|  |
| --- |
| Seminar aus Software-Entwicklung |
| SCRUM2 |
| Abschlussbericht |
|  |
| **Alexander Duml**  **Stefan Müller**  **Thomas Perl**  **Martin Wieser** |

|  |
| --- |
|  |

**Inhaltsverzeichnis**

Prozessbeschreibung 2

Idealtypischer Ablauf 2

Rollen 2

Meetings 3

Artefakte 4

Technologien 4

Anpassungen 5

Beschränkung auf wesentliche Rollen 5

Doppelrollen für Product Owner und Scrum Master 5

Zusammenlegen der Meetings 5

Beschränkung auf wesentliche Artefakte 6

Projektverlauf 6

Retrospektive 11

Prozessbewertung 11

Anpassung 1 11

Anpassung 2 11

Anpassung n 11

Fazit 11

Teilnehmer 1 12

Teilnehmer 2 12

Teilnehmer n 12

Softwareprodukt 12

Einleitung 12

Use Cases 13

Architektur 19

Technologien 20

Fazit 20

Tagebuch Teilnehmer 1 20

Tagebuch Teilnehmer 2 20

Tagebuch Teilnehmer n 20

# Prozessbeschreibung

Scrum ist ein schlanker, agiler Managementprozess, welcher Anfang der 1990er Jahre von Ken Schwaber und Jeff Sutherland für die Umsetzung komplexer Produktentwicklungen konzipiert wurde und heute insbesondere in der Software-Entwicklung Anwendung findet. Kennzeichnend für Scrum ist unter anderem das inkrementelle, interaktive und empirische Vorgehen.

## Idealtypischer Ablauf



Abbildung: Scrum Ablaufschema[[1]](#footnote-1)

Das Ziel von Scrum ist es eine Produkt-Vision schnell, kostengünstig und qualitativ hochwertig umzusetzen. Dafür werden zunächst in einer strategischen Planungsphase, die aus der Vison abgeleiteten Produktfunktionalitäten, im sogenannten Product Backlog festgehalten und entsprechend ihres Geschäftswerts priorisiert.

Anschließend beginnt die eigentliche Umsetzung, wobei diese in einer Serie von Sprints voranschreitet. Ein Sprint ist dabei als eine immer gleichlange, typischerweise zwei bis vier Wochen dauernde Zeitspanne zu verstehen. Zu Beginn jedes Sprints steht das Sprint Planning Meeting. In diesem Meeting wählt das Entwicklungs-Team die am höchsten priorisierten Items aus dem Product Backlog aus und definiert, wie diese während des Sprints umgesetzt werden sollen. Das resultierende Artefakt wird als Sprint Backlog bezeichnet.

Die im Sprint Backlog definierten Items werden während des Sprints entworfend, programmiert und getestet, sodass am Ende des Sprint ein auslieferbares Stück Software steht. Idealerweise trifft sich das Entwicklungs-Team während eines Sprints an jedem Arbeitstag zum sogenannten Daily Scrum, einem ca. 15-minütigen Meeting, welches u.a. der Planung des nächsten Tages dient. Bis zum Projektabschluss werden mehrere Sprints durchgeführt, sodass das Produkt kontinuierlich weiterentwickelt wird.

Das Scrum-Framework wird durch wenige, dafür aber sehr effektive Rollen, Regeln, Meetings und Artefakte determiniert. Im Folgenden werden diese kurz erörtet.

## Rollen

Scrum sieht die folgenden Rollen vor:

* **Der Product Owner:** Zu den Aufgaben des Product Owners gehört es die Bedürfnisse der Kunden und Stakeholder zu erfassen und diese im Product Backlog, welches er verwaltet, abzubilden. Er trägt die Verantwortung für das Produkt und den Projekterfolgt, legt die Prioritäten fest, hat sicherzustellern, dass das Entwicklungsteam die einzelnen Product Backlog-Einträge versteht und akzeptiert oder weist Arbeitsergebnisse zurück. Scrum sieht dabei ausdrücklich vor, dass der Product Owner eine Person und kein Ausschuss oder Gremium ist. Außerdem ist der Product Owner kein Mitglied des Entwicklungs-Teams.
* **Der Scrum Master:** In der Verantwortung des Scrum Master liegt es, dass Scrum verstanden und richtig umgesetzt wird. Dazu führt er gegebenenfalls Coaching der Teammitglieder durch oder versucht den Lernprozess des Teams anzustoßen. Er moderiert die diversen Meetings und bereitet diese vor. Zudem schützt der Scrum Master das Team vor äußeren Störungen und beseitigt etwaige Hindernisse. Der Scrum Master ist kein Teil des Entwicklungsteams. Er arbeitet weder inhaltlich, noch hat er eine Weisungsbefugnis gegenüber dem Team.
* **Das Entwicklungs-Team:** Das Team besteht üblicherweise aus fünf bis neun Mitglieder. Es ist interdisziplinär besetzt und verfügt über das gesamte Wissen und Können um das Produkt zu realisieren. Die Teammitglieder verpflichten sich das selbst definierte Sprint-Ziel zu erreichen. Dabei arbeiten sie selbstorganisiertes, was bedeutet, dass niemand vorschreibt wie die Einträge des Product Backlogs konkret umzusetzten sind. Auch werden keine Hierarchien von außen diktiert.

