Softwaretechnik / Software-Engineering

Lecture 9: Live Sequence Charts & RE Wrap-Up

2019-06-03

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

Albert-Ludwigs-Universität Freiburg, Germany

Topic Area Requirements Engineering: Content

Vocabulary

Techniques

informal

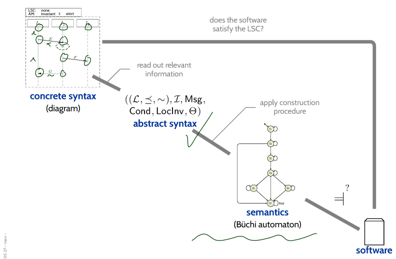
semi-formal

formal

simple DT

complex

The Plan: A Formal Semantics for a Visual Formalism



28/46

Content

LSC Semantics: TBA Construction

LSC Semantics: It's in the Cuts!

Definition. Let $((\mathcal{L}, \preceq, \sim), \mathcal{I}, \mathsf{Msg}, \mathsf{Cond}, \mathsf{LocInv}, \Theta)$ be an LSC body.

A non-empty set $\emptyset \neq C \subseteq \mathcal{L}$ is called a **cut** of the LSC body iff C

- is **downward closed**, i.e.
$$\forall l, l' \in \mathcal{L} \bullet l' \in C \wedge l \preceq l' \implies l \in C,$$
- is **closed** under **simultaneity**, i.e.
$$\forall l, l' \in \mathcal{L} \bullet l' \in C \wedge l \sim l' \implies l \in C, \text{ and}$$
- comprises at least **one location per instance line**, i.e.
$$\forall I \in \mathcal{I} \bullet C \cap I \neq \emptyset.$$

The temperature function is extended to cuts as follows:

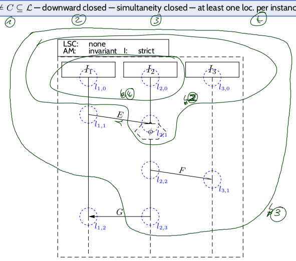
$$\Theta(C) = \begin{cases} \text{hot} & \text{if } \exists l \in C \bullet (\nexists l' \in C \bullet l \prec l') \wedge \Theta(l) = \text{hot} \\ \text{cold} & \text{otherwise} \end{cases}$$

that is, C is **hot** if and only if at least one of its maximal elements is hot.

39/46

Cut Examples

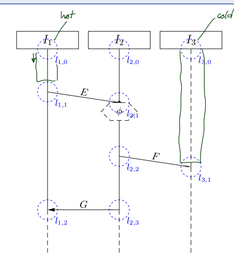
$\emptyset \neq C \subseteq \mathcal{L}$ — downward closed — simultaneity closed — at least one loc. per instance line



40/46

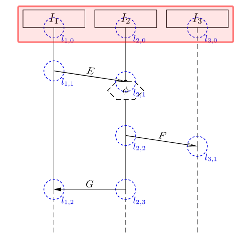
Cut Examples

$\emptyset \neq C \subseteq \mathcal{L}$ — downward closed — simultaneity closed — at least one loc. per instance line



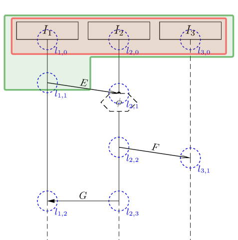
Cut Examples

$\emptyset \neq C \subseteq \mathcal{L}$ — downward closed — simultaneity closed — at least one loc. per instance line



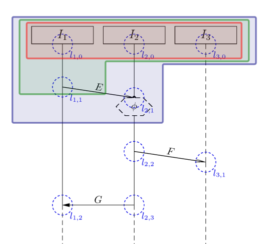
Cut Examples

$\emptyset \neq C \subseteq \mathcal{L}$ — downward closed — simultaneity closed — at least one loc. per instance line



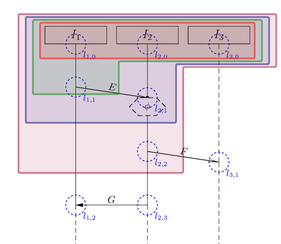
Cut Examples

$\emptyset \neq C \subseteq \mathcal{L}$ — downward closed — simultaneity closed — at least one loc. per instance line



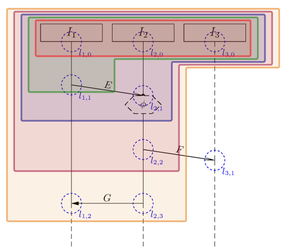
Cut Examples

$\emptyset \neq C \subseteq \mathcal{L}$ — downward closed — simultaneity closed — at least one loc. per instance line



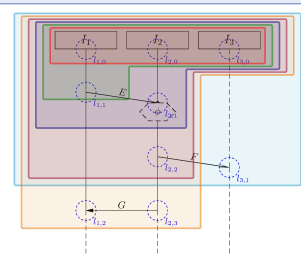
Cut Examples

$\emptyset \neq C \subseteq \mathcal{L}$ — downward closed — simultaneity closed — at least one loc. per instance line



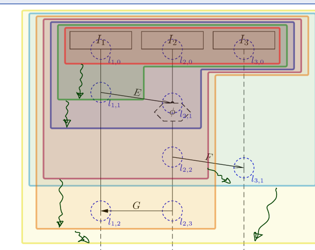
Cut Examples

$\emptyset \neq C \subseteq \mathcal{L}$ — downward closed — simultaneity closed — at least one loc. per instance line



Cut Examples

$\emptyset \neq C \subseteq \mathcal{L}$ — downward closed — simultaneity closed — at least one loc. per instance line



A Successor Relation on Cuts

The partial order "$\preceq$" and the simultaneity relation "$\sim$" of locations induce a **direct successor relation** on cuts of an LSC body as follows:

Definition.
Let $C \subseteq \mathcal{L}$ bet a cut of LSC body $((\mathcal{L}, \preceq, \sim), \mathcal{I}, \mathsf{Msg}, \mathsf{Cond}, \mathsf{LocInv}, \Theta)$.

