charts

A **full LSC** $\mathscr{L} = (PC, MC, ac, am, \Theta_{\mathscr{L}})$ **actually** consists of



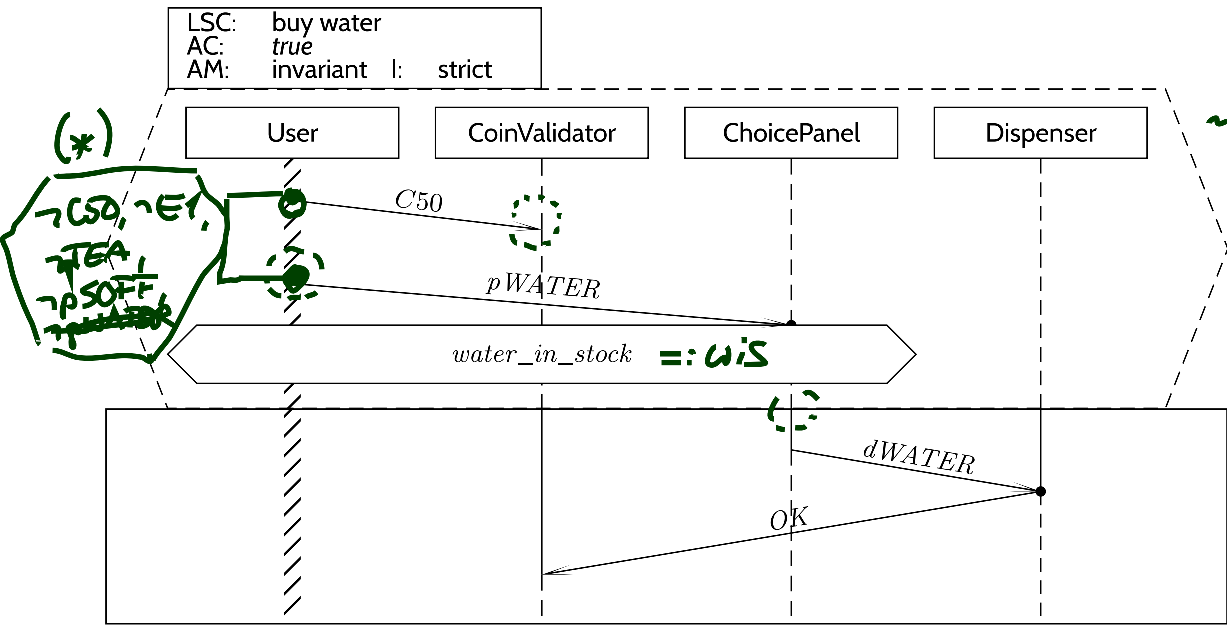
- **pre-chart** $PC = ((\mathcal{L}_P, \preceq_P, \sim_P), \mathcal{I}_P, \mathsf{Msg}_P, \mathsf{Cond}_P, \mathsf{LocInv}_P, \Theta_P)$ (poss. empty),
- **main-chart** $MC = ((\mathcal{L}_M, \preceq_M, \sim_M), \mathcal{I}_M, \mathsf{Msg}_M, \mathsf{Cond}_M, \mathsf{LocInv}_M, \Theta_M)$,
- **activation condition** $ac \in \Phi(\mathcal{C})$, and **mode** *am* $\in$ {initial, invariant},
- **strictness flag** $strict$, **chart mode existential** ($\Theta_{\mathscr{L}}$ = cold) or **universal** ($\Theta_{\mathscr{L}}$ = hot).

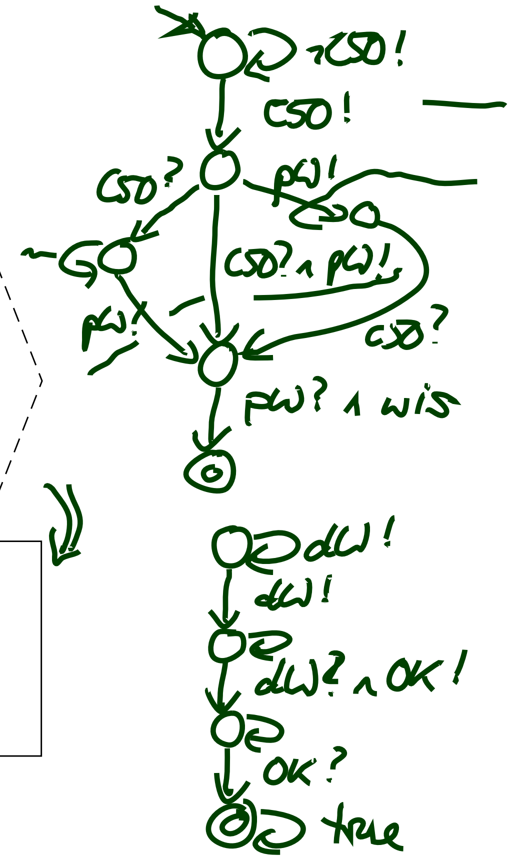
A **set of words** $W \subseteq (\mathcal{C} \to \mathbb{B})^\omega$ is **accepted** by $\mathscr{L}$, denoted by $W \models \mathscr{L}$, if and only if

	am = initial	am = invariant
$\Theta_{\mathscr{L}}$ = cold	$\exists w \in W \, \exists m \in \mathbb{N}_0 \bullet$ $\wedge\, w^0 \models ac \wedge \neg\psi_{exit}(C_0^P) \wedge \psi_{prog}(\emptyset, C_0^P)$ $\wedge\, w/1, \dots, w/m \in Lang(\mathcal{B}(PC))$ $\wedge\, w^{m+1} \models \neg\psi_{exit}(C_0^M)$ $\wedge\, w^{m+1} \models \psi_{prog}(\emptyset, C_0^M)$ $\wedge\, w/m+2 \in Lang(\mathcal{B}(MC))$	$\exists w \in W \, \exists k \leq m \in \mathbb{N}_0 \bullet$ $\wedge\, w^k \models ac \wedge \neg\psi_{exit}(C_0^P) \wedge \psi_{prog}(\emptyset, C_0^P)$ $\wedge\, w/k+1, \dots, w/m \in Lang(\mathcal{B}(PC))$ $\wedge\, w^{m+1} \models \neg\psi_{exit}(C_0^M)$ $\wedge\, w^{m+1} \models \psi_{prog}(\emptyset, C_0^M)$ $\wedge\, w/m+2 \in Lang(\mathcal{B}(MC))$
$\Theta_{\mathscr{L}}$ = hot	$\forall w \in W \, \forall m \in \mathbb{N}_0 \bullet$ $\wedge\, w^0 \models ac \wedge \neg\psi_{exit}(C_0^P) \wedge \psi_{prog}(\emptyset, C_0^P)$ $\wedge\, w/1, \dots, w/m \in Lang(\mathcal{B}(PC))$ $\wedge\, w^{m+1} \models \neg\psi_{exit}(C_0^M)$ $\implies w^{m+1} \models \psi_{prog}(\emptyset, C_0^M)$ $\wedge\, w/m+2 \in Lang(\mathcal{B}(MC))$	$\forall w \in W \, \forall k \leq m \in \mathbb{N}_0 \bullet$ $\wedge\, w^k \models ac \wedge \neg\psi_{exit}(C_0^P) \wedge \psi_{prog}(\emptyset, C_0^P)$ $\wedge\, w/k+1, \dots, w/m \in Lang(\mathcal{B}(PC))$ $\wedge\, w^{m+1} \models \neg\psi_{exit}(C_0^M)$ $\implies w^{m+1} \models \psi_{prog}(\emptyset, C_0^M)$ $\wedge\, w/m+2 \in Lang(\mathcal{B}(MC))$

where C_0^P and C_0^M are the minimal (or **instance heads**) cuts of pre- and main-chart.

