

# The 2010 Salespoint Project Course - What Team WINF02 Actually Did

*described by Lothar Schmitz*

There may be considerable difference between how a project course is designed and what actually happens in the course. The aims and concepts of our project course are described elsewhere (see next page). Here, we try to give you an impression of what it is like to do a project based on the Salespoint framework as a student. We do this by showing you some of the artefacts that a student team has produced during the 2010 project course at our place, the University of the German Federal Armed Forces in Munich.

Naturally, this course was held in German and thus all documents are German, as well. In order to make this material accessible to an international audience, we are going to show you screenshots of the original artefacts as produced by the students and explain them in English. This will hopefully give you a general idea of what was going on ...

We assume that you have already read one of our English papers on the Salespoint framework and its potential advantages when used as the basis of a beginners' project course introducing them to team work in software development (see <http://www.salespoint-framework.org/publications>, e.g., the CSEE&T 2000 paper).

Before turning to team WINF02's materials we briefly describe the contents of the project homepage (<http://www.unibw.de/inf2/Lehre/HT10/progproj>). This is where the students would find all they had to know about the course (and where they would continuously document their own progress - visible only internally):

- General outline of the course and its aims
- Helpful literature on UML, Java, JUnit, Design Patterns etc.
- Persons involved and their respective roles
- Overview of the project phases and deliverables
- Detailed directions and guidance for each project phase
- Choice of twelve different assignments - one for each team

Now let us take a look at team WINF02's materials ...

The text on the right hand side is the **original task assignment** the students were given by their tutor.

The task is to develop a program for managing a *Getränkeller* (German word for a cellar room where beer and other beverages are kept) in the student dormitories.

The program will grant the inhabitants password/barcode-protected access to the cellar and for that purpose will keep accounts for all the students.

Also, the program will help the *Getränkewart* (the manager of the *Getränkeller*) with his duties, in particular to keep the stocks well filled, and to ensure that all debts are cleared regularly.

As described on the project home page, team WINF02 will now have to analyze the assignment in detail ...

## Getränkellerverwaltung

Im Wohnbereich 20/200 gibt es einen Getränkekeller, durch den die 72 Bewohner mit Getränken aller Art versorgt werden. Zurzeit werden alle Getränke anhand einer Strichliste abgerechnet. Um dem Getränkekellerwart die Arbeit zu erleichtern und die Übersicht über den Verbrauch der Getränke zu erhöhen soll nun eine Getränkekellerverwaltungssoftware eingesetzt werden.

Bei der Software werden zwei verschiedene Sichten benötigt, die des Getränkekellerwartes und die des normalen Benutzers, in die man jeweils mit einer Sicherheitsabfrage gelangt. Diese sollte sinnvoll sein, z.B. Benutzername/Passwort und/oder mit einer Barcodekarte.

In der Sicht des Benutzers soll es möglich sein, Getränke zu entnehmen und diese Entnahmen zu protokollieren. Diese Bedienung soll soweit wie möglich mit einem Barcode-Scanner erfolgen können. Dabei soll auch der bisherige Verbrauch und bei Bedarf eine Erinnerung an das Bezahlen der Rechnung angezeigt werden können.

In der Sicht des Getränkekellerwartes soll der komplette Bestand verwaltet werden können. Dabei ist es wichtig, dass man auch nach einer Inventur die Werte manuell anpassen kann, wobei der entstandene Fehlbetrag auf alle Benutzer umgelegt werden soll. Jede Bestandsbewegung, sowie Bewegung von Geld soll in einem Protokoll erfasst werden. Beim Begleichen einer Rechnung beim Getränkekellerwart, soll diese aus dem System ausgetragen werden können, wobei auch jederzeit möglich sein soll, einen höheren Betrag, als der geforderte einzuzahlen. Außerdem soll der Getränkekellerwart nach einer Lieferung den hinzugekommenen Bestand hinzu buchen können. Beim Erstellen einer Abrechnung, soll diese exportiert werden können. Außerdem ist eine denkbare Funktion, die Rechnungen direkt bei der Erstellung personenbezogen per E-Mail zu versenden.



Um eine Bestellung planen zu können, soll eine Statistik angezeigt werden können, in der der Verbrauch pro Tag, Woche und Monat und seit der letzten Bestellung angezeigt werden kann. Eine weitere Statistik soll die offenen Rechnungen und den offenen Gesamtbetrag anzeigen können.

In einer Bewohnerverwaltung soll es möglich sein, diese hinzuzufügen, zu löschen und zu bearbeiten. Falls der Getränkekellerwart wechseln sollte, soll es möglich sein die Rechte zu übergeben, wobei immer nur ein Bewohner der Getränkekellerwart sein kann.

Um die Getränketypen zu verwalten wird eine Getränkeverwaltung benötigt, in der neue Getränke angelegt, vorhandene bearbeitet und gelöscht werden können.



On the project homepage, the **detailed directions** for the **analysis phase** tell the students to

- *fill in details* missing from the assignment description and *resolve ambiguities*
- make a *list of all use cases* and a UML use case diagram
- prepare some *GUI sketches* and possibly a storyboard
- iterate the above steps until a stable analysis model is established delineating the future system's functionality
- draw a coarse-grained UML class diagram showing the static structure of your domain
- write a first version of the user manual
- set up the structure of a test handbook covering all the above use cases

1. Den Anfangspunkt der Analyse bildet verständlicherweise die jeweilige Aufgabenstellung, welche zunächst im Gruppenrahmen diskutiert werden sollte. Hierbei werden sicherlich Lücken in der Themenstellung und Fragen zur gewünschten Funktionalität auftreten, welche zusammen mit dem Auftraggeber geklärt werden müssen. Im Ergebnis entsteht hierbei eine detaillierte bzw. **erweiterte Aufgabenstellung** (in textueller Form).
2. Aus dieser wird dann eine **Use-Case Liste** extrahiert. Diese Anwendungsfälle sollen durch kurze Teilsätze (z. B.: "Kunde anlegen", "Photoauftrag im Internet annehmen") beschrieben werden. Weitere Anwendungsfälle finden sich beim Spielen von Szenarien und deren Protokollierung (Szenarien sollen hierbei konkrete, vorstellbare Abläufe innerhalb des Programms widerspiegeln, also z.B. "Nicht registrierter Kunde sendet Photoauftrag per Post ein"). Die Liste dieser Anwendungsfälle soll alle möglichen, durch die Aufgabe implizierten, Funktionalitäten des Programms wiedergeben. Eine Priorisierung der Anwendungsfälle hilft, Schwerpunkte für die spätere Implementierung zu setzen. Diese ersten zwei Schritte sind entscheidend für den späteren Funktionsumfang des Programms, weshalb die Ergebnisse regelmäßig mit dem Auftraggeber abgestimmt werden sollten.
3. Aus der Use-Case Liste sollen nun **Use-Case Diagramme** erstellt werden. Dazu werden die einzelnen Anwendungsfälle zu Funktionsgruppen zusammengefasst. Die Diagramme sind zur leichteren Verständlichkeit wo immer nötig mit Kommentaren und Beschreibungen zu versehen. Desweiteren werden **Beschreibungen zu den Anwendungsfällen** und **Ablaufbeschreibungen** verfasst.
4. Die gefundenen Anwendungsfälle und durchgespielten Szenarien können zusätzlich als Leitfaden für zu erstellende **Skizzen der späteren Benutzeroberflächen** (sowie einer zugehörigen **Gestaltungsrichtlinie**) dienen, welche in ihrer Gesamtheit gleichzeitig auch die Rolle eines Storyboards für die zu erstellende Anwendung einnehmen können.
5. **Iterieren** Sie nun so lange über die vorangegangenen zwei (evtl. auch drei) Schritte, bis sich ein **stabiles Analysemodell** herauskristallisiert (sie also merken, dass zunächst erstmal keine neue Funktionalität mehr hinzu kommt). Insgesamt gesehen, sollte sich für den Auftraggeber ein attraktives Produkt mit abgerundeter Funktionalität abzeichnen. Das ein oder andere geplante "Highlight" zeigt dem Kunden zusätzlich, dass die Entwickler seine Interessen im Auge haben - bevor man jedoch beim Auftraggeber mit konkreten Vorschlägen Erwartungen weckt, sollte man zumindest prinzipiell deren Umsetzbarkeit durchdacht haben. Legen Sie diesen Produktvorschlag nun nochmals dem Kunden vor und klären Sie mit diesem erneut offene Fragestellungen ab.
6. Im Anschluss werden die bisherigen Ergebnisse aus im Hinblick auf die spätere Umsetzung weiter verfeinert. Zuerst wird hierzu ein **Paketdiagramm** erstellt. Strukturierungsmöglichkeiten finden sich zum Beispiel durch Zuordnung der Anwendungsfälle zu ihren realen Benutzern.
7. Für die Erstellung der **Klassendiagramme** kann wieder auf die erweiterte Aufgabenstellung zurückgegriffen werden. In dieser Phase sollen die Klassendiagramme nur grobgranular gestaltet werden, d.h. die Angabe von Typen und Sichtbarkeiten ist sehr implementierungsnah und kommt deshalb erst in der Designphase zum Einsatz.
8. Weiterhin erstellen Sie ein vorläufiges **Benutzerhandbuch**, welches den anskizzierten Funktionsumfang textuell aus Anwendersicht beschreibt. Im Normalfall spiegeln sich die Anwendungsfallgruppen in der Gliederung wider; die angefertigten GUI-Skizzen können der Verbillidung dienen.
9. Schlussendlich ist noch der erste Abschnitt (die *Systemtests*) des **Testhandbuchs** zu erstellen, welcher die bis dato ersichtlichen Fehlerquellen (siehe Use-Cases und Szenarien) im Programmablauf sowie die erhobenen, funktionalen Anforderungen abdeckt.