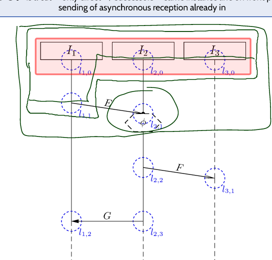
A set $\emptyset \neq \mathcal{F} \subseteq \mathcal{L}$ of locations is called **fired-set** $\mathcal{F}$ **of cut** C if and only if

- $C \cap \mathcal{F} = \emptyset$ and $C \cup \mathcal{F}$ is a cut, i.e. $\mathcal{F}$ is closed under simultaneity,
- all locations in $\mathcal{F}$ are direct $\prec$-successors of the front of C, i.e.
$$\forall l \in \mathcal{F}\, \exists l' \in C \bullet l' \prec l \wedge (\nexists l'' \in \mathcal{L} \bullet l' \prec l'' \prec l),$$
- locations in $\mathcal{F}$ that lie on the same instance line are pairwise unordered, i.e.
$$\forall l \neq l' \in \mathcal{F} \bullet (\exists I \in \mathcal{I} \bullet \{l, l'\} \subseteq I) \implies l \not\preceq l' \wedge l' \not\preceq l,$$
- for each asynchronous message reception in $\mathcal{F}$, the corresponding sending is already in C,
$$\forall (l, E, l') \in \mathsf{Msg} \bullet l' \in \mathcal{F} \implies l \in C.$$

The cut $C' = C \cup \mathcal{F}$ is called **direct successor of** C **via** $\mathcal{F}$, denoted by $C \rightsquigarrow_{\mathcal{F}} C'$.

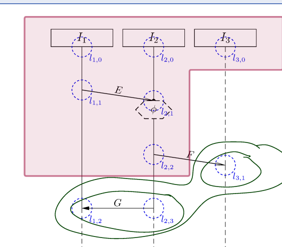
Successor Cut Example

$C \cap \mathcal{F} = \emptyset$ — $C \cup \mathcal{F}$ is a cut — only direct $\prec$-successors — same instance line on front pairwise unordered — sending of asynchronous reception already in

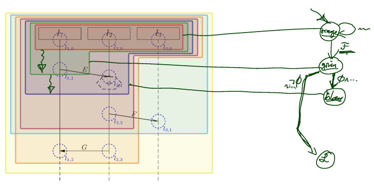


Successor Cut Example

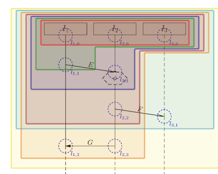
$C \cap \mathcal{F} = \emptyset$ — $C \cup \mathcal{F}$ is a cut — only direct $\prec$-successors — same instance line on front pairwise unordered — sending of asynchronous reception already in

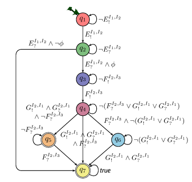


Language of LSC Body: Example



Language of LSC Body: Example





The TBA $\mathcal{B}(\mathscr{L})$ of LSC $\mathscr{L}$ over $\mathcal{C}$ and $\mathcal{E}$ is $(\mathcal{C}_\mathcal{B}, Q, q_{ini}, \to, Q_F)$ with

- $\mathcal{C}_\mathcal{B} = \mathcal{C} \,\dot\cup\, \mathcal{E}^\mathcal{I}_{!?}$, where $\mathcal{E}^\mathcal{I}_{!?} = \{E_!^{i,j}, E_?^{i,j} \mid E \in \mathcal{E}, i, j \in \mathcal{I}\}$,
- Q is **the set of cuts** of $\mathscr{L}$, q_{ini} is the **instance heads** cut,
- $\to$ consists of **loops**, **progress transitions** (from $\rightsquigarrow_\mathcal{F}$), and **legal exits** (cold cond./local inv.),
- $Q_F = \{C \in Q \mid \Theta(C) = \text{cold} \vee C = \mathcal{L}\}$ is the set of cold cuts and the maximal cut.

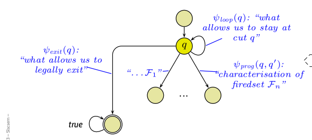
TBA Construction Principle

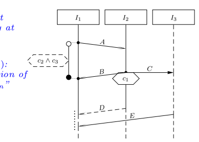
Recall: The TBA $\mathcal{B}(\mathscr{L})$ of LSC $\mathscr{L}$ is $(\mathcal{C}, Q, q_{ini}, \to, Q_F)$ with

- Q is **the set of cuts** of $\mathscr{L}$, q_{ini} is the **instance heads** cut,
- $\mathcal{C}_\mathcal{B} = \mathcal{C} \,\dot\cup\, \mathcal{E}^\mathcal{I}_{!?}$,
- $\to$ consists of **loops**, **progress transitions** (from $\rightsquigarrow_\mathcal{F}$), and **legal exits** (cold cond./local inv.),
- $Q_F = \{C \in Q \mid \Theta(C) = \text{cold} \vee C = \mathcal{L}\}$ is the set of cold cuts.

