Gottfried Wilhelm
Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Praktische Informatik
Fachgebiet Software Engineering

Analyse der Umsetzung zwischen User Stories und formalen Szenarien im Requirements Engineering

Masterarbeit

im Studiengang Informatik

von

Sebastian Föllmer

Prüfer: Prof. Dr. Kurt Schneider
Zweitprüfer: Prof. Dr. Joel Greenyer
Betreuer: Dipl.-Math. Olga Boruszewski, Dipl.-Inform. Daniel Gritzner

Hannover, den 19. April 2016

Abstract

It is difficult to describe requirements in an easy and intuitive but also precise and unambiguous way. The most important aspect in communication with a costumer is an easy to understand notation. In contrast to this, a hard to understand, but precise formal specification provides the advantages of formal model checking for errors or generating program code. However, such a formal specification is usually only created from an informal specification after the requirements analysis process is over. In this thesis a method is described which enables creation of a formal specification during requirements analysis by translating easy to understand user stories into formal scenarios. Consequently, the problems identified by the formal model checking can be solved during the requirement analysis phase already. Additionally, advantages and disadvantages of this approach are discussed. Moreover, it is mentioned how this method can be integrated into different engineering processes. Finally, possible areas of application are pointed out.

Zusammenfassung

Software- und Systemanforderungen so zu dokumentieren, dass sie einerseits leicht zu kommunizieren und andererseits präzise und unmissverständlich sind, ist eine schwierige Aufgabe. Für die Kommunikation zwischen einem Requirements Engineers und einem Kunden sollte die Anforderungsnotation leicht und intuitiv zu verstehen sein, um mit dem Auftraggeber über die Anforderungen diskutieren zu können. Im Gegensatz dazu bietet eine formale Notation die Möglichkeit, Anforderungen zwar schwer verständlich, dennoch präzise zu beschreiben. Dadurch kann das Systemverhalten exakt modelliert, simuliert und auf Fehler überprüft werden. Eine formale Spezifikation wird allerdings erst nach der Anforderungsanalyse anhand einer informellen Spezifikation erstellt. Um schon während der Anforderungsanalyse eine formale Spezifikation erstellen und Anforderungen dennoch leicht kommunizieren zu können, wird in dieser Arbeit eine Methode beschrieben, die die Umsetzung von leicht verständlichen User Stories zu formalen Szenarien ermöglicht. Anschließend wird die Integration dieser Methode in verschiedene Entwicklungsprozesse, sowie deren Vor- und Nachteile diskutiert. Abschließend werden mögliche Anwendungsgebiete beschrieben.

Danksagung

Zunächst möchte ich mich bei Prof. Schneider und Prof. Greenyer für die Vergabe dieses interessanten Themas sowie der fachlichen Betreuung bedanken.

Außerdem danke ich meinen Betreuern Dipl.-Math. Olga Boruszewski, Dipl.-Inform. Daniel Gritzner für ihre Zeit und Unterstüzung, die sie mir wöchentlich gewidmet haben.
Besonderer Dank gilt meiner Freundin Sabrina Kubinski für die große moralische Unterstützung und weiterer Dank gilt meinen Freunden, die bereitwillig meine Arbeit gegengelesen haben

Schließlich danke ich meinen Eltern für ihre Geduld und finanzielle Unterstützung.

Inhaltsverzeichnis

1. Einleitung **1**
1.1. Motivation . . . 1
1.2. Ziel . . . 2
1.3. Struktur . . . 2

2. Grundlagen **4**
2.1. Konzeptionelle Grundlagen . . . 4
2.1.1. Entwicklungsprozesse . . . 4
2.1.2. Requirements Engineering . . . 6
2.1.3. Formale Methoden . . . 7
2.2. Verwendete Notationstechniken . . . 9
2.2.1. Use Cases . . . 9
2.2.2. User Stories . . . 11
2.2.3. SML . . . 12
2.3. Verwandte Arbeiten . . . 16

3. Fallbeispiel: Orangenpresse **18**
3.1. Einleitung und Personas . . . 18
3.2. Kundengespräch und User Stories . . . 19
3.3. SML und Überführungstabelle . . . 27
3.4. Die Erstellung von Scenarios aus User Stories . . . 37

4. Methoden- und Modellentwicklung **43**
4.1. Modell der vorhandenen Artefakte . . . 43
4.2. Herausforderungen . . . 46
4.3. Methode zur Umsetzung zwischen User Stories und SML . . . 47
4.3.1. Erstellen von Enhanced User Stories . . . 48
4.3.2. Erstellen von Enhanced Scenarios . . . 53
4.3.3. Mark User Stories and validate Changes . . . 60
4.4. User Stories mit offenen Fragen . . . 64
4.4.1. Fragekarten erstellen . . . 64
4.4.2. Änderungsvorschag erstellen . . . 65
4.5. Schlussfolgerungen und ungelöste Probleme . . . 67
4.5.1. Schlussfolgerungen . . . 67
4.5.2. Bestehende Probleme . . . 69

5. Auswirkung und Nutzung in der Systemanalyse **71**
5.1. Anwendungsgebiete . . . 71
5.2. Einbettung in den Entwicklungsprozess . . . 73
5.2.1. Einsatz im RE nach dem RE-Referenzmodell . . . 73
5.2.2. Einsatz im agilen RE . . . 75
5.2.3. Einsatz in der Entwicklung mit vollständigem formalen Modelchecking . . . 76
5.3. Nutzen und entstehende Kosten . . . 77
5.3.1. Mögliche Verwendung nach Erstellung der SML Specification . . . 78

6. Zusammenfassung **79**
6.1. Ausblick 81

A. Literaturverzeichnis **83**

B. Anhang **86**

C. Inhalt der DVD **87**

1. Einleitung

1.1. Motivation

Seit es Software- und Requirements Engineering (RE) gibt, besteht auch die zentrale Frage, wie man einerseits Anforderungen möglichst einfach und verständlich aufschreiben kann, und andererseits die Anforderungen in eine Form bringen kann, die man präzise darstellen, modellieren und testen kann. Dies ist unter anderem notwendig, um die Kommunikation, auch mit Nicht-Informatikern (fachfremden Personen), zu erleichtern.

Eine etablierte Methode für die Kommunikation von Anforderungen stellen neben den *Use Cases* die *User Stories* dar, die nicht nur in *agilen* Projekten verwendet werden. Mit dieser Technik kann der Kunde ganz intuitiv mit eigenen Worten und ohne Vorkenntnisse, Anforderungen und Wünsche auf eine Karte schreiben. Damit können Anforderungen besonders einfach erfasst und kommuniziert werden.

Wenn es nun möglich wäre, User Stories direkt in eine *formale Form* zu überführen, hätte man eine präzise und unmissverständliche Darstellung der Anforderungen. Mit einer formalen Spezifikation wäre es anschließend möglich die Anforderungen direkt zu simulieren und sie auf Widersprüche und Anforderungslücken zu prüfen oder sie später für den Entwurf oder die Testfalle Generierung zu Hilfe zu nehmen.
Ein Nachteil von *formalen Methoden* ist die schwierige Anwendung. Deshalb bietet sich für die formale Notation in dieser Arbeit besonders das *Scenario Based Modelling* und speziell die *Scenario Modelling Language (SML)* an. Diese bietet eine (für formale Methoden) besonders einfache und intuitive Möglichkeit Anforderungen formal aufzuschreiben.

Dabei kann der Unterschied in dem Detailgrad von User Stories und SML als Vorteil verwendet werden, um neue Anforderungen aufzudecken, die dem Kunden anfangs noch nicht klar waren. Mit Hilfe einer Simulation der model-

lierten formalen Spezifikation und den User Stories ist es nun einfach Konflikte aufzuzeigen, über fehlende Informationen zu diskutieren und das Modell bzw. die Anforderungen mit dem Kunden in einer frühen Phase zu validieren.

Diese Arbeit untersucht die Möglichkeiten einer solchen Umwandlung zwischen User Stories und SML und beschreibt eine Methode sowie deren Kosten, Nutzen und Einbettung in die Systemanalyse.

1.2. Ziel

Das Ziel dieser Arbeit ist es, zunächst an einem ausführlichen Fallbeispiel die Schwierigkeiten der Umwandlung von User Stories in eine SML Specification zu beschreiben und dabei herauszuarbeiten, welche Informationen für eine erfolgreiche Umwandlung benötigt werden.

Darauf aufbauend soll ein Modell der benötigten Artefakte sowie eine Methode zur Umsetzung zwischen User Stories und SML erarbeitet werden, die die zuvor aufgezeigten Probleme löst und die benötigten Informationen erfasst.

Anschließend soll die praktische Anwendbarkeit und Integration in verschiedenen Entwicklungsprozessen sowie die Vor- und Nachteile der Methode analysiert und beschrieben werden.

1.3. Struktur

Nachdem ein Überblick über die behandelten Themen dieser Arbeit gegeben worden sind, werden zunächst die Grundlagen beschrieben, die für dieses Themengebiet benötigt werden, bevor anschließend kurz auf verwandte Arbeiten hingewiesen wird.

Am Fallbeispiel Orangenpresse werden in Kapitel 3 die Schwierigkeiten einer Umsetzung zwischen User Stories und der formalen Notation SML aufgezeigt und Lösungsvorschläge erarbeitet.

In dem Kernkapitel 4, ”Methoden und Modellentwicklung”, werden die aus dem dritten Kapitel gewonnenen Erkenntnisse verwendet, um eine Methode und ein Artefaktmodell zu erstellen, damit aus User Stories eine SML Specification abzuleiten werden kann. Die einzelnen Schritte werden ausführlich

und mit Beispielen beschrieben. Dabei wird auf besondere Eigenschaften und Variationsmöglichkeiten bei der Ausführung eingegangen. Zusätzlich werden die Erkenntnisse dieser Methode und die noch ungelösten Probleme erläutert.

Das Kapitel 5 geht zunächst auf mögliche Anwendungsgebiete für die Verwendung der erarbeiteten Methode ein und beschäftigt sich mit ihrer praktischen Eingliederung in verschiedene Entwicklungsprozesse und den daraus resultierenden Vor- und Nachteilen sowie möglichen Folgen.

Das letzte Kapitel dieser Arbeit beinhaltet die Zusammenfassung, die noch einmal die gewonnenen Ergebnisse aufzeigt, einen Ausblick liefert und die Arbeit kritisch reflektiert.

2. Grundlagen

In diesem Kapitel werden zum Verständnis benötigte Grundlagen der Arbeit erläutert. Dazu gehören die konzeptionellen Ideen des Requirements Engineering und der formalen Methoden im Systemengineering, sowie die Vorstellung von User Stories und Scenario Modelling Language (SML).

2.1. Konzeptionelle Grundlagen

Die hier betrachteten konzeptionellen Ideen umfassen die Entwicklungsprozesse des V-Modells und eines agilen Entwicklungsprozesses sowie Requirements Engineering und formale Methoden.

2.1.1. Entwicklungsprozesse

Das **V-Modell** ist ein grundlegendes Vorgehensmodell im Software-und Systemengineering. Der Prozess beginnt, wie in Abb. 2.1 gezeigt, bei *System Requirements Analysis* und verläuft anhand der durchgezogenen Pfeile bis hinunter zur *Implementation* und verläuft wieder nach oben zu *Client Acceptance* und *Operation*. Die linke Seite des Vs repräsentiert den Entwicklungsprozess und die rechte Seite die dazugehörigen Tests. Die gestrichelten Linien bedeuten, dass aus den Anforderungs- und Entwurfsdokumenten die Testfälle für die gegenüberliegenden Prozesse abgeleitet werden. In der Abb. 2.1 ist zu erkennen, dass System Requirements Analysis vor der Software Requirements Elicitation geschieht.
Hier werden die einzelnen Schritte nicht weiter erläutert, sondern werden nur zur Übersicht benutzt, da der Schwerpunkt der Arbeit auf der *System Requirement Analysis* liegt. Weitere Informationen hierzu finden sich in Elizabet [Hul10].

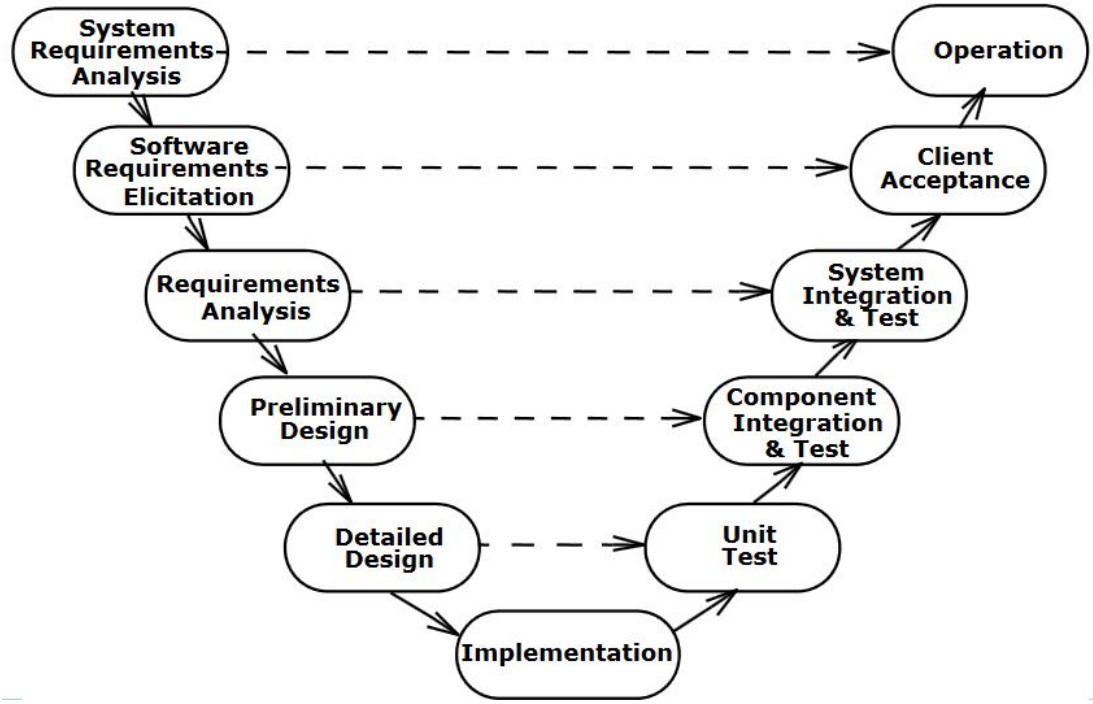

Abbildung 2.1.: System Engineering V-Modell [Pre15]

Bei einem **agilen Ansatz** wird das System in mehreren Iterationen, Schritt für Schritt entwickelt. Die Vorteile sind eine hohe Flexibilität bei neuen Anforderungen oder Anforderungsänderungen (werden in den nachfolgenden Iterationen behandelt) und die Möglichkeit, in einem frühen Entwicklungsstadium eine funktionierende Implementierung zu liefern. Zusammen mit einer Priorisierung der Anforderungen, z.B. durch *Planning Poker* [Coh05] können so schon die wichtigsten Anforderungen umgesetzt und andere Zusatzfunktionen mit jeder Iteration nachgeliefert werden. Die bekanntesten agilen Methoden sind *Extreme Programming* und *Scrum*. In Scrum werden die Iterationen *Sprints* genannt. Genauere Informationen und Beispiele zu beiden Methoden finden sich in dem Buch User Stories applied von Mike Cohn [Coh04]

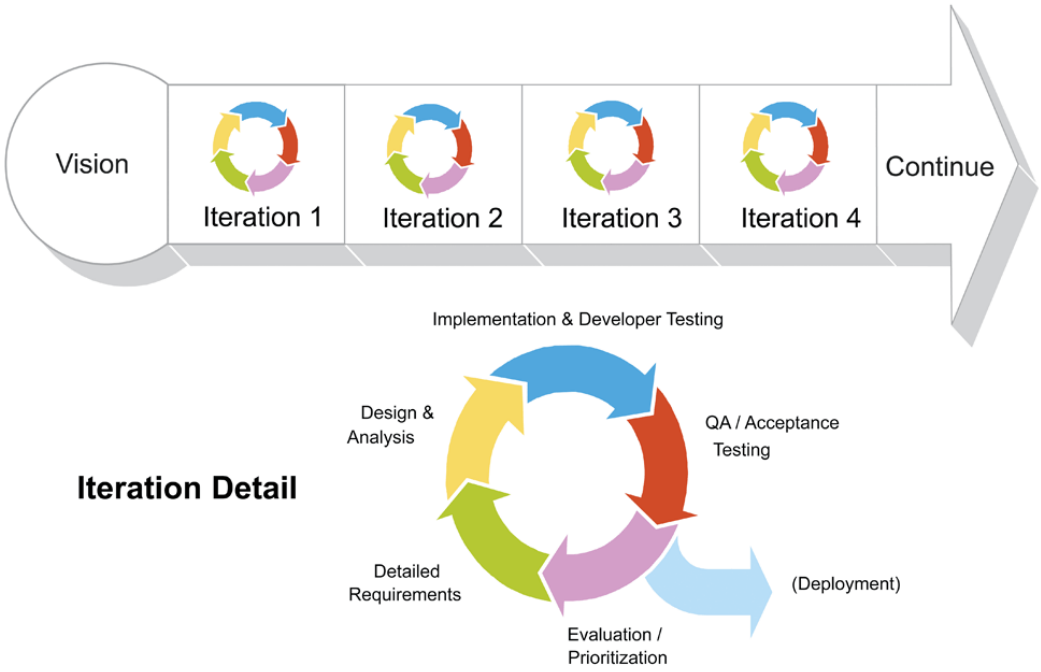

Abbildung 2.2.: agile Entwicklung mit SCRUM [Jam12]

2.1.2. Requirements Engineering

Da sich diese Arbeit mit der Darstellung und Verwendung von Anforderungen befasst, folgen zunächst die Begriffserklärungen für *Anforderungen*, *Requirements Engineering* und *Anforderungsanalyse*:
Anforderungen (nach Definition IEEE610) sind eine Beschaffenheit oder Fähigkeit, die von einem Benutzer zur Lösung eines Problems oder zur Erreichung eines Ziels benötigt werden.
Einfacher ausgedrückt: Eine Anforderung ist etwas, das ein Produkt tun oder haben muss [Sch15].
Die Aufgaben von Anforderungen nach Schneider [Sch15] sind:

- Kunden die Gelegenheit zu geben, ihre Wünsche und deren Hintergründe besser zu verstehen und zu sortieren
- Gemeinsames Verständnis zwischen allen Beteiligten zu schaffen
- Eine Grundlage für die sich ständig ändernden Kundenwünsche schaffen
- Referenz der zu erbringenden Leistungen gegen Projektende

Requirements Engineering (RE) umfasst *Anforderungsanalyse* und *-Management* mit ingenieurmäßigem Vorgehen.

Anforderungsanalyse umfasst die Aktivitäten zur Ermittelung, Formulierung, Dokumentation, Abstimmung und schließlich der Ermittlung und Verifikation.

Anforderungsmanagement umfasst die Verwaltung von Anforderungen und zugehörigen Informationen, um Anforderungsänderungen und Verfolgbarkeit zu gewährleisten.

Das RE- Referenzmodell in Abb.2.3 teilt alle Aufgaben in Phasen ein. Diese Phasen werden in der Anforderungsanalyse von links nach rechts durchlaufen, wobei *Änderungsmanagement* und *Verfolgbarkeit* (Tracing) aus dem Anforderungsmanagement parallel zur Anforderungsanalyse ablaufen.

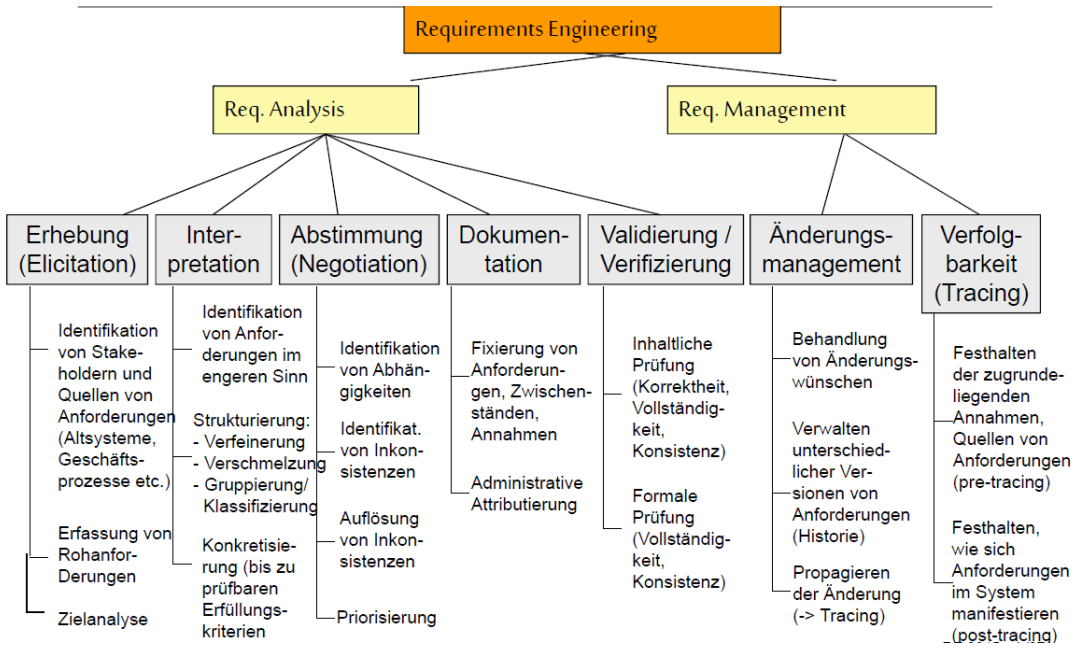

Abbildung 2.3.: Das RE-Referenzmodell (Beitrag DaimlerChrysler, Dagstuhl-Seminar 1998; aus der Vorlesung Requirements Engineering von Prof. Schneider [Sch15]).

2.1.3. Formale Methoden

Formale Methoden nach[Nas02] sind eine auf mathematischer Logik aufbauende Technik für das Erstellen, Entwickeln und Verifizieren von Soft- und Hardware-Systemen. Das heißt, sowohl die formale Spezifikation und das

formale Modell als auch die formalen Analysemethoden unterliegen einer wohldefinierten und logischen Syntax und Semantik, mit der es möglich ist, zu beweisen, dass das Modell eines Systems eine Spezifikation erfüllt.

Formale Spezifikation ist ähnlich zu einer informellen Spezifikation, denn beide beschreiben die Anforderungen des Systems. Allerdings nutzt eine formale Spezifikation eine auf Basis von mathematischer Logik aufbauende präzise Beschreibung.

Scenario Based Modelling Der Entwicklungsprozess mit einer formalen Methode entspricht zunächst dem V-Modell. Wie in Abb.2.4 zu erkennen ist, erfolgt nach der *informellen* Spezifikation, die am Ende der Dokumentation (bzw. Validation/Verifikation) Phase stattfindet, das Erstellen einer *formalen* Spezifikation und eines formalen Modells des Systems. Diese formale Spezifikation kann mit einem Modellchecking-Tool geprüft werden, womit vor der Implementierung geprüft werden kann, ob das erstellte Modell die formale Anforderungsspezifikation erfüllt.