Universal LSC: Example





(i) trivially satisfy 'buy water'

w = (C50!), (C50?), (pW!), (pW?, ¬wis), ...

(ii) w = (C50!), (C50!), (E1!), (E1?), (pSOFT!), (pSOFT?), (pWATER!), (pWATER?, wis), ✓ PC

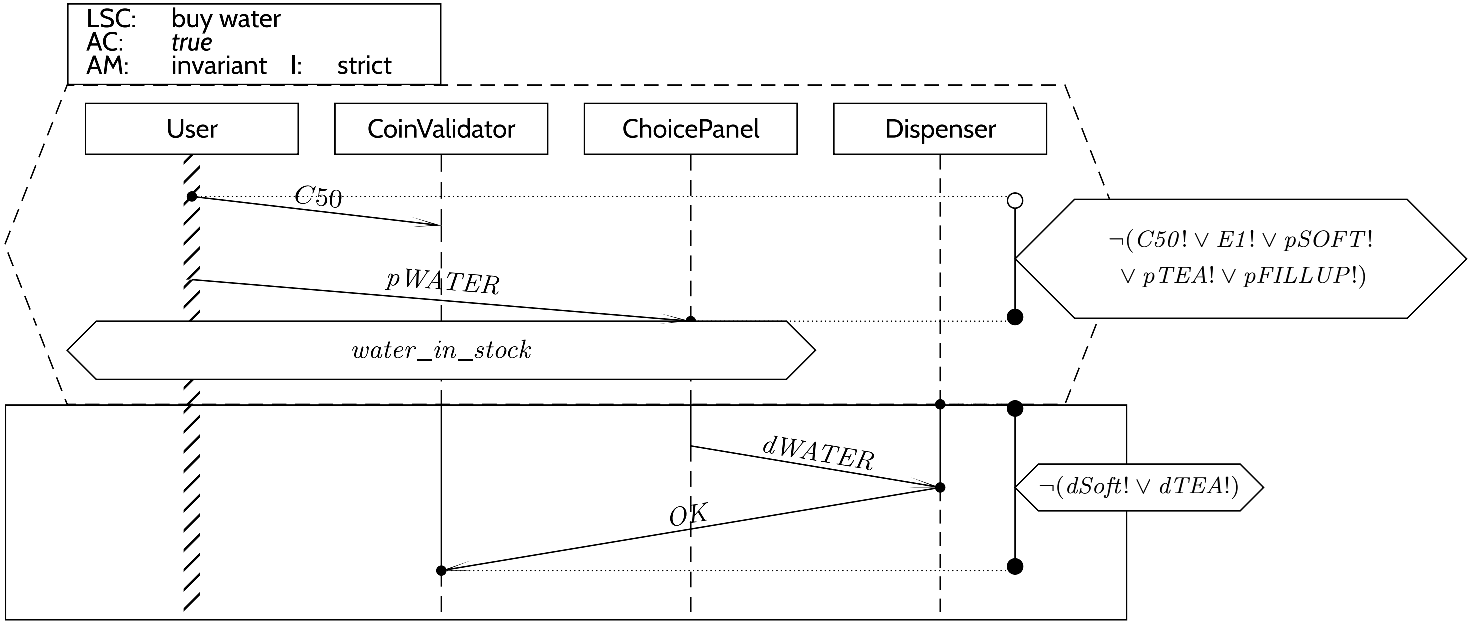
(dSOFT!), (dSOFT?, OK!), (OK?), (C50_chg!), ... ← silence ✗ PC with (*)

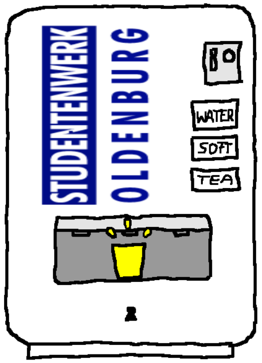
(iii) satisfy chart (non-trivially)

w = (C50!), (C50?), (pW!), (pW? ∧ wis), (dW!), (dW?, OK!), (OK?), ...

(distinguish: by legal exit in main-chart, or not)

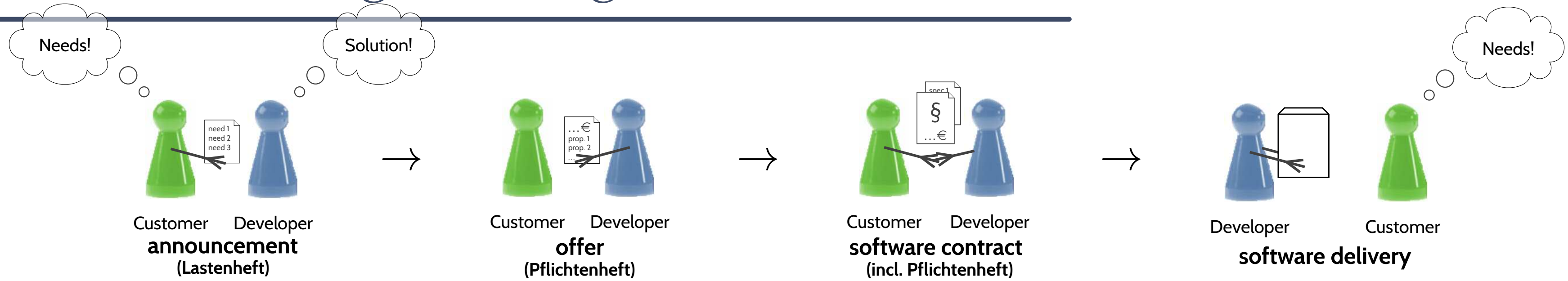
Universal LSC: Example





Requirements Engineering with Scenarios

Requirements Engineering with Scenarios



One quite effective approach:

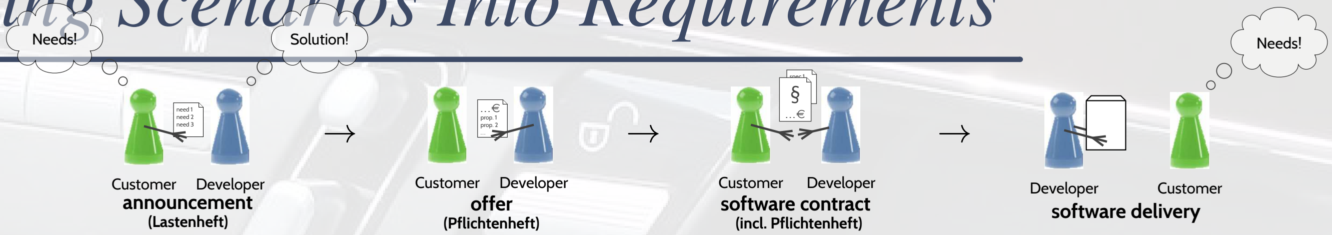
(i) **Approximate** the software requirements: ask for positive / negative **existential scenarios**.

(ii) **Refine** result into **universal scenarios** (and validate them with customer).