So in the following, we "only" need to construct the transitions' labels:

$$\to = \{(q, \psi_{loop}(q), q) \mid q \in Q\} \cup \{(q, \psi_{prog}(q, q'), q') \mid q \rightsquigarrow_\mathcal{F} q'\} \cup \{(q, \psi_{exit}(q), \mathcal{L}) \mid q \in Q\}$$





TBA Construction Principle

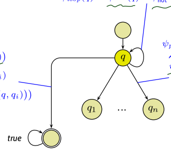
"Only" construct the transitions' labels:

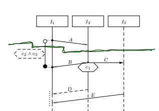
$$\to = \{(q, \psi_{loop}(q), q) \mid q \in Q\} \cup \{(q, \psi_{prog}(q, q'), q') \mid q \rightsquigarrow_\mathcal{F} q'\} \cup \{(q, \psi_{exit}(q), \mathcal{L}) \mid q \in Q\}$$

$$\psi_{loop}(q) = \underbrace{\psi^{\mathsf{Msg}}(q) \wedge \psi^{\mathsf{LocInv}}_{\mathsf{hot}}(q)}_{=:\psi^{\mathsf{hot}}_{loop}(q)} \wedge \psi^{\mathsf{LocInv}}_{\mathsf{cold}}(q)$$

$$\psi_{exit}(q) = \big(\psi^{\mathsf{hot}}_{loop}(q) \wedge \neg\psi^{\mathsf{LocInv}}_{\mathsf{cold}}(q)\big) \vee \bigvee_{1 \le i \le n} \big(\psi^{\mathsf{hot}}_{prog}(q, q_i) \wedge \big(\neg\psi^{\mathsf{LocInv},\bullet}_{\mathsf{cold}}(q, q_i) \vee \neg\psi^{\mathsf{Cond}}_{\mathsf{cold}}(q, q_i)\big)\big)$$

$$\psi_{prog}(q, q_n) = \underbrace{\psi^{\mathsf{Msg}}(q, q_n) \wedge \psi^{\mathsf{Cond}}_{\mathsf{hot}}(q, q_n) \wedge \psi^{\mathsf{LocInv},\bullet}_{\mathsf{hot}}(q, q_n)}_{=:\psi^{\mathsf{hot}}_{prog}(q, q_n)} \wedge \psi^{\mathsf{Cond}}_{\mathsf{cold}}(q, q_n) \wedge \psi^{\mathsf{LocInv},\bullet}_{\mathsf{cold}}(q, q_n)$$





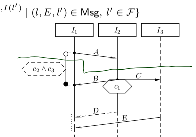
Loop Condition

$$\psi_{loop}(q) = \psi^{\mathsf{Msg}}(q) \wedge \psi^{\mathsf{LocInv}}_{\mathsf{hot}}(q) \wedge \psi^{\mathsf{LocInv}}_{\mathsf{cold}}(q)$$

- $\psi^{\mathsf{Msg}}(q) = \neg \bigvee_{1 \le i \le n, \psi \in \mathsf{Msg}(q_i \setminus q)} \psi \wedge \underbrace{\Big(strict \implies \bigwedge_{\psi \in \mathcal{E}^\mathcal{I}_{!?} \cap \mathsf{Msg}(\mathcal{L})} \neg\psi\Big)}_{=:\psi_{\mathsf{strict}}(q)}$
- $\psi^{\mathsf{LocInv}}_\theta(q) = \bigwedge_{\ell = (l, \iota, \phi, l', \iota') \in \mathsf{LocInv},\ \Theta(\ell) = \theta,\ \ell \text{ active at } q} \phi$

  A location l is called **front location** of cut C if and only if $\nexists\, l' \in C \bullet l \prec l'$.

  Local invariant $(l_0, \iota_0, \phi, l_1, \iota_1)$ is **active** at cut (!) q
  if and only if $l_0 \preceq l \prec l_1$ for some front location l of cut q or $l = l_1 \wedge \iota_1 = \bullet$.
- $\mathsf{Msg}(\mathcal{F}) = \{E_!^{I(l), I(l')} \mid (l, E, l') \in \mathsf{Msg},\ l \in \mathcal{F}\} \cup \{E_?^{I(l), I(l')} \mid (l, E, l') \in \mathsf{Msg},\ l' \in \mathcal{F}\}$
- $\mathsf{Msg}(\mathcal{F}_1, \ldots, \mathcal{F}_n) = \bigcup_{1 \le i \le n} \mathsf{Msg}(\mathcal{F}_i)$



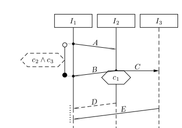
Progress Condition

$$\psi^{\mathsf{hot}}_{prog}(q, q_i) = \psi^{\mathsf{Msg}}(q, q_n) \wedge \psi^{\mathsf{Cond}}_{\mathsf{hot}}(q, q_n) \wedge \psi^{\mathsf{LocInv},\bullet}_{\mathsf{hot}}(q_n)$$

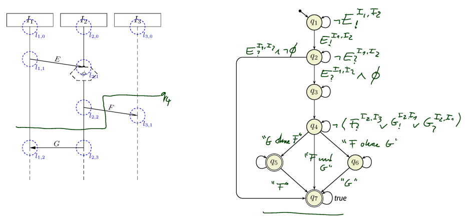
- $\psi^{\mathsf{Msg}}(q, q_i) = \bigwedge_{\psi \in \mathsf{Msg}(q_i \setminus q)} \psi \wedge \bigwedge_{j \ne i} \bigwedge_{\psi \in (\mathsf{Msg}(q_j \setminus q) \setminus \mathsf{Msg}(q_i \setminus q))} \neg\psi$
  $\underbrace{\wedge \Big(strict \implies \bigwedge_{\psi \in (\mathcal{E}^\mathcal{I}_{!?} \cap \mathsf{Msg}(\mathcal{L})) \setminus \mathsf{Msg}(\mathcal{F}_i)} \neg\psi\Big)}_{=:\psi_{\mathsf{strict}}(q, q_i)}$
- $\psi^{\mathsf{Cond}}_\theta(q, q_i) = \bigwedge_{\gamma = (L, \phi) \in \mathsf{Cond},\ \Theta(\gamma) = \theta,\ L \cap (q_i \setminus q) \ne \emptyset} \phi$
- $\psi^{\mathsf{LocInv},\bullet}_\theta(q, q_i) = \bigwedge_{\lambda = (l, \iota, \phi, l', \iota') \in \mathsf{LocInv},\ \Theta(\lambda) = \theta,\ \lambda\ \bullet\text{-active at } q_i} \phi$

