Risks Implied by Bad Requirements Specifications

- Without specification, programs may just “ask around” when in doubt, possibly yielding different interpretations.
- Without specification, the user’s manual author can only describe what the system does, not what it should do (“every observation is a feature”).
- Without a description of allowed outcomes, tests are randomly searching for generic errors (like crashes).
- Without specification, it is unclear at delivery time whether behavior is an error (developer needs to fix) or correct (customer needs to accept and pay), leading to nasty disputes.
- Without specification, re-use needs to be based on re-reading the code, additional effort.
- 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.

Structure of Topic Areas

Example: Requirements Engineering

Vocabulary
- e.g. consistent, complete, tacit, etc.
- Techniques
  - informal
  - semi-formal
  - formal

In the course:
- e.g. “Whenever a crash...”
- e.g. “Always, if crash at t...”
- e.g. “t, t...”
- e.g. “Every observation is a feature”

Use Cases
- Pattern Language
- Decision Tables
- Live Sequence Charts

Use Cases
- Pattern Language
- Decision Tables
- Live Sequence Charts

Content

- Introduction
- Requirements Specification
- Desired Properties
- Kinds of Requirements
- Analysis Techniques
- Documents
- Dictionary, Specification
- Specification Languages
- Natural Language
- Decision Tables
- Syntax, Semantics
- Completeness, Consistency, ...
- Scenarios
- User Stories, Use Cases
- Live Sequence Charts
- Syntax, Semantics

Definition:

Software & SW Specification

Wrap-Up
The idea of scenarios (sometimes without negative or anti-scenarios) re-occurs in many process models or software development approaches. In the following, we will discuss two-and-a-half notations:

- **Use Cases** and Use Case Diagrams (OOSE)
- **User Stories** (part of Extreme Programming)
- **Sequence Diagrams** (here: Live Sequence Charts (Damm and Harel, 2001))

### Use Case: Definition

**Use Case** — A sequence of interactions between an actor (or actors) and a system triggered by a specific actor, which produces a result for an actor. (Jacobson, 1992)

### Use Case Example: ATM Authentication

- **Authentication goal** the client wants access to the ATM
- **pre-condition** the ATM is operational, the welcome screen is displayed, card and PIN of client are available
- **post-condition** client accepted, services of ATM are offered
- **post-cond. in exception case** access denied, card returned or withheld, welcome screen displayed
- **actors** client (main actor), bank system
- **open questions** none

#### Normal case
1. client inserts card
2. ATM reads card, sends data to bank system
3. bank system checks validity
4. ATM shows PIN screen
5. client enters PIN
6. ATM reads PIN, sends to bank system
7. bank system checks PIN
8. ATM accepts and shows main menu

#### Exception cases
- **exc. case 2a** card not readable
  1.1 ATM displays "card not readable"
  1.2 ATM returns card
  1.3 ATM shows welcome screen
- **exc. case 2b** card readable, but not ATM card
- **exc. case 2c** no connection to bank system
- **exc. case 3a** card not valid or disabled
- **exc. case 5a** client cancels
- **exc. case 5b** client doesn't react within 5 s
- **exc. case 6a** no connection to bank system
- **exc. case 7a** first or second PIN wrong
- **exc. case 7b** third PIN wrong

(Ludewig and Lichter, 2013)
Authentication

Use Case Example: ATM Authentication

Use Case Diagrams: Basic Building Blocks

Example: Use Case Diagram of the ATM Use Case

Use Case Diagrams: More Building Blocks

Example: Use Case Diagram of the ATM Use Case
User Stories (Beck, 1999)

"A User Story is a concise, written description of a piece of functionality that will be valuable to a user (or owner) of the software."

Per user story, use one file card — proposed card layout (front side):

- priority
- unique identifier, name
- estimation

As a [role] I want [something] so that [benefit].

and in addition:

- unique identifier (e.g. unique number),
- priority (from 1 (highest) to 10 (lowest)) assigned by customer,
- effort, estimated by developers,
- back side of file card: (acceptance)

- test case(s), i.e., how to tell whether the user story has been realised.

Natural Language Patterns

Natural language requirements can be (tried to be) written as an instance of the pattern "hAi hBi hCi hDi hEi hFi hG. " (German grammar) where

- A clarifies when and under what conditions the activity takes place
- B is MUST (obligation), SHOULD (wish), or WILL (intention); also: MUST NOT (forbidden)
- C is either "the system" or the concrete name of a (sub-)system
- D one of three possibilities:
  - "does," description of a system activity,
  - "offers," description of a function offered by the system to somebody,
  - "requires," description of a constraint, i.e., a condition that the system must satisfy

User Stories: Discussion

✔ easy to create, small units
✔ close contact to customer
✔ objective / testable: by fixing test cases early
✘ may get difficult to keep overview over whole system to be developed
→ maybe best suited for changes / extensions (after first iteration).
✘ not designed to cover non-functional requirements and restrictions
✘ agile spirit: strong dependency on competent developers
✘ estimation of effort may be difficult