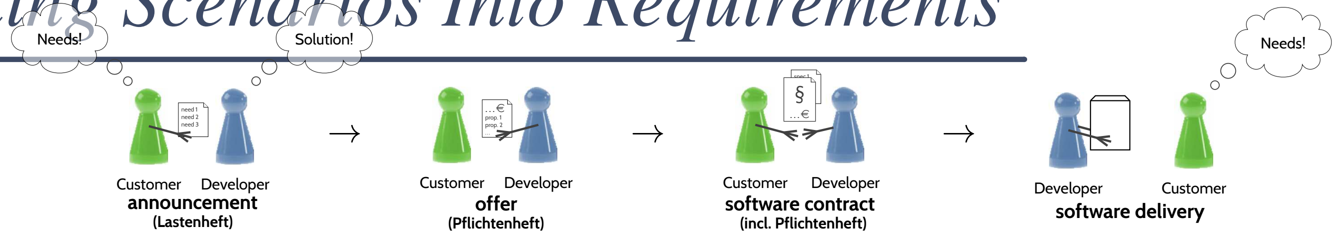
That is:

- Ask the customer to describe **example usages** of the desired system.
  In the sense of: **"If the system is not at all able to do this, then it's not what I want."**
  (→ positive use-cases, existential LSC)
- Ask the customer to describe behaviour that **must not happen** in the desired system.
  In the sense of: **"If the system does this, then it's not what I want."**
  (→ negative use-cases, LSC with pre-chart and hot-*false*)
- Investigate preconditions, side-conditions, exceptional cases and corner-cases.
  (→ extend use-cases, refine LSCs with conditions or local invariants)
- Generalise into universal requirements, e.g., **universal LSCs**.
- **Validate** with customer using new positive / negative scenarios.

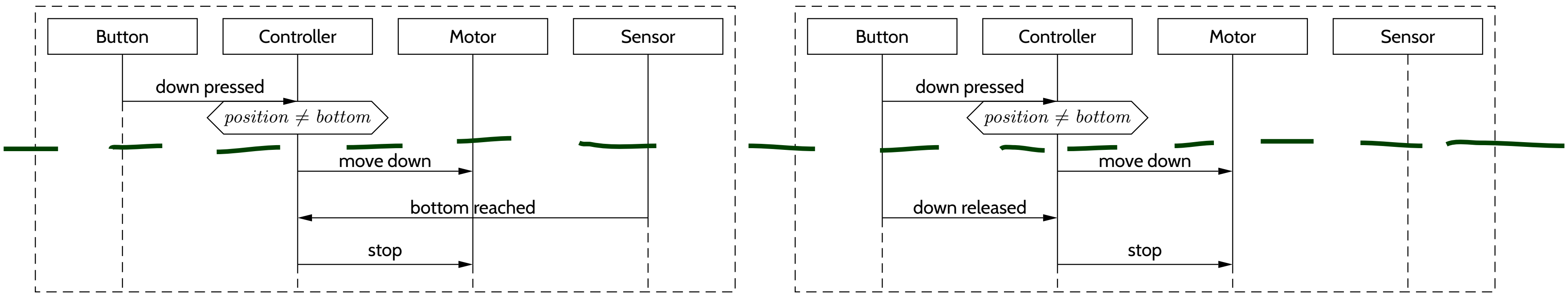
Strengthening Scenarios Into Requirements

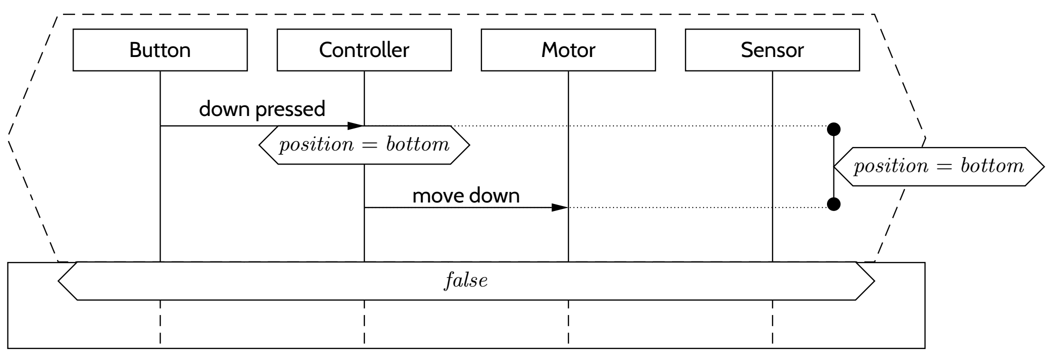


Strengthening Scenarios Into Requirements

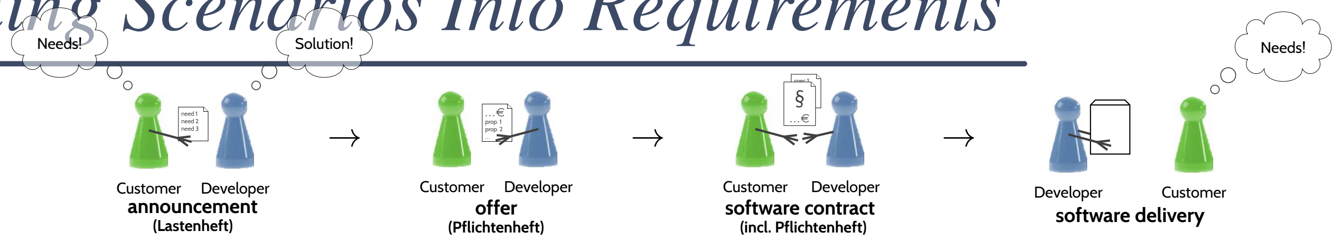


- **Ask customer for (pos./neg.) scenarios**, note down as existential LSCs:

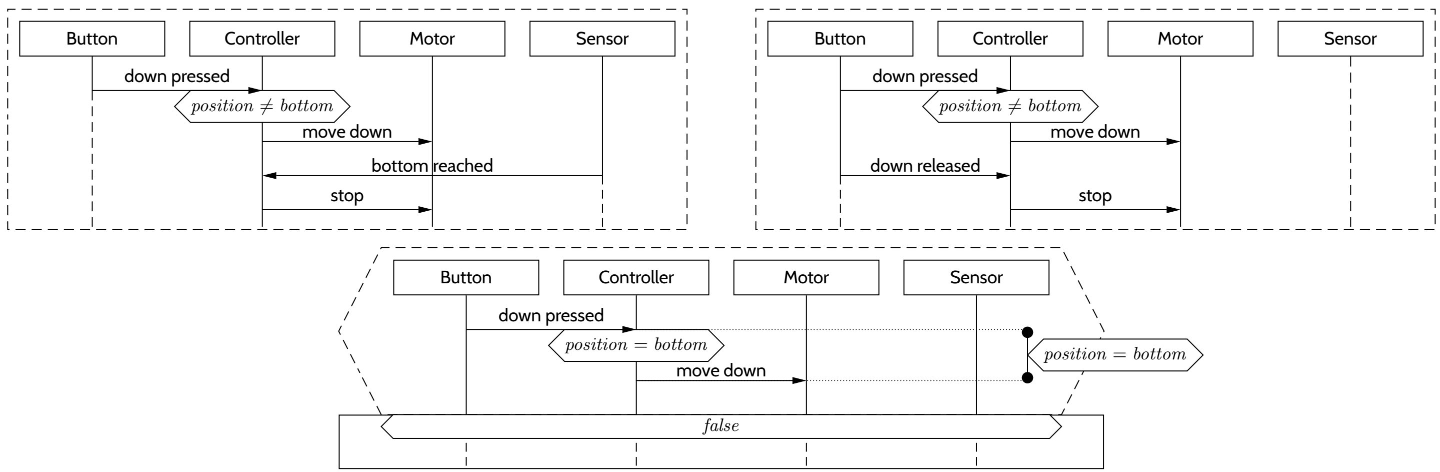




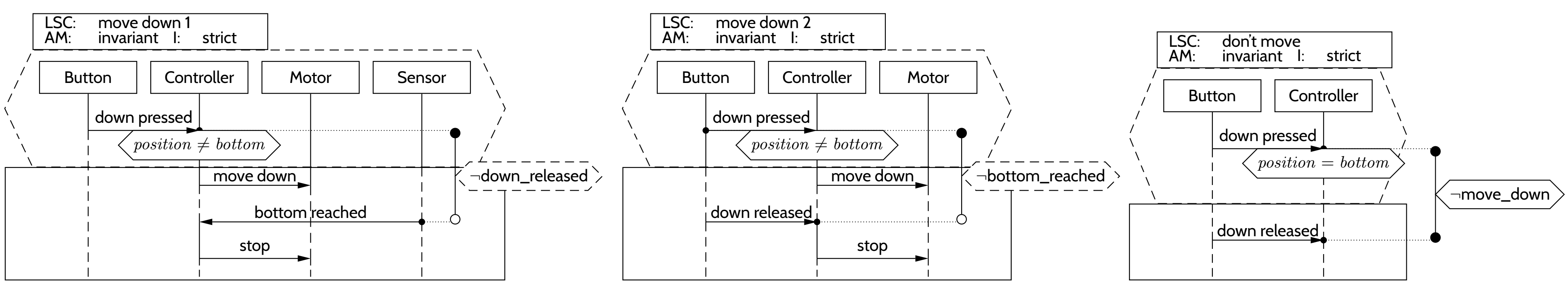
Strengthening Scenarios Into Requirements