  Local invariant $(l_0, \iota_0, \phi, l_1, \iota_1)$ is **$\bullet$-active** at q if and only if
  - $l_0 \prec l \prec l_1$, or
  - $l = l_0 \wedge \iota_0 = \bullet$, or
  - $l = l_1 \wedge \iota_1 = \bullet$

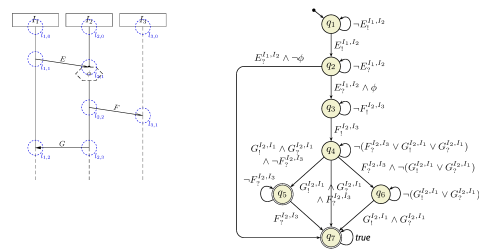
  for some front location l of cut (!) q.



Example (without strictness condition)



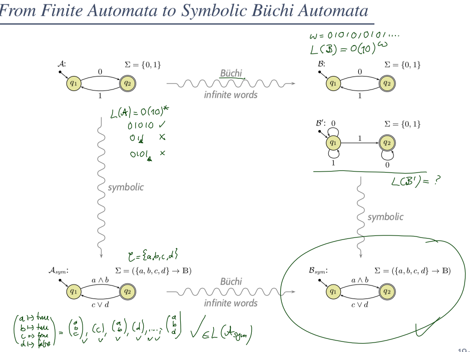
Example (without strictness condition)



Content

Excursion: Symbolic Büchi Automata

From Finite Automata to Symbolic Büchi Automata



Symbolic Büchi Automata

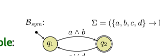
Definition. A **Symbolic Büchi Automaton** (TBA) is a tuple

$$\mathcal{B} = (\mathcal{C}_\mathcal{B}, Q, q_{ini}, \rightarrow, Q_F)$$

where

- $\mathcal{C}_\mathcal{B}$ is a set of atomic propositions,
- Q is a finite set of **states**,
- $q_{ini} \in Q$ is the initial state,
- $\rightarrow \subseteq Q \times \Phi(\mathcal{C}_\mathcal{B}) \times Q$ is the finite **transition relation**. Each transitions $(q, \psi, q') \in \rightarrow$ from state q to state q' is labelled with a propositional formula $\psi \in \Phi(\mathcal{C}_\mathcal{B})$.
- $Q_F \subseteq Q$ is the set of **fair** (or accepting) states.

Example:



Run of TBA

Definition. Let $\mathcal{B} = (\mathcal{C}_\mathcal{B}, Q, q_{ini}, \rightarrow, Q_F)$ be a TBA and

$$w = \sigma_1, \sigma_2, \sigma_3, \dots \in (\mathcal{C}_\mathcal{B} \rightarrow \mathbb{B})^\omega$$

an infinite word, each letter is a valuation of $\mathcal{C}_\mathcal{B}$.

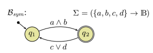
An infinite sequence

$$\varrho = q_0, q_1, q_2, \ldots \in Q^\omega$$

of states is called run of $\mathcal{B}$ over w if and only if

- $q_0 = q_{ini}$,
- for each $i \in \mathbb{N}_0$ there is a transition $(q_i, \psi_i, q_{i+1}) \in \rightarrow$ s.t. $\sigma_i \models \psi_i$.

Example:



$$w = \underbrace{\{a \mapsto \textit{true}, b \mapsto \textit{true}, c \mapsto \textit{false}, d \mapsto \textit{false}\}}_{\{a,b\} \text{ for short}}, \{c\}, \{a, b\}, (\{d\}, \{a, b\})^\omega$$

The Language of a TBA

Definition.
We say TBA $\mathcal{B} = (\mathcal{C}_\mathcal{B}, Q, q_{ini}, \rightarrow, Q_F)$ **accepts** the word

$$w = (\sigma_i)_{i \in \mathbb{N}_0} \in (\mathcal{C}_\mathcal{B} \rightarrow \mathbb{B})^\omega$$

if and only if $\mathcal{B}$ has a run

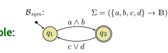
$$\varrho = (q_i)_{i \in \mathbb{N}_0}$$

over w
such that fair (or accepting) states are visited infinitely often by ϱ, i.e.,

$$\forall\, i \in \mathbb{N}_0\; \exists\, j > i : q_j \in Q_F.$$

We call the set $Lang(\mathcal{B}) \subseteq (\mathcal{C}_\mathcal{B} \rightarrow \mathbb{B})^\omega$ of words that are accepted by $\mathcal{B}$ the **language of $\mathcal{B}$**.

Example:



LSCs vs. Software

Software, formally

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

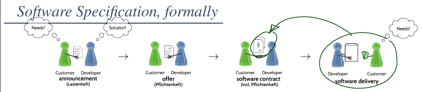
$$\sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots$$

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.

Software Specification, formally



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

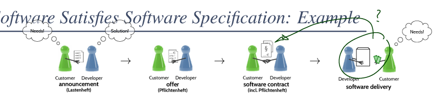
$$[\![\mathscr{S}]\!] = \{(S_1, [\![\,\cdot\,]\!]_1), (S_2, [\![\,\cdot\,]\!]_2), \ldots\}.$$

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

Definition. Software $(S, [\![\,\cdot\,]\!])$ **satisfies** software specification $\mathscr{S}$, denoted by $S \models \mathscr{S}$, if and only if

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

Software Satisfies Software Specification: Example ?