(Balzert, 2009)
Use Case Example: ATM Authentication

**name** Authentication

**goal** the client wants access to the ATM

**pre-condition**
- the ATM is operational,
- the welcomescreen is displayed,
- card and PIN of client are available

**post-condition**
- client accepted,
- services of ATM are offered

**post-cond. in exception case**
- access denied, card returned or withheld,
- welcome screen displayed

**actors**
- client (main actor),
- bank system

**open questions**
- none

**normal case**
1. client inserts card
2. ATM reads card, sends data to bank system
3. bank system checks validity
4. ATM shows PIN screen
5. client enters PIN
6. ATM reads PIN, sends to bank system
7. bank system checks PIN
8. ATM accepts and shows main menu

**exception case 2a**
- card not readable
  1.1 ATM displays "card not readable"
  1.2 ATM returns card
  1.3 ATM shows welcome screen

**exception case 2b**
- card readable, but not ATM card

**exception case 2c**
- no connection to bank system

**exception case 3a**
- card not valid or disabled

**exception case 5a**
- client cancels

**exception case 5b**
- client doesn't react within 5 s

**exception case 6a**
- no connection to bank system

**exception case 7a**
- first or second PIN wrong

**exception case 7b**
- third PIN wrong

(見本: 芦沢と夜野, 2013)

---

**A Brief History of Sequence Diagrams**

- Message Sequence Charts (MSC), ITU standardized in different versions (ITU Z.120, 1st edition: 1993); often accused of lacking a formal semantics.
- Sequence Diagrams (SD) of UML 1.x (one of three main authors: I. Jacobson)
- SDs of UML 2.x address some issues, yet the standard exhibits unclarities and even contradictions (見本: 芦沢と夜野, 2007; 見本: Störrle, 2003)
- For the lecture, we consider Live Sequence Charts (LSCs) (見本: Damm and Harel, 2001; 見本: Klose, 2003; 見本: 芦沢と夜野, 2003), LSCs have a common fragment with UML 2.x SDs (見本: 芦沢と夜野, 2007).
\[ \{ \text{LocInv} \cup \text{Cond} \} \rightarrow \{ \text{LocInv}, \text{Cond}, \text{Msg} \} \]

- Inclusive, \( \circ \)
- Exclusive, \( \prime \)
- \( l, \iota, \phi, l' \) with \( \Phi(\cdot) \times L \times \{ \circ \} \times \text{LocInv} \times \text{Msg} \)

- \( \Phi(\cdot) \) is a set of \( \bullet \)
- \( \Phi(\cdot) \) is called a partitioning of \( \text{LocInv} \)

\[ E \subseteq L \times L \]

\( \{ I, E, I \} \) are called an LSC body

\( \{ I, E, I \} \) is a symmetric partial order

\( \{ I, E, I \} \) is a sim"
The partial order "⪯" and the simultaneity relation "∼" of locations induce a direct successor relation on cuts of an LSC body as follows:

Definition. Let $C \subseteq L$ be a cut of an LSC body $((L, ⪯, ∼), I, Msg, Cond, LocInv, Θ)$. A set $∅ \neq F \subseteq L$ of locations is called a fired-set $F$ of cut $C$ if and only if:

1. $C \cap F = \emptyset$ and $C \cup F$ is a cut, i.e., $F$ is closed under simultaneity,
2. all locations in $F$ are direct $≺$-successors of the front of $C$, i.e., $∀l ∈ F \exists l' ∈ C : l' ≺ l ∧ (∄ l'' ∈ C : l' ≺ l'' ≺ l),$
3. locations in $F$, that lie on the same instance line, are pairwise unordered, i.e., $∀l \neq l' ∈ F : (∃ I ∈ I : \{l, l'\} ⊆ I) ⇒ l \not⪯ l' ∧ l' \not⪯ l$,
4. for each asynchronous message reception in $F$, the corresponding sending is already in $C$, i.e., $∀(l, E, l') ∈ Msg : l' ∈ F ⇒ l ∈ C$.

The cut $C' = C \cup F$ is called a direct successor of $C$ via $F$, denoted by $C \Rightarrow F C'$.
Tell Them What You've Told Them...

- Use-Cases:
  - Interactions between system and actors,
  - Be sure to elaborate exceptions and corner cases,
  - In particular effective with customers lacking technical background.

- Use-Case Diagrams:
  - Visualise which participants are relevant for which use-case,
  - Pretty useless without the underlying use-case.

- User Stories:
  - Simple example of scenarios
  - Strong point: naming tests is necessary,
  - Weak point: hard to keep overview; global restrictions.

- Sequence Diagrams:
  - A visual formalism for interactions, i.e.,
    - Precisely defined syntax,
    - Precisely defined semantics (construct automaton from abstract syntax)
  - Can be used to precisely describe the interactions of a use-case.

References