- **Ask customer for (pos./neg.) scenarios**, note down as existential LSCs:



- **Strengthen into requirements**, note down as universal LSCs:



- **Re-Discuss** with customer using example words of the LSCs' language.

Analysing LSC Requirements

Requirements on Requirements Specifications

A **requirements specification** should be

- **correct** – it correctly represents the wishes/needs of the customer,
- **complete** – all requirements (existing in somebody's head, or a document, or ...) should be present,
- **relevant** – things which are not relevant to the project should not be constrained,
- **consistent**, **free of contradictions** – each requirement is compatible with all other requirements; otherwise the requirements are **not realisable**,
- **neutral**, **abstract** – a requirements specification does not constrain the realisation more than necessary,
- **traceable**, **comprehensible** – the sources of requirements are documented, requirements are uniquely identifiable,
- **testable**, **objective** – the final product can **objectively** be checked for satisfying a requirement.

- **Correctness** and **completeness** are defined **relative** to something which is usually only **in the customer's head**.

  $\rightarrow$ is is **difficult** to **be sure of correctness** and **completeness**.

- **"Dear customer, please tell me what is in your head!"** is in almost all cases **not a solution**!

  It's not unusual that even the customer does not precisely know...!

  For example, the customer may not be aware of contradictions due to technical limitations.

– 6 – 2016-05-12 – Sre –

12/37

Definition. [*LSC Consistency*] A set of LSCs $\{\mathscr{L}_1, \dots, \mathscr{L}_n\}$ is called **consistent** if and only if there exists a set of words W such that $\bigwedge_{i=1}^{n} W \models \mathit{Lang}(\mathscr{L}_i)$.

Content

- **Pre-Charts**
  - Semantics, once again
  - Requirements Engineering with scenarios
  - Strengthening scenarions into requirements
- **Software**, formally
  - Software specification
  - Requirements Engineering, formally
  - Software **implements** specification
- **LSCs vs. Software**
  - Software **implements** LSCs
  - **Scenarios and tests**
  - Play In/Play Out
- **Requirements Engineering Wrap-Up**

Software and Software Specification, formally

Software, formally

Definition. **Software** is a finite description S of a (possibly infinite) set $[\![S]\!]$ of (finite or infinite) computation paths of the form

$$\sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots$$

where

- $\sigma_i \in \Sigma$, $i \in \mathbb{N}_0$, is called **state** (or **configuration**), and
- $\alpha_i \in A$, $i \in \mathbb{N}_0$, is called **action** (or **event**).

The (possibly partial) function $[\![\cdot]\!] : S \mapsto [\![S]\!]$ is called **interpretation** of S.

Example: Software, formally

Software is a finite description S of a (possibly infinite) set $[\![S]\!]$ of (finite or infinite) computation paths of the form $\sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots$. σ_i: **state**/**configuration**; α_i: **action**/**event**.

- **Java Programs**.

```
1: public int f( int x, int y ) {
2:     x = x + y;
3:     y = x / 2;
4:     return y;
5: }
```

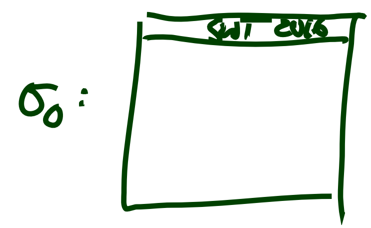
σ_0: pc = 1, x = 27, y = 13
↓ τ
σ_1 pc = 2, x = 40, y = 13
↓ τ
σ_2 pc = 3, x = 40, y = 20
↓ τ
σ_3 pc = ..

Example: Software, formally

Software is a finite description S of a (possibly infinite) set $[\![S]\!]$ of (finite or infinite) computation paths of the form $\sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots$. σ_i: **state**/**configuration**; α_i: **action**/**event**.

- **Java Programs**.
- **HTML**.

```
<html>
<head>
<title>SWT 2016</title>
</head>
<body/>
</html>
```

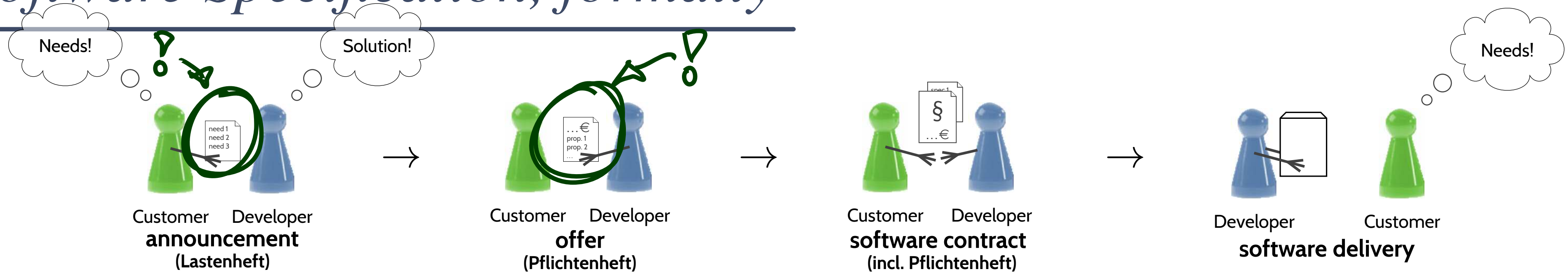


Example: Software, formally

Software is a finite description S of a (possibly infinite) set $[\![S]\!]$ of (finite or infinite) computation paths of the form $\sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots$. σ_i: **state**/**configuration**; α_i: **action**/**event**.

- **Java Programs**.
- **HTML**.
- **User's Manual.**
- **etc. etc.**

Software Specification, formally



Definition. A software specification is a finite description $\mathscr{S}$ of a (possibly infinite) set $[\![\mathscr{S}]\!]$ of softwares, i.e.

$$[\![\mathscr{S}]\!] = \{(S_1, [\![\cdot]\!]_1), \dots\}.$$

The (possibly partial) function $[\![\cdot]\!] : \mathscr{S} \mapsto [\![\mathscr{S}]\!]$ is called interpretation of $\mathscr{S}$.

Example: Software Specification

Alphabet:

- M – dispense cash only,
- C – return card only,
- $\genfrac{}{}{0pt}{}{M}{C}$ – dispense cash and return card.

- **Customer 1**: "don't care"

$$\mathscr{S}_1 = \left(M.C \,\middle|\, C.M \,\middle|\, \genfrac{}{}{0pt}{}{M}{C}\right)^\omega$$

- **Customer 2**: "you choose, but be consistent"

$$\mathscr{S}_2 = (M.C)^\omega \text{ or } (C.M)^\omega$$

- **Customer 3**: "consider human errors"

$$\mathscr{S}_3 = (C.M)^\omega$$

More Examples: Software Specification, formally

A **software specification** is a finite description $\mathscr{S}$ of a set $[\![\mathscr{S}]\!]$ of softwares $\{(S_1, [\![\cdot]\!]_1), \dots\}$.