Software Specification

$\mathscr{S}$:

T: room ventilation		r_1	r_2	r_3
b	button pressed?	×	×	–
off	ventilation off?	×	–	*
on	ventilation on?	–	×	*
go	start ventilation	×	–	–
$stop$	stop ventilation	–	×	–

Define: $(S, [\![\,\cdot\,]\!]) \in [\![\mathscr{S}]\!]$ if and only if for all

$$\sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots \in [\![S]\!]$$

and for all $i \in \mathbb{N}_0$,

$$\exists\, r \in T \bullet \sigma_i \models \mathcal{F}(r).$$

Software

- Assume we have a program S for the room ventilation controller.
- Assume we can **observe** at well-defined points in time the conditions b, off, on, go, $stop$ when the software runs.
- Then **the behaviour** $[\![S]\!]$ of S can be viewed as computation paths of the form

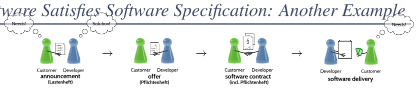
$$\sigma_0 \xrightarrow{\tau} \sigma_1 \xrightarrow{\tau} \sigma_2 \cdots$$

where each σ_i is a valuation of b, off, on, go, $stop$, i.e. $\sigma_i : \{b, off, on, go, stop\} \rightarrow \mathbb{B}$.

- **For example**:

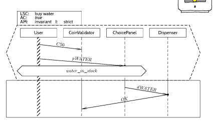
$$(\ off\ ) \xrightarrow{\tau} \begin{pmatrix} b \\ off \\ go \end{pmatrix} \xrightarrow{\tau} (\ on\ ) \xrightarrow{\tau} \begin{pmatrix} b \\ on \\ stop \end{pmatrix} \xrightarrow{\tau} (\ off\ ) \ldots$$

Software Satisfies Software Specification: Another Example



Software Specification

$\mathscr{S}$:



Define: $(S, [\![\cdot]\!]) \in [\![\mathscr{S}]\!]$ if and only if

- **tja...** (in a minute)

Software

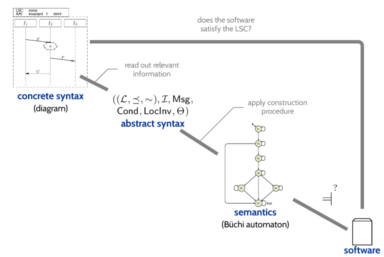
- Assume we can **observe** at well-defined points in time the observables relevant for the LSC (conditions and messages) when the software S runs.
- Then **the behaviour** $[\![S]\!]$ of S can be viewed as computation paths over the LSC's observables.
- **For example**:

$$\sigma_0 \xrightarrow{\tau} \sigma_1 \xrightarrow{E1^{U,V}} \sigma_2 \xrightarrow{pSOFT^{U,V}} \sigma_3 \xrightarrow{\tau} \sigma_4 \xrightarrow{\tau} \sigma_5 \xrightarrow{\tau} \sigma_6 \xrightarrow{SOFT^{V,U}} \dots$$

machine switched on, *user inserts 1 €*, *buttons light up*, *user presses 'SOFT' button*, *prepare dispenser*, *drink ready*, *notify user*

- And then we can relate S to $\mathscr{S}$.

The Plan: A Formal Semantics for a Visual Formalism



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

- $\Sigma = (C \to \mathbb{B}), \mathcal{C} \subseteq C$, i.e. the **states** comprise valuations of the conditions in $\mathcal{C}$,
- $A = (B \to \mathbb{B}), \mathcal{E}^{\mathcal{I}}_{!?} \subseteq B$, i.e. the **events** comprise valuations of $E^{i,j}_{!}, E^{i,j}_{?}$.

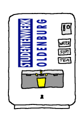
A computation path $\pi = \sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \dots \in [\![S]\!]$ of software S **induces** the word

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

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

$$W_S = \{w(\pi) \mid \pi \in [\![S]\!]\}.$$

LSCs vs. Software (or Systems)



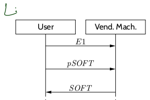
$\pi = \sigma_0 \xrightarrow{\tau} \sigma_1 \xrightarrow{E1^{U,V}} \sigma_2 \xrightarrow{pSOFT^{U,V}} \sigma_3 \xrightarrow{\tau} \sigma_4 \xrightarrow{\tau} \sigma_5 \xrightarrow{\tau} \sigma_6 \xrightarrow{SOFT^{V,U}} \dots \in [\![S]\!]$

$w(\pi) = \{\}, \{E1\}, \{pSOFT\}, \{\}, \{\}, \{\}, \{SOFT\}, \{\}, \dots \in \mathcal{L}(\mathcal{A}_L)$

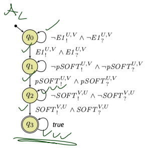
$\Leftrightarrow \pi \models L$

$$w = \{\}, \{E1^{U,V}_!, E1^{U,V}_?\}, \{pSOFT^{U,V}_!, pSOFT^{U,V}_?\}, \{\}, \{\}, \{\}, \{SOFT^{V,U}_!, SOFT^{V,U}_?\}, \{\}, \dots \in Lang(\mathcal{B}(\mathscr{L}))$$

L:



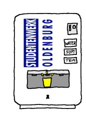
$\mathcal{A}_L$



TBA over $\mathcal{C}_\mathcal{B} = \mathcal{C} \cup \mathcal{E}^{\mathcal{I}}_{!?}$, where $\mathcal{C} = \emptyset$ and $\mathcal{E}^{\mathcal{I}}_{!?} = \{E1^{U,V}_!, E1^{U,V}_?, pSOFT^{U,V}_!, pSOFT^{U,V}_?, SOFT^{V,U}_!, SOFT^{V,U}_?, \dots\}$.