Zusätzlich sieht Scrum weitere Rollen, die zwar nicht Teil des eigentlichen Prozesses sind aber dennoch auf diesen einwirken, vor. Darunter fallen beispielsweise die Rollen User, Customer oder das höhere Management.

## Meetings

Scrum sieht die folgenden Meetings vor:

* **Sprint Planning:** Das Sprint Planning Meeting steht zu Beginn eines jeden Sprints und gliedert sich in zwei Teile:
  + Im ersten Teil erfolgt die Auswahl der zu bearbeitenden Product Backlog-Einträge. Dazu stellt der Product Owner dem Team die geordneten Einträge vor, um ein gemeinesames Verständnis zu erhalten. Die Anzahl der im Sprint umzusetzenden Items wird dann vom Team bestimmt und richtet sich danach, ob die Umsetzung in der fest vorgegebenen Sprintdauer erfolgen kann. In Folge verpflichtet sich das Team das Sprint-Ziel zu erreichen.
  + Im zweiten Teil des Meetings entscheidet das Team, wie genau die ausgewählten Einträge umgesetzt werden. Das Team definiert alle dafür notwenigen Aktivitäten und es erfolgen erste Diskussionen über Architektur, Design etc. Das resultierende Artefakt wird als Sprint Backlog bezeichnet.
* **Daily Scrum:** Dieses Meeting dauert 15 Minuten und findet täglich zur selben Zeit am selben Ort statt.In der Regel nehmen Scrum Master und Entwicklungs-Team daran teil. Ziel ist die Synchronisation der Aktivitäten, der Informationsaustausch und das Planen der nächsten 24 Stunden. Es werden Hindernisse diskutiert und durch Scrum-Master in der Impediment List erfasst.
* **Sprint Review:** Am Ende jedes Sprints steht das Sprint Review Meeting. Es dient der Präsentation der Ergebnisse durch das Team, der Überprüfung der Zielerreichung, der Abnahme der Arbeitsfortschritte durch den Product Owner und der Reflektion. Je nach Sprintlänge dauert es zwischen zwei und vier Stunden.
* **Sprint Retrospective:** Dieses Meeting findet ebenfalls am Ende des Sprints statt und dauert ca. 2-3 Stunden. Es dient dazu, die Zusammenarbeit des Teams zu verbessern, die Produktivität und Softwarequalität zu steigern und die Anwendung des Scrum-Prozesses zu optimieren. Der Scrum Master steht dem Team dabei unterstützend zur Seite und die erarbeiteten Verbesserungen werden im nächsten Sprint umgesetzt.

## Artefakte

Beschreiben Sie alle Artefakte Ihres Prozesses (Backlog, Sprint Backlog, Level of Done, ...)

Product Backlog

Sprint Backlog

Impediment List

Weitere Artefakte

Definition of Done

Produktvision

Burndown Charts

## Technologien

Beschreiben Sie alle Technologien die Sie für die Umsetzung und Unterstützung des Prozesses eingesetzt haben

Email

GitHub

Excel

Handy

## Anpassungen

### Beschränkung auf wesentliche Rollen

In unserem Projekt gestaltete sich die Rollenverteilung wie folgt:

* ProductOwner – Martin Wieser
* Scrum Master – Stefan Müller
* Entwicklungs-Team – Thomas Perl und Alexander Duml

Alle weiteren Rollen wurden von uns nicht berücksichtig. Da diese – wie oben beschrieben – nicht Teil des eigentlichen Prozesses sind, sahen wird dafür keinen Bedarf. Lediglich die Vorgaben der LVA-Leitung hinsichtlich der umzusetzenden Anwendung wurden von uns als Customer-Input behandelt.

### Doppelrollen für Product Owner und Scrum Master

Entgegen der Scrum-Vorgaben arbeiteten sowohl der Product Owner als auch der Scrum Master im Entwicklungs-Team mit. Uns ist sehr wohl bewusst, dass dies in einem realen Projekt zu Konflikten führen würde und dieses Vorgehen daher nicht akzeptabel wäre. In unserem Projekt erachteten wir dies allerdings als vernachlässigbar.

Zudem gingen wir davon aus, dass Product Owner und Scrum Master eine geringere Arbeitslast zu tragen haben werden als das Entwicklungs-Team, da gewisse Aufgaben – wie beispielsweise das Verhandeln mit Kunden oder das beseitigen äußerer Störungen – nur in einem realen Projekt durchgeführt werden müssen. Im Sinne einer faireren Verteilung der Arbeitslast entschieden wir uns daher für die beschriebene Lösung.

**Kürzere Sprintdauer als Vorgeschlagen**

Wir haben uns entgegen der Empfehlung von Scrum (2-4 Wochen) für eine Sprintdauer von lediglich einer Woche entschieden. Die Idee war, dass wir kleine, dafür aber relative viele Sprints durchführen wollten, um damit in Verbindung mit einem wöchentlichen Meeting eine bessere Kontrolle über den Projektvortschritt zu gewährleisten.

**Verzicht auf Daily Scrum**

Das Daily Scrum wurde von uns nicht durchgeführt.