- **Decision Tables.**

T: room ventilation		r_1	r_2	r_3
b	button pressed?	$\times$	$\times$	$-$
off	ventilation off?	$\times$	$-$	$*$
on	ventilation on?	$-$	$\times$	$*$
go	start ventilation	$\times$	$-$	$-$
stop	stop ventilation	$-$	$\times$	$-$

$\sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \dots$

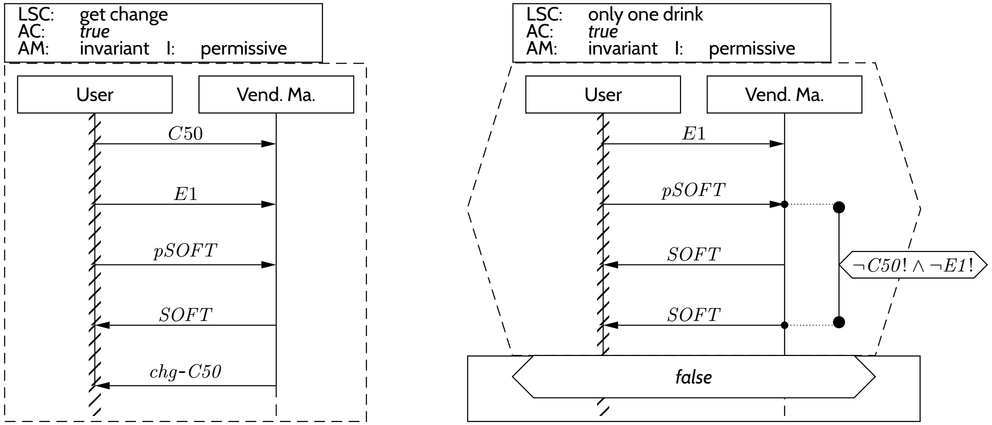
if $\sigma_i \models \mathcal{F}_{pre}(r_j)$

then $\alpha_{i+1} \models \mathcal{F}_{eff}(r_j)$

More Examples: Software Specification, formally

A **software specification** is a finite description $\mathscr{S}$ of a set $[\![\mathscr{S}]\!]$ of softwares $\{(S_1, [\![\cdot]\!]_1), \dots\}$.

- **Decision Tables**.
- **LSCs**.



More Examples: Software Specification, formally

A **software specification** is a finite description $\mathscr{S}$ of a set $[\![\mathscr{S}]\!]$ of softwares $\{(S_1, [\![\cdot]\!]_1), \dots\}$.

- **Decision Tables**.
- **LSCs**.
- **Global Invariants**.

$$x \geq 0$$

More Examples: Software Specification, formally

A **software specification** is a finite description $\mathscr{S}$ of a set $[\![\mathscr{S}]\!]$ of softwares $\{(S_1, [\![\cdot]\!]_1), \dots\}$.

- **Decision Tables**.
- **LSCs**.
- **Global Invariants**.
- **State Machines**.

$\rightarrow$ later

More Examples: Software Specification, formally

A **software specification** is a finite description $\mathscr{S}$ of a set $[\![\mathscr{S}]\!]$ of softwares $\{(S_1, [\![\,\cdot\,]\!]_1), \dots\}$.

- **Decision Tables**.
- **LSCs**.
- **Global Invariants**.
- **State Machines**.
- **Java Programs**.

```
public int f( int x, int y ) {
    x = x + y;
    y = x / 2;
    return y;
}
```

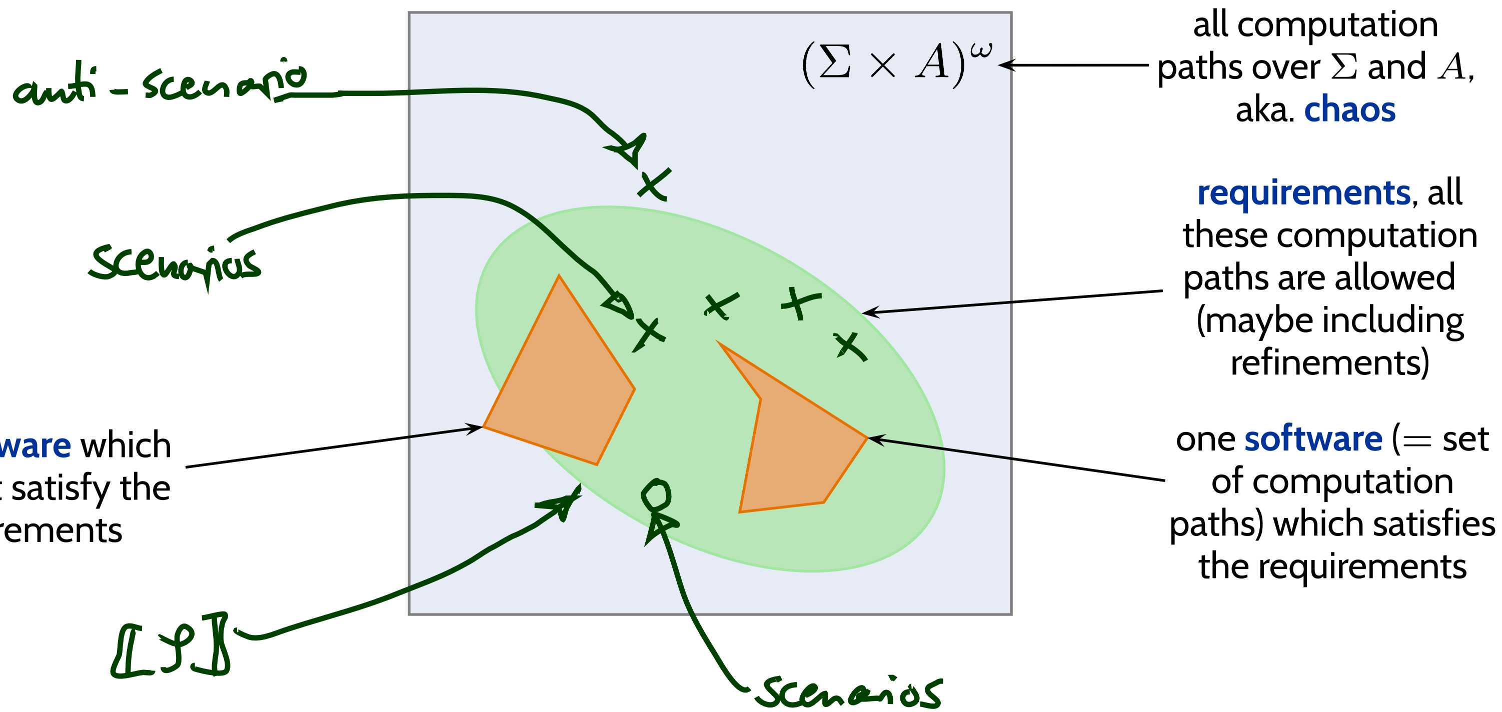
$[\![\mathscr{S}]\!] = \{(S, [\![\cdot]\!])\}$

More Examples: Software Specification, formally

A **software specification** is a finite description $\mathscr{S}$ of a set $[\![\mathscr{S}]\!]$ of softwares $\{(S_1, [\![\cdot]\!]_1), \ldots\}$.

- **Decision Tables**.
- **LSCs**.
- **Global Invariants**.
- **State Machines**.
- **Java Programs**.
- **User's Manual.**
- **etc. etc.**

The Requirements Engineering Problem Formally



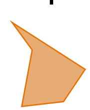
- **Requirements engineering**:

  Describe/specify the set of the **allowed** softwares as $\mathscr{S}$.