Team WINF02 started by producing an **extended version** of the original task description they had been given by the customer - who supposedly does not know much about computers.

The aim was to fill in all the details that were missing in the original description and to make precise the meaning of all the requirements.

On the right hand side, you can see the (first part of the) extended description as produced by team WINF02. The original description is still visible in black; any extensions and clarifications added by the team are shown in orange.

This document is the result of intensive discussions within the team and of negotiations with the customer (who is represented by the tutor), who finally approved it as **new contractual basis**.

## Erweiterte Aufgabenstellung

Im Wohnbereich 20/200 gibt es einen Getränkekeller, durch den die 72 Bewohner mit Getränken aller Art versorgt werden. **Zudem sollen 10 hausfremde Gäste den Getränkekeller benutzen können. Ein Bewohner wird durch Stubennummer(ID), Name, Kontostand, Verbrauch, eMail- Adresse und eigenes Passwort charakterisiert. Die Gäste erhalten anstatt der Stubennummer eine 4stellige ID der Form 3XXX.** Zurzeit werden alle Getränke anhand einer Strichliste abgerechnet. Um dem Getränkekellerwart die Arbeit zu erleichtern und die Übersicht über den Verbrauch der Getränke zu erhöhen soll nun eine Getränkekellerverwaltungssoftware eingesetzt werden.

Bei der Software werden zwei verschiedene Sichten benötigt, die des Getränkekellerwartes und die des normalen Benutzers, in die man jeweils mit einer Sicherheitsabfrage gelangt. Diese sollte sinnvoll sein, z.B. Benutzername/Passwort und/oder mit einer Barcodekarte.

In der Sicht des Benutzers soll es möglich sein, Getränke zu entnehmen und diese Entnahmen zu protokollieren. **Dabei werden neben dem Benutzernamen und des Datum und der Zeit auch die Getränkesorte, -menge und der aktuelle Preis erfasst.** Diese Bedienung soll soweit wie möglich mit einem Barcode-Scanner erfolgen können. Dabei soll auch der bisherige Verbrauch **(mit Datum und Uhrzeit sowie Sorte und Menge)** und bei Bedarf eine Erinnerung an das Bezahlen der Rechnung angezeigt werden können. **Der Verbrauch wird mit Datum, Uhrzeit, Getränkesorte und -menge, sowie Betrag angezeigt. Die Rechnungs-Erinnerung wird mit den 3 letzten ausstehenden Rechnungsbeträgen, dem Datum der Rechnungsstellung und dem aktuellen Kontostand ausgegeben. Ein angemeldeter Bewohner soll 3 Minuten nach seiner letzten Aktion automatisch abgemeldet werden (Timeout). Sollte sich ein anderer Bewohner während dieser 3 Minuten mit seiner Barcodekarte am System anmelden, wird der letzte Nutzer abgemeldet. Ein Bewohner soll sein eigenes Passwort ändern können.**

In der Sicht des Getränkekellerwartes soll der komplette Bestand verwaltet werden können. **Der Bestand ist die Menge aller vorhandenen Flaschen im Getränkekeller und der Geldbetrag in der Getränke-Kasse.** Dabei ist es wichtig, dass man auch nach einer Inventur die Werte manuell anpassen kann, wobei der entstandene Fehlbetrag auf alle Benutzer umgelegt werden soll(.), **die in der Rechnungsperiode Getränke entnommen haben.** Jede Bestandsbewegung, sowie Bewegung von Geld soll in je einem Protokoll erfasst werden.

Next, team WINF02 identified and listed all the **use cases** implied by the task description, gave a **detailed description** of each use case and produced **UML use case diagrams**:

**Use Case:**  
**Actors:**  
**Purpose:**  
**Entry Cond:**  
**Overview:**

/UC-202/ Eigenen Kontostand anzeigen  
 Nutzer  
 Kontrolle des eigenen Kontostands  
 Angemeldet sein  
 Der Nutzer kann seinen Kontostand kontrollieren.

Actors	System Response
	1. Das System zeigt den aktuellen Kontostand des angemeldeten Bewohners an

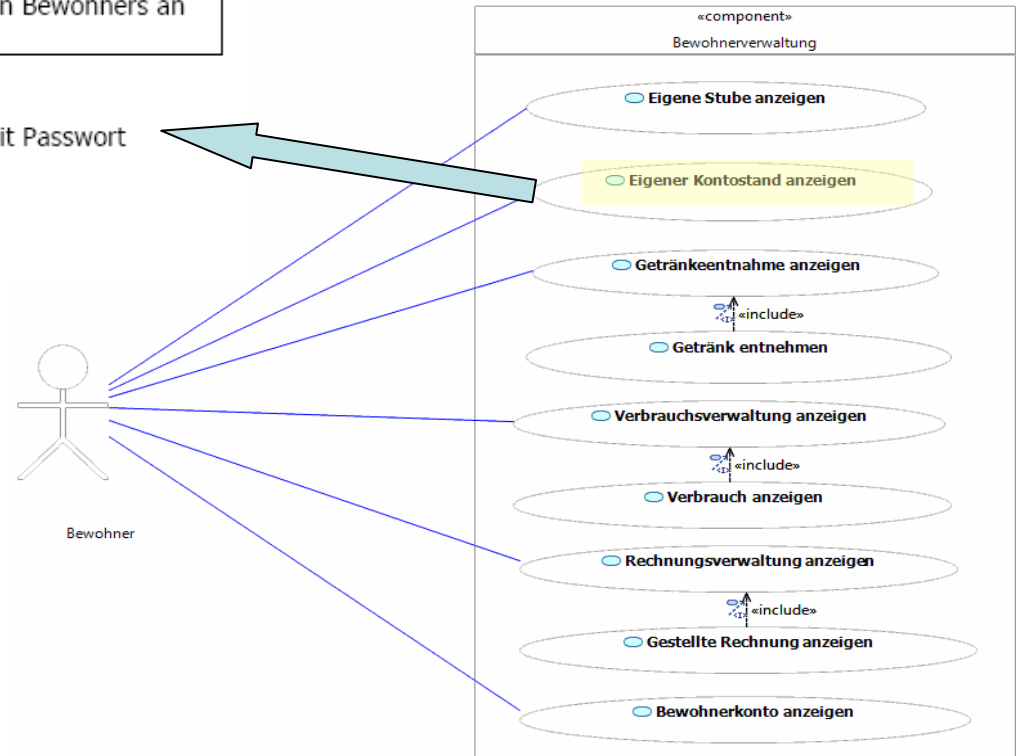
**Exit Cond:**  
**Uses:**  
**Special Req:**  
**Category:**  
**Cross Ref:**

Kontostand wird angezeigt  
 ID und Passwort prüfen, Kontostand anzeigen  
 Überprüfen ob eingegebene Stubennummer mit Passwort übereinstimmt bzw. ob Barcode korrekt ist.  
 mittlere Priorität  
 /UC-202/ und /GUI-\*.Bewohner/

## Bewohner

/UC-201/ Eigene Stube anzeigen  
 /UC-202/ Eigener Kontostand anzeigen  
 /UC-203/ Letzte Entnahme anzeigen  
 /UC-204/ Bewohner automatisch abmelden

/UC-210/ Getränkeentnahme anzeigen  
 /UC-211/ Getränk entnehmen  
 /UC-212/ Letzte Getränkeentnahme stornieren



From these more technical documents, a first version of the **User Manual** was derived. Here, we just show the page which explains one of the main use cases: *removing a beverage*.

## 4.2 Bewohner

Nach der Anmeldung werden folgende Informationen stets angezeigt:

- Datum und Uhrzeit der letzten Anmeldung
- Betrag der letzten Entnahme
- Stubennummer
- Kontostand

### 4.2.1 Getränkeentnahme

Der Getränkeentnahme-Bildschirm wird nach dem Anmelden automatisch angezeigt und kann danach jederzeit per Scannen des Barcodes „Getränkeentnahme“ oder über die Navigationsleiste erreicht werden.

Die Summe aller aktuellen Entnahmen wird als Betrag angezeigt.