Im studentischen Umfeld schwer zu Realisieren (alle haben einen Unterschiedlichen Stundenplan)

Unnötiger Overhead

Kommunikation über Email als teilweiser Ersatz

Keine Negativen Auswirkungen

### Zusammenlegen der Meetings

....

Regelmäßiges Meeting ein mal pro Woche im Umfang von ein bis zwei Stunden.

### Beschränkung auf wesentliche Artefakte

...

# Projektverlauf

**04.11.2011 – Kickoff-Meeting**

Beim Kickoff Meeting wurden die Rollen der Teammitglieder verteilt und es wurde eine Technologieauswahl getroffen:

* Repository auf GitHub
* Dokumentation in LaTeX
* Programmiert wird in Java
* GUI mit Swing realisiert
* HSQLDB als Datenbank

Weiters sind die nächsten Schritte definiert worden:

* Sich mit SCRUM vertraut machen
* LaTeX Template für die Dokumentation erstellen
* User Stories zum Product Backlog hinzufügen
* GitHub einrichten
* Projekt einrichten (Architektur definieren)

**11.11.2011 – Zweites Meeting**

Review

Das Projekt wurde erfolgreich aufgesetzt und eine Architektur wurde definiert. Außerdem wurden für die Softwareentwicklung zwei neue Technologien eingesetzt, welche vorher noch nicht spezifiziert worden sind:

* Eclipse als Entwicklungsumgebung
* Hibernate als OR-Mapper in Kombination mit JPA Annotations in den Models

Retrospektive

Da es sehr anstrengend ist Swing GUIs mit Hand zu implementieren soll das Windows Builder Eclipse Plugin genutzt werden.

Planning Meeting 1

Es sollen folgende User Stories implementiert werden:

* Reservierung vornehmen:

Als Rezeptionist will ich schnell und einfach neue Reservierungen anlegen, so dass ich mich wieder den Kunden widmen kann.

* Kunde anlegen:

Als Rezeptionist will ich neue Kunden erstellen können, so dass ich ihre Daten für die Rechnung habe.

* Zimmer anlegen:

Als Geschäftsfüherer will ich neue Zimmer hinzufügen, so dass das System nach einen Hotelausbau korrekt läuft.

* Daten löschen:

Als Geschäftsfüherer will ich alte Daten löschen können, so dass das System nach einen Hotelumbau korrekt läuft.

* Daten bearbeiten:

Als Geschäftsfüherer will ich Daten ändern können, so dass fehlerhafte Eingabe korrigiert oder Informationen ergänzt werden können.

Planning Meeting 2

Die User Stories werden folgendermaßen implementiert:

* CRUD (Create, Read, Update und Delete) Mechanismus für Kunde (Model für die Datenbankanbindung + Logik für Berechnungen + GUI für die grafische Darstellung der Daten)
* CRUD Mechanismus für Zimmer (Model + Logik + GUI)
* CRUD Mechanismus für Reservierung (Model + Logik + GUI)

**18.11.2011 - Sprint 1**

Review

Alle geplanten Features wurden implementiert, aber es wurden keine Testfälle spezifiziert und somit wurde nichts getestet. Deshalb werden die Features auf den nächsten Sprint verschoben.

Retrospektive

Da bis jetzt keine Testfälle spezifiziert waren, müssen unbedingt Unit-Tests und funktionale Tests erstellt werden. Die Unit-Tests soll die Logik-Komponente auf korrekte Funktionalität prüfen. Die GUI soll funktionalen Tests unterzogen werden. Gefundene Fehler werden in die Bug-Liste (Excel Tabelle) eingetragen.

Das Entwicklerteam findet die Implementierung der GUI von Reservierung ist unsauber, die Berechnungen sollen in eine Logik-Komponente ausgelagert werden.

Dem Product-Owner gefällt nicht, dass die Ansicht von Kunden-, Zimmer- und Reservierungsliste in einzelnen Fenstern geöffnet wird, es sollte alles in einem Fenster gehalten werden.

Planning Meeting 1

Es sollen folgende User Stories implementiert werden:

* Reservierung mit mehreren Zimmern:

Als Rezeptionist will ich Reservierungen mit mehreren Zimmern anlegen, so dass bei einer Stornierung alle Zimmer frei werden.

* Reservierung mit mehreren Kunden:

Als Rezeptionist will ich Reservierungen mit mehreren Kunden anlegen, so dass zukünftige Discounts korrekt berechnet werden.

Planning Meeting 2

Die User Stories werden folgendermaßen implementiert:

* CRUD Mechanismus für Reservierung (Model + Logik + GUI) aktualisieren, damit mit einer Reservierung mehrere Zimmer von mehreren Kunden gebucht werden können.

Die Verbesserungsvorschläge, welche aus der Retrospektive hervorgehen, werden folgendermaßen erledigt:

* Logik aus GUI in eine Logik-Komponente auslagern
* Unittests für Logik erstellen
* GUI Refactoring
* Template für Testfälle erstellen

**25.11.2011 - Sprint 2**

Review

Alle geplanten Features von Sprint 1 und 2 wurden implementiert und erfolgreich getestet.

Retrospektive

Da alle Tests positiv durchgelaufen sind gab es keine Probleme. Auch der Product-Owner war mit dem überarbeiteten GUI zufrieden.