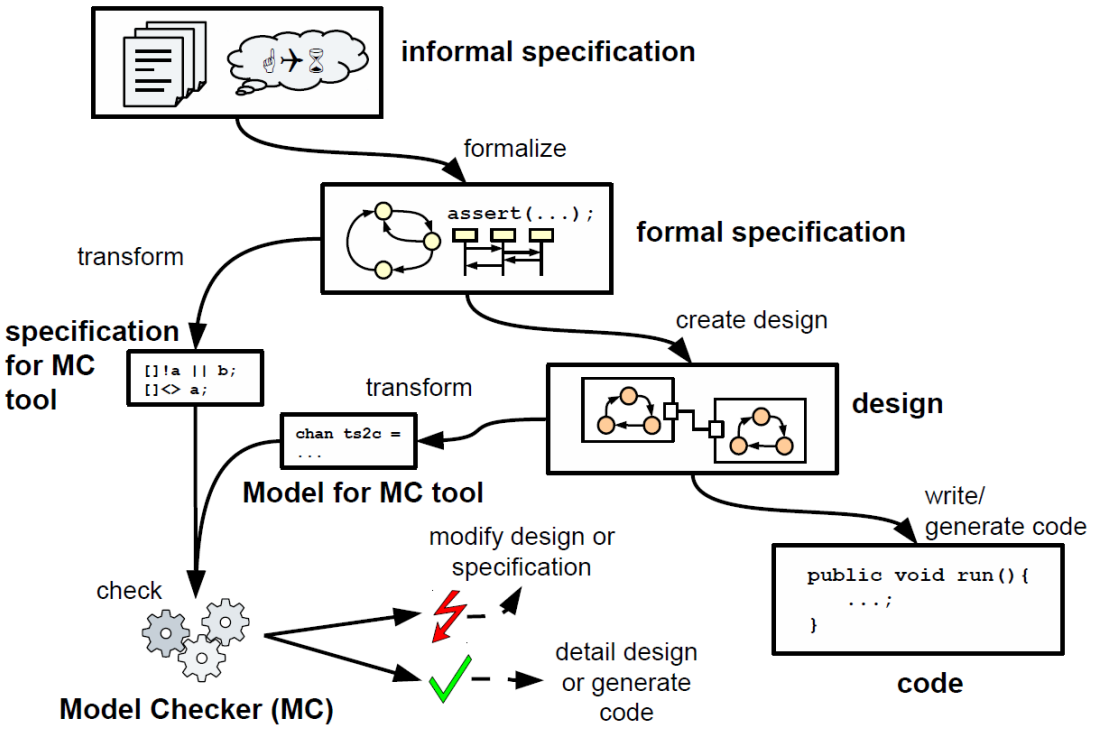

Abbildung 2.4.: formales Model Checking im Entwicklungsprozess (aus der Vorlesung Formale Methode im Softwareengineering Wintersemester 2015, Prof. Greenyer LUH Fachbereich Software Engineering [Gre16c]).

2.2. Verwendete Notationstechniken

Wie der Titel erkennen lässt, wird in dieser Arbeit mit User Stories und formalen Szenarien, genauer gesagt mit der Scenario Modelling Language gearbeitet. Zusätzlich werden noch Use Cases verwendet.

2.2.1. Use Cases

Ein **Use Case** ist eine Beschreibung einer Interaktion zwischen (mindestens) einem Akteur und dem System, durch die ein Ziel des Hauptakteurs erreicht wird.
Use Cases sind im Requirements Engineering eine Standard Technik um funktionale Anforderungen zu erfassen und festzuhalten. Diese beschreiben das extern wahrnehmbare Systemverhalten und werden unter anderem zur Ist-Analyse

und Soll-Analyse sowie zur Ableitung von Testfällen verwendet.
Ein Use Case bestehen aus einer Tabelle in der, je nach gefordertem Detailgrad, eine Vielzahl von Informationen eingetragen werden. Zu den wichtigsten Informationen gehören [Sch15] . Titel, Akteur, Auslösendes Ereignis, Vorbedingung, Essenzielle Schritte, Ausnahmefälle, Nachbedingung.
Weitere mögliche Einträge wären Umfeld, Stakeholder, Garantie oder Erfolgsfall [Rup13]. Eine Exemplarische Use Case Tabelle findet sich im Anhang.

Für die Essenziellen Schritte werden in der Literatur verschiedene Bezeichnungen wie ”Beschreibung” oder *”Szenario”* als Synonym benutzt. In dieser Arbeit wird für die Essenziellen Schritte eines Use Cases niemals der Begriff Szenario, sondern *Use Case Schritte* verwendet, da sonst Verwechslungsgefahr mit den SML Scenarios besteht.

Des weiteren werden in dieser Arbeit vereinfachte Use Cases verwendet, die nur aus der Nummer und dem Titel der Karte, den essenziellen Schritten und den Erweiterungen besteht. Der Grund dafür liegt darin, dass zusätzlich User Stories und SML Scenarios verwendet werden, die mit den Use Cases *verlinkt* werden. Nicht benötigte Informationen werden ausgelassen um mehr Übersichtlichkeit zu erhalten.

Die User Stories und Use Cases werden mit der von der Leibniz Universität Hannover von Institut Software Engineering mitentwickelten Integrated Requirements Environment (IRE) verwaltet und dargestellt2.5, die ein gleichzeitiges Arbeiten an sowohl Use Cases als auch User Stories erlaubt. Zusätzlich können die in der linken Hälfte dargestellten Use Cases mit den dazugehörigen Story Cards verlinkt werden. In den späteren Kapiteln wird dieser Vorteil ausführlich beschrieben. In dieser Arbeit wird auf diese Umgebung aber nicht weiter eingegangen, sondern lediglich die Struktur und die Eigenschaft der Verlinkung als solche verwendet.

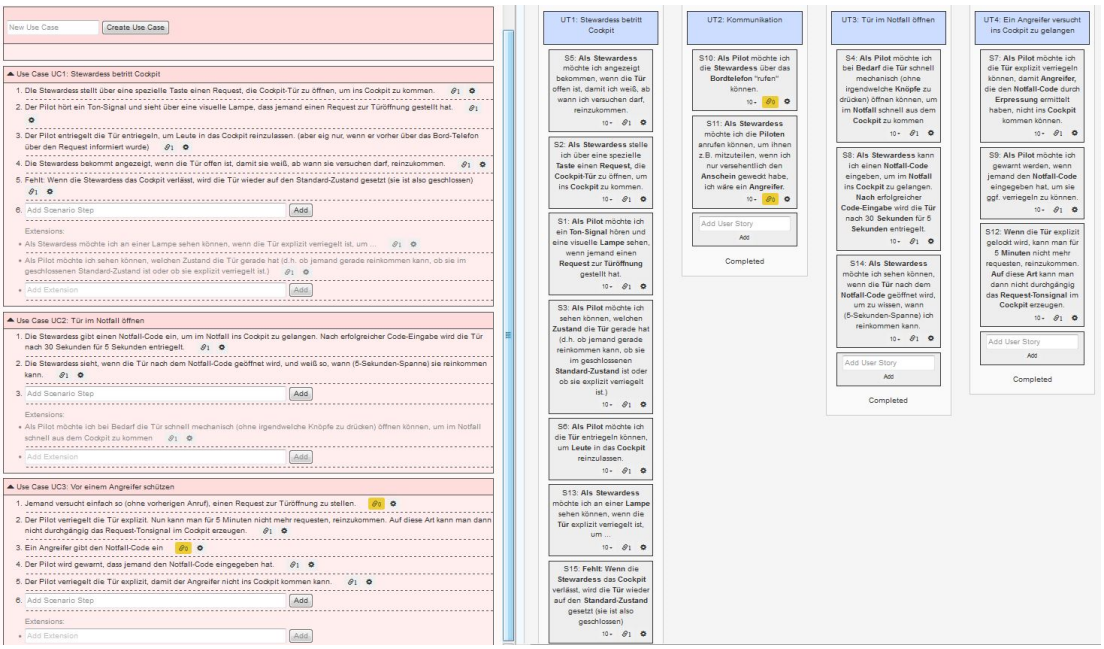

Abbildung 2.5.: Darstellung Use Cases(links) und User Stories(rechts) im Integrated Requirements Environment (IRE) [Ire16]

2.2.2. User Stories

Eine **User Story** oder auch *Story Card* ist eine aus Nutzersicht verfasste Absichtserklärung und beschreibt etwas, das das System für den Benutzer tun soll, damit dieser daraus einen Nutzen hat. User Stories sind also eine kurze und intuitive Technik, um Anforderungen an ein System zu erfassen.

Die Technik stammt aus dem *Extreme Programming Ansatz* von Beck [Bec00]. Dabei schreibt der Kunde seine Wünsche jeweils auf einer Karte, wobei es durch beschränkten Platz gewollt ist, dass nur 1-2 Sätze auf einer Karte stehen. Die User Stories werden gut lesbar zur Planung und Übersicht in *User Tasks* an einer großen Fläche, wie z.B. Whiteboard sortiert, das *Storyboard* genannt wird.

Eine User Story kann entweder formlos angelegt werden oder einem bestimmten Aufbau folgen. Ein Aufbau nach Leffingewell [Lef10] wäre:

> As an <*actor*>, I can <*activity*> so that <*business value*>.

Aber es kann auch eine verkürzte Variante [Wik1] verwendet werden:

Als <Rolle> möchte ich <Ziel/Wunsch>

Eine verkürzte Variante bietet sich dann immer an, wenn der *business value* trivial ist:
"Ich als Benutzer möchte mit dem Startknopf die Maschine einschalten *(, um die Maschine einzuschalten)*."
Für die agile Entwicklung besitzen die User Stories noch weitere Einträge wie Priorität, Risiko und Storypoints, die zur Planung verwendet werden. Auf Abb.2.6 ist eine solche User Story abgebildet.

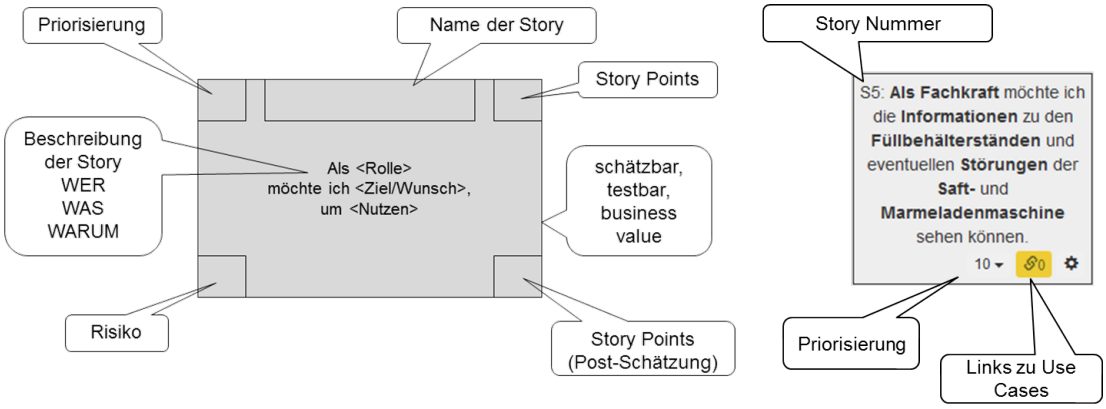

Abbildung 2.6.: Aufbau vollständigen User Story (links)[Ire16] und ein Beispiel der in dieser Arbeit verwendeten verkürzten User Story(rechts).

Die User Stories in dieser Arbeit unterscheiden sich von der Abb.2.6, da sie eine Kartennummer, Priorität und zusätzlich Verlinkungen haben. Mit den Verlinkungen können User Stories mit einem oder mehreren Use Case Schritten verbunden werden.

2.2.3. SML

Die Scenario Modelling Language ist eine eine formale und textuelle Sprache für Scenario Based Modelling [Gre16a].

Diese hat ein Objektmodell und eine System Specification, die in dieser Arbeit **SML Specification** genannt wird, um Verwechslungen zu vermeiden.
Die Objekte aus dem Objektmodell werden in der SML Specification als *control-*

lable und *uncontrollable* definiert und können Variablen und Funktionen besitzen. Objekte sind uncontrollable, wenn sie nicht von dem System kontrolliert werden können, sondern zur Umwelt (*Environment*)gehören. Hierzu zählen z.B. angrenzende Komponenten, die nicht Teil des Systems sind, oder Personen. Objekte, die nicht *uncontrollable* sind, müssen somit *controllable* sein und können vom System gesteuert werden.

Des Weiteren können in einer SML Specification beliebig viele *Collaborations* erstellt werden. In diesen können eng zusammenarbeitende Objekte an *Rollen* gebunden werden. Als Beispiel können auf einem Flugzeug sowohl Pilot und Copilot als auch die Stewardess durch den Lautsprecher zu den Passagieren sprechen. Da alle berechtigt sind den Lautsprecher zu verwenden, kann die Rolle "announcer" beliebigen Creqmitgliedern zugeordnet werden.
Es gibt *statische* und *dynamische* Rollen. Statische Rollen sind fest an ein Objekt gebunden. Dynamische Rollen können unterschiedlichen Objekten zugeordnet werden und werden erst während einer Aktivierung eines Scenarios fest an ein Objekt gebunden. Die Aktivierung eines Scenarios wird später in diesem Abschnitt erklärt.

Das Verhalten des Systems wird beschrieben, indem Objekte über Nachrichten genannte *Messages* kommunizieren. Ein Ablauf des Systems, der in SML als eine Folge von bestimmten Nachrichten dargestellt wird, heißt **Scenario**.

Wenn eine Nachricht auftaucht, die der ersten Nachricht eines Scenarios entspricht, so entsteht eine aktive Kopie dieses Scenarios, und die nächste Nachricht des Sceanrios kann eintreten, was in SML als *enabled* bezeichnet wird.

Messages können verschiedene Eigenschaften der Ausführung besitzen.
Ist eine Message *strict*, so darf die Reihenfolge dieser Message in diesem Scenario nicht verändert werden. Trotzdem können andere Messages auftreten, die nicht Teil dieses Scenarios sind. Geschieht dies trotzdem, so führt das zu einer *Systemviolation*. Ist eine Message nicht *strict* und die Reihenfolge der Message verletzt, wird das Scenario abgebrochen und es gibt es keine Systemviolation.

Ist eine Message *requested*, so darf diese Message nicht unausgeführt bleiben. Ist die Message nicht requested, kann sie unausgeführt und somit enabled bleiben. Die genaue Bedeutung, bis wann diese Message ausgeführt werden muss, wird weiter unten im Kapitel erläutert.

Es gibt auch *strict requested* Messages, die ausgeführt werden müssen und in

dieser Reihenfolge auftreten müssen.

Die kombinatorisch letzte Möglichkeit ist eine Message, die weder strict, noch requested ist. Diese Messages werden in dieser Arbeit als *normal* bezeichnet. Diese müssen nicht ausgeführt werden und brechen bei falscher Reihenfolge ab. Die erste Nachricht eines Scenarios ist immer eine normal Message. Bei Ausführung der SML Specification wird ein in Harels Buch [Har03] beschriebener Playout Algorithmus verwendet. Dabei verhält sich das System passiv, bis von dem Environment eine Message gesendet wird. Daraufhin führt das System solange alle enabeld *requested* Messages, bis es keine requested Message mehr enabled ist. Dies folge von Systemnachrichten wird auch als *Superstep* bezeichnet.

In SML das Verhalten des System in einem Specification Scenario modelliert. Ein Beispiel für ein Specification Scenario liefert der Ablauf einer Kaffeemaschine in Abb.2.7. Darunter ist zur Veranschaulichung der Ablauf in einer vergleichbaren grafischer Form durch erweiterte Sequenz Diagramme (hier LSC [Har03]) dargestellt.

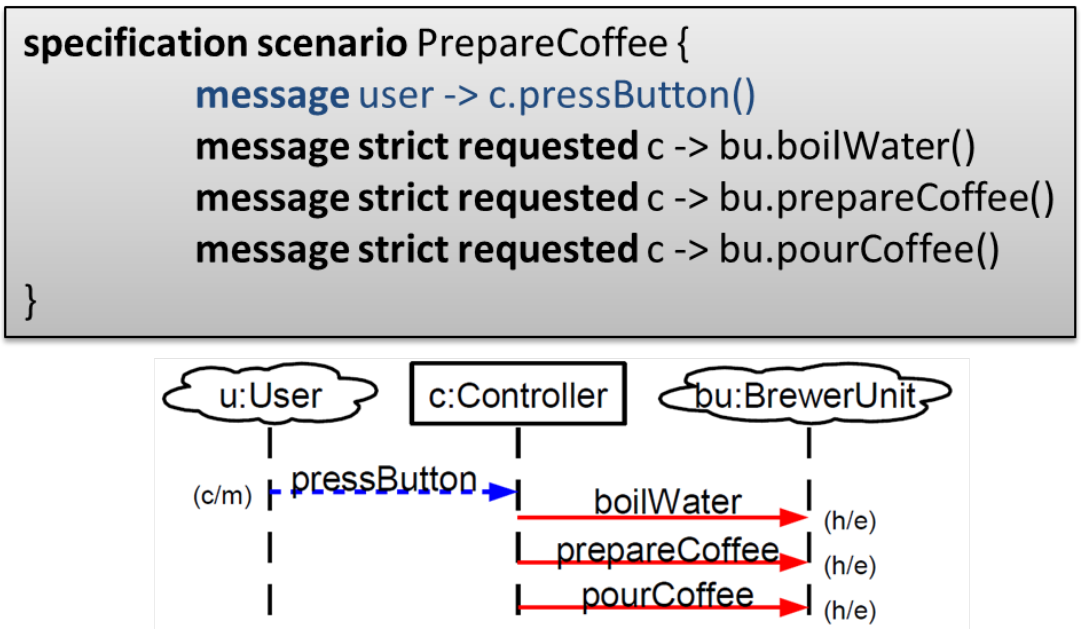

Abbildung 2.7.: Beispiel eines Specification Scenarios und einer vergleichbaren grafischen Notation (LSC)).

Der User betätigt den Knopf der Kaffeemaschine. Daraufhin sendet der Con-

```
1  assumption scenario takeCoffeeCup{
2    message c -> bu.pourCoffee()
3    message strict requested u -> bu.takeCup()
4  }
```

Listing 1: Assumption Scenario für das Entnehmen einer Kaffeetasse

troller der Kaffeemaschine die Nachricht ”Wasser kochen” an die Brüheinheit. Erst danach wird vom Controller das Signal für ”Kaffee vorbereiten”und erst danach ”Kaffee ausgeben” an die Brüheinheit übermittelt.

Mit SML ist es durch Assumption Scenario möglich, Annahmen über die Umwelt zu modellieren. Eine Annahme könnte auf die Kaffeemaschine bezogen sein, dass ein Benutzer nach der Kaffeeausgabe die gefüllte Kaffeetasse aus der Maschine entnehmen muss. Dies ist eine Annahme, da das System nicht garantieren kann, dass die Tasse wirklich entnommen wird, da sie auf das Verhalten Benutzers keinen Einfluss hat.

Der letzte in SML unterstützte Scenariotyp sind Requirement Scenario. Diese drücken eine spezielle Bedingung aus, die später eintreten muss, dies aber nicht im gleichen *Superstep* gefordert ist. Dies könnte für die Kaffemaschine z.B. die Bedingung sein, dass der Kunde irgendwann wieder Wasser in der Brüheinheit auffüllen muss.

Während die zuvor aufgelisteten Scenarios Abläufe beschreiben, die für alle Sequenzen von Messages gelten müssen, gibt es noch einen weiteren Ansatz. Diese Scenarios heißen Existential Scenarios und sind in SML noch nicht integriert. Harel [Har03] benutzt diese zum Testen von Specifications.

In dieser Arbeit wird SML in der an Abb.2.8 Entwicklungsumgebung Scenario-Tools verwendet. Ein Vorteil von SML und Scenario-Tools ist, dass das Modell jederzeit ausgeführt werden kann. Dort hat man die SML Specification mit dem momentanen Fortschritt der Scenarios, die momentan erlaubten Systemnachrichten und den Simulationsgraphen mit dem momentanen Zustand auf einen Blick.

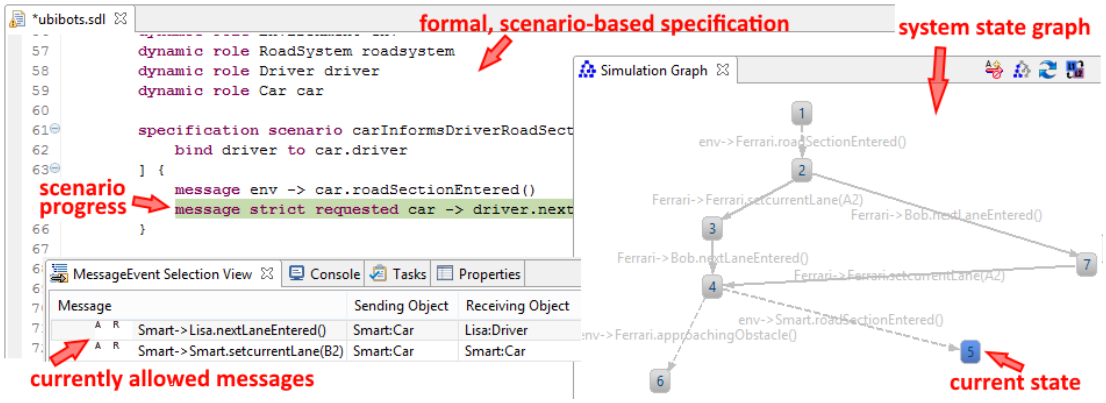

Abbildung 2.8.: Screenshot aus Scenariotools [Sce16].

2.3. Verwandte Arbeiten

Diese Arbeit befindet sich in zwei den Forschungsbereichen Anwendung und Verbesserung von Formalen Methoden und Verknüpfung und Überführung zwischen Anforderungsartefakten.

In dem Forschungsgebiet der Formalen Methoden ist besonders das des Scenario Based Modellings von Bedeutung, da dies in dieser Arbeit verwendet wird. In der Arbeit von Greenyer [Gre16b] wird untersucht, wie Entwickler mit Scenario Based Modelling bei der Erstellung einer Spezifikation eines cyper-phsyical-system unterstützt werden können. Das vorgestellte Tool dieses Artikels ist auch das in dieser Arbeit verwendete Scenario-Tool, welches an dem Projekt Car-to-X evaluiert wird. Das Car-to-X System umfasst autonome Fahrzeugroboter, die miteinander zusammenarbeiten müssen, um einem Straßenhindernis sicher auszuweichen und nicht auf der anderen Straßenseite mit einem Gegenfahrzeug zu kollidieren.

Michael Gordan [Gor64] beschreibt in seiner Arbeit, dass Kinder, die nicht im Umgang mit scenario-based programming geschult wurden und eine Programmieraufgabe lösen mussten, intuitiv Muster aus diesem verwenden. Er begründet dies damit, dass Menschen schon von klein auf lernen, anderen Menschen Abläufe zu erklären, sei es die Einleitung zum Bedienen einer Maschine oder die Zubereitung anhand eines Kochrezeptes. Daher scheint scenario-based programming in gewisser Weise eine natürliche Art der Programmierung für den Menschen zu sein.

Eine anderes naheliegendes Forschungsgebiet sind die Verknüpfung und Überführung zwischen Anforderungsartefakten, da auch in dieser Arbeit Artefakte ineinander überführt werden.
In der Arbeit von Liskin [Lis12] geht es um den Umgang und die Kommunikation von Anforderungsartefakten. Nicht alle Artefakte eignen sich gleich gut, um ganz bestimmte Informationen zu kommunizieren, und es wird in einer Studie untersucht welche Artefakte für Kunden, Manager, verschiedene Entwickler und andere Gruppen wichtig sind und wie die Kombination von anderen Artefakten die Kommunikation verbessern kann. Die Arbeit hat auch gezeigt, dass eine Verknüpfung von Requirements Artefakten oft aus Kostenaspekten und zu geringer Tool-Unterstützung abgelehnt wird und dass dort viel Verbesserungsmöglichkeit existiert.

3. Fallbeispiel: Orangenpresse

Dies ist ein fiktives Beispiel um die Möglichkeiten und den Ablauf einer Umsetzung zwischen User Stories und formalen Szenarien zu untersuchen. Das Fallbeispiel ist kein Musterablauf der in Kapitel 4 beschriebenen Methode, sondern zeigt vielmehr einen Entstehungsprozess zu dieser und beschreibt anschaulich das Erkennen und Lösen der auftretenden Probleme. Zusätzlich finden sich farbige Kästen im Text, die nicht Teil des Beispiels sind, sondern die Probleme und Erkenntnis kurz und allgemein zusammenfassen. In Abb.3 ist ein solches Ereigniskästchen exemplarisch dargestellt.

Schlussfolgerung: Titel der Schlussfolgerung

Problem	Beschreibung des vorliegenden Problems
Lösung	Beschreibung der Lösungsidee
Detail	Weitere Beispiele, Sonderfälle oder Details

3.1. Einleitung und Personas

In diesem Beispiel geht es um die Entwicklung einer industriellen Orangenpresse.
Der Kunde ist Herr Orangenmüller, der seine defekte Orangenpresse ersetzen will und dazu die Firma LTec42 aufsucht, die sich auf die Entwicklung landwirtschaftlicher Geräte spezialisiert hat. Herr Grünwald ist der zuständige Requirements Engineer und wird erstmalig durch den neuen Mitarbeiter und SML Modellierer Herrn Blauhaus unterstützt. Aus Abb.3.1 sind die Personas zu entnehmen. Das Beispiel beginnt mit dem ersten Kundengespräch, zwischen Herrn Orangenmüller und Herr Grünwald.