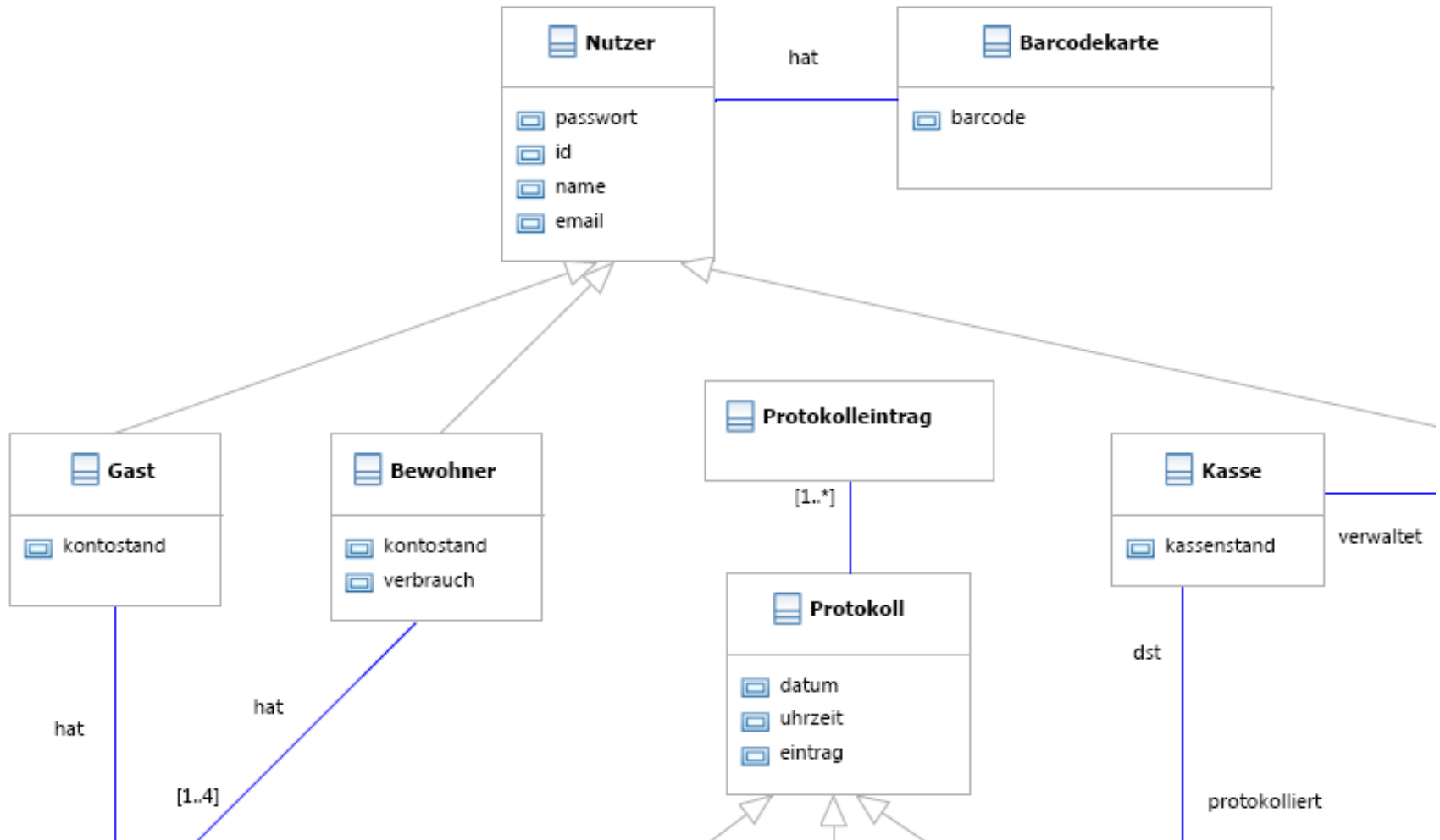
#### 4.2.1.1 Getränk entnehmen

Eine Getränkeentnahme wird gebucht durch

- Scannen des Barcodes der Flasche oder
- Eingabe des Barcodes der Flasche in das Eingabefeld und Bestätigung mit [Enter].

Das Getränk wird automatisch mit Menge und Preis in die Liste der Entnahmen eingefügt. Um mehrere Flaschen eines Getränks zu entnehmen, muss die Getränkeentnahme wiederholt werden oder vereinfacht der entsprechende Multiplikator vor der gewünschten Flasche gescannt oder eingegeben werden.

Team WINF02 also described the **static structure** of the program (as understood so far) in the form of a **UML class diagram**, the top part of which is shown below:



Finally, hand-drawn **GUI sketches** (aka *wireframe models*) like the following were produced. This sample sketch shows the screen to be used for removing beverages.

Logo Glt VS

Logo Unidew

Detail Anmeldung: dt.mm.jj hh:mm / Betrag / Letzte Entf.

Stube: XXXX

KontoStand: X€

Getränke-entnahme

Verbrauchs-Verwaltung

Barcode

Name (Getränk)	Preis	Menge
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____

Rechnung-Verwaltung

Mein Konto

Abmelden

Summe: X€

In an intermediate phase between analysis and design the students learned about the **Salespoint framework**:

- Its general structure and components:  
Catalogues, Stocks, User Management, Logging Facilities etc
- How to model User Interactions as Salespoint Processes
- How to adapt Salespoint components

The rationale for postponing the introduction to Salespoint until after the analysis phase is that we want to keep analysis free from implementation considerations. Once the students have grasped the concepts of Salespoint, functional analysis tends to be guided (or even narrowed) towards solutions that are 'Salespoint-compatible'.

Students acquire the basics of Salespoint by

- attending a *presentation* given by their tutors
- reading an *introductory text*
- then working through a *tutorial*
- and finally *solving small problems*,  
typically adapting some Salespoint code to concrete requirements

The **exercises** for Team WINF02 were all related to a *video rental service*. Among other things, they learned how to *filter the stock* of the video service ...

```
Getraenkekeller/Einarbeitungsaufgabe/Marcus/VideoautomatMeins/src
import videoautomat.core.data.Video;

public class DataVideoFilter extends CatalogFilter<CatalogItem>
{

    public DataVideoFilter(Catalog<CatalogItem> catalog)
    {
        super(catalog);
    }

    protected boolean match(Video item)
    {
        // all notes and coins less worth than 5 Euro are filtered
        if(item.getFSK () >= 18)
            return false;
        else
            return true;
    }
}
```

This comment obviously still refers to the code this filter was modelled on.

The new filter returns videos that are suitable for children (age<18) according to *Freiwillige Selbst-Kontrolle*.

## ... and how to set up a Salespoint process using FormSheets, Gates and Transitions.

Getraenkekeller/Einarbeitungsaufgabe/Marcus/VideoautomatMeins/src

```
import org.salespointframework.core.FormSheet;
import org.salespointframework.core.FormSheetContentCreator;
import org.salespointframework.core.Gate;
import org.salespointframework.core.SaleProcess;
import org.salespointframework.core.SalesPoint;
import org.salespointframework.core.Transition;
import org.salespointframework.core.UIGate;
import org.salespointframework.core.users.User;

import videoautomat.desktop.processes.SaleProcessLogOn;

public class AdministrationContentCreator extends FormSheetContentCreator {

    private static final int BUTTON_BACK = 1001;

    @Override
    protected void createFormSheetContent(FormSheet fs) {
        fs.removeAllButtons();
        fs.addButton("Back", BUTTON_BACK, new Action() {

            public void doAction(SaleProcess p, SalesPoint sp) throws Throwable {
                ((UIGate) p.getCurrentGate())
                    .setNextTransition(new Transition() {

                        public Gate perform(SaleProcess owner, User usr) {
                            return ((SaleProcessLogOn) owner).getMainGate();
                        }

                    });
            }

        });
    }
};
```

On the project homepage, the detailed directions and guidance for the **design phase** tell the students, that while analysis was about *what*, the design is about *how* things are done. This will include a lot of new detailed decisions.

In particular, they have to adapt the *class diagram* from the analysis phase to the scope and options of Salespoint.

Also, they have to recast all user interactions into the Salespoint *process model*. This, again, requires many adaptations.

Ziel dieser Phase ist die Übersetzung des nichttechnischen Analysemodell (das **WAS**) der ersten Phase in ein **technisches Designmodell** (das **WIE**). Dies geschieht typischerweise in mehreren Durchläufen, welche immer feinere "Baupläne" für die zu erstellende Software (vgl. Grobentwurf/ Feinentwurf) zum Ergebnis haben, die dann in der anschließenden Phase der Implementierung fließend in die Codierung übergehen.

#### **Vorgehensweise:**

Es ist zunächst ein erster Abgleich zwischen dem Analyseergebnis und dem genutzten Framework (das Wissen hierüber sollte man sich in der Einarbeitungsphase angeeignet haben!) erforderlich, um festzustellen

- welche der geforderten Funktionen das Framework in Form von **Standardfunktionalität** (Fall 1) abdeckt
- welche weiteren Funktionen es nach **Anpassungen** (Fall 2) erfüllt, und wie diese Anpassungen genau aussehen
- was darüber hinaus an **Erweiterungen** (Fall 3) notwendig ist, und wie diese einzubinden sind

Dies ist sowohl für das sich in Form von statischen und dynamischen UML-Diagrammen äußernde Fachkonzept (hier müssen die vorliegenden Diagramme in eine technische Sicht **detailliert** werden), für den GUI-Prototypen (hier müssen die vorliegenden GUI-Skizzen in eine technische Sicht **übersetzt** werden), als auch für sich erst in technischer Sicht ergebenden Anteile (diese müssen vollständig **neu entworfen** werden) zu erledigen. Diese erste Abschätzung manifestiert sich in einem **Grobentwurf**, welcher auf architektureller Ebene die identifizierten, funktionalen Blöcke beschreibt.