Planning Meeting 1

Es sollen folgende User Stories implementiert werden:

* Kundendaten einsehen:

Als Geschäftsführer will ich Einblick in die Kundendaten, so dass ich mit ihnen Kontakt aufnehmen kann.

* Rechnung erstellen:

Als Rezeptionist will ich Rechnungen erstellen, so dass ich diese den Kunden vorlegen kann.

Planning Meeting 2

Die User Stories werden folgendermaßen implementiert:

* Aktualisierung der Kundenansicht
* CRUD Mechanismus für Rechnung (Model + Logik + GUI)

**02.12.2011 - Sprint 3**

Review

Alle geplanten Features wurden implementiert und erfolgreich getestet.

Retrospektive

Bug-Liste ist im Excel-Dokument schwer zum tracken. Deshalb soll die Issue-Liste von GitHub verwendet werden. Somit können die Fehler den zuständigen zugewiesen werden. Außerdem sollen Fehler und Features den jeweiligen Sprints durch GitHub-Milestones zugeordnet werden.

Der Product Backlog soll auch ins GitHub-Wiki verschoben werden.

Planning Meeting 1

Es sollen folgende User Stories implementiert werden:

* Reservierung stornieren:

Als Rezeptionist will ich Reservierungen stornieren.

* Frühzeitige Abreise erfassen:

Als Rezeptionist will ich ein frühzeitige Abreise erfassen, so dass die Zimmer wieder als "frei" erkannt werden.

* Rechnungen anzeigen:

Als Geschäftsführer möchte ich mir schnell und einfach alle ausgestellten und noch offenen Rechnungen anzeigen lassen.

Planning Meeting 2

Die User Stories werden folgendermaßen implementiert:

* Hinzufügen eines Strono-Flags im Model, Erstellen eines Strorno-Buttons in der Reservierungs-Listen GUI und Implementieren der Storno-Logik
* Überarbeiten der Rechnungs GUI, hinzufügen der Logik für die frühzeitige Abreise und hinzufügen einer Liste mit allen Rechnungen

**09.12.2011 - Sprint 4:**

Review

Alle geplanten Features wurden implementiert. Bei den funktionalen Tests wurde festgestellt, dass die ManyToMany Beziehung nicht ordnungsgemäß funktioniert. Dies hatte aber keine Auswirkung auf die derzeitige Funktionalität.

Retrospektive

Der Benutzer findet, dass die Berechnung des Preises der Reservierung automatisch erfolgen sollte.

Außerdem sollte es beim Löschen eines Datensatzes eine Rückfrage geben bevor dieser gelöscht wird.

Planning Meeting 1

Es sollen folgende User Stories implementiert werden:

* Automatische Berechnung des Preises bei Reservierung:

Als Rezeptionist will ich dass sich der Preis beim Erstellen einer Reservierung automatisch berechnet.

* Löschung mit Rückfrage:

Als Rezeptionist will ich, dass beim Löschen eines Datensatzes noch einmal nachgefragt wird ob ich den Datensatz wirklich löschen will.

Planning Meeting 2

Die User Stories werden folgendermaßen implementiert:

* Es wird eine Berechnung in die Logik-Komponente eingefügt, welche vom GUI durch ein Event aufgerufen wird.
* Der Lösch-Button der Kunden-, Zimmer- und Reservierungslisten GUI erhält eine Rückfrage, die bestätigt werden muss.

Bug-Fix von:

* Reservierungs-Model ManyToMany Annotation zu Kunde.

**16.12.2011 - Sprint 5:**

Review

Alle geplanten Features wurden implementiert. Durch die funktionalen Tests wurde erkannt, dass Kunden, Zimmer und Reservierungen erstellt werden können. Es sollte jedem Editor eine Eingabe-Validierung hinzugefügt werden.

Retrospektive

Der Product-Owner möchte, dass man die Listen mit einer Suchfunktion filtern kann und das Öffnen des Bearbeiten-Menüs sollte durch einen Doppelklick, nicht durch einen Button, erfolgen.

Planning Meeting 1

Es sollen folgende User Stories implementiert werden:

* Zimmerbelegung anzeigen:

Als Geschäftsführer will ich in die Zimmerbelegung einsehen.

* Suchfunktion in Anzeige:

Als Rezeptionist will ich über eine Suche schnell zu gewünschten Kunden, Zimmern oder Reservierungen gelangen.

* Öffnen des Bearbeiten-Menüs nach Doppelklick:

Als Rezeptionist will ich durch einen Doppelklick auf einen Datensatz, diesen bearbeiten.

Planning Meeting 2

Die User Stories werden folgendermaßen implementiert:

* Die Zimmerbelegung wird durch ein GUI mit durch eine Tabelle, in der die Zellen bei freien Tagen grün und bei belegten Tagen rot angezeigt wird, dargestellt.
* Kunden-, Zimmer- und Reservierungslisten GUI erhält eine Suchleiste, die bei einer Eingabe die Liste filtert
* Kunden-, Zimmer- und Reservierungslisten GUI wird ein Doppelklick-Event zum Bearbeiten der Einträge hinzugefügt

Bug-Fix von:

* Eingabe-Validierung für alle Editoren.