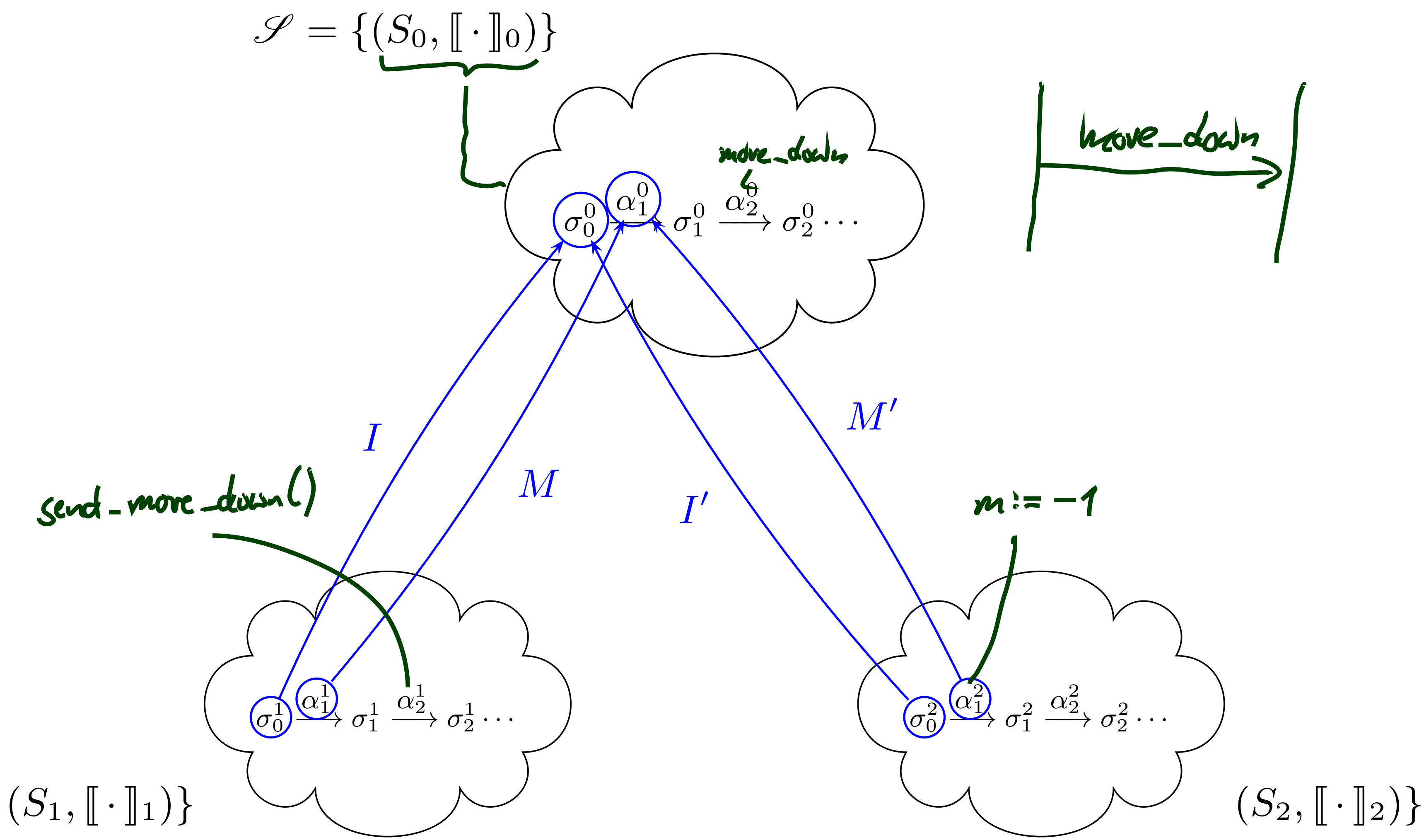
  Note: what is not constrained is allowed, usually!

- **Software development**:

  Create one software S whose computation paths $[\![S]\!]$ are all allowed, i.e. $[\![S]\!] \in [\![\mathscr{S}]\!]$.

  - **Note**: different programs in different programming languages may describe the same $[\![S]\!]$.
  - **Often allowed**: any **refinement** of  ($\rightarrow$ in a minute; e.g. allow intermediate transitions).

Software Specification vs. Software



S_1 **implements** $\mathscr{S}$ via I and M

S_2 **implements** $\mathscr{S}$ via I' and M'

LSCs vs. Software

LSCs as Software Specification

A software S is called **compatible** with LSC $\mathscr{L}$ over $\mathcal{C}$ and $\mathcal{E}$ is if and only if

- $\Sigma = (\mathcal{C} \to \mathbb{B})$, i.e. the **states** are valuations of the conditions in $\mathcal{C}$,
- $A \subseteq \mathcal{E}_{!?}$, i.e. the **events** are of the form $E!$, $E?$ (viewed as a valuation of $E!$, $E?$).

A computation path $\pi = \sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots \in [\![S]\!]$ of software S **induces** the word

$$w(\pi) = (\sigma_0 \cup \alpha_1), (\sigma_1 \cup \alpha_2), (\sigma_2 \cup \alpha_3), \dots,$$

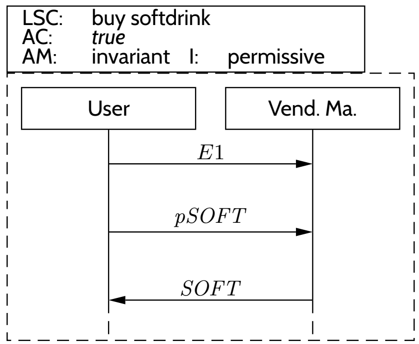
we use W_S to denote the set of words induced by $[\![S]\!]$.

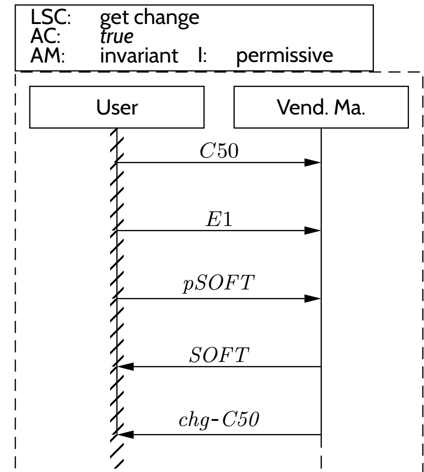
We say software S **satisfies** LSC $\mathscr{L}$ (without pre-chart), denoted by $S \models \mathscr{L}$, if and only if

$\Theta_{\mathscr{L}}$	$am =$ initial	$am =$ invariant
cold	$\exists\, w \in W_S \bullet w^0 \models ac \wedge \neg\psi_{exit}(C_0)$ $\wedge\, w^0 \models \psi_{prog}(\emptyset, C_0) \wedge w/1 \in Lang(\mathcal{B}(\mathscr{L}))$	$\exists\, w \in W_S\, \exists\, k \in \mathbb{N}_0 \bullet w^k \models ac \wedge \neg\psi_{exit}(C_0)$ $\wedge\, w^k \models \psi_{prog}(\emptyset, C_0) \wedge w/k+1 \in Lang(\mathcal{B}(\mathscr{L}))$
hot	$\forall\, w \in W_S \bullet w^0 \models ac \wedge \neg\psi_{exit}(C_0)$ $\implies w^0 \models \psi_{prog}(\emptyset, C_0) \wedge w/1 \in Lang(\mathcal{B}(\mathscr{L}))$	$\forall\, w \in W_S\, \forall\, k \in \mathbb{N}_0 \bullet w^k \models ac \wedge \neg\psi_{exit}(C_0)$ $\implies w^k \models \psi^{\text{Cond}}_{\text{hot}}(\emptyset, C_0) \wedge w/k+1 \in Lang(\mathcal{B}(\mathscr{L}))$