Abbildung 3.1.: Personas des Kunde und der Mitarbeitern

3.2. Kundengespräch und User Stories

Erste Kundenunterhaltung im Besprechungsraum von LTec42

Nach einer kurzen Begrüßung bittet Grünwald (G) Orangenmüller (O) seine Vorstellungen über das Produkt zu erläutern:

O Meine alte Orangenpresse macht Probleme, und ich möchte sie durch eine neue und größere Presse ersetzen. Sie soll ungefähr 3 m*2 m*4 m groß sein, wie alle anderen Maschinen auch orangefarbig sein, wie die vorherige mit dem Fließband sowie der Saft- und der Marmeladenmaschine zusammenarbeiten.

G Könnten sie einmal grob eine Skizze zeichnen, wie der Aufbau aussehen soll?

O zeichnet eine kleine Skizze 3.2 und erklärt, was die jeweiligen Anschlüsse bedeuten.

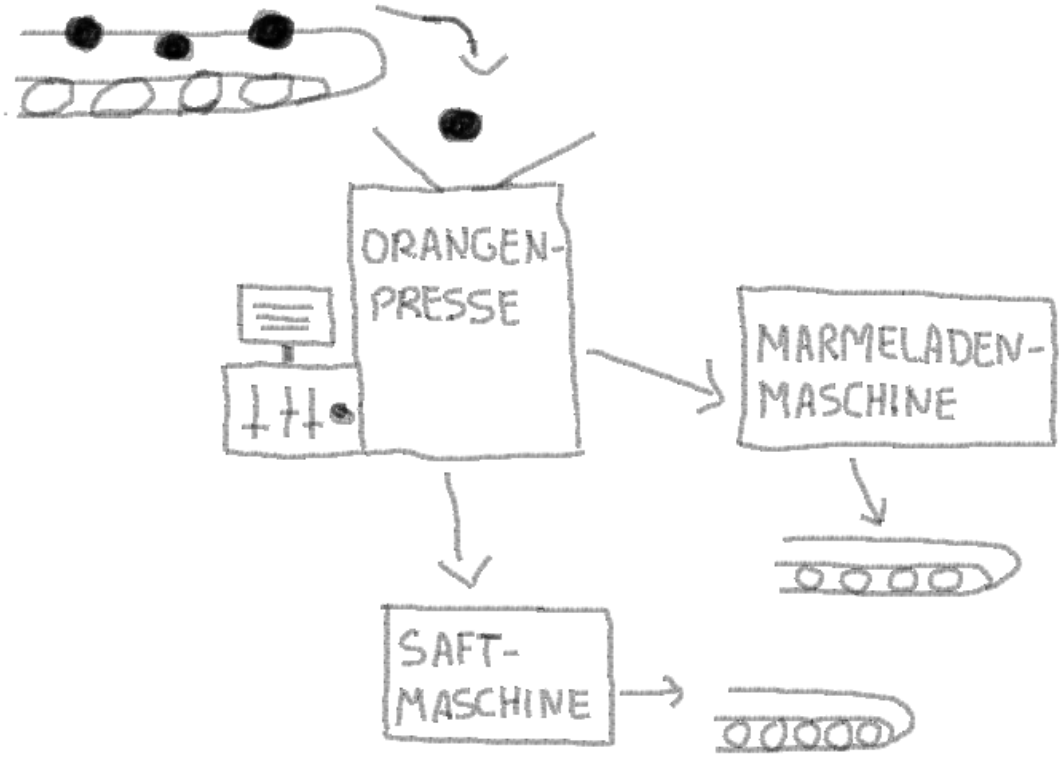

Abbildung 3.2.: Schematische Abbildung der Orangenpresse.

G Gut, schreiben wir nun einmal auf, wie der normale Arbeitsablauf aussehen sollte.

O Das ist eigentlich nicht viel, nach dem Anschalten läuft das meiste automatisiert ab. Wichtig ist, dass es einen Notausschalter gibt, der die Presse sofort anhält.

G Was gibt es denn für Gründe, um die Maschine anhalten zu wollen?

O Falls die anderen Maschinen Störungen melden, wenn z.B. bei der Saftmaschine die Glasflaschen leer sind. Oder aber auf dem Förderband mit den Orangen etwas drauf geraten ist, dass nicht in die Presse soll. Womöglich hat sich ein Kätzchen auf das Band geschlichen.

G Gut, halten wir das schriftlich fest.

G erstellt zusammen mit dem Kunden die Use Cases UC1 und UC2 Abb. 3.3.

Abbildung 3.3.: erstellte Use Cases UC1 und UC2

G Nun haben wir einen generellen Ablauf und eine Skizze. Lassen Sie uns jetzt über die Einzelheiten sprechen, wobei es zunächst die beiden Hauptbereiche ”Inbetriebnahme (Normalbetrieb)” und ”Abschalten im Notfall” gibt. Haben wir einen Teil vergessen?

O Ich glaube das ist fürs Erste alles.

G erstellt mit O die User Tasks UT1 und UT2 3.4.

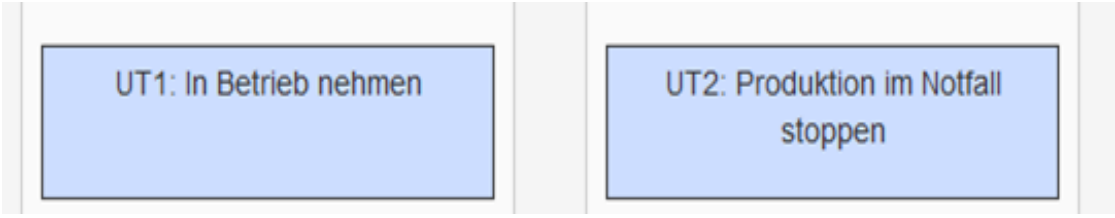

Abbildung 3.4.: erstellte User Task UT1 und UT2

G Gut, also fangen wir beim Anfang von UC1 (und UT1) an: Welche Parameter kann man über „die Regler“ aus UC1.1 alles einstellen? Oder gibt es vorher noch einen Einschalter für das Gerät?

O Einen An-Schalter braucht man eigentlich nicht. Wenn man auf Start drückt geht sie sofort an, sonst ist sie im Haltezustand. Und dann kann man an den Reglern die Pressdauer, Pressstärke und die Ablaufmenge einstellen.

G erstellt mit dem Kunden User Story S1 und S2 3.5.

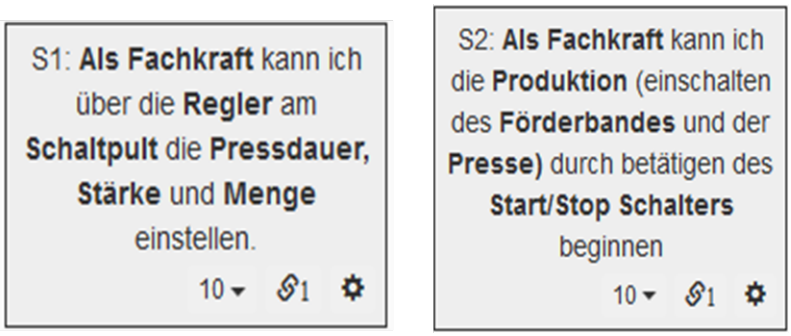

Abbildung 3.5.: User Story S1 und S2

O Und wenn die Warnlampe leuchtet, sollte die Maschine so schnell wie möglich abgeschaltet werden

G Das macht manuell die Fachkraft? Aber was ist, wenn sie kurz abgelenkt ist oder ein Kätzchen auf dem Band ist, das die Fachkraft nicht sieht, weil es zu gut versteckt ist?

O Hmm, daran habe ich noch nicht gedacht, und bis jetzt reichte das so! Vielleicht sollte man aber noch eine Sicherung einbringen.

G Ok, ich notiere mir das für später und wir machen uns darüber später Gedanken. Sie erwähnten ganz am Anfang, dass die Presse mit der Marmeladen- und Saftmaschine zusammenarbeiten soll und in UC1.3 steht, dass sich diese beiden Maschinen dann automatisch nach dem Anschalten der Presse aktivieren. Wie sieht denn nun diese Zusammenarbeit mit den anderen Maschinen aus? Haben Sie eine technische Dokumentation der beiden Maschinen?

O Die genauen technischen Details weiß ich nicht, aber bis jetzt war es so, dass die anderen Maschinen dann automatisch anfingen zu arbeiten, wenn die

Presse aktiviert wurde. Ich habe auch eine "dicke Bedienungsanleitung" für beide Maschinen, die lasse ich Ihnen zukommen. Was die Fachkraft auf einer Anzeige sehen soll, sind z.B. die Informationen zu den Füllbehälterständen.

G Dann lassen wir den Vorgang erst einmal so abstrakt und erweitern den "Zusammenarbeitsaspekt", wenn wir mehr Informationen haben.

G erstellt zusammen mit dem Kunden User Story S3, S4 und S5 3.6.

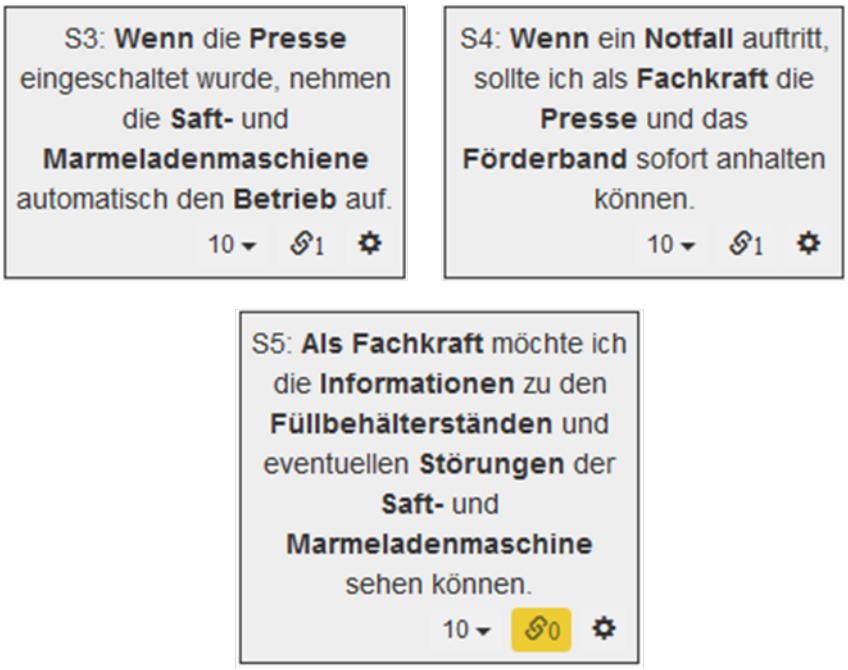

Abbildung 3.6.: User Story S3, S4 und S5

Schlussfolgerung: Generalisierung

Problem	Manchmal ist schon im Vorfeld bekannt, dass während des Erstellens einer User Story zu wenig Informationen vorliegen, um das genaue Verhalten abbilden zu können.
Lösung	Die Erstellung einer vereinfachten User Story, die stellvertretend für einen komplexen Ablauf steht. Dies wird dann auch in SML als vereinfachte Funktion abgebildet. Wenn die Informationen später vorhanden sind oder die Entscheidungen getroffen wurden, können vereinfachte Funktionen durch den komplexen Ablauf ersetzt werden oder diesen einleiten.
Detail	Eine Generalisierung kann auch gewollt sein, wenn es sich z.B. um eine Entscheidung handelt, die erst im Entwurf getroffen werden soll.

G Was halten Sie davon, wenn wir unseren Modellierer das Ganze modellieren lassen? So können Sie sich den gesamten Ablauf einmal in einer Simulation anschauen. Und wir können auch besser überlegen, welche Sicherheitsvorkehrungen sich für Sie gut eignen würden.

O Ok.

G Ich bringe das mal eben zu unserem Mitarbeiter und bin gleich wieder da.

Büro von Tomas Blauhaus (B)

G Hallo Tomas, ich hab mit dem Kunden ein paar Use Cases und User Stories erstellt. Reicht dir das für die SML Modellierung oder brauchst du noch etwas?

B Hi Jan. Tja, du hast noch nichts mit SML modelliert, oder? In der scenariobasierten Modellierung bilden wir Anforderungen an unser System und unsere Umwelt ab, indem wir Abläufe, also Scenarios beschreiben. Die sehen so aus, dass sich Objekte unterschiedliche Nachrichten zuschicken. Wenn das Modell genau werden soll, brauche ich auch die genauen Objekte und deren Funktionen, um sie zu modellieren. Das ist etwas komplizierter und wenn ich eine Karte falsch interpretiere, stimmt hinterher nichts mehr.

Besser du besprichst die Karten mit den Kunden nochmal genauer, bevor ich hinterher etwas raten muss.

G Ok, ich verstehe. Und raten ist ganz schlecht. Am besten du kommst einfach kurz mit und fragst ihn selbst, was du noch brauchst und wissen musst. Ich habe ja mit SML noch nicht gearbeitet und weiß nicht genau, was für dich wichtig ist.

Schlussfolgerung: Tags bilden Objekte

Problem	SML benötigt konkrete Datenobjekte, die sich Nachrichten schicken. Auf User Stories stehen diese Objekte oft verallgemeinert oder implizit, was nicht ausreicht oder zu Verwechslungen und Fehlern bei der Modellierung führen kann.
Lösung	Die User Stories werden mit dem Kunden noch einmal kurz besprochen und zu jeder Karte die dafür (vermuteten) benötigten Objekte als *tags* hinten auf die Karten geschrieben.
Detail	Es ist zu beachten, dass die Objekte manchmal nicht explizit auf einer Karte vom Kunden geschrieben stehen, sondern erst bei einer Diskussion dazukommen. Wird ein neues Objekt entdeckt, sollte es nachträglich in die User Story mit aufnehmen werden.

Besprechungsraum

Nach einer kurzen Vorstellung und Begrüßung

B Herr Orangenmüller, wie mein Kollege Ihnen schon gesagt hat, möchten wir ein detailliertes Modell von der Orangenpresse erstellen, das ihr Verhalten abbildet. Mit diesem Modell können wir dann eine Simulation starten, in der Sie sich ansehen können, ob das Verhalten der Presse nach Ihren Wünschen ist. Dafür müssen wir genau wissen, welche Objekte existieren und angesprochen werden. Nicht, dass der *Start/Stop Schalter* auf einmal ein anderes Förderband einschaltet. Wenn Sie jetzt etwas noch nicht genau wissen, ist es nicht schlimm, dann nehmen wir dafür Platzhalter und fragen Sie zu einem späteren Zeitpunkt noch einmal.

Bei S1 3.5: Fachkraft kann die Regler am Schaltpult bedienen. Können sie auch schon richtig eingestellt sein, d.h. muss man immer an den Reglern drehen, bevor wir die Presse aktivieren können? Soll man mit dem Schaltpult noch etwas anderes einstellen, oder sind nur die drei separaten Regler entscheidend?

O Ob die drei Regler und der Schalter an irgendeinem Schaltpult oder direkt an der Presse befestigt sind, ist eigentlich egal. Die Regler können schon voreingestellt sein, denn vielleicht hatte man nur angehalten weil die Orangen gerade ausgegangen sind und man auf neue wartet.

B Bei S2 3.5: Welche Förderbänder sind betroffen? Das obere Förderband sollten wir dann explizit "Orangenförderband" nennen, um es von den anderen zu unterscheiden. In der Skizze sind die Regler ja direkt an der Presse eingezeichnet. Dann ist also die Software der Presse dafür verantwortlich das Orangenförderband zu steuern, oder geht ein Kabel von den Schaltern direkt zum Orangenförderband? Je nachdem würde ich das anders modellieren, und so würden wir es, wenn Sie mit der Modellierung zufrieden sind, umsetzen.

O Bei der alten Maschine lief kein extra Kabel zum Orangenförderband. Das ging alles über die Presse. Die anderen Förderbänder werden auch von den dazugehörigen Maschinen, also Saft- und Marmeladenmaschine gesteuert.

G Also kann ich zu der User Story S1 3.7 als tags die drei *Schalter* für *Dauer*, *Stärke* und *Menge* aufschreiben.

B Und dazu kannst du auch den *Controller*, also die Software der Orangenpresse, aufschreiben.

O Und bei S2 den Start/Stop Schalter, das Orangenförderband und auch die Software ergänzen.

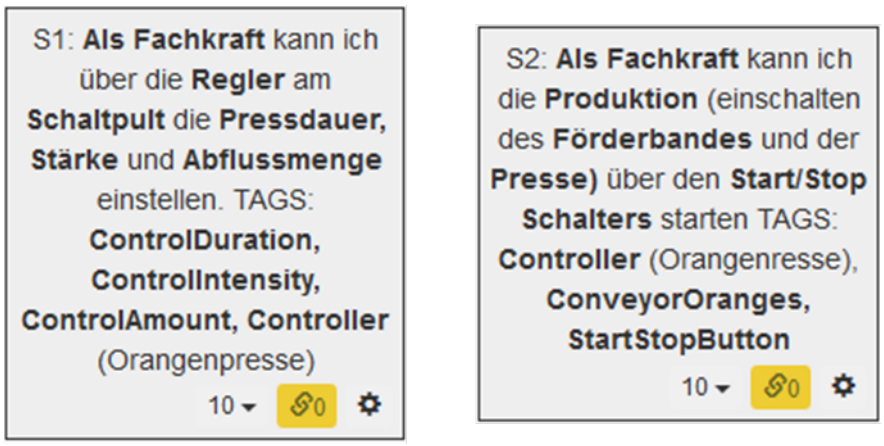

Abbildung 3.7.: User Story S1 und S2 mit Tags

B Ich habe noch zwei Frage zu S5: Wo sollen die Informationen angezeigt werden? Und wissen Sie zufällig, wie die Kommunikation zwischen den anderen Maschinen und der Presse aussieht?

O Angezeigt werden soll es auf einem normalen Display oder Bildschirm. Das mit der Kommunikation habe ich Herrn Grünwald schon gesagt, ist mir zu technisch da schick ich Ihnen die Anleitung der Maschinen von zu Hause zu.

O, G und B besprechen die übrigen User Stories und welche Objekte es wirklich gibt und schreiben diese als tags unter die Karten. Im Anschluss verabschiedet sich O und kehrt zu seiner Orangenfarm zurück. Daraufhin gehen G und B in das Büro von Blauhaus.

3.3. SML und Überführungstabelle

Büro von Tomas Blauhaus

B So, dann schlage ich vor, wir machen das jetzt beim ersten Mal zusammen. Es wird bestimmt noch ein paar offene Fragen geben und ich war ja nicht von Anfang an bei dem Kundengespräch dabei und das mit SML und User Stories ist ja für uns beide neu.

G Ok, na dann zeige mir erst einmal SML.

B Zunächst müssen wir die SML Objekte, die kommunizieren sollen, erstellen und Instanzen bilden. Dazu gucken wir nach allen tags von allen Karten und erstellen für jeden tag ein Objekt. Hier sollten wir uns einigen, dass es genau eine Instanz von den Objekten gibt. Wenn mehrere Instanzen vom selben Typ möglich sind, müssen auf den Karten auch zwei verschiedene *tags* stehen, z.B. *display1* und *display2*.

G und B erstellen ein SML Objekt für alle unterschiedlichen tags der Karten 3.8.

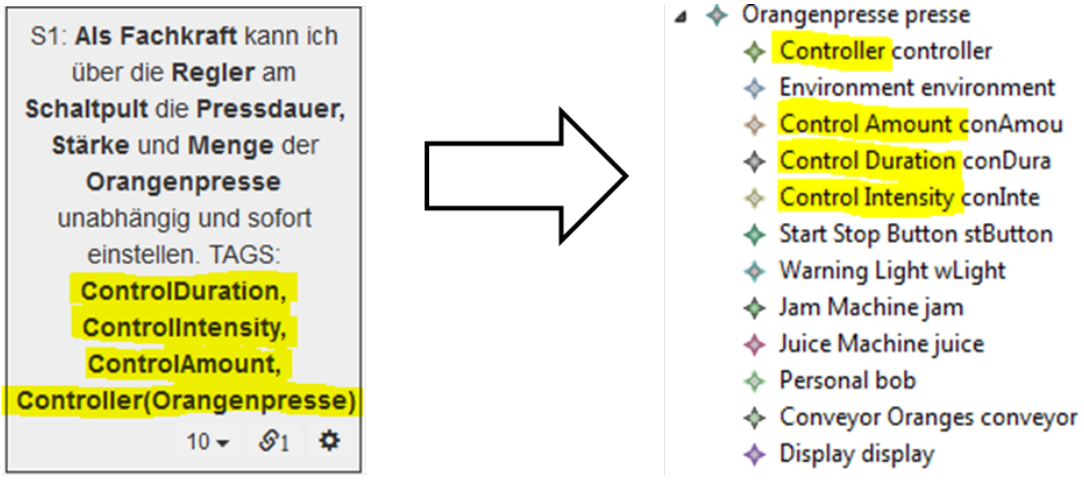

Abbildung 3.8.: B und G leiten aus tags die SML Objekte ab. Die gelben Markierungen veranschaulichen die aus den tags generierten SML Objekte

B Nun erstellen wir Instanzen und definieren passende Funktionen. So etwas wie *stButton.push()* oder *Controller.isRunning:=false*.

G Was sind "passende" Funktionen und wie kommt man von der User Story auf die Funktionen oder Variablen der Objekte?

B Naja, eigentlich kann man die Verben aus den Haupt- und Nebensätzen verwenden. Nehmen wir die Karte S4: Wenn die Warnleuchte blinkt oder ein anderer Notfall auftritt, sollte ich als Fachkraft die Presse und das Orangenförderband schnellstmöglich durch Betätigen des Start/Stop Schalters anhalten. Der Hauptsatz lautet: "*... sollte ich als Fachkraft die Presse ... anhalten*", also wäre die Funktion "*Presse.stop()*". Hier findet sich auch oft ein passiver Ablauf aus Nutzersicht, z.B. "die Fachkraft möchte informiert werden ...". Dies wäre *display.showInformation()*, da dies das Display aktiv

tun muss und der Nutzer passiv ist. Natürlich muss die Fachkraft auch hingucken, aber das ist ja nicht Teil unseres Systems. Und ob die Funktion *"showInformation()"* oder *"displayInformation()"* heißt, ist nicht entscheidend. Es sollte nur verdeutlicht werden, was getan werden soll, und kann später auch ersetzt werden.

G Und was ist mit *"... durch Betätigen des Schalters ..."*?

B Stimmt. Der Schalter wird zwar auch durch User Story S2 modelliert, aber es muss ja einheitlich sein. Nominalisierungen solltest du auflösen, also: Wenn die Warnleuchte blinkt oder ein anderer Notfall auftritt, sollte ich als Fachkraft die Presse und das Orangenförderband schnellstmöglich anhalten, indem ich den Start/Stop Schalter drücke.

Schlussfolgerung: Verben bilden Funktionen

Problem	Ohne Regelung ist es nicht immer klar, wie Funktionen oder Attribute der Objekte aus den User Stories heraus gelesen können oder welche benötigt werden.
Lösung	Eine Vorgehensweise ist, die Verben aus dem Haupt- und Nebensätzen als Funktionen für die entsprechenden Objekte zu verwenden. Dabei sollten Nominalisierungen und passive Verben aus Sicht des Benutzers als aktive Verben aus Objektsicht formuliert werden.
Detail	Beispiel: Als Fachkraft möchte ich, dass mir das Display Informationen anzeigt *anstatt*: Als Fachkraft möchte ich Informationen auf dem Display sehen können.

G und B erstellen Methoden und Variablen für die einzelnen Objekte im SML Modell.