**12.01.2011 - Sprint 6**

Review und Retrospektive

Alle must-have Features wurden implementiert und erfolgreich getestet. Das Projekt wurde vom Product-Owner abgenommen, somit wurde das Projekt erfolgreich beendet.

**18.01.2012 – Endpräsentation**

Das fertig gestellte Produkt wurde präsentiert.

# Retrospektive

Beschreiben Sie Rückblickend Ihren Prozess und die Auswirkung die der Prozess auf das Projekt und den Projekterfolg hatte.

* 1. Prozessbewertung

Bewerten Sie rückblickend den von Ihnen eingesetzten Prozess

* 1. Anpassung 1

Bewerten Sie rückblickend die von Ihnen durchgeführten Anpassungen.

* 1. Anpassung 2
  2. Anpassung n
  3. Fazit

Verfassen Sie pro Teilnehmer ein Fazit über den Prozess der kurz und prägnant Ihre subjektiven Erfahrungen und Eindrücke mit dem Prozess darstellt.

* + 1. Teilnehmer 1
    2. Teilnehmer 2
    3. Teilnehmer n

# Softwareprodukt

In diesem Kapitel wird das von uns erstellte Software-Produkt beschrieben, wobei vorallem auf Use Cases, die Architektur und die verwendeten Technologien eingegangen wird.

* 1. Einleitung

Beschreiben Sie überblicksmäßig die Rahmenbedingungen der Implementierung sowie die Implementierung selbst.

Unsere Implementierung der Hotel-Reservierungs-Lösung wurde vom Projektteam unter dem Arbeitstitel „BlueHotel“ geführt. BlueHotel ist für kleine Hotels (Familienbetriebe, Urlaub am Bauernhof, Ferienhäuser) konzipiert, in bei denen es eine überschaubare Anzahl an Räumen, und keine Raumklassen gibt (d.h. statt zum Beispiel 200 freie Räume in der Klasse „Premium“ und 100 freie Räume in der Klasse „Business“ gibt es bei uns im Hotel dedizierte Räume, die eigenständige Namen haben, und nur einzeln existieren, zum Beispiel „Präsidenten-Suite“, „Garçonnière A“ und „Das Loft“). Aus diesem Grund wird auch jeder Raum eigenständig in der Datenbank erfasst.

Als Rahmenbedingung gilt zusätzlich, dass in einem Hotelzimmer jeweils 1-3 Personen untergebracht sind, wobei sich dieses beliebig aus Erwachsenen und Kindern zusammensetzen können (mit der Ausnahme, dass kein Kind alleine ein Zimmer belegen kann). Für alle Kombinationen (bis zu 6: 1E, 2E, 3E, 1E+1K, 1E+2K und 2E+1K) von Erwachsenen und Kindern kann jedem Zimmer ein eigener Nächtigungspreis zugewiesen werden.

Weiters gehen wir davon aus, dass diese Lösung auf einem Einzelplatzrechner verwendet wird, und es keine Anforderung gibt, die Daten über das Web zugänglich zu machen.

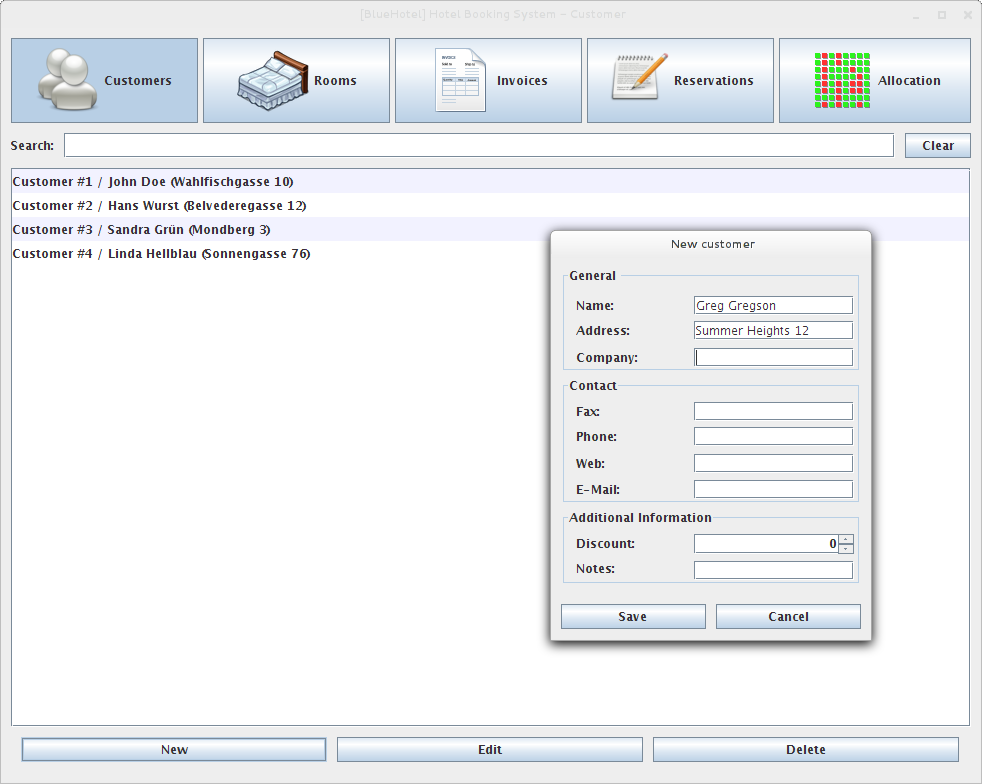
Als zusätzliche Einschränkung gehen wir davon aus, dass die Preise keiner saisonalen Schwankung unterliegen – das User Interface wurde trotzdem so konzipiert, dass man die Preise der einzelnen Zimmer leicht manuell ändern kann, eine einmalige Eingabe der Preise für jede Saison mit automatischer, zeitlicher Anpassung wurde nicht vorgenommen.