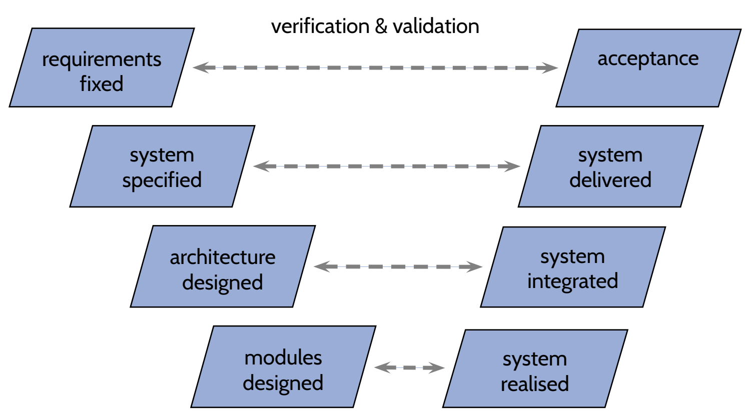
Software S satisfies **a set of** LSCs $\mathscr{L}_1, \dots, \mathscr{L}_n$ if and only if $S \models \mathscr{L}_i$ for all $1 \leq i \leq n$.

How to Prove that a Software Satisfies an LSC?

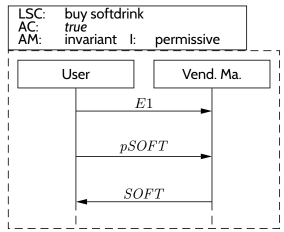


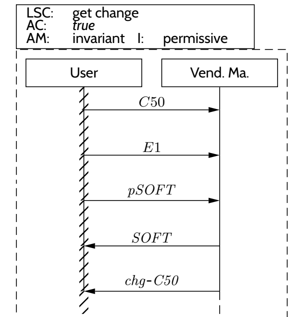


- Software S satisfies **existential** LSC $\mathscr{L}$ if there **exists** $\pi \in [\![S]\!]$ such that $\mathscr{L}$ accepts $w(\pi)$. Prove $S \models \mathscr{L}$ by demonstrating π.
- Note: **Existential** LSCs* may hint at **test-cases** for the **acceptance test**!
  ($*$: as well as (positive) scenarios in general, like use-cases)

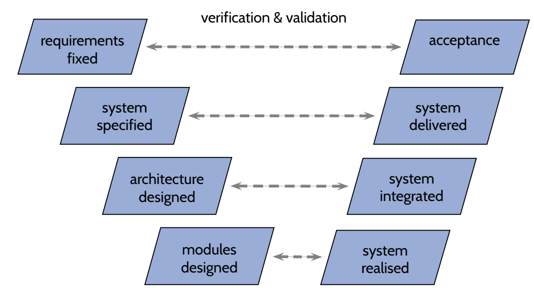


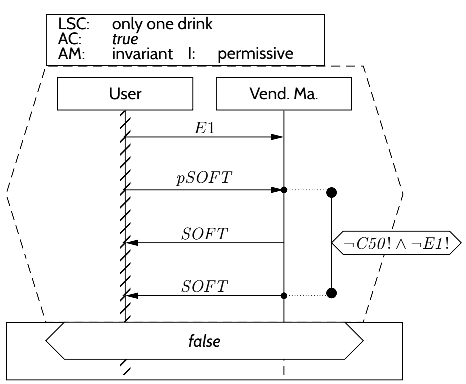
How to Prove that a Software Satisfies an LSC?

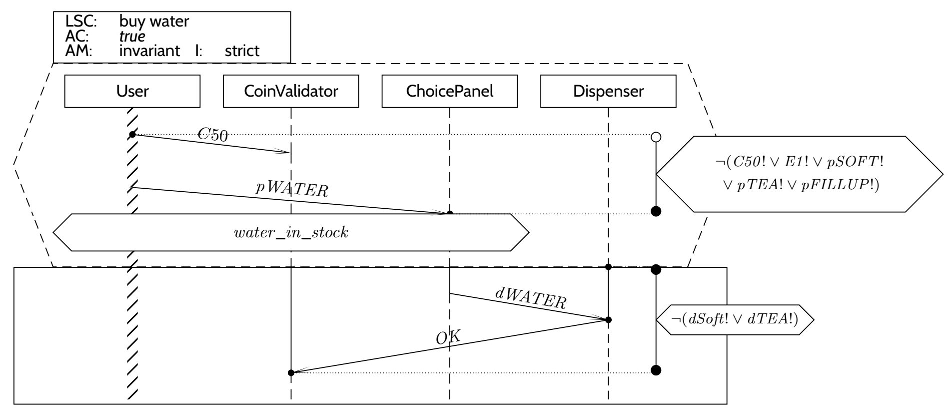




- Software S satisfies **existential** LSC $\mathscr{L}$ if there **exists** $\pi \in [\![S]\!]$ such that $\mathscr{L}$ accepts $w(\pi)$. Prove $S \models \mathscr{L}$ by demonstrating π.
- Note: **Existential** LSCs* may hint at **test-cases** for the **acceptance test**!
  ($*$: as well as (positive) scenarios in general, like use-cases)







- **Universal** LSCs (and negative/anti-scenarios!) in general need an **exhaustive analysis**!
  (Because they require that the software **never ever** exhibits the unwanted behaviour.)

  Prove $S \not\models \mathscr{L}$ by demonstrating one π such that $w(\pi)$ **is not accepted** by $\mathscr{L}$.

Pushing It Even Further

(Harel and Marelly, 2003)

Tell Them What You've Told Them...

- **Live Sequence Charts** (if well-formed)
  - have an abstract syntax: instance lines, messages, conditions, local invariants; mode: hot or cold.
- From an abstract syntax, mechanically construct its **TBA**.
- **Pre-charts** allow us to
  - specify **anti-scenarios** ("this must not happen"),
  - contrain **activation**.
- An **LSC** is **satisfied** by a software S if and only if
  - **existential** (cold):
    - **there is a word** induced by a computation path of S
    - which is **accepted** by the LSC's pre/main-chart TBA.
  - **universal** (hot):
    - **all words** induced by the computation paths of S
    - are **accepted** by the LSC's pre/main-chart TBA.
- **Method**:
  - discuss (anti-)scenarios with customer,
  - generalise into universal LSCs and re-validate.

Requirements Engineering Wrap-Up

Topic Area Requirements Engineering: Content

VL 6	• **Introduction**
	• **Requirements Specification**
	– Desired Properties
	– Kinds of Requirements
	– Analysis Techniques
⋮	• **Documents**
	– Dictionary, Specification
	• **Specification Languages**
	– Natural Language
	– Decision Tables
VL 7	– Syntax, Semantics
⋮	– Completeness, Consistency, …
VL 8	– Scenarios
⋮	– User Stories, Use Cases
	– Live Sequence Charts
VL 9	– Syntax, Semantics
⋮	
VL 10	• Definition: **Software** & **SW Specification**
⋮	• **Wrap-Up**

Example: Software Specification

Alphabet:

- M – dispense cash only,
- C – return card only,
- $\substack{M \\ C}$ – dispense cash and return card.