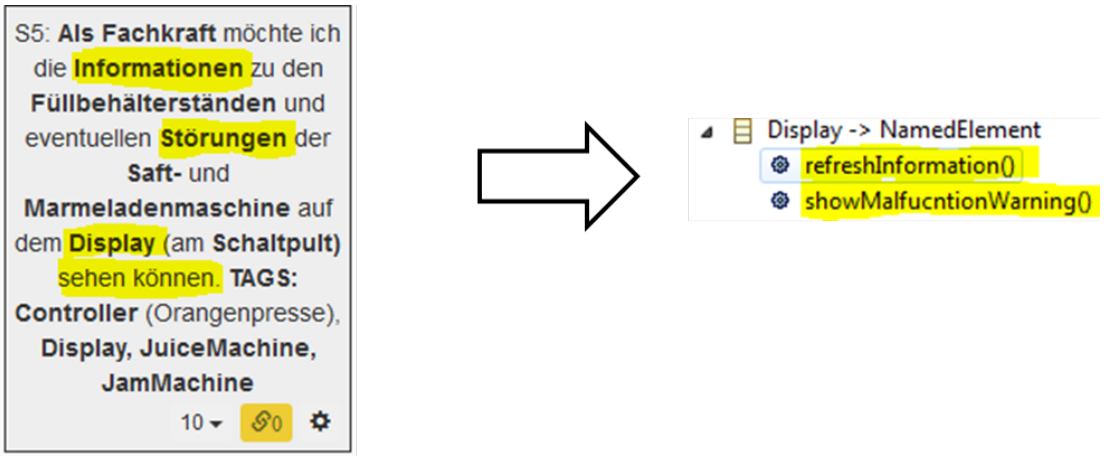

Abbildung 3.9.: Funktionen der Objekte ermitteln. Die gelben Markierungen zeigen die aus den Schlagwörtern (links) generierte Funktionen des Displays.

G Warum *refreshInformation()*?

B So wie die Karten geschrieben sind, muss die Fachkraft keine Interaktion vornehmen, um sich die Informationen anzeigen zu lassen. Daher werden die Informationen immer angezeigt und immer aktualisiert. Das ist natürlich schon etwas weiter gedacht. Und man könnte dies zur Sicherheit noch einmal mit dem Kunden absprechen. Im Moment machte diese Benennung aber noch keinen Unterschied, da es darauf ankommt, in welchem Zusammenhang die Nachricht verschickt wird.

G Und dann sind wir jetzt endlich soweit, dass wir Scenarios schreiben können?

B Richtig, jetzt kommt der schwere Teil: Also wenn alle Objekte einer Karte vorhanden sind, erstellen wir ein Scenario, in dem sich diese Objekte Nachrichten schicken. Es kann aber sein, dass eine User Story mehrere Scenarios erstellt, oder der schwierigere Fall, dass zu einem Scenario verschiedene Karten gehören.

G Und wie findet man das heraus?

B Lass uns die User Stories mit den Use Cases *verlinken*. Dadurch sehen wir alle User Stories, die im gleichen oder direkt angrenzenden Use Case Schritt auftreten. Die Use Cases stellen ja schon eine Reihenfolge von Ereignissen dar, und wenn die User Stories, die diese Use Cases genauer beschreiben, nicht im selben oder benachbarten Schritt auftauchen, ist es sehr unwahr-

scheinlich, dass die User Stories ein einziges Scenario beschreiben. Ob es dennoch Fälle geben kann, in denen dieses möglich ist, kann ich allerdings nicht ausschließen.

G Das Verlinken kann ich nächstes Mal direkt beim Erstellen der User Stories machen. Dann sieht man auch auf einen Blick, welche Schritte von einem Use Case noch mit keiner User Story beschrieben sind. Aber wir sollten uns in den nächsten Tagen noch einmal über den Fall Gedanken machen, wenn eine User Story nicht verlinkt ist, also in keinen Use Case passt, oder wenn ein Use Case Schritt keine User Story hat.

B Mir fällt gerade ein, dass wir zusätzlich über die tags nach den Objekten suchen können, um nachzusehen, welche Karten noch von dem Start/Stop Schalter beeinflusst werden.

Schlussfolgerung: Verlinkung von User Stories und Use Cases

Problem	Ein Scenario kann mehrere User Stories enthalten: Nicht immer ist ersichtlich, welche anderen User Stories an einem Scenario beteiligt sind.
Lösung	Verlinken der User Stories mit den Use Cases und prüfen, welche User Stories im selben oder direkt benachbarten Schritt vorkommen. Zusätzlich können Stories über tags gefiltert werden, um alle Karten, in denen z.B. ein Schalter auftaucht, zu finden.
Detail	Eine andere Möglichkeit ohne Verlinkung könnte darin bestehen, verwandte User Stories bei Erstellung direkt zu gruppieren (d.h. S1.1, S1.2 ...)

B Als nächstes schreiben wir, da es noch keine Tool-Unterstützung dafür gibt, die Nummern der User Stories und den Stakeholder einer User Story als Kommentar über das zu erstellende Scenario. Damit können wir später zurückverfolgen, welche Karte zu welchem Scenario gehört und wissen beim Lesen des Scenarios auch ohne auf die USer Stories zu schauen, welcher Stakeholder betroffen ist. Mit andere Worten: Wir verlinken die User Stories mit den Scenarios.

G Damit findest du dann auch schneller die Scenarios, die geändert werden

müssen, wenn der Kunde Änderungen bei den User Stories vornimmt. Aber reicht da nicht die User Story Nummer? Warum auch der Stakeholder?

B Das hilft mir nur beim Verständnis von den Scenarios, wenn ich sie in Zukunft lese oder ändern soll.

Schlussfolgerung: Verlinkung der Szenarien und User Stories

Problem	Ohne einen weiteren Mechanismus ist es nicht möglich, später zu entscheiden, welches Scenario von welcher User Story stammt. Gerade bei Anforderungsänderungen ist es schwierig alle beteiligten Scenarios effizient zu finden, die geändert werden müssen.
Lösung	Beim Erstellen oder Bearbeiten eines Scenarios werden die beteiligten User Story IDs als Kommentar über das Scenario geschrieben. So entsteht ein Link zu den dazugehörigen User Stories, und es können mit der Suche oder in Zukunft durch Tool-Unterstützung die entsprechenden User Stories und Scenarios gefunden werden. Die Verlinkung zwischen den Use Cases und User Stories existiert bereits im IRE Tool.

G Und wie wählst du den Scenarionamen?

B Der Name des Scenarios ist das Ziel (”goal”) der Karte, wenn die User Story nicht ein bestehendes Scenario erweitert. Bei anderen Projekten ist dies der Titel der User Story, aber bei unserer Variante muss ich das Ziel der Karte selbst herauslesen, da wir in unserem Betrieb ja mit dem IRE Tool arbeiten, das keine Titel hat, dafür aber Verlinkungen zwischen Use Cases und User Stories unterstützt. Das Ziel einer Karte zu erkennen ist aber nicht schwer.

G Erstellt denn eine User Story immer maximal ein Scenario?

B Es gibt auch User Stories, die mehrere Scenarios erstellen. Das passiert z.B. immer, wenn man eine Fallunterscheidung machen muss. Nehmen wir die erste User Story ”Die Fachkraft kann die Regler (Stärke, Menge und Dauer) unabhängig einstellen” .

Schlussfolgerung: Erleichtertes Verständnis durch Einfügen des Stakeholders

Problem	Scenarios mit ähnlichen Namen können sehr verwirren, da zum Teil Domänenwissen notwendig ist, um z.B. zu entscheiden, ob die Tür von innen oder außen geöffnet wird. Dies kann zu Fehlern führen und erschwert zusätzlich das Erstellen von User Storys oder Fragekarten.
Lösung	Durch das Einfügen des Stakeholders, als Kommentar im Scenario, einer User Story ist es leichter den Überblick zu erhalten, wer etwas tut oder für wen etwas getan wird. Natürlich sollte auch eine sinnvolle Namensgebung stattfinden, sodass die Scenarionamen im Notfall länger aber ausführlicher werden.

B Da man alle einzeln und unabhängig voneinander oder auch gar nicht einstellen kann, macht man hier eine Fallunterscheidung. Das gleiche kommt auch bei einer Sicherheitsabfrage vor. Ein Scenario für die Abfrage und dann ein Scenario für den positiven und negativen Ereignisfall. Die genauen Angaben dazu können natürlich wieder auf anderen Karten stehen.

G Ach so. Also damit ich nichts falsch verstehe: *Fachkraft* ist auch unsere Instanz der Fachkraft, *conAmou.setValue()* ist der Regler, der gerade den Wert verstellt und *c.update* ist der *controller*, also die Software unserer Presse, die den neuen Wert des Reglers übernimmt. Und die Pfeile symbolisieren Pfeile wie in einem Sequenzdiagramm.

Aber was bedeutet das *strict* und *requested*?

B *Strict* bedeutet, dass wenn die Nachricht *enabled* ist und keine andere Nachricht aus diesem Scenario kommen darf, sondern nur diese. Enabled bedeuted, dass die Nachricht als nächstes in dem Scenario kommen sollte. Nachrichten, die nicht in diesem Scenario enthalten sind, können trotzdem dazwischen auftreten, da sie dieses Scenario dann auch nicht beeinflussen. Ist eine Nachricht nicht *strict*, kann eine andere Nachricht auftauchen, die die Anforderungen nicht verletzt, allerdings das aktive Scenario abbricht.

Requested bedeutet, dass die Nachricht irgendwann ausgeführt werden

muss, also nicht mehr *enabled* sein darf, bevor die nächste Environment Nachricht eintrifft. Eine Nachricht, die nicht *requested* ist, muss nicht ausgeführt werden und kann beliebig lange *enabled* bleiben.

Bevor du fragst, gebe ich dir noch einmal eine kurze Einleitung zu Scenario basierter Modellierung: Es gibt *controllable* und *uncontrollable* Objekte. Also alles, was nicht zu unserem System gehört, ist Teil der Umgebung (*Environment*) und uncontrollable. Dazu gehört auch die Fachkraft. Wir können ja schließlich keinen Einfluss darauf nehmen, ob sie einen Knopf drückt oder doch einen Kaffee trinken geht. Die Presse ist controllable, da wir ihr Verhalten kontrollieren und genau vorschreiben. Der Algorithmus, um ein Modell zu prüfen ist der *Playout-Algorithmus*. In welchem das Environment Nachrichten schickt und das System solange Systemnachrichten verschicken darf, bis es zu einem akzeptierenden Zustand kommt, einem Zustand, in dem es keine requested Nachrichten mehr gibt. Daraufhin schickt das Environment wieder eine Nachricht und das System reagiert. Kann das System immer zu einem Endzustand gelangen, ist das System konstant ausführbar und Verklemmungen und unendliche Systemaufrufschleifen können immer vermieden werden.

G Und wo sieht man in den User Stories, ob eine Nachricht strict und/oder requested ist?

B Die erste Nachricht ist in einem Szenario immer eine *normal* (non_stict/ non_requested) Message, die entweder von dem Environment gesendet wird oder durch eine andere Systemnachricht eines anderen Szenarios erstellt wurde. Beinhaltet die User Story einen *Auslöser* oder eine *Bedingung*, ist dies die erste Nachricht. Falls das Szenario von einer Vorbedingung wie z.B. *PIN.isValid(true/false)* kommt, so ist das Ergebnis die erste Nachricht. Ob danach eine Nachricht strict oder requested ist, ist schwierig aus einem normalen Text herauszulesen, da es dort immer auf die Formulierung ankommt. Außerdem kann man mit SML auch Nebenläufigkeiten und Alternativen modellieren.

G Gut, dann lass uns doch eine kleine *Übersetzungstabelle* festlegen. Ich versuche, die Karten dann mit diesen Schlüsselwörtern aus der Tabelle auszudrücken, sodass du unmissverständlich daraus die konkreten Nachrichten ablesen kannst. Also wenn die Fachkraft *kann* oder *darf*, ist dies optional und eine

non_requested Message. Ohne diese Hilfsverben soll die Aktion dann auch irgendwann ausgeführt werden und die Message ist requested.

G Könnte man strict mit sofort darstellen? Also eine Nachricht muss sofort nach einer anderen kommen?

B Eigentlich kann man *sofort* in SML sehr schlecht abbilden, denn nach einer Environment Message können ja beliebig viele Nachrichten folgen, auch von ganz anderen Scenarios, bevor eine strict Nachricht kommen kann. Und da diese nicht requested ist, muss sie auch niemals ausgeführt werden.

Strictness selbst ist aus Anwendersicht schwer zu beschreiben und dafür fällt mir auch nichts ein. Es ist möglich, einzelne strict messages mit *A darf nicht vor B passieren* auszudrücken, aber auch hier ist der Vergleich nur zwischen A und B gezogen und gäbe es ein C, müsste man dies auch noch genauer beschreiben. So wäre das das ganze Satzkonstrukt sehr hässlich und nicht mehr auf einen Blick ersichtlich. Strict requested sagt aus, dass diese Nachricht in dieser Reihenfolge ausgeführt werden muss, sonst ist es eine Verletzung der Anforderungen. Drückt man z.B. den Startknopf, so erwartet man, dass dieses Signal auch bei dem Controller ankommt.

G Na das ist ja kompliziert. Was können wir denn sonst noch für Konventionen treffen?

B Also Parallelität können wir mit *gleichzeitig* beschreiben. Wenn ich beispielsweise den Schalter drücke, gehen gleichzeitig die Saft- und Marmeladenmaschine an. In SML würde man das mit *"parallel{ message...}and{ message...}"* ausdrücken. Alternativen können als *"entweder ..., oder ..."* dargestellt werden. Hier sollten dann auch beide Fälle abgebildet sein. Ein Lichtschalter kann z.B. an oder aus sein, was man mit drei Scenarios realisieren könnte: Das erste überprüft und leitet beide Alternativen ein: der *Licht an* Fall und der *Licht aus* Fall. In SML wäre das *"alternative{message...}or{ message...}"*. In unserem Fall haben wir Regler, die wir *unabhängig* bedienen können und deswegen verwenden wir kein *"alternative"*, sondern wir erstellen ein eigenes Scenario für jeden Schalter. Bedingungen mit *"wenn ..., dann ..."* bilden immer die erste Message eines Scenarios. Ansonsten können wir *indem* schreiben, wenn es eine weitere Bedingung ist. Außerdem sollen alle tags auf der Karte an mindestens einer Message beteiligt sein, da sie sonst ja nicht im Scenario auftauchen würden.

G und B stellen gemeinsam die besprochenen Regeln in einer Überführungstabelle auf.

Schlussfolgerung: Überführungstabelle

Ausdruck auf User Story	**Bedeutung in SML**
jemand/etwas **"kann/darf"**	non_ Strict und non_ Requested (oft erste Message eines Scenarios)
Verb **ohne** "kann/darf"	requested
1 und 2 **"gleichzeitig"**	parallel [message1] and [message2]
"entweder 1..., oder 2"	alternative [message1] or [message2] kann bei Fallunterscheidungen andere Scenarios einleiten
"wenn 1, ... dann 2"	Bilden Vorbedingung: also einleitende non_strict, non_requested message 1 (Auslöser), danach message 2 (oft erste Message eines Scenarios)
kann **"unabhängig"** getan werden	Indikator für Erstellung mehrere (ähnlicher) Scenarien
1 passiert, **indem** 2 passiert'	2 ist Vorbedingung für 1 aber nicht Auslöser des Scenarios
jemand/etwas **"könnte/sollte/müsste"**	Regel für Assumption Scenario

Schlussfolgerung: Erstellen mehrerer Scenarios aus einer User Story

Problem	Eine User Story kann mehrere Scenarios erstellen. Es ist nicht immer sofort ersichtlich, wann mehrere Scenarios aus einer User Story erstellt werden.
Lösung	Die Grammatik hilft, gängige Verzweigungen leichter zu erkennen.

Schlussfolgerung: Erkennen von Message Typen

Problem	Mit einer normal geschriebenen User Story ist für den SML Modellierer nicht immer ersichtlich, ob eine Nachricht stattfinden muss oder darf (strict, requested, forbidden).
Lösung	Die Grammatik hilft auch hier, um gerade bei requested und normal Messages Probleme zu vermeiden.
Detail	Strictness ist oft schwer zu erkennen, da es nicht immer so explizit auf einer User Story steht.

3.4. Die Erstellung von Scenarios aus User Stories

G Lass uns noch einmal ein paar User Stories beispielhaft durchgehen und SML Scenarios erstellen. Nehmen wir als Beispiel User Story S2 3.5, ich würde die User Story so umschreiben:

> Als Fachkraft kann ich den Start/Stop Schalter drücken, um die Orangenpresse (wenn die Presse still steht) und das Orangenförderband zu starten. TAGS: Controller (Orangenpresse), ConveyorOranges, StartStopButton

Könnte ich sie auch so formulieren?

> Als Fachkraft starte ich die Orangenpresse und das Orangenförderband, wenn ich den Start/Stop Schalter drücke (und die Presse still steht). TAGS: Controller (Orangenpresse), ConveyorOranges, StartStopButton

B Ich denke, es geht beides. In beiden Fällen ist es die *Fachkraft*, die erst den *Star/tStop Schalter* **drücken** muss, um die *Presse* und das *Orangenförderband* zu **starten**. "Wenn" ist nach der Regel der Tabelle eine einleitende Nachricht und dasselbe bezweckt in diesem Fall das "kann". Aber da es ja ein Start/Stop Schalter zum Drücken, also eigentlich ein Knopf, ist und es darauf ankommt, ob die Presse läuft oder stillsteht, erstelle ich zwei

```
1     //Sth: Fachkraft
2     //User Story: S2
3     specification scenario pushButton{
4       message fachkraft -> stButton.push()
5       alternative if [stButton.isRunning == false] {
6         message strict requested stButton -> stButton.
              setIsRunning(true)
7       }or if [stButton.isRunning == true] {
8         message strict requested stButton -> stButton.
              setIsRunning(false)
9       }
10    }
11    //Sth: Fachkraft
12    //User Story: S2
13    specification scenario startProduction{
14      message stButton -> stButton.setIsRunning(true)
15      message strict requested c -> c.start()
16      message strict requested c -> conveyor.start()
17    }
```

Listing 2: Umsetzung der User Story S2 in SML

Scenarios. Eines für das Drücken des Knopfes mit einer Verzweigung, die beide Fälle betrachtet und ein Scenario für den Erfolgsfall. Ach, und als Stakeholder schreibe ich *"fachkraft"* für Fachpersonal und bei User Story natürlich S2 2.

G Dann können wir die User Story S3 fast so lassen. Die Schlüsselwörter "wenn" und "gleichzeitig" sind bereits auf der Karte vorhanden. Ich würde statt "Betrieb aufnehmen" auch "starten" aufschreiben und ein "dann" einfügen 3.10.

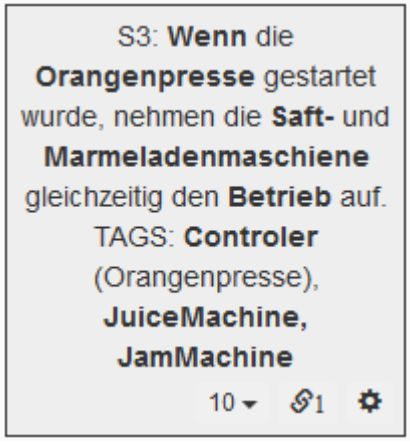

Abbildung 3.10.: Enhanced User Story S3

```
1   //Sth: Fachkraft
2   //User Story: S3
3   specification scenario aktivateJamAndJuice{
4     message c -> c.start()
5     parallel{
6       message strict requested c -> juice.start()
7     }and{
8       message strict requested c -> jam.start()
9     }
10  }
```

Listing 3: Umsetzung der User Story S3 in SML

B Dieses Scenario würde ich dazu erstellen 3. Stakeholder und Storycard sind klar, dann: ”Wenn” ist das Schlüsselwort für den Auslöser, also die erste Nachricht. Wenn die Orangenpresse also *c* eingeschaltet wurde, also *c.start()*, ”dann” starten ”gleichzeitig” also *parallel* die Saft- und Marmeladenmaschine *juice.start()* und *jam.start()*.

G Nun haben wir es geschafft und eine User Story modelliert. Wir sollten diese User Story als ”modelliert” kennzeichnen.

B Das stimmt, aber wir sollten auch Karten markieren, bei denen es noch offene Fragen gibt zu denen wir weitere Informationen von Orangenmüller benötigen. Dazu schreibe ich noch eine ”Fragenkarte”, die du mit Orangenmüller besprichst. Allerdings können wir versuchen eine User Story

umzusetzen ohne mit Orangenmüller gesprochen zu haben, wenn die Bedeutung der Karte eigentlich ersichtlich ist. Diese User Story markieren wir als "geändert". Auch diese besprichst du mit Orangenmüller.

Markierung des Bearbeitungsstatus

Problem	Im späteren Verlauf ist es nicht auf einen Blick ersichtlich, welche User Stories komplett umgesetzt, teilweise umgesetzt oder auf Grund von offenen Fragen nicht umgesetzt werden konnten und welche noch gar nicht bearbeitet wurden.
Lösung	Bearbeitete User Stories werden als modelliert, modelliert mit Änderungsvorschlag oder nicht bearbeitet markiert. Außerdem können bei Unklarheiten neue User Stories als Fragekarten entstehen, die mit dem Kunden besprochen werden müssen.

G Das ist eine gute Idee, also markieren wir die Karten entweder als *not_modeled* und *modeled*, für nicht bearbeitete oder fertige User Stories. *Changed* steht für bearbeitete User Stories mit Fragen oder Änderungsvorschlägen und *information_required* für Karten, die aufgrund von zu wenig Informationen nicht bearbeitet werden können.

B Wobei ich nicht der Meinung bin, dass man alle Fragen in Form von User Stories stellen sollte. Gerade sehr technische Details sind vielleicht besser in einer Tabelle oder sehr fein granularen Anforderungen auf einer handelsüblichen DinA4 Seite aufgehoben.

G Das stimmt. Hauptsache man schreibt sie auf und bespricht sie mit dem Kunden. Ob man sie später in kleine User Stories aufteilt oder dies zur Schnittstellenbeschreibung der externen Komponenten gehört, ist nicht so entscheidend. Es ist nur wichtig, dass man später in seinen Scenarios darauf verweisen kann.

B Und nachdem Fragen und Anregungen geklärt sind, mache ich aus diesen genaueren Karten wieder neue Scenarios oder verändere die zuvor erstellten.

G Ich habe aber noch drei allgemeine Fragen: Was machen wir mit nicht funktionalen Anforderungen? So etwas wie *die Maschine soll wie alles andere in*

der Halle orange sein können wir ja schlecht modellieren oder?

B Das stimmt, so etwas, oder auch *leichte Bedienbarkeit* etc. können wir mit unserem Modell auch überhaupt nicht überprüfen. Das ist eher etwas für den Entwurf.

Nicht-Funktionale Anforderungen

Problem	Nicht-funktionale Anforderungen können nicht modelliert werden.
Lösung	Sie werden nicht mitmodelliert.

G Und das Zweite ist, die *User Tasks* und *Collaborations* haben wir jetzt gar nicht betrachtet. Bringen dir die etwas?

B Nur das, was sie dir auch bringen: Sie gliedern die Karten und helfen ein wenig dem Verständnis und der Orientierung, aber sie erhalten keine Zusatzinformationen. Da die User Tasks die Karten in gröbere Ziele und die Collaborations Verhalten von bestimmten Systemteilen gliedert, weiss ich nicht, wie man das sinnvoll verbinden kann. Theoretisch können wir alle User Stories und Scenarios in einer User Task bzw. einer Collaboration schreiben.

Collaborations und User Tasks

Problem	User Tasks und Collaborations können nicht gegenseitig voneinander abgebildet werden.
Lösung	Diese sind für eine erfolgreiche Umsetzung nicht erforderlich, da sie keine zusätzlichen Informationen enthalten, sondern nur die Orientierung erleichtern. Also werden sie, so verwendet, wie es den Beteiligten am besten hilft und sind nicht fester Bestandteil der Methode.

G Letzte Frage: Warum schreibst du überall *static role*? Was sind bei dir im SML Rollen und können die auch dynamisch sein?

B Ein Objekt kann statisch oder dynamisch gebunden sein. Wenn ein Objekt also immer das Gleiche tut und nicht auf einmal eine andere „*Rolle*“ übernimmt, ist es statisch. Eine Ampel macht immer das Gleiche und ist deshalb statisch. Ein Jäger kann z.B. ein Hund oder eine Katze sein und die Beute

kann eine Maus oder eine Katze sein. Gibt es nur Katze und Maus, bleibt die Rolle des Jägers und des Gejagten gleich. Kommt der Hund in das Modell hinzu, und die Rollen des Jägers und der Beute sollen beibehalten bleiben, so würde man sie in diesem Kontext dynamisch binden. Bei unserer Presse ist aber alles dynamisch. Ob man diese Beziehung aus den User Stories ablesen kann, müssen wir uns später noch einmal überlegen.

B Ok, dann, erstellst du jetzt die restlichen erweiterten User Stories und schickst sie mir. Ich erstelle dann für die restlichen User Stories Scenarios oder, wie besprochen, Fragekarten, die du dann beim nächsten Treffen mit Orangenmüller besprichst.