Als UI-Sprache haben wir Englisch gewählt, durch ein Folgeprojekt könnte Lokalisierung (l10n) bzw. Internationalisierung (i18n) durch Java-eigene Technologien leicht hinzugefügt werden.

* 1. Use Cases

Im Folgenden wird die Implementierung beispielhaft anhand von ausgewählten, repräsentativen Use Cases inklusive Screenshots erklärt.

**Kunde anlegen**

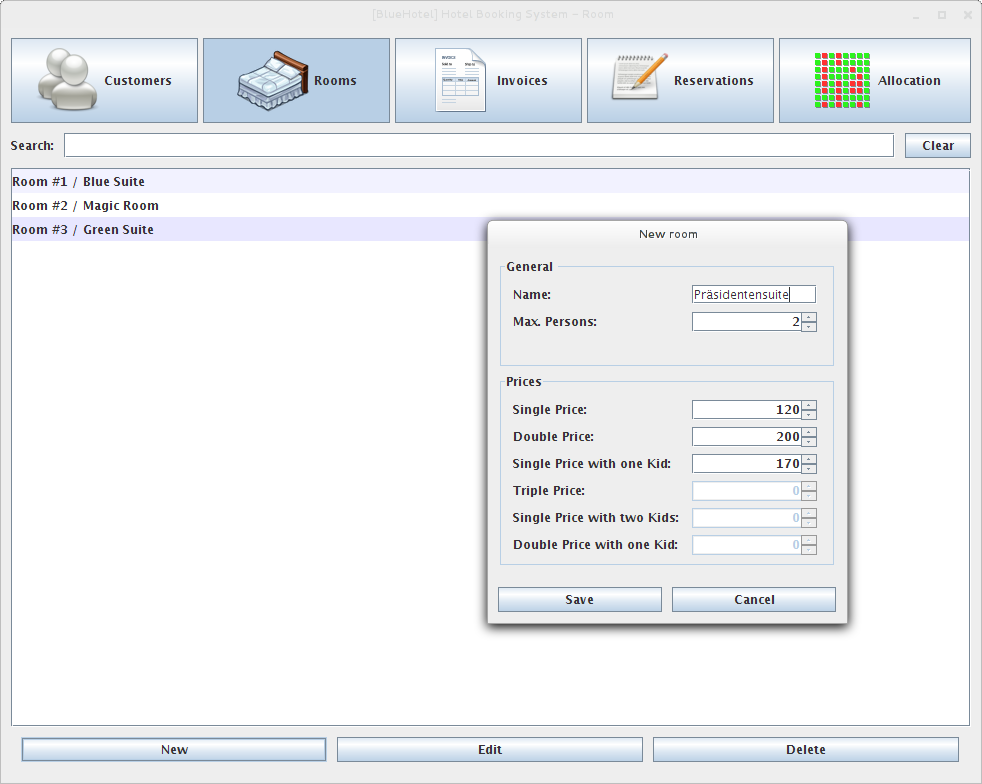
Abbildung: Kunden-Liste mit „Kunde anlegen“-Dialog

Das Anlegen von Kunden erfolgt über den Menüpunkt „Customers“. Diese Ansicht hat die selbe Struktur wie andere Listen im Programm:

* Such- und Filterleiste oben, mit „Clear“-Button
* Liste der Kunden, sortiert nach Erstellungsdatum
* C(R)UD-Buttons unten: New (=Create), Edit (=Read and Update) und Delete (=Delete)

Name und Adresse sind Pflichtfelder, die anderen Felder sind optional. Dies erlaubt ein schnelles Erfassen ankommender Kunden, die Kontaktdaten können dann optional nach der Ankunft eingetragen werden. Bei fehlerhaften Eingaben wird eine Fehlermeldung angezeigt, wenn der „Save“-Button aktiviert wird.