Im hierauf folgenden Feinentwurf erfolgt nun, abhängig vom jeweils notwendigen Grad der Anpassung, die detaillierte Modellierung der **Umsetzung** auf Basis der Standard-Schnittstellen des Frameworks (im Fall 1 genügt dies), eventuell eine **Adaption bzw. Erweiterung** selbiger Schnittstellen (ausreichend für Fall 2) oder sogar der Entwurf **zusätzlicher Komponenten** (siehe Fall 3). Vor allem in letzteren beiden Fällen ist es natürlich notwendig, dies ausführlich (auch textuell) zu dokumentieren.

Regarding the **UML class diagram**, team WINF02 identified the following adaptations to be necessary. The next page shows a small cutout from the resulting, detailed diagram.

## Anpassungen

### Analysephase zu Designphase

Änderung in der Nutzerklasse (v1) zu Nutzerklasse (v2)

- der Nutzer erbt von der SalespointKlasse User, da dort schon viele vordefinierte Funktionen vorhanden sind die für eine Nutzerverwaltung notwendig sind
- Die Klasse Barcodekarte entfällt, der Nutzer erhält dafür ein Attribut

Änderung in der Kasse (v1) zu Kasse (v2)

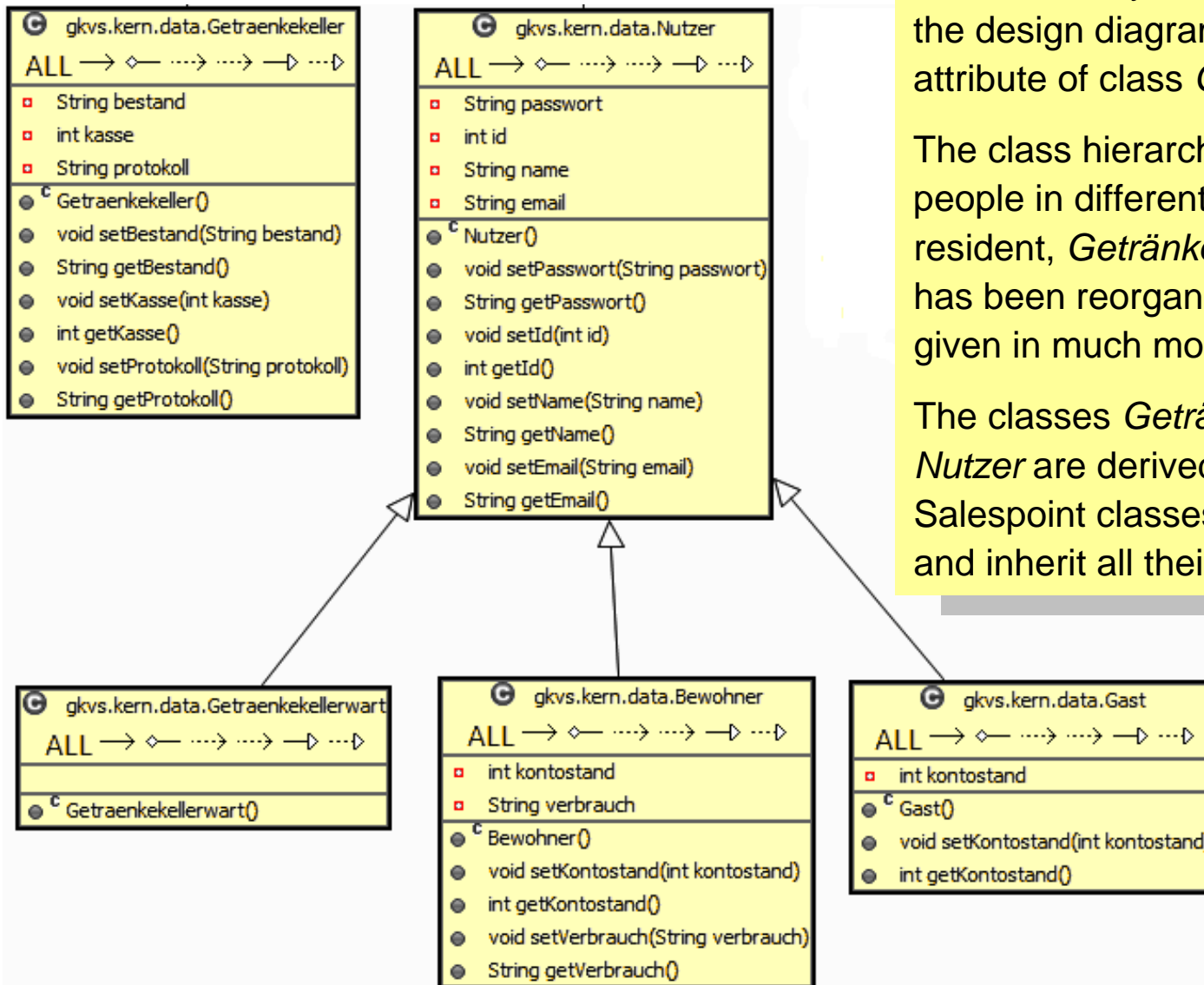
- Die Klasse Kasse entfällt, der Getränkekeeper erhält dafür ein Attribut Kassenstand

Änderung in der Klasse Protokoll (v1) zu Protokoll (v2)

- Protokoll ist Oberklasse und vererbt an Entnahme-, Bestands- und Kassenprotokoll

Grundlegende Veränderungen:

- Getränkekeeper erbt von CountingStock
- Getränk erbt von CatalogItemImpl
- Verbrauch erbt von StoringStockImpl
- Bestand erbt von CatalogImpl
- Protokoll erbt von Log



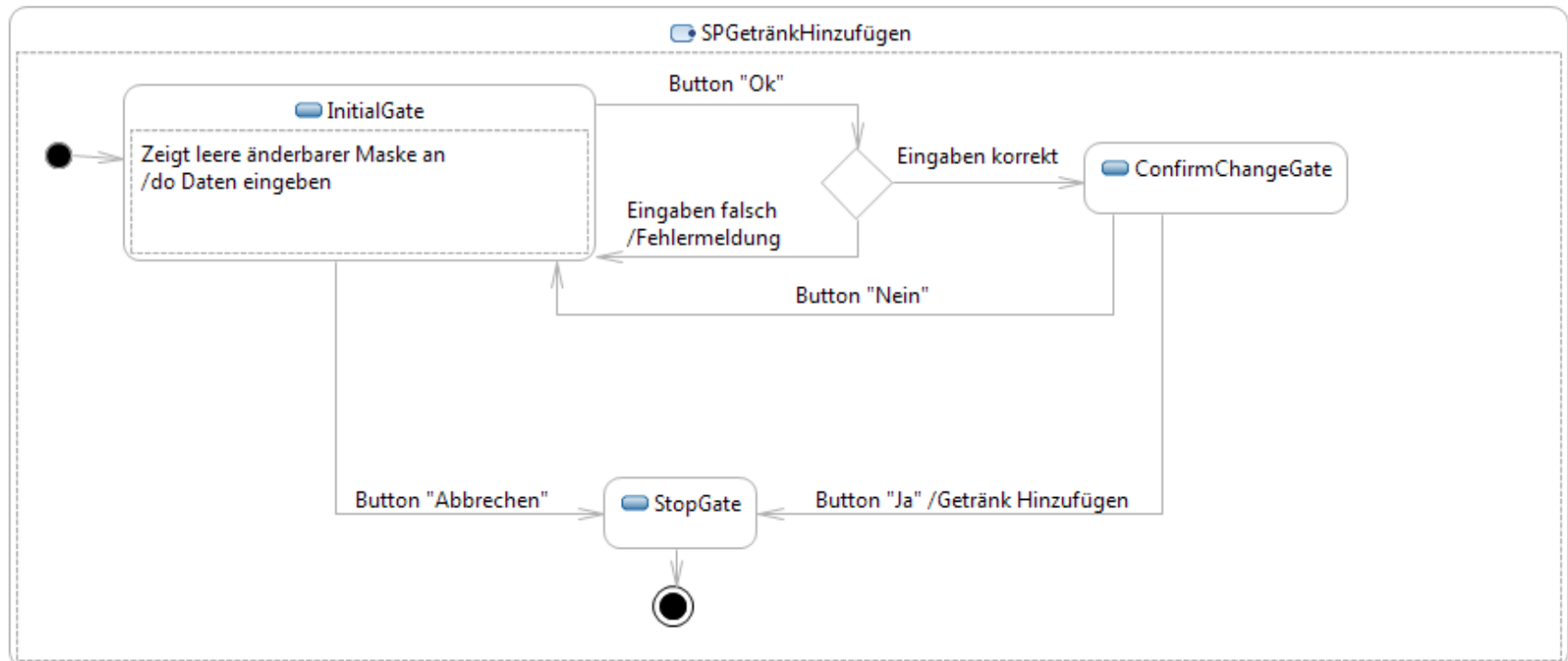
The *Kasse* (engl. cash box) class from the analysis diagram now in the design diagram has become an attribute of class *Getraenkekeller*.