$E1$: insert 1 € coin
$pSOFT$: press 'SOFT' button
$SOFT$: dispense soft drink

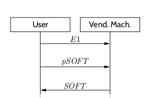
LSCs vs. Software (or Systems)

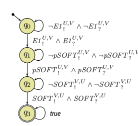


$$\sigma_0 \xrightarrow{\tau} \sigma_1 \xrightarrow{E1^{U,V}} \sigma_2 \xrightarrow{pSOFT^{U,V}} \sigma_3 \xrightarrow{\tau} \sigma_4 \xrightarrow{\tau} \sigma_5 \xrightarrow{\tau} \sigma_6 \xrightarrow{SOFT^{V,U}} \dots \in [\![S]\!]$$

$$w(\pi) = \sigma_0, (\sigma_1 \cup \{E1^{U,V}_!, E1^{U,V}_?\}), (\sigma_2 \cup \{pSOFT^{U,V}_!, pSOFT^{U,V}_?\}), \sigma_3, \sigma_4, \sigma_5, (\sigma_6 \cup \{SOFT^{V,U}_!, SOFT^{V,U}_?\}), \dots$$

$$w = \{\}, \{E1^{U,V}_!, E1^{U,V}_?\}, \{pSOFT^{U,V}_!, pSOFT^{U,V}_?\}, \{\}, \{\}, \{\}, \{SOFT^{V,U}_!, SOFT^{V,U}_?\}, \{\}, \dots \in Lang(\mathcal{B}(\mathscr{L}))$$





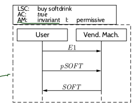
TBA over $\mathcal{C}_\mathcal{B} = \mathcal{C} \cup \mathcal{E}^{\mathcal{I}}_{!?}$, where $\mathcal{C} = \emptyset$ and $\mathcal{E}^{\mathcal{I}}_{!?} = \{E1^{U,V}_!, E1^{U,V}_?, pSOFT^{U,V}_!, pSOFT^{U,V}_?, SOFT^{V,U}_!, SOFT^{V,U}_?, \dots\}$.

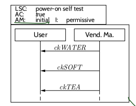
$E1$: insert 1 € coin
$pSOFT$: press 'SOFT' button
$SOFT$: dispense soft drink

Content

Activation Condition and Mode

Full LSC Syntax (without pre-chart)





A **full LSC** $\mathscr{L} = (MC, ac_0, am, \Theta_\mathscr{L})$ consists of

- (non-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** $ac_0 \in \Phi(\mathcal{C})$,
- **strictness flag** $strict$ (if *false*, $\mathscr{L}$ is **permissive**)
- **activation mode** $am \in$ {initial, invariant},
- **chart mode** **existential** ($\Theta_\mathscr{L}$ = cold) or **universal** ($\Theta_\mathscr{L}$ = hot).

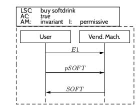
Software Satisfies LSC

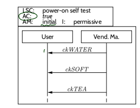
Let S be a software which is **compatible** with LSC $\mathscr{L}$ (without pre-chart).

We say software S **satisfies** LSC $\mathscr{L}$, 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	$\forall w \in W_S \bullet w^0 \models ac \wedge \neg\psi_{exit}(C_0)$ $\implies w^0 \models \psi_{prog}(\emptyset, C_0) \wedge w/1 \in Lang(\mathcal{B}(\mathscr{L}))$	$\forall w \in W_S\, \forall k \in \mathbb{N}_0 \bullet w^k \models ac \wedge \neg\psi_{exit}(C_0)$ $\implies w^k \models \psi^{\mathsf{Cond}}_{\mathsf{hot}}(\emptyset, C_0) \wedge w/k+1 \in Lang(\mathcal{B}(\mathscr{L}))$

where and C_0 is the minimal (or **instance heads**) cut of the main-chart.





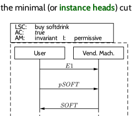
Software Satisfies LSC

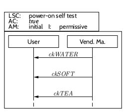
Let S be a software which is **compatible** with LSC $\mathscr{L}$ (without pre-chart).

We say software S **satisfies** LSC $\mathscr{L}$, 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	$\forall w \in W_S \bullet w^0 \models ac \wedge \neg\psi_{exit}(C_0)$ $\implies w^0 \models \psi_{prog}(\emptyset, C_0) \wedge w/1 \in Lang(\mathcal{B}(\mathscr{L}))$	$\forall w \in W_S\, \forall k \in \mathbb{N}_0 \bullet w^k \models ac \wedge \neg\psi_{exit}(C_0)$ $\implies w^k \models \psi^{\mathsf{Cond}}_{\mathsf{hot}}(\emptyset, C_0) \wedge w/k+1 \in Lang(\mathcal{B}(\mathscr{L}))$

where and C_0 is the minimal (or **instance heads**) cut of the main-chart.

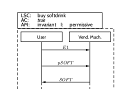




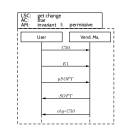
Software S satisfies **a set of** LSCs $\mathscr{L}_1, \dots, \mathscr{L}_n$ if and only if $S \models \mathscr{L}_i$ for all $1 \le i \le n$.

LSCs At Work

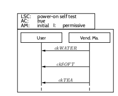
Example: Vending Machine



- **Positive scenario:** Buy a Softdrink
  We (only) accept the software if it **is possible** to buy a softdrink.
  - (i) Insert one 1 euro coin.
  - (ii) Press the 'softdrink' button.
  - (iii) Get a softdrink.