**Raum anlegen**

Abbildung: Zimmer-Liste mit „Raum anlegen“-Dialog

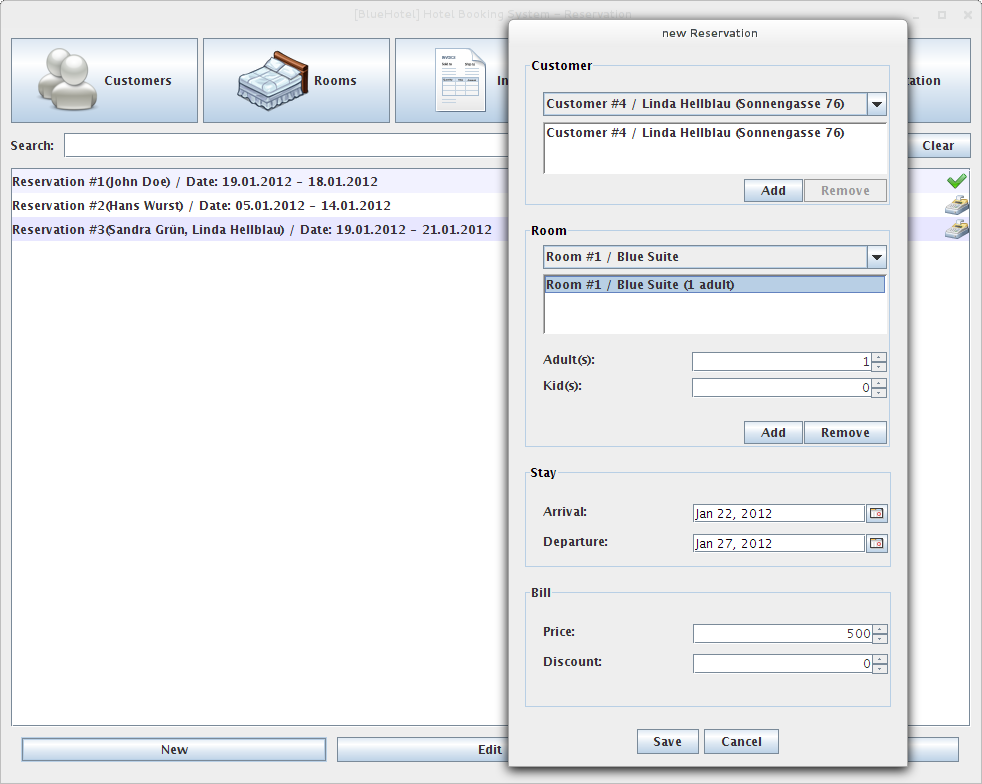
Um Zimmer anzulegen, bzw. die Preise zu editieren kann der Menüpunkt „Rooms“ verwendet werden. Hier haben wir als Pflichtfelder zum einen den Namen des Zimmers und die maximale Anzahl der Personen, die in diesem Zimmer Platz finden.

Je nachdem, welchen Wert das Feld „Max. Persons“ hat, werden im „Price“-Abschnitt des Dialogs unterschiedliche Eingabefelder freigeschalten. Alle freigeschaltenen Pflichtfelder müssen mit einem positiven Preis ausgefüllt werden, ansonsten erscheint einer Fehlermeldung.

Beispiel: Bei Auswahl von maximal 2 Personen müssen die Felder „Single Price“, „Double Price“ und „Single Price with one child“ ausgefüllt werden- die restlichen Felder (die nur bei 3 Personen relevant sind) sind deaktiviert, und können nicht ausgefüllt werden.

Auch hier erscheint wieder eine Fehlermeldung beim Speichern, wenn eine Input-Validierung fehlschlägt. Die Fehlermeldung gibt klar Auskunft darüber, welche Felder den Fehler verursacht haben, und wie der Fehler zu beheben ist.

**Reservierung erstellen**

Abbildung: Reservierungs-Liste mit „Reservierung anlegen“-Dialog

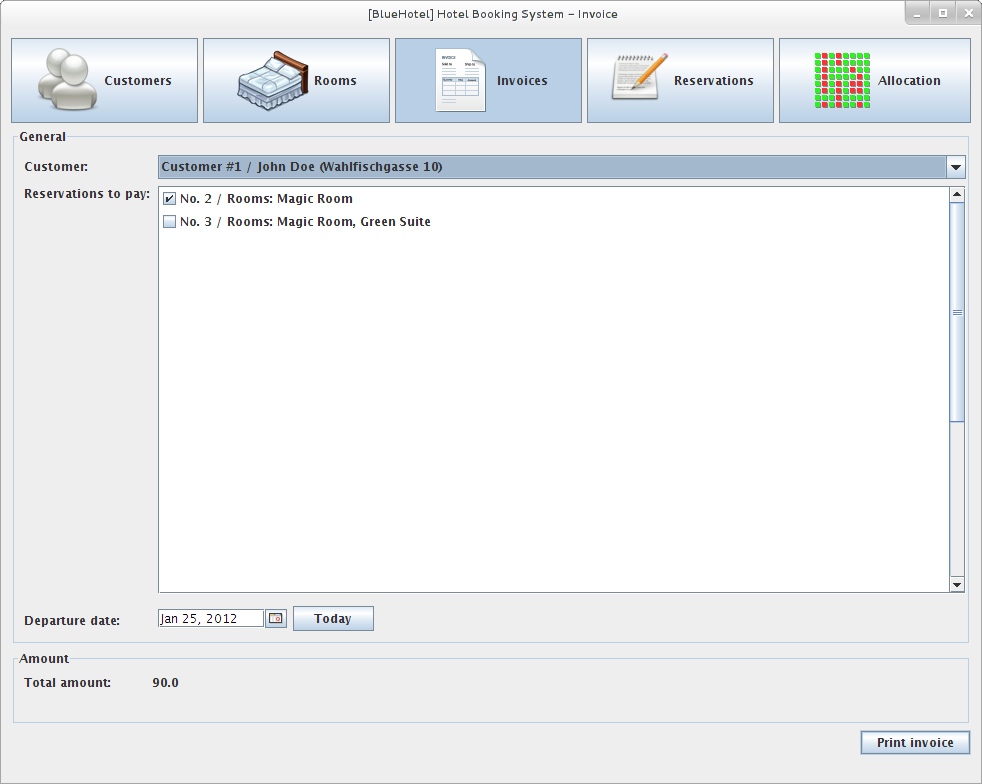
Die am meisten benutzte Funktion betrifft das Verwalten der Reservierungen. Um einen möglichst reibungslosen Workflow zu gewährleisten, ist diese Ansicht auch die Start-Ansicht, wenn man das Programm öffnet.