The class hierarchy representing people in different roles (*Bewohner* - resident, *Getraenkewart* - manager) has been reorganized and is now given in much more detail.

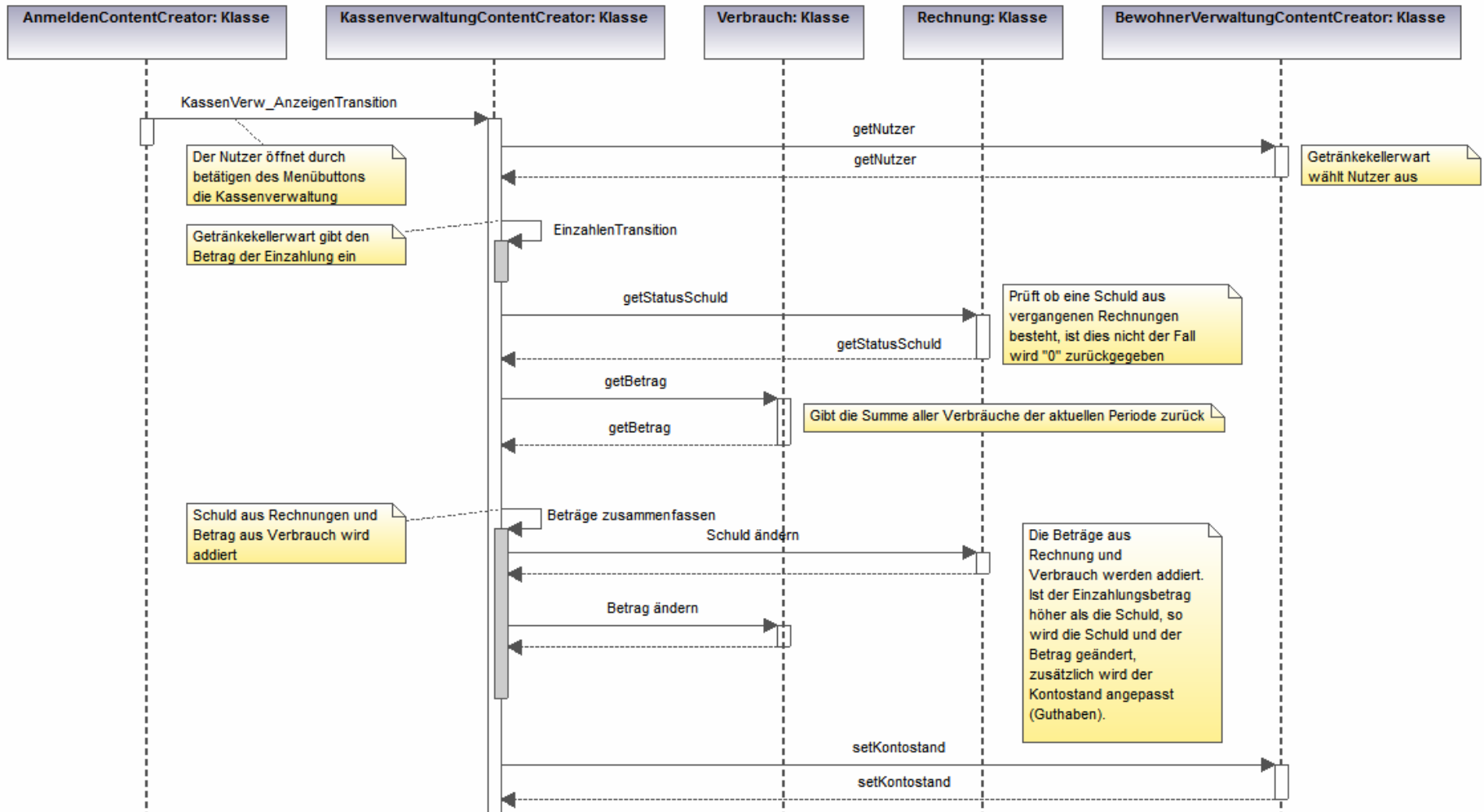
The classes *Getraenkekeller* and *Nutzer* are derived from the standard Salespoint classes *Stock* and *User* and inherit all their functionality.

The **dynamic structure** of a Salespoint application is described by its **processes**, which in turn are defined by special **UML state diagrams**. In Salespoint terminology, the states are called **Gates**. State diagrams show all the Gates and all the **Transitions** between them.

Below, we have a typical Salespoint process diagram defining how beverages are added to the system: Processing starts at the InitialGate, where data for new beverages are input. Once the button OK is pressed, either a transition to ConfirmChangeGate occurs (if data are valid) or (otherwise) with an error message back to the InitialGate. And so on ...



More complicated processes are additionally clarified by **UML sequence diagrams** like the one below which defines what happens in detail, when a user deposits some money.



Still in the design phase, team WINF02 has established a **test handbook** in order to ensure the quality of the resulting program.

Below, a few entries from the middle of the book are shown. Each entry, first, gives a short description of the test, second, names the classes that are affected, third, states the desirable outcome and, fourth, indicates whether the test is to be carried out manually (M) or automatically (A).

*/T-420/ Lagerstatus anzeigen*

<i>/T-421/ Prüfen ob Lagerstatus richtig angezeigt wird</i>	*ContentCreator (Getränkewart)	Lagerstatus wird richtig angezeigt	M
---	--------------------------------	------------------------------------	---

*/T-430/ Kassenverwaltung anzeigen*

<i>/T-431/ Prüfen ob Kassenverwaltungs Fenster richtig angezeigt wird</i>	KassenverwaltungContentCreator KassenVerw_AnzeigenTransition	Kassenverwaltungs Fenster wird angezeigt	M
---	---	--	---

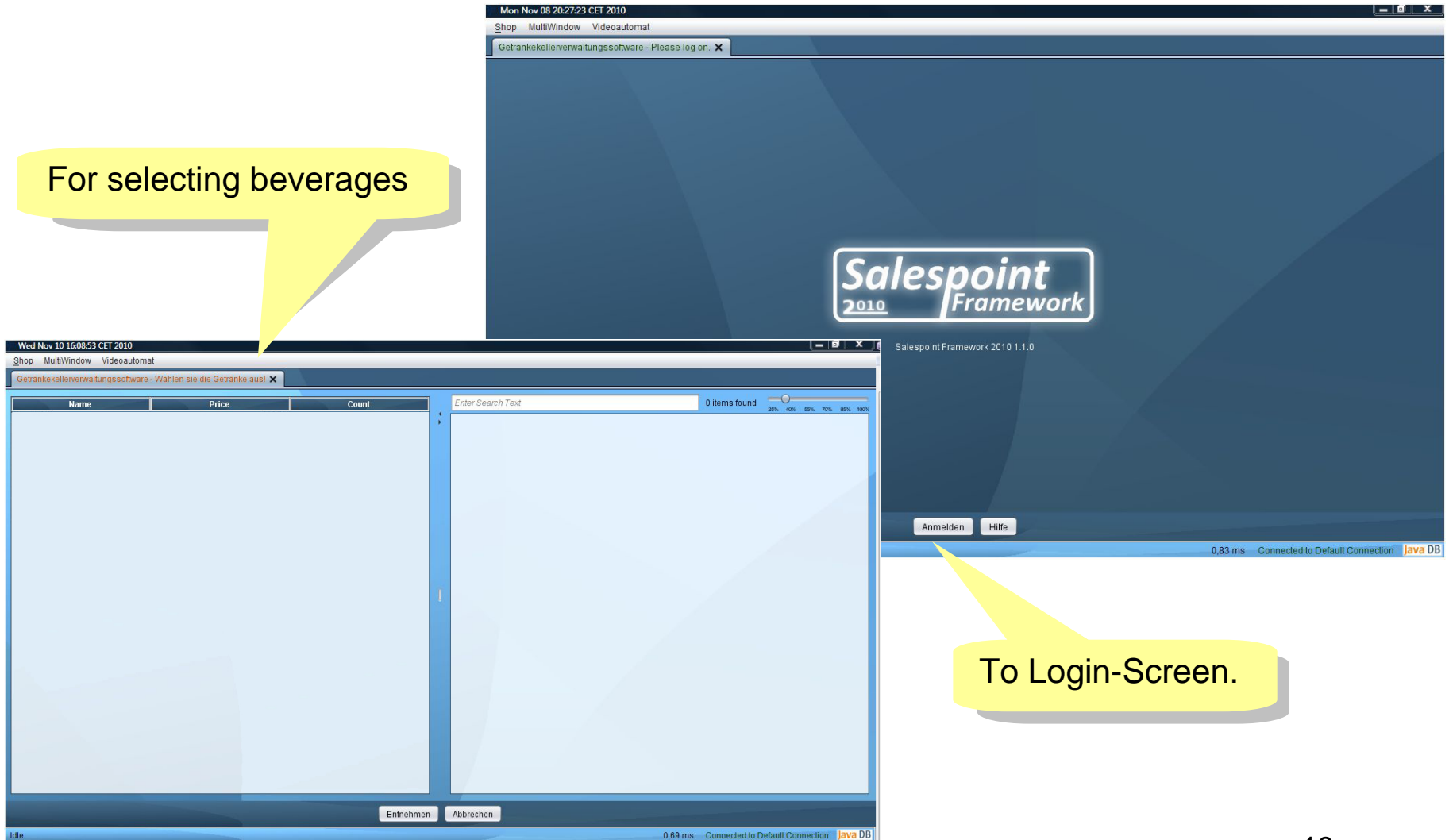
*/T-440/ Einzahlung Bewohner*

<i>/T-441/ Einzahlung Tätigen</i>	KassenverwaltungContentCreator EinzahlenTransition	Schulden des Bewohners werden getilgt	A
-----------------------------------	---	---------------------------------------	---