- **Positive scenario:** Get Change
  We (only) accept the software if it **is possible** to 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.



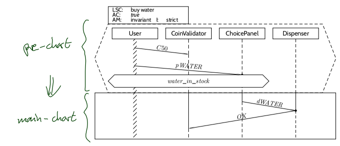
- **Requirement:** Perform Self-Test on Power-on
  We (only) accept the software if it **always** performs a self-test on power-on.
  - (i) Check water dispenser.
  - (ii) Check softdrink dispenser.
  - (iii) Check tea dispenser.

Content

- **Live Sequence Charts**
  - **TBA Construction**
  - **LSCs vs. Software**
  - **Full LSC** (without pre-chart)
    - Activation Condition & Activation Mode
  - (Slightly) **Advanced LSC Topics**
    - Full LSC with **pre-chart**
  - **LSCs in Requirements Engineering**
    - **strengthening** existential LSCs (scenarios) into universal LSCs (requirements)
  - **LSCs in Quality Assurance**
- **Requirements Engineering Wrap-Up**
  - Requirements Analysis in a Nutshell
  - Recall: Validation by **Translation**

(Slightly) Advanced LSC Topics

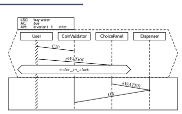
Full LSC Syntax *(with pre-chart)*



A **full LSC** $\mathscr{L} = (PC, MC, ac_0, am, \Theta_\mathscr{L})$ consists of

- **pre-chart** $PC = ((\mathcal{L}_P, \preceq_P, \sim_P), \mathcal{I}_P, \mathsf{Msg}_P, \mathsf{Cond}_P, \mathsf{LocInv}_P, \Theta_P)$ (possibly empty),
- (non-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** $ac_0 \in \Phi(\mathcal{C})$,
- **strictness flag** $strict$ (if *false*, $\mathscr{L}$ is **permissive**)
- **activation mode** *am* $\in$ {initial, invariant},
- **chart mode existential** ($\Theta_\mathscr{L}$ = cold) or **universal** ($\Theta_\mathscr{L}$ = hot).

LSC Semantics with Pre-chart



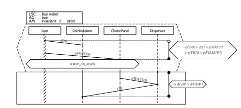
	am = initial	am = invariant
$\Theta_\mathscr{L}$ = cold	$\exists\, w \in W\; \exists\, m \in \mathbb{N}_0 \bullet$ $\wedge\, w^0 \models ac \wedge \neg\psi_{exit}(C_0^P) \wedge \psi_{prog}(\emptyset, C_0^P)$ $\wedge\, w/1, \dots, w/m \in Lang(\mathcal{B}(PC))$ $\wedge\, w^{m+1} \models \neg\psi_{exit}(C_0^M)$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

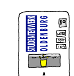
where C_0^P and C_0^M are the minimal (or **instance heads**) cuts of pre- and main-chart.

Pre-Charts At Work

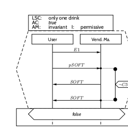
Example: Vending Machine

- **Requirement:** Buy Water
  We (only) accept the software if,
  (i) **Whenever** we insert 0.50 €,
  (ii) and press the 'water' button (and no other button),
  (iii) and there is water in stock,
  (iv) **then** we get water (and nothing else).





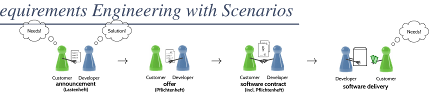
- **Negative scenario:** A Drink for Free
  We **don't** accept the software if it is possible to get a drink for free.
  (i) Insert one 1 euro coin.
  (ii) Press the 'softdrink' button.
  (iii) Do not insert any more money.
  (iv) Get **two** softdrinks.



Content

LSCs in Requirements Analysis

Requirements Engineering with Scenarios



One quite effective approach:

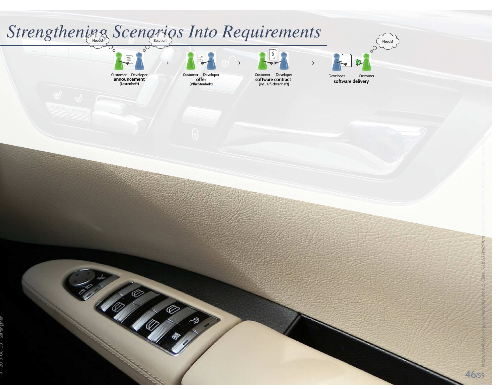
(i) **Approximate** the software requirements: ask for positive / negative **existential scenarios.**

- **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*)

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

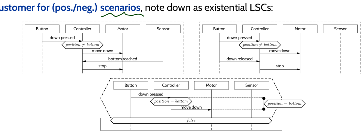
- **Investigate preconditions, side-conditions, exceptional cases** and **corner-cases.**
  (→ extend use-cases, refine LSCs with conditions or local invariants)
- **Generalise** into universal requirements, e.g., **universal LSCs.**
- **Validate** with customer using new positive / negative scenarios.

Strengthening Scenarios Into Requirements

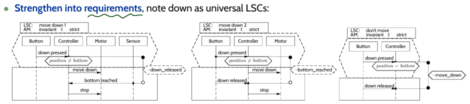


Strengthening Scenarios Into Requirements

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



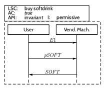
- **Strengthen into requirements**, note down as universal LSCs:

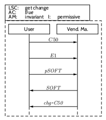


- **Re-Discuss** with customer using example words of the LSCs' language.

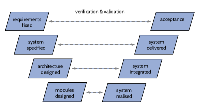
LSCs vs. Quality Assurance

How to Prove that a Software Satisfies an LSC?