G Mach ich, bis dann.

Nach zwei Tagen kommt O zum zweiten Kundengespräch. Dort spielt er mit G die soweit fertiggestellte Simulation durch. G macht dazu Notizen, bespricht danach mit O die Fragekarten von B und ändert die entsprechend markierten Karten. Damit endet das zweite Kundengespräch.

4. Methoden- und Modellentwicklung

In diesem Kapitel werden die Herausforderungen einer Umsetzung zwischen User Stories und SML sowie die entwickelte Methode und das daraus resultierende Artefaktmodell ausführlich beschrieben.

4.1. Modell der vorhandenen Artefakte

In Kapitel 2 wurden die hier verwendeten User Stories und SML im Allgemeinen beschrieben. Um beide gegenseitig ineinander überführen zu können, wurden zunächst die Artefakte eines Storyboards und einer SML Specification untersucht und in einem in Abb.4.1 Klassendiagramm dargestellt. Damit soll veranschaulicht werden, welche Informationen zur Verfügung stehen und welche benötigt werden. Zur besseren Lesbarkeit sind die zum Storyboard gehörenden Klassen in weiß und die zur SML Specification gehörenden Klassen in grau dargestellt. Die blauen Linien sind eine eigene Erweiterung zum Standard-UML und zeigen, wie die beiden getrennten Klassen des Storyboards und der SML Specification in Zusammenhang stehen.

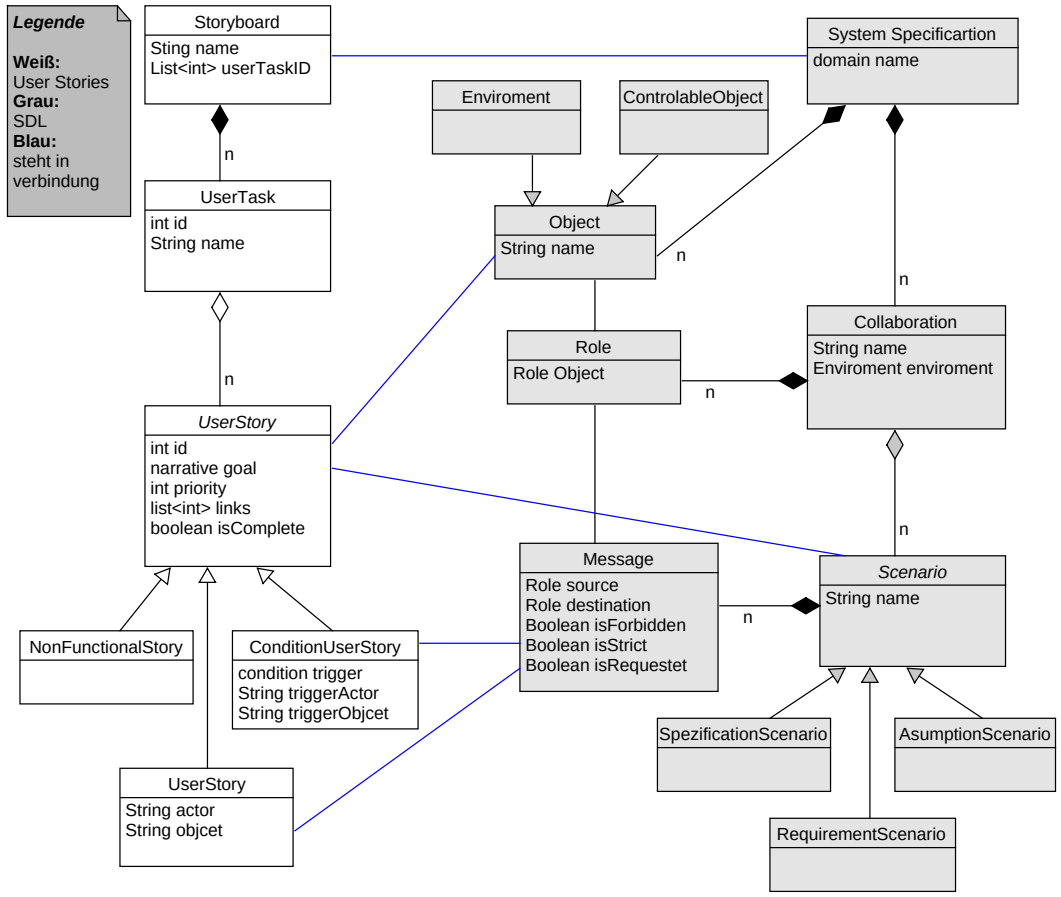

Abbildung 4.1.: Klassendiagramm eines Stroyboards und einer SML Specification

Ein Storyboard besteht aus mehreren User Tasks, die wiederum beliebig viele User Stories enthalten. Es konnten drei verschiedene Arten von User Stories identifiziert werden, wobei sich diese ein wenig von dem *Standardschema* [Lef10] (siehe hierzu Kapitel 2) unterscheiden.

UserStory User Stories, die nach dem Standardschema aufgebaut sind.

ConditionUserStory User Stories, die mit einer Bedingung beginnen und anschließend dem Standardschema folgen.

NonFunctionalUserStory User Stories, mit denen nicht funktionale Anforderungen aufgenommen werden und nicht dem Standardschema entsprechen.

Eine SML Specification kann beliebig viele Collaborations beinhalten, in denen

beliebig viele Szenarien enthalten sind. Daneben hat SML auch noch ein Objektmodell, so dass in einem Scenario beliebig viele Messages von einem Objekt zu einem anderen verschickt werden. Das Objekt kann an unterschiedliche Rollen gebunden werden. Da in dem Beispiel aus Kapitel 2 nur statische Rollen verwendet werden, wird jedes Objekt nur an eine einzige Rolle gebunden. Die Objekte selbst lassen sich in *Environment* (uncontrollable) und *Controllable* einteilen und werden innerhalb der Collaborations an Rollen gebunden. Eine Message hat immer ein Start- und Zielobjekt und kann zudem die Attribute *forbidden*, *strict* und *requested* haben. Wie auch bei den User Stories, lassen sich die Scenarios in drei verschiedene Arten unterteilen:

SpecificationScenario Scenario, das das Systemverhalten beschreibt.

AssumptionScenario Scenario, das eine Annahme über die Umgebung beschreibt.

RequirementScenario Scenario, das eine spezielle Bedingung ausdrückt, die später eintreten muss, dies aber nicht im gleichen *Superstep* tun muss.

Die blauen Linien verbinden einige Klassen des Storyboards und der SML Specification und sollen zeigen, dass es zwischen diesen Klassen eine Beziehung geben muss:
Das Storyboard und die Specification weisen denselben Namen auf. Da in dem Storyboard, bis auf die User Tasks, nur User Stories zur Verfügung stehen, müssen alle Informationen (Objekte, Verlauf und Typ der Message) für eine SML Specification aus diesen gewonnen werden. Umgekehrt gibt es in SML ein Objektmodell, aber die Informationen *Akteur*, *Nutzen* oder *Aktivität* müssen in das Scenario bzw. in die einzelnen Messages einfließen. Soll in umgekehrter Richtung aus einem Scenario eine User Story erstellt werden, müssen alle wichtigen Informationen wie Objects, Messages und Scenarionamen in eine User Story einfließen, da diese Informationen sonst verloren gehen. Die dabei auftretenden Probleme werden im nächsten Unterkapitel behandelt.

Eine Eigenschaft, die aus dem Klassendiagramm abgeleitet werden kann, ist, dass User Tasks und Collaborations keine Verbindungen haben. Diese sorgen für die Strukturierung und Übersicht der User Stories bzw. Scenarios, enthalten aber keine extra Informationen[1] und helfen nicht bei der Umsetzung in die jeweils andere Form. In dem beschriebenen Beispiel können Collaborations aber

[1] Collaborations enthalten für dynamische Rollen wichtige Informationen zu der Rollenbindung.

vernachlässigt werden, da lediglich eine benötigt wird in der alle Rollen statisch gebunden werden. Auch die Attribute *isComplete* und *priority* sind nur für die agile Planung sowie Entwicklung notwendig und werden für eine Modellierung nicht benötigt.

4.2. Herausforderungen

Die Methode, die in diesem Kapitel erarbeitet wird, konzentriert sich auf das Systemengineering, System Requirements und das sich auf einer abstrakten Stufe befindliche SML Modell.

Hier folgt eine Zusammenfassung der Probleme, die schon im Beispiel der Orangenpresse behandelt wurden:

Um Scenarios erstellen zu können und eine Message zu schreiben, werden die genauen Objekte, die sich gegenseitig Nachrichten schicken, benötigt. Auf normalen User Stories stehen diese nicht immer explizit beschrieben und der Modellierer müsste bei einigen User Stories raten, was ohne Domänenwissen zu Fehlern führen kann.

Die Funktionen der Objekte müssen aus den User Stories entnommen werden. Was der Modellierer meist intuitiv aus den Karten entnehmen kann, ist allerdings ohne Domänenwissen und ohne Regeln nicht immer einfach abzuleiten.

Eine User Story kann entweder ein oder mehrere Scenarios erstellen oder ein bestehendes erweitern.

Es gibt keinen Mechanismus, um später nachzuvollziehen, welches Scenario zu welcher User Story gehört und welche User Stories schon modelliert wurden. Dies macht ein *Tracing* sehr schwierig.

Soll aus einem Scenario eine User Story erstellt werden, wird ein Akteur, bzw. Stakeholder benötigt. In manchen Scenarios ist dies ohne Hintergrundwissen nicht leicht zu entscheiden.

Ohne Konventionen ist es schwierig, die Art der Message (strict/re-

quested), die Art des Scenarios (Specification/Assumption) oder die Programmiermöglichkeiten wie Bedingungen, Verzweigungen und Nebenläufigkeit einheitlich zu erkennen.

4.3. Methode zur Umsetzung zwischen User Stories und SML

Die Methode besteht aus drei Aktivitätsblöcken, die iterativ ausgeführt werden, wobei sie selbst keine agile Technik ist. Dabei wird versucht möglichst viele Anforderungen mit dem Kunden zu erheben und diese dann bis zum nächsten Kundentermin in SML Scenarios umzuwandeln, um gemeinsam mit dem Kunden festgestellte Unklarheiten aufzulösen.

Im Folgenden werden die einzelnen Aktivitäten der Methode 4.2 erläutert.

Create Enhanced User Stories

Diese Aktivity findet in einem Kundengespräch statt und es werden mit dem Kunden zusammen Anforderungen in Form von Use Cases und User Stories zusammengestellt. Die Anforderungen werden dann allerdings noch einmal in *Enhanced User Stories* umgewandelt, damit der SML Modellierer mit diesen User Stories effektiv arbeiten kann.

Create Enhanced Scenarios

In dieser Aktivität werden die erweiterten User Stories genutzt, um anhand bestimmter *Übersetzungsregeln* 4.3.2 aus den einzelnen User Stories zunächst die Objekte mit ihren Methoden und anschließend Scenarios zu erstellen. Außerdem werden die User Stories und Stakeholder für jedes Scenario vermerkt, um *Tracing* zu ermöglichen.

Mark and validate User Stories

In dem letzten Schritt der Methode werden die bearbeiteten User Stories markiert, Fragen oder Änderungsvorschläge bezüglich der Unklarheiten von User

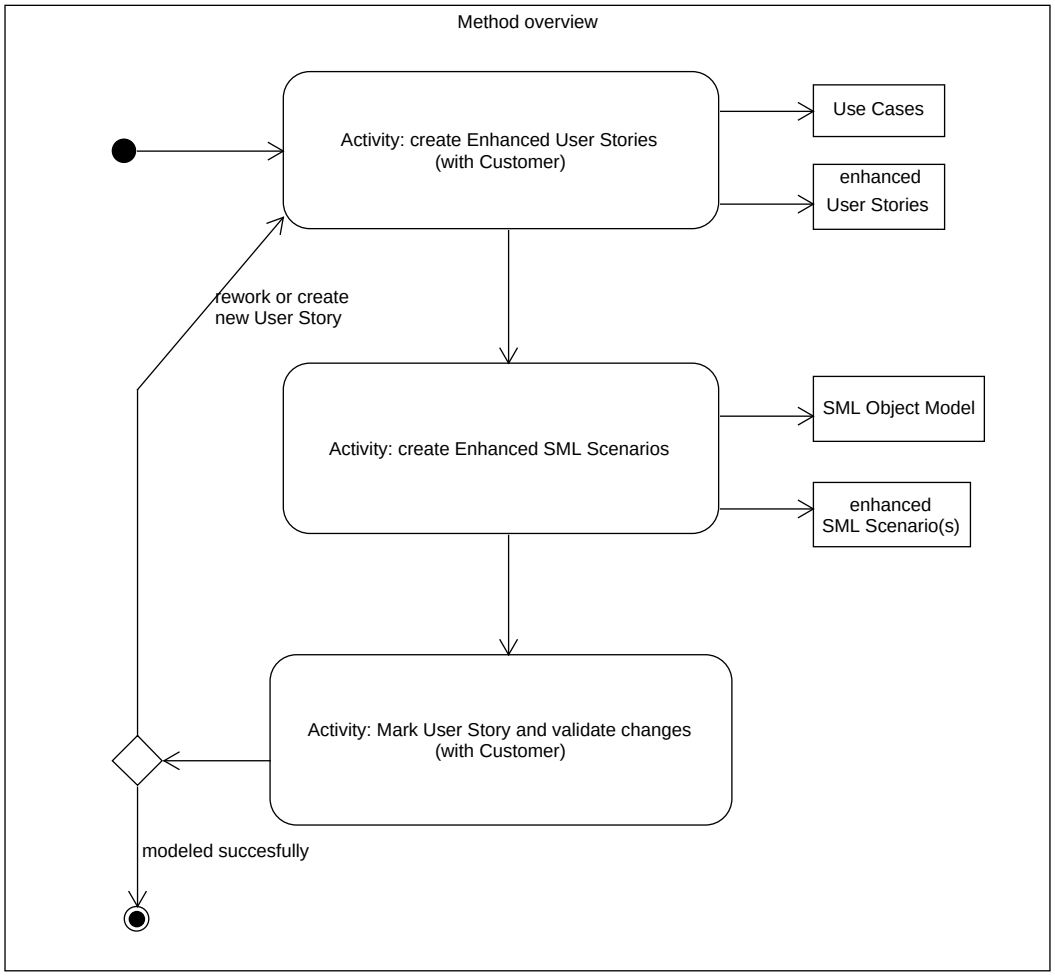

Abbildung 4.2.: Übersicht der vorgestellten Methode

Stories erstellt und mit dem Kunden besprochen.

Nach der Validierung können anhand neuer Informationen neue User Stories entstehen oder Änderungen an vorhandenen Stories vorgenommen werden, sodass daraus im Anschluss wieder erweiterte User Stories erstellt und in SML umgesetzt werden können.

4.3.1. Erstellen von Enhanced User Stories

Die Activity *creating Enhanced User Stories* 4.3 beginnt mit dem Sammeln von Anforderungen der Umgebung (*describe Environment*).

describe Environment

Der Kunde beschreibt, was für eine Art System er entwickelt haben möchte, wer es benutzen soll und wie die Umgebung und angrenzenden Systeme aussehen. Dies wird von dem Requirements Engineer informell aufgeschrieben, der somit einen ersten Überblick bekommt und sein Domänenwissen verbessert. Gleichzeitig werden Quellen wie Bilder, Skizzen oder Beschreibungen der verwendeten externen Systeme gesammelt. Wie auch in dem Beispiel der Orangenpresse kann es vorkommen, dass es beim ersten Gespräch nur bei einer Skizze und ein paar Notizen bleibt, und Informationen zu einem späteren Zeitpunkt des Analyseprozesses hinzukommen müssen.

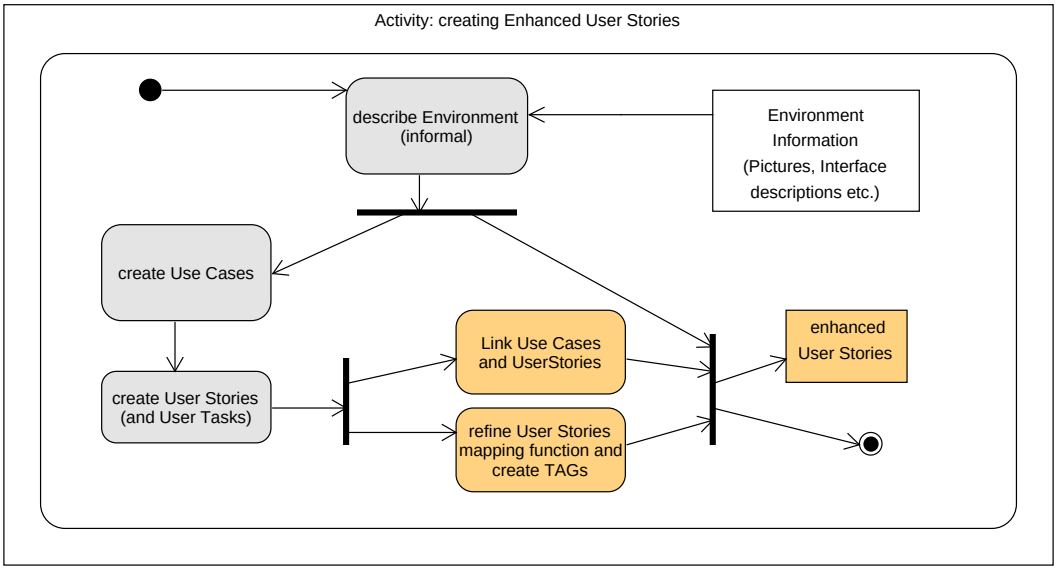

Abbildung 4.3.: Activity: creating Enhanced User Stories.

create Use Cases

Anschließend werden Use Case Abläufe erstellt (*create Use Cases*), um damit die Benutzerinteraktion und die allgemeinen Arbeitsabläufe darzustellen. Wie schon erwähnt, werden in vorigen Beispiel aufgrund der IRE Umgebung und der einfacheren Darstellung nur die Use Case Abläufe verwendet. Zudem kön-

nen stattdessen auch vollständige Use Cases[2] erstellt oder zusätzlich Use Case Diagramme verwendet werden. Oder es kann zur einfacheren Kommunikation mit dem Kunden erst die kürzere Fassung verwendet werden und später nur dem Entwickler die vollständigen Use Cases vorgelegt werden.

create User Stories

Anhand der einzelnen Use Cases Schritte ist es leicht die Kundenwünsche anhand eines Storyboards zu konkretisieren und mehr Informationen oder Spezialfälle aufzunehmen und zu diskutieren. Hier handelt es sich wie im Beispiel der Orangenpresse um vereinfachte User Stories aus dem IRE Tool, die zur Übersicht keine Risiko- und Storypoints oder Testfälle enthalten. Diese Zusatzinformationen werden für die Umsetzung in eine SML Specification nicht benötigt, können aber trotzdem aufgenommen werden, sollten diese für andere Entwicklungsprozesse benötigt werden. Eine Priorisierung der Karten kann zudem hilfreich sein, wenn zum Beispiel nicht *agil* entwickelt werden soll, um diese Information in eine spätere *Negotiation* einzubringen. Wenn sich hierfür entschieden wird, sollte nicht zu viel Zeit darauf verwendet werden.

Link Use Cases and User Stories

Bis zu diesem Zeitpunkt wurden bekannte Techniken verwendet um Anforderungen zu erheben. Die nachfolgenden farblich orange dargestellten Aktivitäten 4.3 sind extra für einen Systemanalyseprozess mit User Stories und SML ausgelegt. Die Verlinkung von User Story und Use Cases - *genauer einem Use Case Schritt* - ist auch eine sinnvolle Technik für andere Arbeitsmethoden und keine neue Erfindung dieser Arbeit. Die Verlinkung von Use Cases mit User Stories bietet mehrere Vorteile:

- Kunde und Entwickler erkennen im späteren Entwicklungsverlauf schneller, zu welchem Ablauf eine User Story beiträgt und hilft beim *Change Management* sowie *Tracing*.
- Der SML Modellierer kann im späteren Verlauf anhand der Links leichter entscheiden, ob eine User Story ein neues Scenario erstellt oder ein beste-

[2] Allerdings könnten zu viele vollständige Use Cases später bei der Verknüpfung der User Stories zu Unübersicht führen

hendes erweitert. Dies wird im späteren Verlauf genauer erläutert.

- Es wird sofort ersichtlich, wenn ein Use Case Schritt mit noch keiner User Story verknüpft ist.

Da nur User Stories in Scenarios umgewandelt werden, ist es wichtig, dass jeder Use Case Schritt mindestens einer User Story zugeordnet ist. Dies garantiert nicht, dass die Anforderungen des Use Case Schritts auch vollständig abgebildet sind. Hat dieser Schritt jedoch keinen Link, so ist davon auszugehen, dass der Schritt nicht modelliert wird und Anforderungen fehlen (oder der Use Case Schritt redundant ist). Lässt sich eine User Story zu keinem Use Case Schritt zuordnen, muss dieser Fall genauer untersucht werden, ob vielleicht ein ganzer Use Case, ein Teilschritt fehlt oder die Karte unwichtig ist.
Zum Verlinken wird in dem IRE Tool die User Story einfach auf den gewünschten Use Case Schritt gezogen. Wird mit analogen Karten gearbeitet, kann auf der User Story der Use Case mit der Schrittnummer notiert werden (z.B. UC1.3 für Use Case 1 Schritt 3) und/oder die User Story neben den Schritt im Use Case eintragen werden.

Refine User Stories with mapping function and create tags

Gleichzeitig kann der letzte Schritt zu Enhanced User Stories begonnen werden, der gleich mehrere der in Kapitel 4.2 aufgeführten Probleme löst. Jede User Story wird noch einmal mit dem Kunden besprochen und ermittelt, welche realen Objekte oder Personen seiner Ansicht nach an der User Story beteiligt sind. Diese Namen werden, wenn mit Karten aus Papier gearbeitet wird, auf der Rückseite der Karte notiert. Ist das genaue Objekt nicht bekannt, aber es ist ersichtlich, dass es existiert, so kann zunächst ein generischer Platzhalter mit der benötigten Funktion verwendet werden. Soll z.B. der Fachkraft ein Warnsignal übermittelt werden, es wurden aber noch keine Überlegungen darüber getroffen, ob es ein Warnton, eine blinkende Lampe oder eine Anzeige auf einem Display (oder alles zugleich) sein soll, so kann dies zunächst als "Warnindikator" modelliert werden, bis es für die genaue Umsetzung mehr Informationen gibt um diese später zu ersetzen. Mit diesen *tags* können später die Objekte für das SML Objektmodell erstellt und als Filter bei einer User Story Suche verwendet werden.
Zusätzlich werden die Kartenformulierungen noch einmal anhand der Über-

führungstabelle und einigen Konventionen geändert, um Unklarheiten für den Modellierer im Vorfeld auszuräumen. Dabei sollen Nominalisierungen und passive Verben aus Benutzersicht als aktive Verben aus Objektsicht formuliert werden. Aus diesen Verben werden später die Funktionen der Objekte generiert. Die *Schlüsselwörter* der Überführungstabelle können in der 2. Activity der Methode verwendet werden, um die Messages daraus zu erstellen.

Dies soll am Beispiel von User Story S5 4.4 verdeutlicht werden.

Abbildung 4.4.: Normale User Story S5 aus dem Beispiel der Orangenpresse

”Als Fachkraft möchte ich ... sehen können.” Ob die Fachkraft hinsieht, kann das System nicht gewährleisten, aber es kann ihr die Informationen anzeigen. Nach Nachfrage sagt der Kunde, dass ein Display die Informationen bereitstellen solle. Wird der Satz aktiv aus der Objektsicht beschrieben, ergibt sich: ”Als Fachkraft möchte ich, dass das Display mir Informationen ... auf dem Display anzeigt”. Nun wird notiert, welche Objekte daran beteiligt sind: zunächst die steuernde Software (Controller), dann das Display und die Saft- sowie Marmeladenmaschine. Hierdurch entsteht die Enhanced User Story S5 4.5.