- **Customer 1**: "don't care"

$$\mathscr{S}_1 = \left(M.C \middle| C.M \middle| \substack{M \\ C} \right)^\omega$$

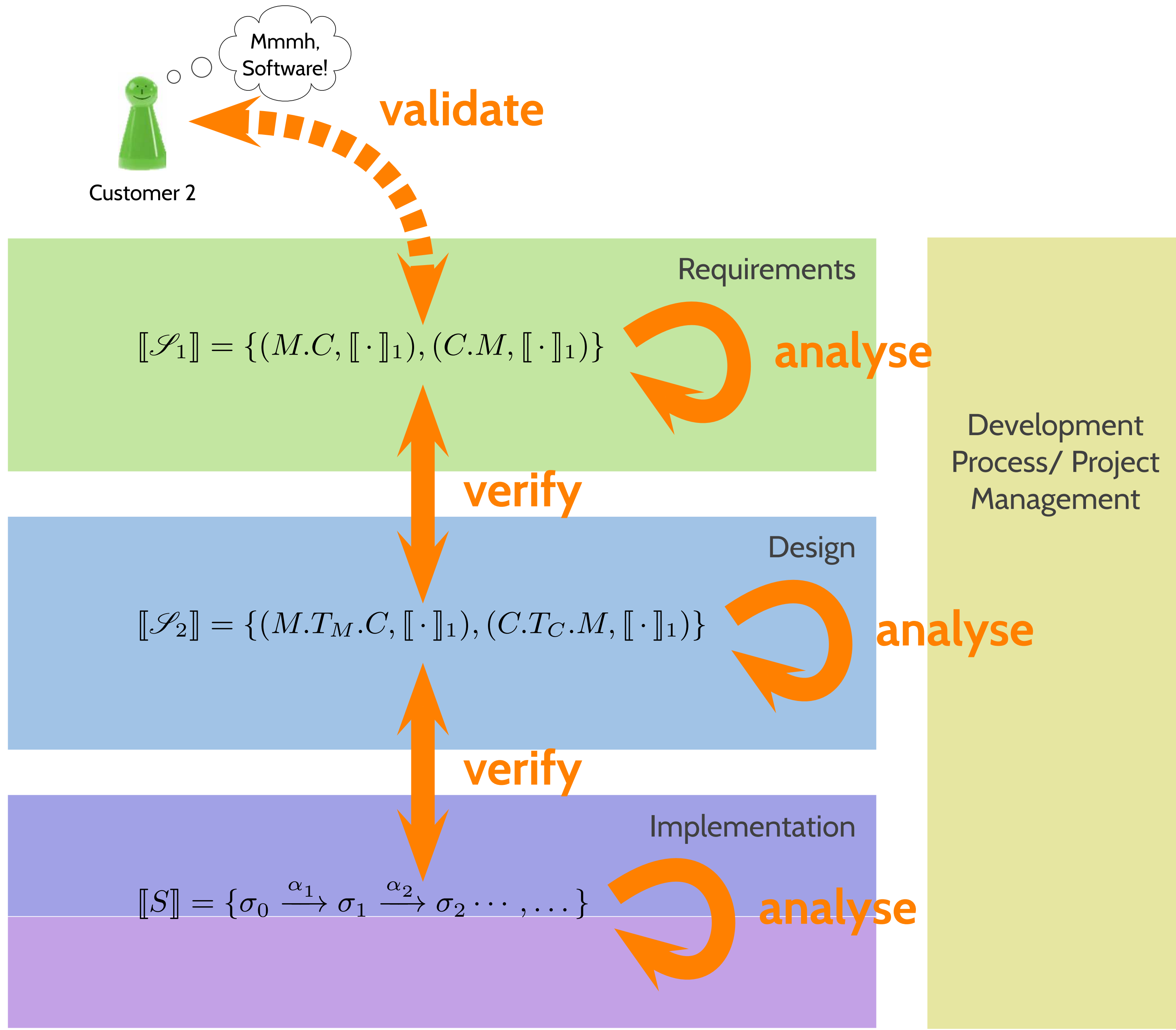
- **Customer 2**: "you choose, but be consistent"

$$\mathscr{S}_2 = (M.C)^\omega \text{ or } (C.M)^\omega$$

- **Customer 3**: "consider human errors"

$$\mathscr{S}_3 = (C.M)^\omega$$

Formal Methods in the Software Development Process



Tell Them What You've Told Them. . .

- A **Requirements Specification** should be
  - correct, complete, relevant, consistent, neutral, traceable, objective.
- **Requirements Representations** should be
  - easily understandable, precise, easily maintainable, easily usable.
- **Languages / Notations** for Requirements Representations:
  - Natural Language Patterns
  - **Decision Tables**
  - User Stories
  - Use Cases
  - **Live Sequence Charts**
- **Formal representations**
  - can be very **precise**, objective, testable,
  - can be **analysed** for, e.g., completeness, consistency
  - can be **verified** against a formal design description.

(Formal) inconsistency of, e.g., a decision table **hints at** inconsistencies in the requirements.

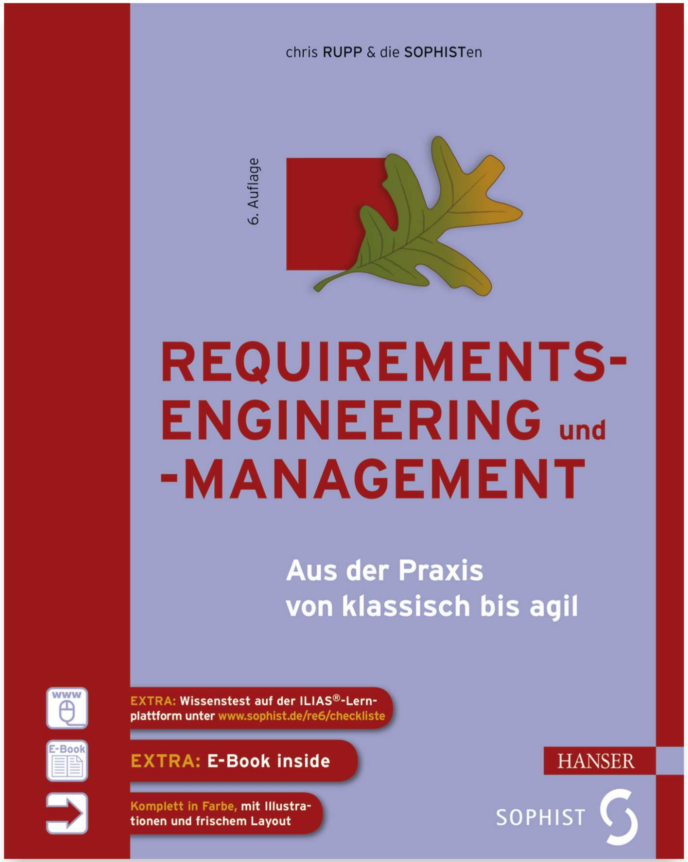
Requirements Analysis in a Nutshell

- Customers **may not know** what they want.
  - That's in general not their "fault"!
  - Care for **tacit** requirements.
  - Care for **non-functional** requirements / constraints.
- For **requirements elicitation**, consider starting with
  - **scenarios** ("positive use case") and **anti-scenarios** ("negative use case")

  and elaborate corner cases.

  Thus, **use cases** can be **very useful** – use case **diagrams** not so much.
- Maintain a **dictionary** and high-quality descriptions.
- Care for **objectiveness** / **testability** early on.

  Ask for each requirements: what is the **acceptance test**?
- **Use formal notations**
  - to **fully understand requirements** (precision),
  - for **requirements analysis** (completeness, etc.),
  - to communicate with your developers.
- If in doubt, **complement** (formal) **diagrams with text**
  (as safety precaution, e.g., in lawsuits).



(Rupp and die SOPHISTen, 2014)

References

References

Harel, D. and Marelly, R. (2003). *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer-Verlag.

Ludewig, J. and Lichter, H. (2013). *Software Engineering*. dpunkt.verlag, 3. edition.

Rupp, C. and die SOPHISTen (2014). *Requirements-Engineering und -Management*. Hanser, 6th edition.