<i>/T-442/ Falsche Einzahlung Tätigen (Negativer Betrag)</i>	KassenverwaltungContentCreator EinzahlenTransition	Fehlermeldung, Fenster wird weiterhin angezeigt	A
--	---	---	---

At the end of the design phase, some **GUI mockups** like the ones below are prepared and discussed with the customer (tutor).

For selecting beverages



As described on the project homepage, in the following **implementation phase** the students are expected to apply an *iterative strategy* (code-evaluate-redesign).

Another demand is to *keep* the test handbook and other *artefacts* from previous phases *up-to-date* (and also visible on their homepage) all the time.

Additionally, at every intermediate milestone both a *functional program prototype* (i.e., an executable jar file) and its complete jadadoc have to be presented.

Finally, by the end of this phase the *user handbook* must be completed.

## Die Implementierungsphase

### Ziel:

Das Ziel der Implementierungsphase ist die **iterative Umsetzung** des Designmodells in ein funktionierendes Programm.

### Vorgehensweise:

Entsprechend der zuvor festgelegten und priorisierten Liste von **Use-Cases** (vgl. Implementierungsstatus) wird nun schrittweise das anskizzierte Programm implementiert. Hierbei werden in dieser Phase drei gleich geartete Iterationen durchlaufen, wobei der Priorisierung folgend implementiert wird. Die Iterationen ihrerseits folgen dabei dem **Zyklus Implementierung, Evaluation** und **Re-Design**, wobei die Evaluation sowohl das Testen auf technischer Ebene (funktioniert das Programm?), als auch die Nutzertests (entspricht das Programm den Erwartungen?) umfasst. Das Re-Design bezieht sich auf Modellanteile, deren Design sich im Laufe der Implementierung noch anpasst - dies lässt sich auch mit einem intensiven Vorabdesign nicht vollständig vermeiden und muss an dieser Stelle nun verbessert bzw. vervollständigt werden.

### Phasenartefakte:

Zwischenpräsentationen:

- das übersetzte (und funktionstüchtige!) **Programm als ausführbares JAR**
- eine **vollständige JavaDoc-Dokumentation** mit Erläuterungen zur Implementierung (mit Beschreibung von allen Klassen, Attributen, Parametern, Methoden, Rückgabewerten und Ausnahmen) **in Form eines ZIP-Archivs** (die direkte Ablage im BSCW ist i.A. zu ineffizient!)
- ein aktualisiertes **Testhandbuch** (welches nun auch die Funktionstests der einzelnen Klassen umfasst) sowie Testprotokolle und ggfs. Testklassen der bereits durchgeführten Tests
- die **aktualisierten Modellanteile** der vorherigen Phase - die Darstellung und Erläuterung der Änderungen genügt hier

Zusätzlich Endpräsentation: Das fertige **Benutzerhandbuch** zum Programm

### Hinweise:

Unabhängig von der verwendeten Sprache (deutsch oder englisch) für den Programmcode sind die gängigen Konventionen einzuhalten:

- Klassennamen beginnen mit einem Großbuchstaben, Paketnamen mit einem Kleinbuchstaben
- "Getter" und "Setter" werden nicht übersetzt, also z.B. nicht "holeName" statt "getName"

Ferner ist darauf zu achten, dass das eigene Programm in sinnvolle Pakete eingeteilt wird (das default-Package darf nicht verwendet werden). Insbesondere die Testklassen sollten sich in einem eigenen Paket befinden.

This is the **WINF02 team homepage**, where the implementation phase status shows that by now most of the use cases have been implemented successfully (hence, marked **green**):

**Programmierprojekt HT10**

der Bundeswehr  
**Universität München**

## Getränkellerverwaltungssoftware - GVS

**ANALYSE**

**EINARBEITUNG**

**DESIGN**

**IMPLEMENTIERUNG**

**Projektfortschritt**  
[ChangeLog](#)  
[Präsentationsprotokolle](#)  
[Implementierungsstatus](#)

**Phasenartefakte**  
[Anpassungen](#)  
[Paket-/ Klassendiagramme](#)  
[Sequenzdiagramme](#)  
[Programm](#)  
[JavaDoc](#)  
[Testhandbuch](#)  
[Benutzerhandbuch](#)

**Status**

100% MUSS

70% KANN

**Administration**

- /UC-101/ Nutzer anmelden
- /UC-102/ Nutzer abmelden
- /UC-103/ Nutzerpasswort ändern
- /UC-104/ Letzte Anmeldung anzeigen

**Bewohner**

- /UC-201/ Eigene Stube anzeigen
- /UC-202/ Eigener Kontostand anzeigen
- /UC-203/ periodische Entnahmen anzeigen
- /UC-210/ Getränkeentnahme anzeigen
- /UC-211/ Getränk entnehmen
- /UC-212/ Letzte Getränkeentnahme stornieren
- /UC-220/ Verbrauchsverwaltung anzeigen
- /UC-221/ Verbrauch anzeigen
- /UC-230/ Rechnungsverwaltung anzeigen
- /UC-231/ Gestellte Rechnungen zeigen

**Team WINF02's javadoc** shows the packages their GKVS (GetränkKellerVerwaltungsSoftware, engl.: GetränkeKeller management software) consists of: model, GUI, test classes etc.

[All Classes](#)

Packages

- [gkvs.anzeige](#)
- [gkvs.anzeige.oberflaechen](#)
- [gkvs.anzeige.prozesse](#)
- [gkvs.kern](#)
- [gkvs.kern.aktionen](#)
- [gkvs.kern.daten](#)
- [gkvs.kern.tabellen](#)
- [gkvs.kern.transitionen](#)
- [gkvs.Tests](#)

---

[gkvs.anzeige.prozesse](#)

Classes

- [AnmeldungsProzess](#)
- [BackupProzess](#)
- [HilfeProzess](#)
- [SPBestandsVerwaltung](#)
- [SPBewohnerVerwaltung](#)
- [SPGetraenkeEntnahme](#)
- [SPGetraenkeVerwaltung](#)
- [SPGetraenkHinzufuegen](#)
- [SPKassenVerwaltung](#)
- [SPMeinKontoBewohner](#)
- [SPMeinKontoWart](#)
- [SPRechnungen](#)
- [SPRechnungsVerwaltung](#)
- [SPVerbraeuche](#)

**Overview** [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV](#) [NEXT](#)      [FRAMES](#) [NO FRAMES](#)

## Getraenkekellerverwaltungssoftware

**Packages**

<a href="#">gkvs.anzeige</a>	Paket GKVS.Anzeige Dieses Paket enthält alle Pakete der Anzeige.
<a href="#">gkvs.anzeige.oberflaechen</a>	Paket GKVS.Anzeige.Oberflaeche Dieses Paket enthält alle Klassen der Oberfläche um diese zu generieren.
<a href="#">gkvs.anzeige.prozesse</a>	Paket GKVS.Anzeige.Prozesse Dieses Paket enthält alle Prozessklassen.
<a href="#">gkvs.kern</a>	Paket GKVS.Kern Dieses Paket enthält alle Kernklassen, sowie Klassen, welche keinem anderen Paket zuzuordnen waren.
<a href="#">gkvs.kern.aktionen</a>	Paket GKVS.Kern.Aktionen Dieses Paket enthält alle Aktionen.
<a href="#">gkvs.kern.daten</a>	Paket GKVS.Kern.Daten Dieses Paket enthält alle Datenklassen des Programms.
<a href="#">gkvs.kern.tabellen</a>	Paket GKVS.Kern.Tabellen Dieses Paket enthält alle Tabellen die für das Programm wichtig sind.
<a href="#">gkvs.kern.transitionen</a>	Paket GKVS.Kern.Transitionen Dieses Paket enthält alle Transitionen.
<a href="#">gkvs.Tests</a>	Paket GKVS.Tests Dieses Paket enthält alle Testklassen.