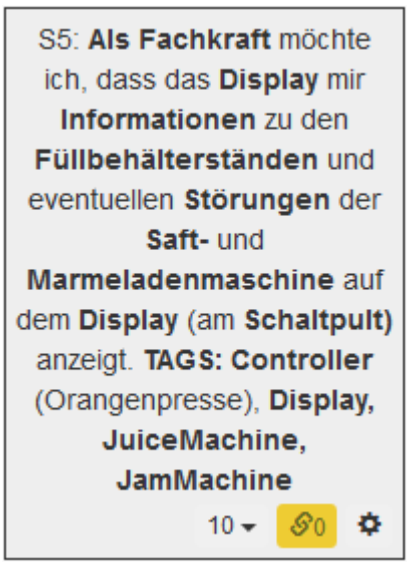

Abbildung 4.5.: Enhanced User Story S5 aus dem Beispiel der Orangenpresse

Zudem können noch weitere Regeln für bessere Verständlichkeit [Sch15] angewendet werden, dies ist jedoch nicht Gegenstand dieser Arbeit.

Die Reihenfolge der Aktivitäten dieser Methode kann und wird in vielen Fällen von der Abbildung 4.3 abweichen. Dies liegt daran, dass erst nach dem Verlinken der User Stories auffällt, dass ein Use Case oder Task fehlt oder eine weitere User Story angelegt werden muss. Die abgebildete Reihenfolge soll lediglich die nachfolgenden Schritte erleichtern. Aber es ist ebenso möglich, direkt beim Erstellen der User Stories Schlüsselwörter sowie tags aufzuschreiben und mit dem entsprechenden Use Case zu verlinken, wenn dies den Kunden nicht verwirrt oder ablenkt.

4.3.2. Erstellen von Enhanced Scenarios

Die Enhanced User Stories erlauben uns daraus eine SML Specification, also das SML Objektmodell und enhanced SML Scenarios zu erstellen. Es wird davon ausgegangen, dass die Activity *creating Enhanced Scenarios* nicht mehr Teil eines Kundengespräches oder Workshops ist. Daher steht der Kunde für Rückfragen nicht mehr zur Verfügung. Jede *Enhanced User Story* wird einzeln betrachtet, um erweiterte Scenarios zu erstellen oder bestehende zu erweitern 4.6.

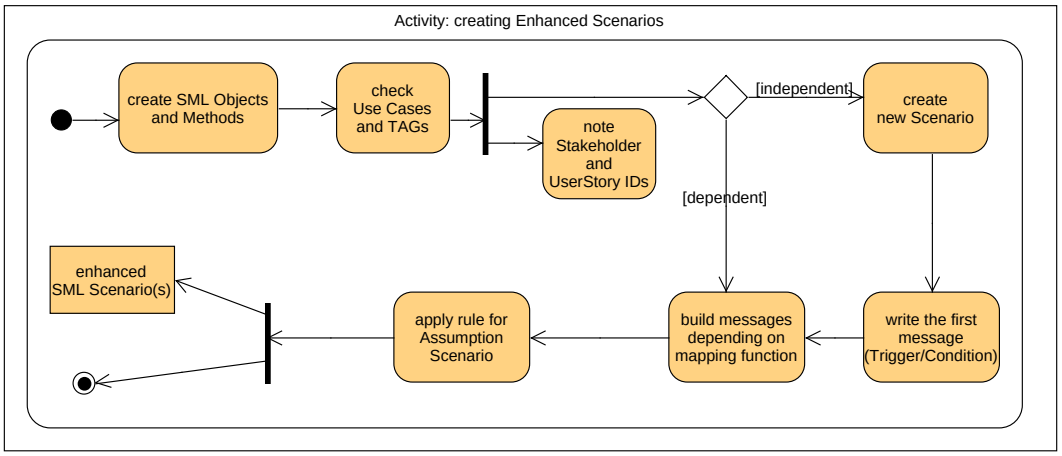

Abbildung 4.6.: Activity: creating Enhanced Scenarios

Create SML Objects and Methods

Auf einer User Story stehen nun *tags*, die die Objekte im SML Modell bilden. Steht auf einer Karte ein *tag*, der noch nicht im SML Objektmodell enthalten ist, wird dieser in das Objektmodell mit aufgenommen. Es ist möglich auch mit Vererbungen zu arbeiten, was einerseits einen übersichtlicheren und schöneren Programmierstil ermöglicht. Andererseits ist es nicht immer selbstverständlich, dass ähnliche Objekte identisch sind. Beispielsweise können die drei Regler dieselben sein oder es gibt zwei Dreh- und einen Schiebeschalter, die sich im späteren Entwurf anders ansteuern lassen. Als Beispiel werden aus der erweiterten User Story S5, die zuvor erstellt wurde, Objekte abgeleitet 4.7.

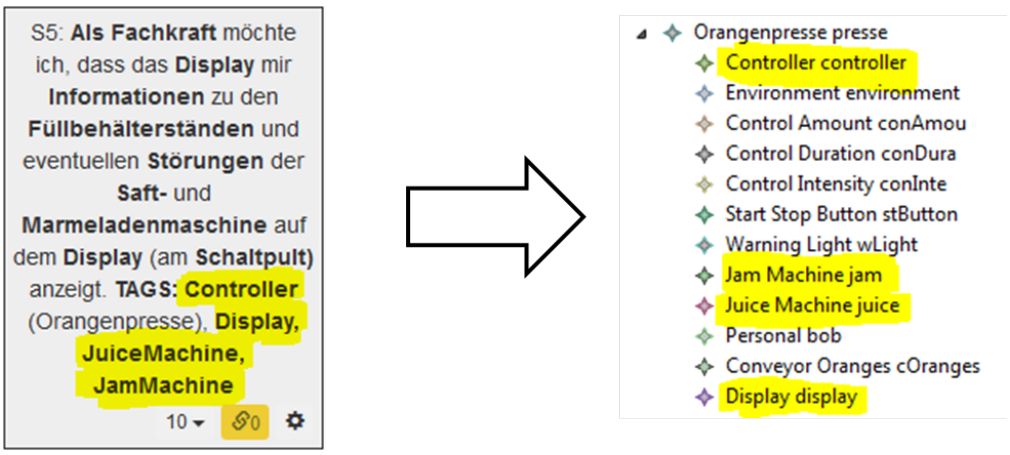

Abbildung 4.7.: Objekte aus *tags* erstellen. Die gelben Markierungen zeigen die *tags* und die dazugehörigen erstellten Objekte.

Sind alle Objekte, die die User Story betreffen, vorhanden, so werden im nächsten Schritt die Funktionen abgeleitet. Hier gilt, dass ein Verb, was dieses Objekt betrifft, eine Funktion bildet. Für die Enhanced User Story S5 heißt dies, dass "das Display Informationen zu (...) auf dem Display (...) anzeigt". Hieraus ergibt sich *display.showInformation()* und *display.showErrorMessage()* 4.8.

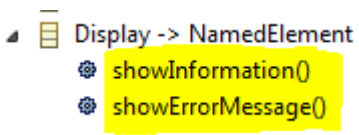

Abbildung 4.8.: Methoden aus Verben der Enhanced User Stories ableiten

Check Use Cases and *tags*

Die nächste Activity befasst sich mit einer der schwierigsten Herausforderung, die noch nicht vollständig bewältigt ist. Und zwar ist das Erkennen, ob eine User Story ein neues Scenario erstellt oder ob es ein bestehendes Scenario erweitert, oft schwierig und erfordert besondere Denkleistung vom Modellierer. Ein Richtwert ist, dass mehrere User Stories, die im selben oder direkt benachbarten Use Case Schritt verlinkt sind, wahrscheinlicher von einander abhängen, da sie in der logischen Abfolge nah beieinander sind. Eine andere Annahme ist, dass User Stories mit gleichen Objekten sich gegenseitig beeinflussen können. Ob eine Suche über *tags* auch mit vielen User Stories effektiv ist, müsste an

einem größeren Beispiel untersucht werden. Dies ist nicht sinnvoll, wenn es viele Karten mit gleichen Objekten gibt.

> Die User Story ist nicht verlinkt, also ist sie keinem Use Case zugeordnet.

Nach dem Verständnis der Karte lässt sie sich in UC1 als letzten Schritt oder als Extension einordnen. Die Informationen sollten dann immer automatisch angezeigt werden, da keine Interaktion mit dem Benutzer erfolgt. Sollen diese Informationen allerdings vor dem Start der Orangenpresse angezeigt werden oder der Kunde will doch einen Schalter um sich Informationen per Knopfdruck anzeigen zu lassen. Also wird hierzu eine Frage und/oder ein Vorschlag für die Verlinkung notiert, welche in der 3. Activity mit dem Kunden besprochen wird. Die User Story besitzt die *tags Display, JuiceMachine, JamMachine*. Die letzten beiden *tags* sind nur in der User Story S3 zu finden, welche die Maschinen einschaltet und nichts über Informationen, die man anzeigen könnte, enthält. Also wird für diese User Story ein neues Scenario erstellt.

Note Stakeholder and User Story ID

Es ist egal welches Ergebnis die vorherige Activity liefert, zu dem neu erstellten oder zu den erweiternden Scenario wird der Stakeholder und die User Story ID hinzugefügt. Dadurch ist später leichter erkennbar, von welcher User Story ein Scenario entstanden ist, und es kann bei einer Änderung einer User Story schnell die betroffenen Scenarios gefunden werden. Der Stakeholder trägt zum Verständnis bei, da in späteren Entwicklungsphasen evtl. nicht immer das Storyboard ”griffbereit” ist, wenn mit der SML Specification gearbeitet wird (z.B. eine Simulation durchführt).
Für die User Story S5 wird entsprechend ”S5” und ”Fachkraft” eingetragen.

Create new Scenario

Beim Erstellen eines neuen Scenarios liefert das *Ziel* der Karte den Scenarionamen. In dem Beispiel der Orangenpresse haben die User Stories durch das IRE Tool bedingt keinen Titel. Diese lassen sich sonst als Scenarionamen über-

```
1 message fachkraft -> controlDuration.setValue()
```

Listing 4: Umsetzung der geänderten enhanced User Story S4 in SML

nehmen. Ohne Titel muss der SML Modellierer das *Ziel* der Karte selbst "herauslesen". Dies macht jeder Modellierer intuitiv und es wird dazu keine Anleitung benötigt. Sollte tatsächlich einmal eine Karte so unklar geschrieben sein, dass nicht einmal das *Ziel* ersichtlich ist, so wird wie bei allen anderen Unklarheiten ebenso eine Fragekarte geschrieben. Für die User Story S5 könnte z.B. der Name *showInformationOnDisplay* oder *showStatusOnDisplay* gewählt werden.

Write first Message

Wie aus den Grundlagen bekannt, muss die erste Nachricht eines Scenarios eine non_strict, non_requested Message sein. Entweder wird sie von dem Environment, beispielsweise von der Fachkraft oder der Marmeladenmaschine eingeleitet oder die Nachricht wird von einem anderen Scenario mit einer Bedingung eingeleitet. Letzteres ist durch die 5. Regel in der Überführungstabelle 4.3.2 leicht zu erkennen. In der bisher ungenutzten User Story S5 ist dies nicht mehr möglich, da nicht beschrieben ist, wie das Display diese Informationen erhält. Die User Story S5 wird im nächsten Abschnitt 4.4.1 behandelt.

Stattdessen wird der Erfolgsfall bei der Umsetzung von User Story S1 betrachtet: "Als Fachkraft kann ich über die Regler (...) die Pressdauer unabhängig einstellen." "Kann" ist als erstes Element in der Überführungstabelle aufgeführt und gibt an, dass es sich um eine non_strict und non_requested Message handelt, die oft eine erste Message in einem Scenario bildet. Also die Fachkraft stellt ControlDuration den Wert ein 4.

Building Messages depending on mapping function

Dieser Abschnitt der Methode ist für die Modellierung einer der wichtigsten, da die Messages erstellt und die Scenarios mit Inhalt gefüllt werden. Dies ist ein kreativer Prozess des Modellierers, der noch nicht durch Texterkennung ausgereift genug ist, sodass es von einem Tool übernommen werden kann. Durch die Überführungstabelle (*mapping function*) hat der Modellierer eine

Vorgabe, welche Messagetypen (strict, requested) verwendet werden, und ob sich das Scenario in mehrere Unterscenarios aufteilt oder Verzweigungen und Nebenläufigkeiten gibt:

Nr.	Ausdruck auf User Story	Bedeutung in SML
1	jemand/etwas **"kann/darf"**	non_strict und non_requested (oft erste Message eines Scenarios)
2	Verb **ohne** "kann/darf"	requested
3	1 und 2 **"gleichzeitig"**	parallel [message1] and [message2]
4	**"entweder 1... oder 2"**	alternative [message1] or [message2] kann bei Fallunterscheidungen andere Scenarios einleiten
5	**"wenn 1 ..., dann 2"**	Bilden Vorbedingung: Also einleitende non_strict; non_requested message 1(*trigger*) , danach message 2
6	kann **"unabhängig"** getan werden	Indikator für Erstellung mehrere (Änhlicher) Scenarios
7	1 passiert,**indem** 2 passiert'	2 ist eine Vorbedingung für 1 aber nicht *trigger* des Scenarios
8	jemand/etwas **"könnte/sollte/müsste"**	Regel für Assumption Scenario

Eine wichtige Feststellung ist, dass jeder *tag* an mindestens einer Nachricht beteiligt sein muss. Sonst ist der *tag* auf der Karte redundant. Sollten Objekte an Messages beteiligt sein, die nicht als *tag* auf der User Story vermerkt sind, so ist dies nur möglich, wenn sie nicht von dieser User Story stammen. Diese stammen von einer vorher umgesetzten, die auch an dem Scenario beteiligt ist.

Werden die Regeln auf User Story S1 angewendet, so wird außer dem "kann" noch das Schlüsselwort "unabhängig" erhalten. Die User Story erstellt für jeden Schalter eines also drei ähnliche Scenarios. Nachdem ein Schalter betätigt wurde, muss der Schalter die Einstellung an den Controller weitergeben, der die Wertänderung verarbeitet und speichert. Ausgeschrieben würde die User Story S1 also lauten:

> Als Fachkraft kann ich über die Regler die Stärke, Menge und Pressdauer der Orangenpresse einstellen, um die Werte am Controller ein-

```
1     //Sth: Fachkraft
2     //User Story: S1
3     specification scenario changeAmoumt{
4       message fachkraft -> conAmou.setValue()
5         message strict requested conAmou -> c.updateAmount()
6     }
7     //Sth: Fachkraft
8     //User Story: S1
9     specification scenario changeIntensity{
10      message fachkraft -> conInte.setValue()
11        message strict requested conInte -> c.updateIntensity()
12    }
13    //Sth: Fachkraft
14    //User Story: S1
15    specification scenario changeDuration{
16      message fachkraft -> conDura.setValue()
17        message strict requested conDura -> c.updateDuration()
18    }
```

Listing 5: Umsetzung der User Story S1 in SML

zustellen, der die Presse steuert.

Dies ist der *Nutzen* der Karte und steht in diesem Fall nicht auf der verkürzten User Story, da dies als trivial angesehen wird und die User Story unnötig verlängert. Der Schalter übermittelt den Wert und der Controller aktualisiert diesen, woraus sich *controlDuration -> c.updateDuration()* ergibt. Nachdem der Schalter betätigt wurde, *muss* der Wert im Controller geändert werden, wodurch die Message ”requested” sein *muss* 5.

Apply Assumption Scenario rule

Zuletzt wird kontrolliert, ob es sich bei dem Scenario, um ein *Assumption Scenario* handelt. Dies ist ein Scenario, das Annahmen an die Umwelt modelliert. Generell ist dies der Fall, wenn ein Environment Objekt eine Message verschickt, die nicht die erste Message eines Scenarios ist. Dies bedeutet, dass eine Interaktion von einem *uncontrolleable* Objekt verlangt wird. Wenn der Konjunktiv genauer ”könnte”, ”sollte” oder ”müsste” auf einer User Story benutzt wird,

so drückt dies auch in der Alltagssprache eine Annahme aus. In diesem Fall wird die letzte Regel der Überführungstabelle 4.3.2 angewendet und ändert das Specification Scenario in ein Assumption Scenario. Steht diese Regel nicht auf der User Story, kann es sich trotzdem um ein Assumption Scenario handeln (z.B. weil bei dem Verfassen der Nachricht nicht daran gedacht wurde). Dies fällt anschließend in der Simulation auf.

4.3.3. Mark User Stories and validate Changes

In der letzten *Activity: mark and validate Changes* 4.9 werden alle zuvor bearbeiteten User Stories einzeln durchgegangen und Fragen oder Änderungsvorschläge, die bei dem Modellierungsprozess aufgetreten sind, mit dem Kunden besprochen. Außerdem werden die User Stories entsprechend ihres Status markiert, damit ein Überblick gewährleistet bleibt, welche User Stories schon fertig umgesetzt wurden, bei welchen es Fragen oder Probleme gab und welche noch nicht bearbeitet wurden. Beispiele für die Umsetzung von Fragen und Änderungsvorschlägen werden in dem Abschnitt *User Stories mit offenen Fragen* behandelt.

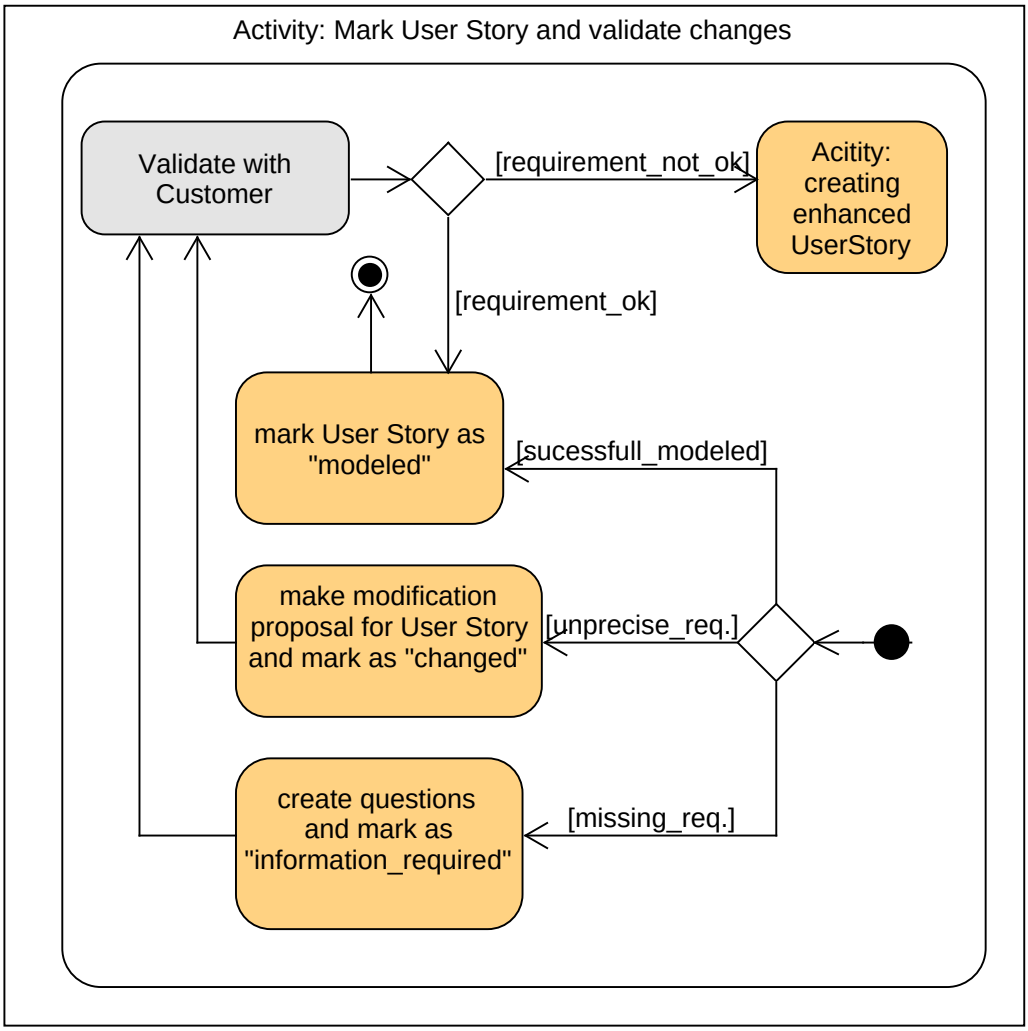

Abbildung 4.9.: Activity: Mark and validate changes

Mark User Story as "modeled"

Die betrachtete User Story wurde ohne irgendwelche Fragen oder Probleme in der letzten Activity *create Enhanced Scenarios* erfolgreich umgesetzt und wird als *modeled* markiert. Daher wird sie nicht noch einmal mit dem Kunden besprochen. Sollte der Modellierer die Karte falsch umgesetzt haben, ist dies nicht die letzte Instanz und es sollte bei der Simulation auffallen.

Make modification proposal and mark User Story as "changed"

Der SML Modellierer hat etwas an einer User Story geändert, um diese umsetzen zu können. Es könnte z.B. ein *tag* fehlen oder etwas nicht der Überführungstabelle entsprechen. Er könnte auch erkennen, dass eine wichtige Information fehlt, die auf die Karte gehört. Diese Änderung würde also den Text der User Story ändern, weshalb der Modellierer seinen Änderungsvorschlag separat aufschreiben und die User Story als *changed* markieren muss. In dem Schritt v*alidate with Customer* muss der Vorschlag mit dem Kunden besprochen werden. Dieser Schritt eignet sich, um eine User Story trotzdem umzusetzen, auch wenn die Karte nicht richtig formuliert ist oder etwas fehlt. Der Ansatz, dass Lösungsvorschläge zum Kundengespräch mitgebracht werden, kann viel Zeit und Nerven sparen. Diese Möglichkeit sollte nur genutzt werden, wenn die Bedeutung einer Karte sicher jedoch nicht entsprechend formuliert ist. Keinesfalls sollten Anforderungen erraten und anschließend als umformulierte Kundenanforderungen ausgegeben werden. Es kann aber auch nur mit Fragekarten gearbeiten werden, um auf Vorschläge zu verzichten.

Der Modellierer schreibt eine geänderte *Enhanced User Story*, welche dem Kunden zusammen mit der ursprünglichen **Enhanced User Story** vorgelegt wird. Es ist auch denkbar, dass er seine Änderungen als Normaltext aufschreibt und das Erstellen einer möglichst kundennahen und korrekten User Story mit dem Requirements Engineer zusammen erstellt. Der Änderungsvorschlag kann zusätzlich auch direkt in der Simulation dem Kunden präsentiert werden.

Create Question and mark as ”informationRequired”

Dieser Schritt ist dem *make modification proposal and mark User Story as ”changed”* sehr ähnlich. Der Unterschied ist, dass hier keine eigenen Vorschläge (ob in Prosatext, User Story oder SML Scenarios) vorbereitet wurden, sondern nur Fragen erstellt werden, da die User Story zu ungenau formuliert ist oder zu wenig Informationen vorliegen um die Karte in SML zu modellieren.
Während die als ”changed” markierten User Stories später die Möglichkeit haben, angenommen und als modelled markiert zu werden, so müssen alle mit ”informationRequired” markierten User Stories in der nächsten Iteration geändert werden. Hierzu existieren noch keine Vorschläge in Form von Scenarios.

Validate with Customer

In diesem letzten Schritt bespricht der Requirements Engineer die Fragen und Änderungsvorschläge mit dem Kunden. Die Änderungsvorschläge werden zusammen mit den dazugehörigen ursprünglich als ”changed” markierten *Enhanced User Stories* vorgelegt und der Kunde muss entscheiden, ob diese seinen Wünschen entsprechen. Zu den Änderungsvorschlägen kann dem Kunden auch die Simulation mit der momentanen Änderung gezeigt werden. Stimmt der Kunde dieser Änderung zu, so wird die *Enhanced User Story* vom Requirements Engineer entsprechend geändert und als erfolgreich modelliert markiert. Genauso wäre es möglich, dass der SML Modellierer die vom REler geänderte User Story noch einmal kontrolliert und die Karte erst dann als erfolgreich modelliert markiert.
Stimmt der Kunde den Änderungen nicht zu, müssen die Unklarheiten beseitigt und notiert werden. Anschließend ändert der Requirements Engineer in der nächsten Iteration in der 1. Activity *create Enhanced User Stories* die User Story oder erstellt eine neue, sodass sie der Modellierer umsetzen kann. Das Gleiche gilt auch für die vom Modellierer notierten Fragen und den dazugehörigen als ”information_required” markierten User Stories.
Sind alle User Stories (abgesehen von nicht-funktionalen) als ”modeled” markiert, der Kunde hat keine weiteren Wünsche und ist mit der Simulation zufrieden, so ist die Methode beendet. Natürlich können im späteren Verlauf Anforderungsänderungen oder neue Anforderungen dazukommen und mit der 1.Activity der Methode gestartet werden.