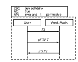


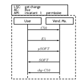


- Software S satisfies **existential** LSC $\mathscr{L}$ if there **exists** $\pi \in [\![S]\!]$ such that $\mathscr{L}$ accepts $w(\pi)$. Prove $S \models \mathscr{L}$ by demonstrating π.
- Note: **Existential** LSCs* may hint at **test-cases** for the **acceptance test**!
(*: as well as (positive)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

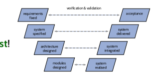


How to Prove that a Software Satisfies an LSC?

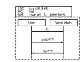


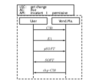


- Software S satisfies **existential** LSC $\mathscr{L}$ if there **exists** $\pi \in [\![S]\!]$ such that $\mathscr{L}$ accepts $w(\pi)$. Prove $S \models \mathscr{L}$ by demonstrating π.
- Note: **Existential** LSCs* may hint at **test-cases** for the **acceptance test**!
(*: as well as (positive)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

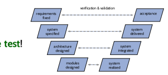


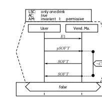
How to Prove that a Software Satisfies an LSC?

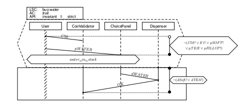




- Software S satisfies **existential** LSC $\mathscr{L}$ if there **exists** $\pi \in [\![S]\!]$ such that $\mathscr{L}$ accepts $w(\pi)$. Prove $S \models \mathscr{L}$ by demonstrating π.
- Note: **Existential** LSCs* may hint at **test-cases** for the **acceptance test**!
(*: as well as (positive)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

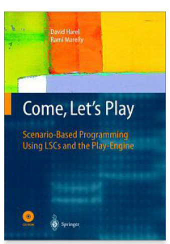






- **Universal** LSCs (and negative/anti-scenarios!) in general need an **exhaustive analysis**!
(Because they require that the software **never ever** exhibits the unwanted behaviour.)
Prove $S \not\models \mathscr{L}$ by demonstrating one π such that $w(\pi)$ **is not accepted** by $\mathscr{L}$.

Pushing Things 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**.
- 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.
- **Pre-charts** allow us to
  - specify **anti-scenarios** ("this must not happen"),
  - contrain **activation**.
- **Method:**
  - discuss (anti-)scenarios with customer,
  - generalise into universal LSCs and re-validate.

Requirements Engineering Wrap-Up

Topic Area Requirements Engineering: Content

- VL 5
  - **Introduction**
  - Definition: **Software** & **SW Specification**
  - **Requirements Specification**
    - Desired Properties
    - Kinds of Requirements
    - Analysis Techniques
- VL 6
  - **Documents**
    - Dictionary, Specification
  - **Specification Languages**
    - Natural Language
    - Decision Tables
      - Syntax, Semantics
      - Completeness, Consistency, …
- VL 7
- VL 8
  - Scenarios
    - User Stories, Use Cases
    - Live Sequence Charts
      - Syntax, Semantics
- VL 9
  - **Wrap-Up**

Vocabulary

Techniques

informal

semi-formal

formal

52/59

Risks Implied by Bad Requirements Specifications

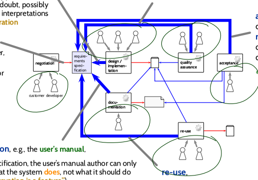
preparation of **tests**,

- without a description of allowed outcomes, tests are randomly searching for generic errors (like crashes) → systematic testing hardly possible

design and **implementation**,

- without specification, programmers may just "ask around" when in doubt, possibly yielding different interpretations → difficult integration

negotiation (with customer, marketing department, or …)



acceptance by customer, **resolving** later objections or regress claims,

- without specification, it is unclear at delivery time whether behaviour is an error (developer needs to fix) or correct (customer needs to accept and pay) → nasty disputes, additional effort

documentation, e.g., the **user's manual**,

- without specification, the user's manual author can only describe what the system does, not what it should do ("every observation is a feature")

re-use,

- without specification, re-use needs to be based on re-reading the code → risk of unexpected changes

- later **re-implementations**.
  - the new software may need to adhere to requirements of the old software; if not properly specified, the new software needs to be a 1:1 re-implementation of the old → additional effort

– 5 – 2019-05-13 – Sreintro –

12/49

53/59

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.

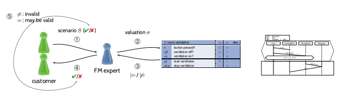
  Ask for each requirements: what is the **acceptance test**?
- **Use formal notations**
  - to **fully understand requirements** (precision),
  - for **requirements analysis** (completeness, etc.),
  - to communicate with your developers.
- If in doubt, **complement** (formal) **diagrams with text** (as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

54/59

Formalisation Validation

Two broad directions:

- **Option 1**: teach formalism (usually not economic).
- **Option 2**: serve as translator / mediator.



① domain experts **tell** system scenario S (maybe keep back, whether allowed / forbidden),
② FM expert **translates** system scenario to valuation σ,
③ FM expert **evaluates** DT on σ,
④ FM expert **translates** outcome to "allowed / forbidden by DT",
⑤ compare expected outcome and real outcome.

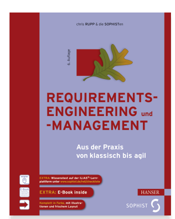
- **Recommendation**: (Course's Manifesto?)
  - use formal methods for the **most important/intricate requirements** (formalising **all requirements** is in most cases **not possible**),
  - use the **most appropriate formalism** for a given task,
  - use formalisms that **you know (really) well**.

– 7 – 2019-05-20 – Setdisc –

39/41

56/59

(Strong) Literature Recommendation



(Rupp and die SOPHISTen, 2014)

57/59

References

58/59

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.