**Overview** [Package](#) [Class](#) [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV](#) [NEXT](#)      [FRAMES](#) [NO FRAMES](#)

Class *GetraenkeKatalog* inherits from the standard Salespoint class *CalalogImpl*. Its javadoc description shows how many interfaces have to be considered in this adaptation.

[All Classes](#)

Packages

[gkvs.anzeige](#)  
[gkvs.anzeige.oberflaechen](#)  
[gkvs.anzeige.prozesse](#)  
[gkvs.kern](#)  
[gkvs.kern.aktionen](#)  
[gkvs.kern.daten](#)  
[gkvs.kern.tabellen](#)  
[gkvs.kern.transitionen](#)  
[gkvs.Tests](#)

[gkvs.kern.daten](#)

Classes

[BenutzerBestand](#)  
[Bewohner](#)  
[EntnommenesGetraenk](#)  
[Gast](#)  
[Getraenk](#)  
[GetraenkeBestand](#)  
[GetraenkeKatalog](#)  
[Getraenkekellerwart](#)  
[Nutzer](#)  
[Verbrauch](#)

[Overview](#) [Package](#) **Class** [Use](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

`gkvs.kern.daten`

## Class *GetraenkeKatalog*

`java.lang.Object`

└ `org.salespointframework.core.data.AbstractNameable`

└ `org.salespointframework.core.data.CatalogItemImpl`

└ `org.salespointframework.core.data.CatalogItemImpl<org.salespointframework.core.data.CatalogItemImpl>`

└ `gkvs.kern.daten.GetraenkeKatalog`

### All Implemented Interfaces:

`java.lang.Cloneable`, `java.lang.Comparable<java.lang.Object>`,

`java.lang.Iterable<org.salespointframework.core.data.CatalogItemImpl>`,

`org.salespointframework.core.data.database.Recoverable`, `org.salespointframework.core.data.events.DataSourceChangeListener`,

`org.salespointframework.core.data.events.ExternalModificationListener`,

`org.salespointframework.core.data.interfaces.Catalog<org.salespointframework.core.data.CatalogItemImpl>`,

`org.salespointframework.core.data.interfaces.CatalogItem`,

`org.salespointframework.core.data.interfaces.DataBasketEntryDestination`,

`org.salespointframework.core.data.interfaces.DataBasketEntrySource`,

`org.salespointframework.core.data.interfaces.DataBasketKeys`,

`org.salespointframework.core.data.interfaces.ListenableCatalog<org.salespointframework.core.data.CatalogItemImpl>`,

`org.salespointframework.core.data.interfaces.Nameable`, `org.salespointframework.core.data.interfaces.NameContext`,

`org.salespointframework.core.data.interfaces.SpAggregate`, `org.salespointframework.core.data.interfaces.SpItem`,

`org.salespointframework.core.data.SelfManagingDBEDestination<org.salespointframework.core.data.CatalogItemImpl>`,

`org.salespointframework.core.data.SelfManagingDBESource<org.salespointframework.core.data.CatalogItemImpl>`

Meanwhile, quite a few tests have been completed successfully, marked green in the **test handbook**, while those that have not been done yet are marked yellow. Unsuccessful test would be marked red.

Some of the test are carried out manually (M), others automatically (A) using JUnit tests.

Programmierprojekt HT10 – WIN02

Getränkkelerverwaltungssoftware

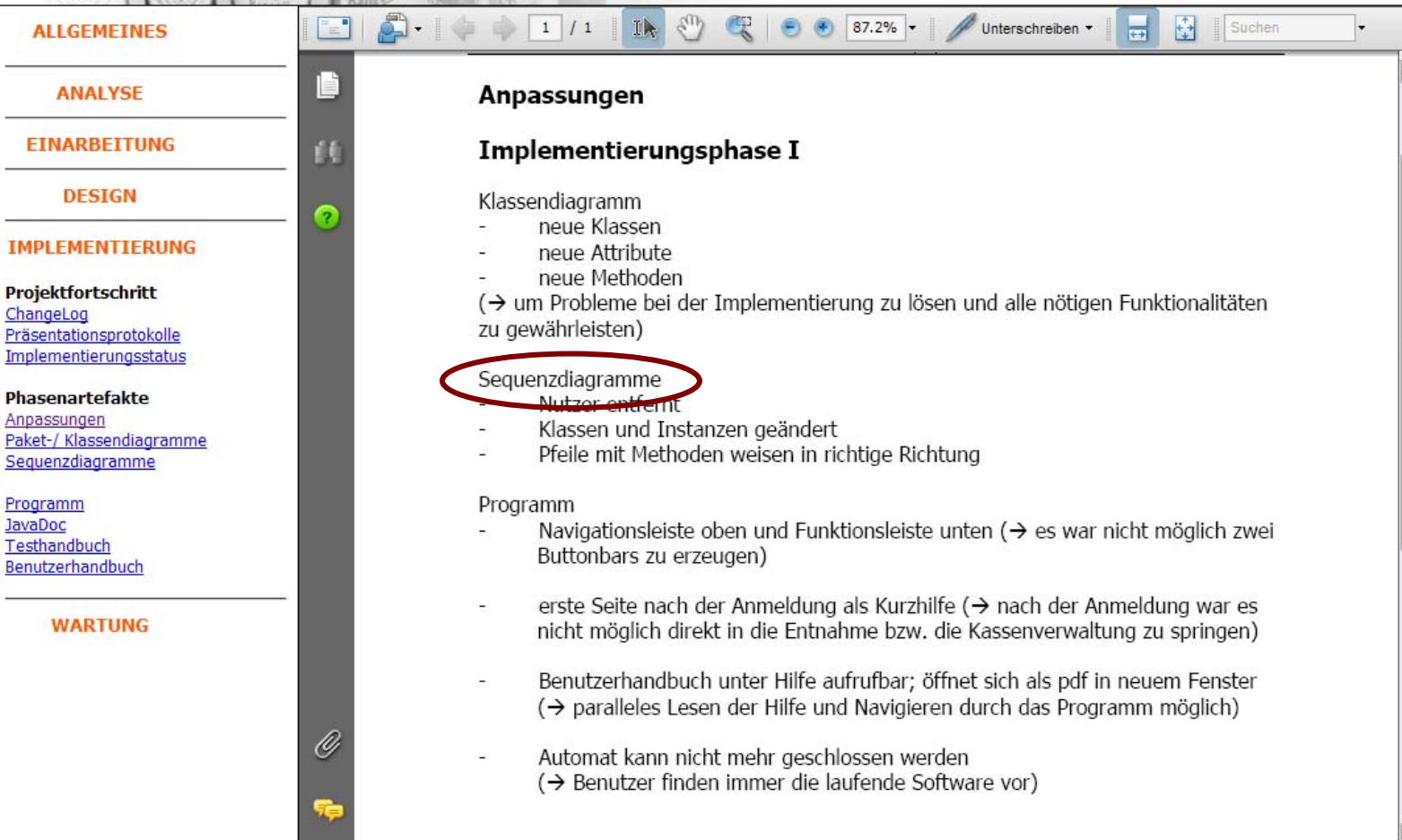


Tätigen (Negativer Betrag)			
/T-443/ Prüfen ob beim Bezahlen der Rechnung, Statusfarbe ändert	RechnungOberflaeche	Statusfarbe ändert sich	M
/T-444/ Prüfen ob Kassenstand sich ändert	AnmeldenProzess MeinKontoWartAnzeigenOberflaeche	Kassenstand ändert sich um den Betrag	A
/T-445/ Prüfen ob Bewohner entsperrt wird	AnmeldungsProzess AnmeldenTransition	Bewohner kann sich wieder anmelden	M
/T-450/ Bestandsverwaltung anzeigen			
/T-451/ Prüfen ab Bestandsverwaltungsfenster richtig angezeigt wird	BestandsverwaltungContetnCreator BestandsVerwAnzeigenTransition	Bestandsverwaltungsfenster wird angezeigt	M
/T-460/ Bestand anpassen mit Inventur			
/T-461/ Bestand anpassen	BestandsverwaltungContetnCreator BestandAnpassenTransition	Anzahl der Getränke wird verändert	A
/T-470/ Rechnungsverwaltung anzeigen			
/T-471/ Gesamtverbrauch anzeigen lassen	RechnungsVerwAnzeigenTransition RechnungsverwaltungOberflaeche	Gesamtverbrauch wird angezeigt	M
/T-472/ Gesamtverbrauch anzeigen lassen wenn noch	RechnungsVerwAnzeigenTransition RechnungsverwaltungOberflaeche	Gesamtverbrauch bleibt leer	M

The *Anpassungen* page lists all the required modifications in order to bring the artefacts from previous phases up-to-date. This among other things includes the sequence diagrams.