4.4. User Stories mit offenen Fragen

Die gezeigten Beispiele entsprachen bis jetzt zum größten Teil dem idealen Ablauf. Häufig kommt es vor, dass eine User Story noch nicht in der ersten Iteration umgesetzt werden kann. Dies ist ein wichtiger Aspekt dieser Methode und Teil des Prozesses der Systemanalyse.

4.4.1. Fragekarten erstellen

Manchmal ist bereits im Vorfeld bekannt, dass Informationen fehlen, wie es beispielsweise bei der User Story S5 während des Kundengespräches vorgefallen ist. Die Frage, wie die Kommunikation und "Zusammenarbeit" der Presse mit der Saft- und Marmeladenmaschine aussieht, konnte der Kunde im ersten Kundengespräch noch nicht beantworten. Dies wurde zusätzlich während der Erstellung der ersten Message festgestellt, dass nirgendwo beschrieben ist, woher das Display die Informationen bekommt. Außerdem fehlt der • "Controller" auf der User Story, was bedeutet, dass die Saftmaschine die Informationen direkt an das Display überträgt. Eine Fragekarte, die später mit dem Kunden besprochen wird, könnte so aussehen:

Fragen zu User Story S5

1. Senden die Saft- und Marmeladenmaschine ihre Statusmeldungen und Änderungen erst an den Controller, der diese dann an das Display weiterleitet?
2. Wird die Anzeige nur aktualisiert, wenn die Saft- oder Marmeladenmaschine den Impuls dazu gibt?
3. Welche Füllbehälterstände gibt es? Sollen Füllbehälterstände und Störungen verschiedene Informationen sein oder können diese als Statusinformationen zusammengefasst werden?

Zusätzlich markiert der Modellierer die User Story als "information_required", damit sie in der 3. Activity der Methode mit dem Kunden besprochen wird.

4.4.2. Änderungsvorschag erstellen

Am Beispiel von User Story S4 sind ebenfalls Schwierigkeiten bei der Umsetzung vorhanden:
Die *tags* und damit die Objekte *Controller, StartStopButton* und *ConveyorOrange* stehen zur Verfügung. Das Wort "wenn" leitet nach der Überführungstabelle die erste Message ein. Hier taucht aber ein Problem auf, denn ein Notfall ist nirgendwo beschrieben. Nun lässt sich dies als Frage formulieren wie bei S5 oder es wird versucht einen Vorschlag zu liefern. Der Start/Stop Schalter, der in User Story S2 modelliert wurde, unterscheidet die beiden Fälle *st.Button.setIsRunning(true/false)*. Die User Story S2 modelliert aber nachfolgend nur den Fall des Anschaltens. Es ist aber sinnvoll, die Maschine stoppen zu können, ob nun ein Notfall vorliegt oder nicht. Außerdem kann ein Notfall auch ein Ereignis aus der Umwelt sein, das nicht modelliert wird.

Wird die erste Bedingung der Karte weggelassen, ergibt sich folgendes: "Sollte ich als Fachkraft die Presse (und das Orangenförderband) anhalten, indem ich den Start/Stop Schalter drücke." Ohne den "wenn Notfall"-Zusatz, gibt es keinen Auslöser, und der "indem" Teil ist eine Vorbedingung (Regel 7) für das Stoppen der Maschinen. Wird der "indem"-Ausdruck vorgezogen und zum Auslöser, so entsteht folgende User Story:

> Wenn ich als Fachkraft den Start/Stop Schalter drücke (und die Maschine eingeschaltet ist), dann hält die Presse und danach das Orangenförderband an.

Das "sollte" fällt weg, da die Fachkraft nicht mehr auf einen Notfall reagiert, sondern der Controller auf die Aktion der Fachkraft. Damit wird das Verhalten des Controllers und der Presse spezifiziert, was einem Specification Scenario entspricht. Der in Klammern angefügte Nebensatz gibt an, was logisch ist: Die Maschine muss in Betrieb sein um sie stoppen zu können. Da die User Story S4 geändert bzw. eine geänderte Kopie angelegt wurde, muss diese wie im Normalfall die 2. Activity der Methode durchlaufen 6. Neue Objekte werden für diese Karte nicht benötigt, da sie bereits durch *Enhanced User Story* S2 erstellt wurden. Die Funktion *stButton.push()* ist ebenso in S2 enthalten. Das Verb "anhalten" bezieht sich auf die Presse und das Förderband, sodass beide Objekte die Funk-

```
1     //Sth: Fachkraft
2     //User Story: S2, S4
3     specification scenario pushButton{
4     message fachkraft -> stButton.push()
5     alternative if [stButton.isRunning == false] {
6       message strict requested stButton -> stButton.setIsRunning
            (true)
7     }or if [stButton.isRunning == true] {
8       message strict requested stButton -> stButton.setIsRunning
            (false)
9     }
10    }
11    //Sth: Fachkraft
12    //User Story: S4
13    specification scenario stoptProduction{
14      message stButton -> stButton.setIsRunning(false) //
            stButton= StartStopButton
15      message strict requested stButton -> c.stop()  // c =
            Controller
16      message strict requested c -> cOranges.stop() // cOranges
            = ConveynorOranges
17    }
```

Listing 6: Umsetzung der geänderten Enhanced User Story S4 in SML

tion *.stop()* erhalten. S4 ist die einzige verlinkte User Story in Use Case UC2, aber User Story S2 hat dieselben *tags* wie S4. Das Scenario *pushButton* wird von beiden User Stories verwendet, und die Verzweigung *stButton.setRunning(false)* leitet das neue Scenario *stopProduction* ein. Nach der Übersetzungstabelle sind beide Nachrichten zur Presse und Förderband *requested*. Der Schalter überträgt die Nachricht *stop* zum Controller, welcher danach eine Stoppnachricht zum Orangenband schickt. Da diese Reihenfolge eingehalten werden muss, sind beide Nachrichten *strict*.

Dem Kunden kann in der 3. Activity der Methode die veränderte *Enhanced User Story* mit der Simulation neben der alten *Enhanced User Story* gezeigt werden. Daraufhin entscheidet dieser, ob die Änderungen seinen Wünschen entsprechen oder ob die User Story noch einmal geändert werden muss.

4.5. Schlussfolgerungen und ungelöste Probleme

In diesem Abschnitt werden die Schlussfolgerungen und Erkenntnisse, die während der Methodenentwicklung entstanden sind, erläutert und es wird zusätzlich auf ungelöste Probleme aufmerksam gemacht.

4.5.1. Schlussfolgerungen

Kapitel 3 und 4 zeigen, dass eine Umsetzung zwischen User Stories und formalen Szenarien durchführbar ist. Da aber User Stories zielorientiert und SML Scenarios verhaltensorientiert sind und beide einen unterschiedlichen Detailgrad aufweisen, muss man die fehlenden Informationen für die Umsetzung vorher gewonnen werden. Die erarbeitete Methode kann die benötigten Informationen erfassen. Einige Informationen sind jedoch nicht überführbar. Die User Tasks und User Stories mit nicht-funktionalen Anforderungen werden in SML nicht abgebildet. Collaborations kommen dafür in der SML Specificationen vor, werden in User Stories aber nicht verwendet.

In der beschriebenen Methode wird mit den vom Kunden erstellten User Stories gearbeitet. Diese werden in der Methode erweitert, geändert und entsprechen nun keinen Standard User Stories mehr. Eine zentrale Frage lautet, ob es sinnvoll ist, die User Story des Kunden mit einer Überführungstabelle anzupassen und die Sätze so umzustellen, dass die Verben aktiv und aus Objektsicht geschrieben sind. Dies und das weitere Notieren von *tags* führt dazu, dass die Karten des Kunden stark umgewandelt werden können und es dann nicht mehr ”seine” Karten sind. In der 3. Activity ist sogar vorgesehen, dass der Modellierer die Karte ändern, bzw. einen völlig neuen Vorschlag machen kann.

Die Antwort lautet, dass die Änderungen an den User Stories überaus sinnvoll sind, solange das Erstellen der *Enhanced User Stories* dem Kunden und Entwicklerteam weiterhilft. Indem eine ”normale” User Story vom Kunden in eine *Enhanced User Story* umgewandelt wird, werden dessen Karten genutzt und diskutiert um weitere Anforderungen herauszuarbeiten. Diese werden vom Modellierer interpretiert und durch *Enhanced Scenarios* formal aufgeschrieben. Fehlen dem Modellierer dazu Informationen, so werden Vorschläge oder Fragen zu diesen Karten erstellt und über sie diskutiert. Die einfache Kommunikation zwischen Kunden und Modellierer und die Erfassung von Anforderungen war

der Grund, warum User Stories verwendet wurden, sodass dieser Vorteil durch die entwickelte Methode noch weiter ausgebaut wird.

Hier könnte theoretisch der ungünstige Fall auftreten, dass die Kommunikation über User Stories und den Verbesserungsvorschlägen in einem "hin und her" zwischen Vorschlägen des Modellierers und des Kunden ausartet. Beiden steht aber der Requirements Engineer mit seinem Fachwissen zur Seite, sodass nach wenigen Schritten eine vom Kunden akzeptierte *Enhanced User Story entsteht*, die in SML umgesetzt werden kann.

Die erarbeitete Methode wirkt sich auf agile Aspekte wie Priorisierung und Kostenabschätzung einer User Story nicht aus und bleibt bei der Umsetzung in *Enhanced User Stories* erhalten.

Die Verwendung von User Stories zusammen mit Use Cases und einer Verlinkung zwischen ihnen hat sich als sehr nützlich herausgestellt. Die Use Cases bieten bei der Erstellung der User Stories eine Grundlage, an der mit dem Kunden schrittweise User Stories erstellt werden. Durch die Verlinkung fällt sofort auf, wenn zu einem Use Case Schritt noch keine User Story zugeordnet wurde, und dass Anforderungen in Form von User Stories fehlen, um den ganzen Ablauf informell zu beschreiben. Diese werden auch später bei der formellen Beschreibung fehlen. Ist eine User Story vom Kunden erstellt worden, die keinem Use Case Schritt zugeordnet werden kann, sollte noch einmal darüber diskutiert werden, ob die User Story wirklich relevant ist oder ein Use Case Ablauf geändert werden muss bzw. ein neuer noch unbetrachteter Use Case vorliegt.

Die Use Cases sind auch während des Erstellens der *Enhanced Scenarios* von Bedeutung. Da die Use Cases mit den *Enhanced User Stories* und diese mit den *Enhanced Scenarios* verlinkt sind, sind folglich auch Use Cases indirekt mit den *Enhanced Scenarios* verlinkt. Die Use Cases werden betrachtet, um zu entscheiden, ob eine User Story ein neues Scenario erstellt oder ein existierendes erweitert und dienen dem Modellierer zum Nachschlagen bei Verständnisfragen. Eine denkbare Weiterentwicklung der Methode wäre, aus den Use Cases *Existential Scenarios* zu generieren, mit denen getestet werden kann, ob die Standard Use Cases von der SML Specification erfüllt werden.

Wie bereits am Beispiel der Orangenpresse gezeigt werden konnte, kann es andere Anforderungsdokumente geben, in denen Anforderungen stehen. Zu

diesen Dokumenten zählen beispielsweise Schnittstellenbeschreibungen und Bedinungsanleitungen. Falls aus diesen Anforderungen keine User Stories erstellt werden sollen, weil sie z.B. zu technisch sind, sie trotzdem in SML aufgenommen werden sollen, müssen die Scenarios als Referenz das jeweilige Dokument mit der entsprechenden Position der Anforderung enthalten. So kann zurückverfolgt werden, wo diese Anforderungen herkommen.

4.5.2. Bestehende Probleme

Schwierigkeiten mit Enhanced User Stories
Schreibt der Kunde User Stories, so sind diese oft verkürzt (wie dem Beispiel User Story S1), denn er hält manche Dinge für selbstverständlich, die es bei einer genauen Modellierung nicht immer sind. Selbst für einen Requirements Engineers ist es schwierig genaue User Stories in einem frühen Kundengespräch zu schreiben, da noch kein Gesamtüberblick über die genauen Abläufe und Informationen vorhanden ist. Ein Beispiel dafür ist die User Story S5, die anfangs mit den verwendeten Regeln richtig klingt, die trotzdem nicht umgesetzt werden kann.

Die Überführungstabelle kann noch um zusätzliche Regeln erweitert werden, um beispielsweise Schleifen oder Timer einzubauen. Doch gleichzeitig sollen für den Wiedererkennungswert des Kunden so viele Wörter und Formulierungen wie möglich von seiner (standard) User Story erhalten bleiben. Daher muss abgewägt werden, wie viele Regeln man verwenden möchte. Aufschluss darüber, welche zusätzlichen Regeln bei der Entstehung von User Stories sinnvoll wären, könnte durch eine Evaluation mit konkreten Beispielen anhand realer Kunden erarbeitet werden.

Schwierigkeiten in SML:
In dieser Methode sind noch nicht alle Möglichkeiten von SML ausgeschöpft. Hier werden ausschließlich boolesche Variablen für Fallunterscheidungen verwendet. Es kann als sinnvoll erachtet werden andere Variablentypen mit einzubeziehen. Dafür müsste noch untersucht werden, wie man aus einer User Story Variablen ableiten kann bzw. wann extra Variablen benötigt werden.

Diese Methode betrachtet des Weiteren nur statische Rollen. Ein dynamisches System kann zwar so spezialisiert und vereinfacht werden, dass daraus ein statisches System entsteht, verliert dadurch aber viel an Aussagekraft, da einzig

ein Spezialfall betrachtet wird. Eventuell soll explizit die Dynamik des Systems simuliert und überprüft werden. Die Annahme, dass der Rollentyp (statisch oder dynamisch) sich aus dem Kontext ableiten lässt, muss noch überprüft werden.

Strict Messages können mit der Methode nicht aus User Storys abgeleitet werden. Strictness sagt aus, dass die Reihenfolge der Messages eingehalten werden muss. Werden mehrere Abläufe auf einer User Story beschrieben, so kann Strictness aus dem Kontext erkannt werden. Eine allgemeine Regel konnte dafür aber noch nicht abgeleitet werden.

Schwierigkeiten bei der Methode:
Anforderungen tauchen auch in anderen Dokumenten (siehe oben) auf, und manchmal ist es nicht sinnvoll, diese noch einmal in User Stories aufzuschreiben. Die Möglichkeit wie eine Verlinkung mit anderen Quellen in die Methode eingebaut kann, ist nicht Teil dieser Arbeit.

Der Modellierer sollte immer gegenprüfen, ob das Scenario, das er erstellt hat, auch wirklich mit den Kundenanforderungen auf der User Story übereinstimmt. Wenn ein Modellierer hierbei einen Fehler macht, ist nicht sicher, ob er durch die Simulation oder im späteren Projektverlauf gefunden wird. Auch ist nicht sicher, ob ein Fehler eine echte und nicht erfüllte Anforderung ist, wenn er durch die Simulation nicht auffällt und der Kunde mit der Simulation zufrieden ist. Ebenfalls müsste untersucht werden, welche Folgen das Finden eines Fehlers im Modell während der Entwurfs- oder Implementierungsphase hat.

5. Auswirkung und Nutzung in der Systemanalyse

Dieses Kapitel befasst sich mit dem Thema, welche Anwendungsgebiete sich für den Einsatz von User Stories und formalen Szenarien eignen, wie die in Kapitel 4 beschriebene Methode in verschiedene Software/System Entwicklungsprozesse eingebunden werden können und welchen Aufwand sowie Nutzen sie mit sich bringen.

5.1. Anwendungsgebiete

Zunächst soll in diesem Abschnitt unabhängig vom RE-Ansatz diskutiert werden, wann sich die Verwendung dieser Methode anbietet, und wann nicht.

Günstige Anwendungsgebiete

Die Methode soll zur Systemanalyse genutzt werden, um das Verhalten der Hardware und der Mensch-Maschine-Interaktion darzustellen. Die Software wird als eine Black Box angesehen oder als Server, der die anderen Komponenten steuert, es sei denn, es geht aus dem Domänenwissen direkt hervor, dass explizit mehrere Softwarekomponenten benötigt werden. Eine besondere Stärke der Methode ist, dass durch die Umsetzung von User Stories und SML viele Anforderungen konkretisiert werden oder neue Anforderungen entstehen, Widersprüche und Fehler gefunden sowie mit dem SML Modell "gespielt" und exploriert werden kann.

Das in Kapitel 2 beschriebene Projekt über die Entwicklung einer Orangenpresse ist ein Beispiel für eine Systemanalyse, in dem die Vorteile der Methode ausgenutzt werden können. Der fiktive Kunde Orangenmüller entspricht einem Klienten, der von Informatik, Anforderungsanalyse oder Softwareentwicklung nicht viel versteht. Er möchte vielmehr seine Wünsche zu einem funktionierenden und für ihn zufriedenstellenden Resultat umgesetzt haben, wobei ihn

die genaue Implementierung nicht interessiert. Auch hat er wenig Wissen über die Softwarekomponenten der angrenzenden Systeme. Daher ist der Ansatz ideal, die Software zunächst als Black Box zu betrachten und die Wünsche des Kunden auf Karten zu schreiben und bei fehlenden Informationen Umsetzungsvorschläge zu liefern, die dem Kunden später präsentiert werden können. An einer Simulation kann der Kunde zusätzlich die Abläufe durchspielen und ihm beim Verständnis des zu entwickelnden Systems und dem Finden neuer Anforderungen helfen.
Tritt der Fall ein, dass der Kunde ein Ingenieur, Informatiker oder Fachkundiger ist, der sehr konkrete Anforderungen und Einschränkungen hat und der keine Vorschläge möchte, ist dies für die Methode kein Nachteil. Die User Stories werden dadurch gleich zu Anfang detaillierter und bei fehlenden Informationen während dem create Enhanced Scenarios Schritt (2. Activity der Methode) werden nur "Fragekarten" verwendet . Die Fragen können in vielen Fällen durch den fachkundigen Kunden schnell beantwortet werden. Dennoch ist die Verwendung von User Stories zur Kommunikation auch hier geeignet, da die Ingenieure in den wenigsten Fällen SML Specificationen lesen oder schreiben können.

Soll die Methode bei einer Systemanalyse verwendet werden, bei der das System mit sehr vielen Komponenten besitzt, die viele einzelne und verschiedene Nachrichten verschicken, muss auf eine Sache geachtet werden: Wenn Komponenten nicht irgendwie zusammengefasst oder generalisiert werden können, könnten zu viele User Stories entstehen. Dadurch wird ein Storyboard und die Simulationsgraphen von SML sehr groß und unübersichtlich. Sind beispielsweise 500 Schalter und Knöpfe vorhanden, könnten diese sich zu 50 Schaltpulten oder Armaturen zusammengefügt werden und die Funktionen abstrakter beschreiben.
Wie bereits in Kapitel 3 und 4 beschrieben, ist eine Generalisierung auch nötig, wenn anfangs nicht genug Informationen vorliegen. Die *setValue* Funktion der Regler aus dem Beispiel der Orangenpresse muss in einer realen Umsetzung auch Werte übergeben. Solange über den Wertebereich noch keine Informationen vorhanden sind, ist es anfangs nur wichtig zu wissen, dass etwas verändert wird. Später können die Informationen entweder ergänzt oder in der Implementierung eingesetzt werden, ohne vorher in SML simuliert worden zu sein.

Ungünstige Anwendungsgebiete

Entstammen die meisten Anforderungen aus anderen Dokumentationen oder informellen Spezifikationen mit vielen Details, weil beispielsweise Middleware für größere Systeme gebaut wird, ist es sinnvoll, eine formale Spezifikation zu schreiben. Hierdurch kann die Integration in den anderen Systemkomponenten sichergestellt werden. User Stories hingegen bieten sich zu diesem Zweck nicht an, da diese nicht detailliert genug sind und mit zunehmendem Detailgrad unübersichtlicher werden, daher eignet sich die Methode hier nicht.

Wird festgestellt, dass ein Projekt wesentlich mehr nicht-funktionale Anforderungen als funktionale besitzt, sollte die Methode auch nicht angewendet werden. Hier bietet sich mehr ein Prototyp an, der diese Anforderungen einbezieht, als eine Formale Spezifikation, die mehr als die Hälfte aller Anforderungen nicht einbeziehen kann. Ein Beispiel dafür ist das Erstellen einer Werbewebseite. Diese besitzt wenig Funktionalität im Vergleich zu den Anforderungen an Oberflächendesign und Usability (abgesehen davon, dass es Software und kein System zu entwickeln wäre).

In manchen Anwendungsfällen ist der Einsatz der Methode auf das gesamte System nicht sinnvoll. Ein System, das in ähnlicher Art schon mehrfach für Kunden entwickelt wurde und nur angepasst werden muss, benötigt keine so umfangreiche Systemanalyse, da der Aufwand zu hoch ist. Die Methode kann aber partiell für einzelne kritische Systemteile verwendet werden.

5.2. Einbettung in den Entwicklungsprozess

Im Folgenden wird beschrieben, wie sich die erarbeitete Methode in die verschiedenen Entwicklungsprozesse einbetten lässt, und welcher Phase des RE-Referenzmodells diese entsprechen.

5.2.1. Einsatz im RE nach dem RE-Referenzmodell

Nach dem RE-Referenzmodell aus Abb. 2.3 ist die erste Phase einer Anforderungsanalyse die *Elicitation*, in welcher Anforderungen gesammelt werden. Die Activity create enhanced User Stories der Methode befindet sich mit der Erfas-

sung der Use Cases und User Stories in dieser Phase. Das Erstellen von tags und Anwenden der Überführungstabelle kann aber schon als kleine *Interpretation Phase* aufgefasst werden. Werden aus den *Enhanced User Stories* Enhanced Scenarios erzeugt, wobei Unklarheiten auftreten, so kann man das Aufschreiben der Fragen und Vorschläge auch als *Negotiation Phase* betrachtet werden. In größeren Projekten kann es sich anbieten, erst mehr Hintergrundinformationen über das zu entwickelnde Produkt und die Domäne einzuholen, bevor begonnen wird User Stories zu erstellen.

Eine andere Möglichkeit, die Methode einzusetzen, ist die Verwendung in einem Workshop zur Lösung von Konflikten mit verschiedenen Stakeholdern. Oft werden User Stories mit einer agilen Entwicklung gleichgesetzt. Genauso können User Stories aber auch als Unterstützende Technik [Rup13] angesehen werden, um Anforderungen aufzuschreiben. Ähnlich wie die Technik mit dem "Moderationskoffer" oder der "Mindmap" [Sch15]. So können durch Interviews vorab Anforderungen, Use Cases und Domänenwissen gesammelt werden, welche die Grundlagen eines Workshops bilden. Ein Workshop kann dann die verschiedenen Parteien zusammenführen und jeder schreibt Storycards aus seiner Perspektive auf. Die User Stories werden dann diskutiert und benutzt, um Konflikte zu lösen. Dies entspricht im RE-Prozess der Negotiation-Phase. Nach der Diskussion einer Karte formuliert der Requirements Engineer diese Karten in eine *Enhanced User Story* um und kann diese am Ende des Workshops als Ergebnis präsentieren und mit den Beteiligten noch einmal durchgehen. Soll kein Workshop durchgeführt werden oder kommen nicht alle Parteien zusammen, kann das Erstellen der User Stories auch mit bestimmten Stakeholdern einzeln durchgeführt werden. So kann die 1. Activity *create Enhanced Scenarios* der Methode als Elicitation oder Negotiationtechnik verwendet werden. Die Erstellung der Enhanced Scenarios sind nicht Teil des Workshops und werden wie im Beispiel der Orangenpresse ohne Anwesenheit des Kunden durchgeführt. Außer es handelt sich bei den Kunden um Modellierer und das Ziel des Workshops ist es, ein SML Modell zu erstellen. Zwischen Activity 2 und 3 findet eine Interpretation statt, wenn Vorschläge oder Fragen zu User Stories aufgeschrieben werden.

5.2.2. Einsatz im agilen RE

Soll *agil* (z.B. mit SCRUM [Coh04]) gearbeitet werden, bleibt der erste Teil der Methode *create Enhanced User Stories* gleich und wie im Beispiel der Orangenpresse kann auch so ein Kundengespräch ablaufen. Zusätzlich werden die User Stories mit dem Kunden priorisiert. Anschließend müssen noch die Kosten bestimmt werden. Ob dies zusammen mit dem Kunden oder als *Planning Poker* [Coh05] mit dem Entwicklerteam durchführt wird, kommt auf den Umfang und die Komplexität des Projektes an.