## Programmierprojekt HT10

### Getränkellerverwaltungssoftware - GVS



The screenshot shows a web application interface with a sidebar on the left and a main content area. The sidebar contains navigation links for 'ALLGEMEINES', 'ANALYSE', 'EINARBEITUNG', 'DESIGN', 'IMPLEMENTIERUNG', 'Projektfortschritt', 'Phasenartefakte', and 'WARTUNG'. The main content area is titled 'Anpassungen' and 'Implementierungsphase I'. It lists several changes, with 'Sequenzdiagramme' circled in red. The changes include:

- Klassendiagramm
  - neue Klassen
  - neue Attribute
  - neue Methoden(→ um Probleme bei der Implementierung zu lösen und alle nötigen Funktionalitäten zu gewährleisten)
- Sequenzdiagramme**
  - Nutzer entfernt
  - Klassen und Instanzen geändert
  - Pfeile mit Methoden weisen in richtige Richtung
- Programm
  - Navigationsleiste oben und Funktionsleiste unten (→ es war nicht möglich zwei Buttonbars zu erzeugen)
  - erste Seite nach der Anmeldung als Kurzhilfe (→ nach der Anmeldung war es nicht möglich direkt in die Entnahme bzw. die Kassenverwaltung zu springen)
  - Benutzerhandbuch unter Hilfe aufrufbar; öffnet sich als pdf in neuem Fenster (→ paralleles Lesen der Hilfe und Navigieren durch das Programm möglich)
  - Automat kann nicht mehr geschlossen werden (→ Benutzer finden immer die laufende Software vor)

```

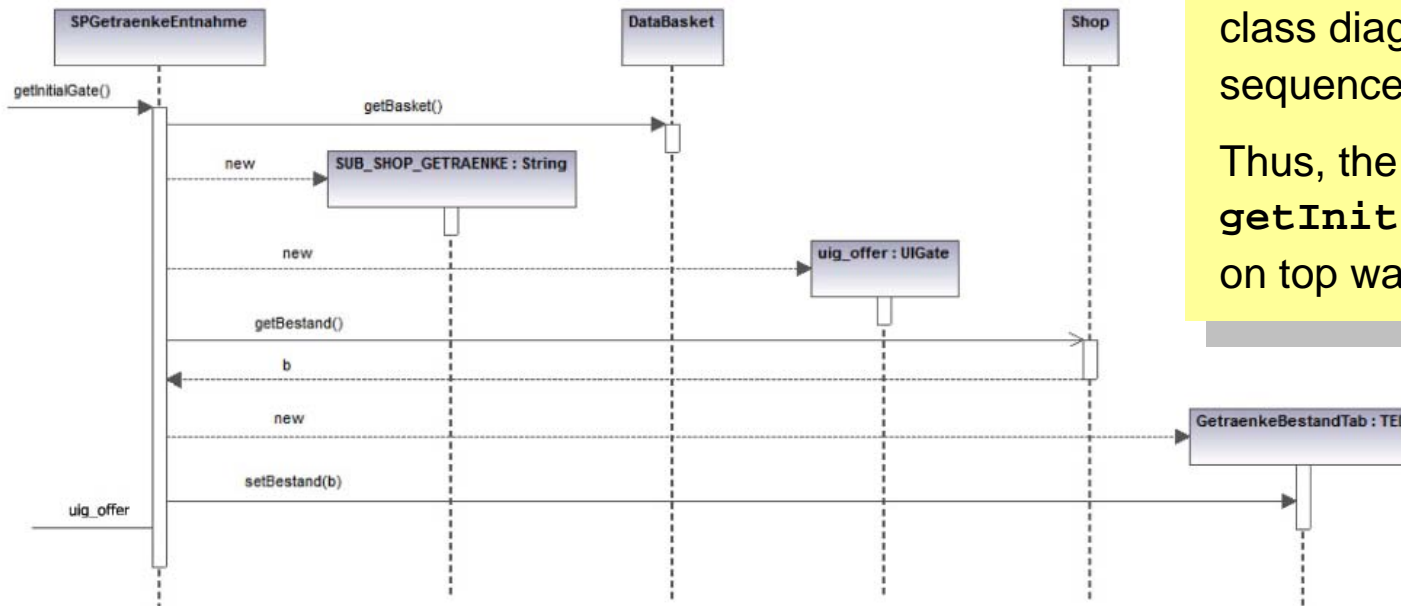
public SPGetraenkeEntnahme() {
    super("SPGetraenkeEntnahme");
}
//Abstrakt @ Sequenzdiagramm
@Override
protected Gate getInitialGate() {
    getBasket().setCurrentSubBasket(SUB_SHOP_GETRAENK);

    CSDBStrategy csdbs = new CSDBStrategy();
    csdbs.setErrorHandler(FormSheetStrategy.MSG_POPUP_ERROR_HANDLER);
    UIGate uig_offer = new UIGate(null, null);

    DataExchangeOptions options = DataExchangeFormSheet.OPTION_TABLE_LISTVIEW.copy();
    ((TableDataExchangeOption) options.getSrcOption()).setTed(new GetraenkeBestandTabelle());
    ((ListViewDataExchangeOption) options.getDestOption()).setCellConstraints(.25f, 1f, .35f, 1f);
    DataExchangeFormSheet ttfs_rent = DataExchangeFormSheet.create(
        "Wählen sie die Getränke aus!", GetraenkekellerShop.getGetraenkeBestand(), getBasket(),
        uig_offer, csdbs, options);
    ttfs_rent.addContentCreator(new GetraenkeEntnahmeOberflaeche());
    uig_offer.setMenuSheet(MenuLeiste.getMenuBewohner());
    return uig_offer;
}

public Gate getEntnahmeGate() {
    return m_gCurGate;
}

```



Since in the implementation phase the source code of classes and methods is available, we can use the *round-trip-engineering* features of modern IDEs and generate the diagrams directly from the code.

That way, the diagrams and their corresponding source code are guaranteed to be in sync! This works well for class diagrams, but not for sequence charts.

Thus, the diagram for method `getInitialGate()` shown on top was produced by hand.

## 1. Inhaltsverzeichnis

2. Systemvoraussetzungen .....	3
3. Einführung.....	4
4. Funktionen .....	5
4.1 Anmelden.....	5
4.2 Bewohner.....	6
4.2.1 Getränkeentnahme.....	6
4.2.1.1 Getränk entnehmen.....	7
4.2.1.2 Entnahme stornieren.....	7
4.2.2 Verbrauch.....	7
4.1.1 Rechnung.....	7
4.1.2 Mein Konto.....	8
4.1.2.1 Kennwort ändern.....	8
4.2 Getränkekellerwart.....	9
4.2.1 Kassenverwaltung.....	9
4.2.1.1 Einzahlung durch Bewohner.....	9
4.2.1.2 Lieferung anpassen und buchen.....	9
4.2.2 Bestandsverwaltung.....	10
4.2.2.1 Bestand nach Inventur anpassen.....	10
4.2.3 Rechnungsverwaltung.....	11
4.2.3.1 Gesamtkosten anzeigen.....	11
4.2.3.2 Gesamtverbrauch anzeigen.....	11
4.2.3.3 Rechnungen versenden.....	11
4.2.4 Getränkeverwaltung.....	12

This is the top part of the **table of contents** from the **GKVS user handbook**.

It describes what users and the *Getränkellerwart* can with GKVS.

The more interesting part is the administrative functionality offered to the *Getränkellerwart*: taking inventory, some accounting, and also managing the stocks.

We take a look at how the users get their drinks ...

The window below shows (in the left hand side) the beverages offered in the *Getränkeller*, giving the name, the price per bottle, and the number of bottles currently in stock.

The right hand side is the user's data basket. Bottles are moved via direct manipulation. A purchase is concluded (or cancelled) by pressing the buttons in the lower part of the window.


#### 4.2.1.1 Getränk entnehmen

Um ein Getränk zu entnehmen klickt man links auf ein Getränk, hält die Maustaste gedrückt und zieht das Getränk rechts in den Warenkorb. Bei mehreren Getränken wiederholt man diesen Vorgang. Durch [Entnehmen] bestätigt man die gesamte Entnahme der im Warenkorb befindlichen Getränke.

Name	Preis(Euro)	Anzahl
Bier	1.20	60
Cola	1.00	49
Wasser	0.90	40

Enter Search Text 1 item found 25% 40% 55% 70% 85% 100%

General (1)

 1

Cola 1,00 €

Entnehmen Abbrechen

Idle 0,45 ms Connected to Default Connection Java DB

Was that it?

Have we seen everything team WINF02 did?

No: what we have seen is only the proverbial tip of the iceberg!

It is neither possible nor sensible to show you all the results produced by seven people in about three months. Still, the material you have seen gives you some impression of what our students typically do in their programming project courses.

Actually, after implementation there is one more phase to be overcome by the teams. In the **maintenance phase**, the teams not only have to eliminate any bugs found so far; they also have to carry out some small modifications and/or extensions the customer wishes. This will be a minor effort for the team if their design was sound and easy to maintain. Otherwise, they will suffer from their own sloppy work.

For most of our students, this is the first time they tackle a substantial programming assignment and experience the highs and lows of team work. Most of them like it - at least in retrospect!