Nun werden nicht alle User Stories in Enhanced Scenarios umgesetzt, sondern nur die, welche die höchste Priorität haben und zum nächsten Sprint bearbeitet werden sollen. Es entsteht ein Zwischenschritt, so dass zunächst eine funktionierende SML Simulation entsteht, bevor in darauffolgenden Sprints das echte System dazu entwickelt wird. In jedem Sprint kann so das vorhandene SML Modell erweitert und der Stand, der zuvor in SML modelliert wurde, "umgesetzt werden". Die User Stories bieten die Möglichkeit, schrittweise zu modellieren. Mit einem OnSite Customer oder regelmäßiger Kommunikation, die in der agilen Entwicklung üblich ist, können rechtzeitig zu einem Sprint die Fragekarten der markierten unklaren User Stories besprochen werden. Durch die Verlinkung zwischen User Stories, Use Cases und SML kann leicht mit nachträglichen Änderungen und neuen User Stories umgegangen werden. So unterstützt auch die Methode der Umwandlung zwischen User Stories und SML ein agiles Arbeiten und trägt zur Dokumentation bei.

Was im letzten Absatz als "Umsetzen des Systems" bezeichnet wurde, wird von der erarbeiteten Methode allerdings nicht betrachtet. Sie liefert nur die Anforderungen an das System. Um dieses System wirklich implementieren zu können, muss die benötigte Hardware- und Softwareanforderung, die auf anderem Wege erhoben wird, vorhanden sein. Dieses Problem besteht unabhängig von dem Einsatz der Methode und SML.

5.2.3. Einsatz in der Entwicklung mit vollständigem formalen Modelchecking

Der übliche Ablauf der Entwicklung mit formalem Modelchecking unterscheidet sich von dem normalen V-Modell Prozess 2.1 darin, dass nach der informellen Spezifikation eine formale Spezifikation angefertigt wird. In der erarbeiteten Methode wird schon während der Elicitation, also zusammen mit den User Stories, eine erste formale Spezifikation angelegt.

Das Verhalten des Systems und dessen benötigten Objekten ist auch hier die ersten Überlegung. Gerade wenn der Modellierer nicht selbst der Kunde ist, so ist auch hier eine Kommunikation über User Stories anschaulich. Der große Vorteil bei der erarbeiteten Methode liegt im Tracing. So kann später leicht und schnell zurückverfolgt werden, wo eine Anforderung herkommt, wo sie in der SML Specification steht und welche User Stories umgekehrt geändert werden müssen, wenn bei der Modellierung widersprüchliche Anforderungen gefunden werden. Scenario-Based-Modelling und SML zählen zwar von den formalen Methoden zu den leichter lesbaren Spezifikationen. Aber gerade bei größeren SML Specificationen findet man leichter, welche Scenarios von einer Anforderungsänderung betroffen sind (durch Links zwischen User Stories und SML) und welche Scenarios von anderen abhängen(Use Case Schritte und eingetragene beteiligte User Stories).

Durch eigene praktische Erfahrungen und Aussagen von Kommilitonen, erleichtert die Methode den Einstieg in Scenario Based Modelling und speziell in Scenario-Tools. Mit Use Cases und Enhanced User Stories (tags) können vorab auf einem strukturierten Weg die für SML benötigten Abläufe und Objekte verdeutlicht werden. Die einzelnen Use Cases bieten mit der Überführungstabelle gemäß dem Ansatz Teilen und Herrschen kleine und übersichtliche ”Portionen”. Diese können in SML einfacher umgesetzt werden.
Durch die 3. Activity der Methode erhält man eine konkrete Vorgabe, wie man mit fehlenden Informationen bzw. unvollständigen User Stories umgehen kann.

Nach der Systemanalyse wird die SML Specification in der Softwareanalyse weiterverwendet und mit dem Ziel erweitert, eine vollständige formale Spezifikation des Gesamtsystems (mit System- und Softwareanforderungen) zu erstellen.

5.3. Nutzen und entstehende Kosten

Mit Hilfe der Methode lassen sich User Stories auf einfache Weise in SML umformen. Mit der Überführungstabelle 4.3.2 ist auch der umgekehrte Fall möglich, der hier aber nicht betrachtet wird, da in den wenigsten Fällen ein Projekt mit einer gegebenen SML Specification beginnt. Durch die Entscheidung für den Einsatz der erarbeiteten Methode wird zunächst durch Use Cases und User Stories eine leichte Technik erlangt, um Anforderungen aufzuschreien und zu diskutieren. Wird diese als Teil der Elictiation eingearbeitet, z.B. als Teil der normalen Interviews, oder werden sowieso User Stories verwendet, so entsteht dadurch nicht mehr Aufwand, da die Erstellung der User Stories schon im Prozess enthalten ist. Die Enhanced User Stories beanspruchen allerdings mehr Zeit des Kunden, da nach der 1. Activity der Methode erst User Stories und Use Cases aufgenommen werden. Anschließend werden noch einmal mit dem Kunden die *tags* besprochen, mit der Überführungstabelle umformuliert und mit den Use Cases verlinkt. Je mehr User Stories in einem Projekt vorliegen, desto mehr Kundenzeit, welche oft rar und teuer ist, wird für die Erweiterung benötigt. Schafft es ein erfahrener Requirements Engineer, die Erstellung der Enhanced User Stories gleich in einem Schritt zu bewerkstelligen, also direkt beim Erstellen der User Stories tags und Schlüsselwörter der Überführungstabelle mit aufzunehmen, ohne vorher "normale" Use Cases zu erstellen, kann der Kundenzeit-Nachteil wieder ausgeglichen werden. Es ist wichtig den Kunden dabei nicht zu überfordern. Den Aufwand, der mehr betreiben werden muss, wird dafür in dieser Phase mit mehr und besser dokumentierten Anforderungen belohnt, da bereits konkrete Objekte durch tags erhalten wurden und Anforderungen durch die Verlinkung verknüpft wurden. Hierbei besteht allerdings auch die Gefahr zu genau zu werden und in der Elicitation schon Entwurfsentscheidungen zu treffen, die keine echten Anforderungen sind. Wenn die Enhanced User Stories in einem Workshop (z.B. in der Negotiation Phase) entstehen, müssen zu diesem Zeitpunkt schon viele Anforderungen bekannt sein und einige Anforderungen werden daher doppelt aufgenommen.

Die 2. Activity create Enhanced Scenarios benötigt keine Kundenzeit, dafür aber die zusätzliche Zeit für das Erstellen der SML Specification, die sonst nicht erstellt worden wären. Dabei fällt die Modellierungszeit umso länger aus, je ungenauer die User Stories geschrieben sind, da der Modellierer Fragen und Vor-

schläge selbst liefern oder die Stories evtl. noch einmal mit dem Requirements Engineer besprechen muss. Dies entfällt jedoch, falls Modellierer und Requirements Engineer dieselbe Person sind. Dafür werden zu einem frühen Zeitpunkt die Vorteile eines formalen Modells erhalten. Zu den Vorzügen gehören präzise und unmissverständliche Schreibweise, *Verklemmung* und *Consistently Executable* Überprüfung, Aufspüren von sich gegenseitig ausschließenden Scenarios und Ausführbarkeit einer Simulation. Diese Simulation kann mit dem Kunden "durchgespielt" werden, was zur Veranschaulichung, Ideengewinnung oder Validierung genutzt werden kann. Die Enhanced Scenarios bieten hier zusätzlich den Vorteil, dass durch das Einfügen von Stakeholdern und dem Link zu den User Stories die Verständlichkeit des SML Codes und das Change Management verbessert wird.

Das Aufschreiben der Fragen und Vorschläge sollte nicht viel Zeit in Anspruch nehmen, und evtl. schon während der Umsetzung der Enhanced Scenarios durchgeführt werden. Diese müssen aber wieder mit dem Kunden besprochen werden und evtl. müssen neue User Stories erstellt werden, so dass diese Activity wieder Zeit vom Kunden und Requirements Engineer benötigt.

5.3.1. Mögliche Verwendung nach Erstellung der SML Specification

Nach erfolgreicher Verwendung der Methode ist aus allen User Stories eine SML Specification erstellt worden. Aus den Use Cases können nun existential Scenarios[1] erstellt werden. Damit könnte überprüft werden, ob die Anfangs mit den Use Cases dokumentierten Systemabläufe auch wirklich von der SML Specification erfüllt werden. Eine korrekte SML Specification sollte die "Durchläufe" dieser Existential Scenarios erfüllen. Ist dies nicht der Fall, müssen Use Cases und User Stories auf Aktualität und Konsistenz überprüft werden. Liegt hier nicht der Fehler, sollten die Scenarios der User Stories der betroffenen Use Case Schritte zuerst überprüft werden.

Codegenerierung ist speziell bei Scenario-Tools noch nicht implementiert, aber prinzipiell möglich. Die Scenarios können auch in für Programmierer evtl. leichter lesbare grafische Notation überführt werden, die dann im Entwurf genutzt werden können.

[1] Das Erstellen von Existential Scenarios aus Use Cases ist nicht Teil dieser Arbeit

6. Zusammenfassung

In dem Beispiel der Orangenpresse wurde ein erstes Kundengespräch nachgespielt und zunächst Use Cases und User Stories erstellt. Der Versuch, diese in formale Szenarien (mit SML) umzusetzen, hat gezeigt, dass dies ohne Vorarbeit nicht systematisch und einheitlich möglich ist. Daraufhin wurde eine Methode entwickelt, die die meisten Probleme löst:

- Nicht-funktionale Anforderungen können nicht modelliert werden und Collaborations und User Tasks helfen bei einer Überführung in die jeweils andere Notation nicht. Darum werden diese in der Methode nicht betrachtet[1], sie können aber trotzdem angelegt werden und stören die Methode nicht.
- Um zu wissen, welche genauen Objekte an einer User Story beteiligt sind, wurden *Tags* eingeführt.
- Um die Funktionen dieser Objekte zu erfassen, muss die Karte so umformuliert werden, dass alle Verben aktiv und aus Objektsicht geschrieben werden.
- Manchmal ist es schwierig Messagetypen und Messagereihenfolge in SML aus den User Stories abzuleiten. Um dies zu erleichtern werden die User Stories -wenn möglich- mit einigen wenigen Formulierungsregeln aufgeschrieben, zu denen es eine Überführungsfunktion gibt, um daraus direkt einen Code zu generieren.
- Um auf Anforderungsänderungen, also Änderungen an einer User Story in SML reagieren zu können, werden die Scenarios mit den User Stories verlinkt. Zusätzlich wird auch noch der Stakeholder in das Scenario eingetragen. Dies hilft nicht nur bei dem schnellen Finden von den Änderungen betroffener Scenarios, sondern erleichtert das Verständnis beim Lesen des SML Codes.

[1] es gibt in der Methode eine einzige Collaboration

Dennoch ist die Methode ein kreativer Prozess, der (noch) nicht automatisiert werden kann und einen Modellierer benötigt.
So ist die *Strictness* einer Message eine Technik der Modellierung und kann nicht immer aus den User Stories extrahiert werden. Verwendet man die vereinfachten User Stories ohne Titel, so muss der Modellierer das *Ziel* einer Karte selbst herauslesen und dies als Titel verwenden, wenn er ein neues Scenario erstellt. Die schwierigste Aufgabe zu erkennen ob eine User Story ein anderes Scenario erweitert, ist noch nicht vollständig gelöst. Die Metrik, dass sich beeinflussende User Stories zu dem selben oder einem angrenzenden Use Case Schritt verlinkt sind ist sinnvoll, deckt aber nicht alle möglichen Fälle ab.

Nun ist es möglich, aus den User Stories SML Scenarios zu erstellen. In vielen Fällen fehlen aber noch Informationen und eine User Story kann nicht eins zu eins in SML umgesetzt werden. Dies ist nicht überraschend, da die Methode sich immer noch im Prozess der Systemanalyse befindet und noch nicht alle Anforderungen erhoben wurden. Mit der 3. Activity der Methode werden die unklaren User Stories noch einmal mit dem Kunden besprochen und entsprechend markiert. Dadurch werden Verständnisfragen geklärt, Vorschläge mit dem Kunden validiert, um danach die User Stories wie besprochen zu ändern oder neue User Stories zu erstellen. Durch die Markierung findet man schnell die betroffenen User Stories und kann diese in der nächsten Iteration mit der 1. Activity (*create Enhanced User Stories*) der Methode ändern und dank der Verlinkungen die betroffenen Scenarios in der 2. Activity (*create Enhanced Scenarios*)schnell finden und aktualisieren.

Anschließend wurde analysiert, für welche Probleme sich diese Methode anbietet, und für welche sie sich als ungeeignet erweist. Außerdem wurde untersucht, wie man die Methode am besten in verschiedene RE Prozesse einbaut und welche Kosten und welchen Mehrwert die Anwendung mit sich bringt. So kann die Methode in einer Systemanalyse nach dem RE-Referenzmodell in der Elicitation und Negotiation eingesetzt werden. Die Aufwandserhöhung liegt sowohl in der Erstellung der SML Specification und in der Besprechung der Fragekarten mit dem Kunden. Die Kundenzeit könnte in vielen Projekten eine der knappsten Ressourcen sein. Verwendet man die Methode agil, entsteht ein Zwischenschritt bei der Implementierung. So könnten neue User Stories erst in SML umgesetzt und simuliert werden, bevor sie im nächsten Sprint implementiert werden. Hier können sich allerdings Wartezeiten akkumulieren, falls Fragekarten nicht be-

sprochen werden können. Das kann die Erstellung SML Specification verzögern, was wiederum zu einer Verzögerung der Implementierung führt (und damit zu zusätzlichen Kosten führt).

Abschließend zeigt diese Arbeit, dass eine Umsetzung zwischen User Stories und formalen Szenarien durchaus möglich ist, viele Vorteile bietet und sowohl im normalen RE-Prozess als auch in einem agilen oder formal betriebenen Arbeitsprozess einsetzbar wäre. Allerdings ist die entwickelte Methode nicht für jeden Aufgabenbereich geeignet und müsste für eine Softwareanalyse noch weiter angepasst und für eine sichere Aufwandsabschätzung evaluiert werden. Auch kann die Methode (noch) nicht automatisiert werden, und es gibt ein paar ungelöste Probleme und viel Verbesserungsbedarf in der Tool-Unterstützung.

6.1. Ausblick

Der nächste Schritt wäre eine Evaluation der Methode, um die wirklichen Kosten und Nutzen herauszufinden und zu untersuchen, ob sie in der Praxis so angewendet werden kann, oder ob sich nach kurzer Zeit die Probanden nicht mehr an die Methode halten und die Scenarios "aus dem Bauch heraus" erstellen.

Zunächst könnte die Erstellung der *Enhanced User Stories* beobachtet werden. Dazu wird einem Probanden ein Problem mit fertigen Use Cases und User Stories zugeteilt und dieser fängt nach einer Einlesezeit an Enhanced User Storys zu erstellen. Anschließend wird der Proband nach seiner Einschätzung zur Schwierigkeit befragt. Anschließend versucht dieser, die gerade erstellten Enhanced User Stories in SML umzusetzen und Vorschläge und Fragen zu erstellen. Eine andere Gruppe bekommt die gleiche Aufgabe und versucht sie ohne die erarbeitete Methode durchzuführen. Hierbei wird jeweils die Zeit gemessen und gezählt, wie viele Anforderungen in SML sinnvoll umgesetzt und wie viele neue Anforderungen entstehen würden.

Um das Arbeit mit der Methode zu erleichtern, können noch einige Tool-Verbesserungen durchgeführt werden:

- Momentan sind die Verknüpfungen zwischen User Stories und Scenarios durch Kommentare im Quellcode realisiert. Würde sich die Integrated Requirements Environment (IRE) in Eclipse und Scenario-Tools integrieren

lassen, so könnten Links in einer Datenstruktur gespeichert sein. Dadurch könnte man sich per Knopfdruck die User Stories zu einem bestimmten Scenario anzeigen lassen, effizient nach Tags filtern und sich alle unmodellierten, modellierten oder als "changed" markierten User Stories anzeigen lassen.

- Eine Tool-unterstütztes Testverfahren für Existential Scenarios und eine entsprechende Verknüpfung zu den Use Case, oder sogar eine Generierung aus diesen, würde die praktische Anwendbarkeit von Existential Scenarios erhöhen. Dadurch könnte man diese in die Methode aufnehmen.
- Eine Erkennung oder Einfärbung der Schlüsselwörter der Überführungstabelle würde das Schreiben von Scenarios erleichtern.
- Die Möglichkeit, auch andere Quellen wie Schnittstellenbeschreibungen mit Scenarios zu verlinken, würde bei der Umsetzung genauerer Scenarios helfen. So könnten Informationen, die für User Stories zu detailliert sind, in die Modellierung mit einfließen und das Tracing vereinfachen.
- Vorlagen für Fragen- und Vorschlagkarten wären hilfreich. Die könnten dann mit der User Story verbunden und bei deren Aufruf automatisch angezeigt werden.

Die Methode selbst besitzt noch einige Verbesserungsmöglichkeiten.
Zunächst gibt es, wie in Kapitel 4 beschrieben, noch einige ungelöste Probleme, wie beispielsweise das Einbinden von dynamischen Rollen oder eine Technik um strict Messages zu erkennen.

Eine interessante Idee wäre, die Methode um Miss Use Cases zu erweitern. Es könnte auch untersucht werden, was verändert werden müsste, um die Methode für die Softwareanalyse zu verwenden und wie dort die Mensch-Maschine-Schnittstelle modelliert werden müsste.

A. Literaturverzeichnis

Literaturverzeichnis

[Hul10] HULL, ELIZABETH; KEN JACKSON; JEREMY DICK und JEREMY DICK: *Requirements Engineering*, Springer Science & Business Media, 2010

[Pre15] PRECHELT, L., BRÜGGE B., DUTOIT, A. und SCHNEIDER, K.: *Vorlesung SSoftwaretechnik": Software-Prozessmodelle*, Freie Universität Berlin, Institut für Informatik, Arbeitsgruppe Software Engineering URL: `www.inf.fu-berlin.de/inst/ag-se/teaching/V-SWT-2015/44_Prozessmodelle.pdf` (abgerufen: 18.04.16)

[Coh05] COHN, MIKE: *Agile estimating and planning*, Pearson Education, 2005

[Coh04] COHN, MIKE: *User stories applied: For agile software development*, Addison-Wesley Professional, 2004

[Jam12] JAMES, MICHAEL *Scrum Reference Card*, URL: `http://scrumreferencecard.com/scrum-reference-card/`, (abgerufen 19.04.16)

[Nas02] NATIONAL AERONAUTIC AND SPACE ADMINISTRATION: *What is Formal Methods?*, URL: `http://shemesh.larc.nasa.gov/fm/fm-what.html` (abgerufen: 19.04.16)

[Rup13] RUPP, CHRIS: *Systemanalyse kompakt*, Springer-Verlag, 2013

[Bec00] BECK, KENT: *Extreme programming explained: embrace change*, Addison-Wesley Professional, 2000

[Lef10] LEFFINGWELL, DEAN: *Agile software requirements: lean requirements practices for teams, programs, and the enterprise*, Addison-Wesley Professional, 2010.

[Wik1] WIKIMEDIA FOUNDATION INC.: *User-Story*, URL: `https://de.wikipedia.org/wiki/User-Story` (abgerufen: 19.04.16)

[Gre16a] GREENYER, JOEL, ET AL.: *Scenarios@ run. time–Distributed Execution of Specifications on IoT-Connected Robots*, URL: `http://jgreen.de/wp-content/documents/2015/scenarios-at-runtime.pdf`, 2016, (abgerufen: 19.04.16) (non published)

[Gre16b] GREENYER, JOEL, ET AL.: *Scenario-based Specification of Car-to-X systems*, URL: `http://ceur-ws.org/Vol-1559/paper12.pdf`, 2016, (abgerufen: 19.04.16) (non published)

[Sch15] SCHNEIDER, KURT: *Aus der Vorlesung "Requirements Engineering"*, Leibniz Universität Hannover, Fachgebiet Software Engineering, Sommersemester 2015

[Gre16c] GREENYER, JOEL: *Aus der Vorlesung "Formal Methods in Software Engineering"*, Leibniz Universität Hannover, Fachgebiet Software Engineering, Sommersemester 2015

[Har03] HAREL, DAVID und RAMI MARELLY: *Come, let's play: scenario-based programming using LSCs and the play-engine*, Springer Science & Business Media, 2003

[Sce16] SCENARIOTOOLS: URL: `http://scenariotools.org/` (abgerufen: 19.04.16)

[Gor64] GORDON, MICHAL, ASSAF MARRON und ORNI MEERBAUM-SALANT: *Spaghetti for the main course?: observations on the naturalness of scenario-based programming*,Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education. ACM, 2012.

[Ire16] LEIBNIZ UNIVERSITÄT HANNOVER FACHGEBIET SOFTWARE ENGINEERING: *Das IRE-Tool* URL: `http://www.se.uni-hannover.de/pages/de:ire-tool` (abgerufen: 19.04.16)

[Lis12] LISKIN, OLGA *How artifacts support and impede requirements communication*: Requirements Engineering: Foundation for Software Quality. Springer International Publishing, 2015 S. 132-147

B. Anhang

- Schlussfolgerung Tabellen

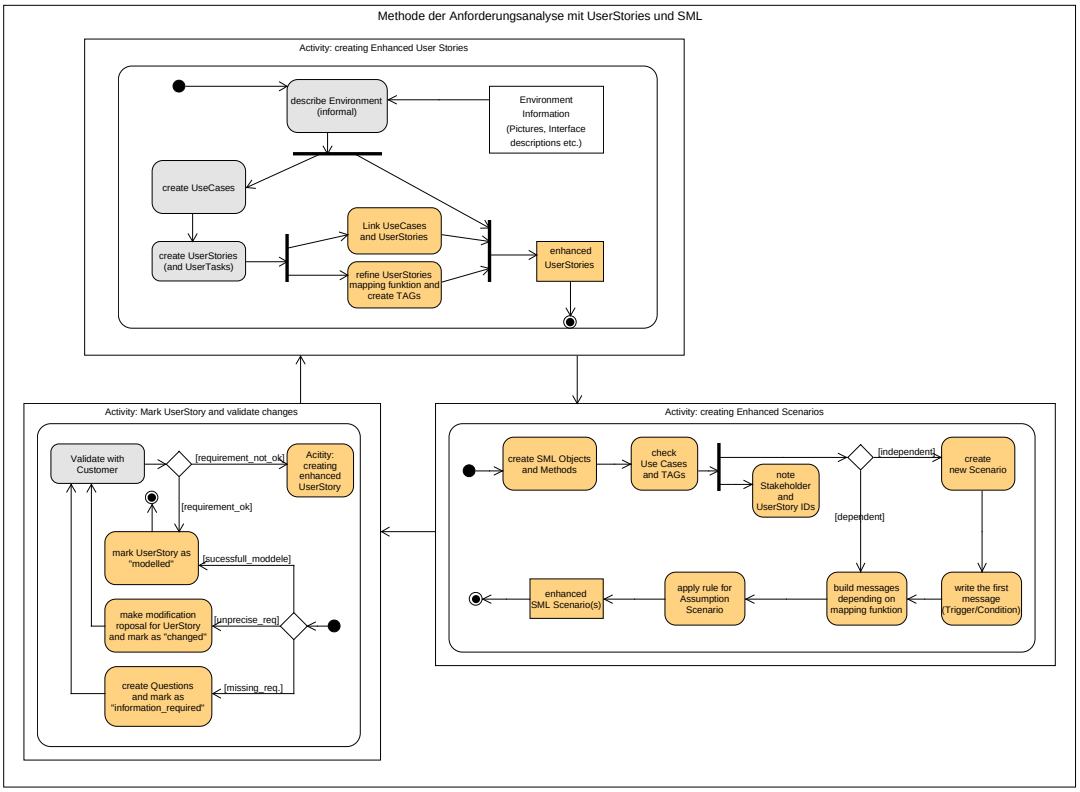

Abbildung B.1.: Activity Diagramm der Methode

C. Inhalt der DVD

Zusätzlich zur gedruckten Ausgabe liegt dieser Arbeit eine DVD bei.

Die DVD enthält:

- Die vollständige schriftliche Ausarbeitung im PDF- und LATEX -Format
- Das Beispiel der Orangenpresse als Eclipse-Scenario-Tools-Projekt und StoryBoard mit Use Cases als Textdatei und als Bilder

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 19. April 2016

_____

Sebastian Föllmer