Zum Erstellen einer Reservierung wählt man zuerst die Kunden aus, die in dieser Reservierung inkludiert sein sollen. Danach kann man die Zimmer verbuchen, und dabei die Anzahl der Erwachsenen und Kinder angeben. Falls man einen Raum überbucht, wird das als Fehlermeldung beim hinzufügen des Raums zur Reservierung angezeigt. Nach der Zuweisung der Zimmer kann nun per Date-Picker das Start- und Ende-Datum des Aufenthalts angegeben werden.

Basierend auf der Raum-Auswahl wird dann der Preis berechnet. Dieser Preis kann bei Bedarf vom Ersteller der Buchung noch verändert werden – ein auf der Rechnung ausgewiesener Rabatt (ohne Minderung des Originalpreises) kann hier ebenfalls eingegeben werden.

Fehler in der Eingabemaske werden wie schon zuvor beim Speichern entdeckt und per Fehlermeldung angezeigt.

**Rechnung erstellen**

Abbildung: Rechnung erstellen mit Kunden- und Reservierungs-Auswahl

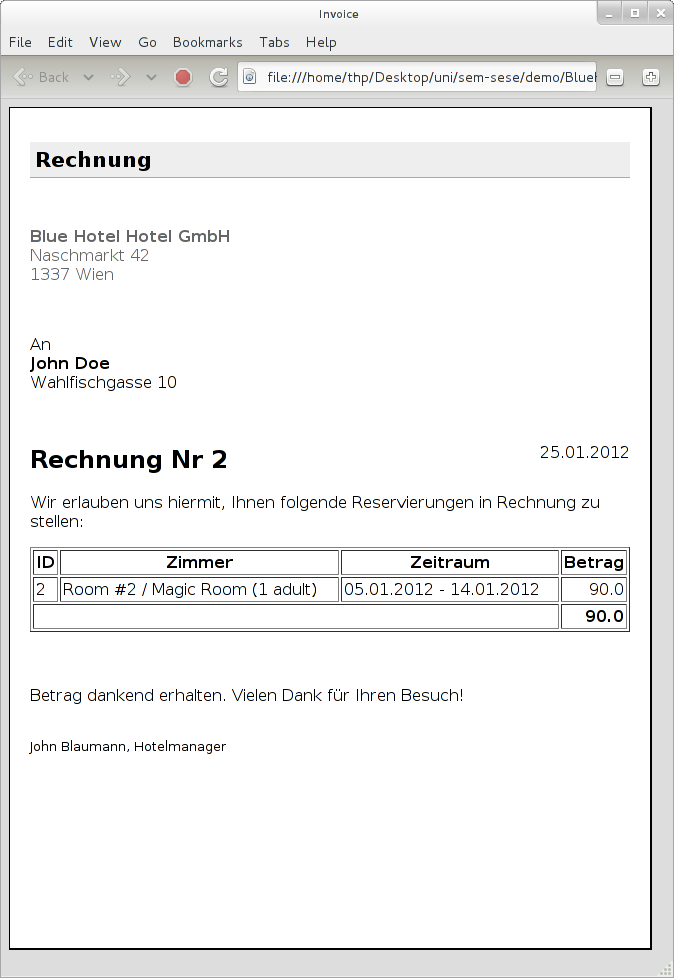
Die Rechnungslegung ist ein wichtiger Bestandteil des Systems, denn das bisherige manuelle Erstellen von Rechnung ist mühsam, und war einer der Gründe für die Beauftragung eines neuen Software-basierten Systems zum Management der Zimmer, Reservierungen und Rechnungen.

Ausgangspunkt für unsere Designentscheidungen sind wieder die üblichen Szenarien von kleinen Hotels: Ein Kunde nähert sich der Rezeption und will eine oder mehrere Reservierungen bezahlen – dies können auch Reservierungen sein, die nicht vom Kunden selbst gebucht worden sind. Als Beispiel sei dem Leser hier ein Firmenausflug vor Augen geführt, bei dem die Angestellten selbstständig ihre Zimmer reservieren, die Endabrechnung dann aber direkt vom Buchhalter übernommen und gesammelt bezahlt wird.

Weiters ist es möglich, das Abreisedatum – und somit das Rechnungsdatum – festzulegen. Für den unwahrscheinlichen Fall dass die Gäste verfrüht abreisen, steht ein „Today“-Button zur Verfügung. Die Reservierungen werden beim Erstellen der Rechnung wenn nötig automatisch verkürzt.

Rechnungen werden im HTML-Format gespeichert und angezeigt.

**Rechnung einsehen**

  
Abbildung: Erstellte Rechnung, direkt im Dateisystem abgelegt und geöffnet

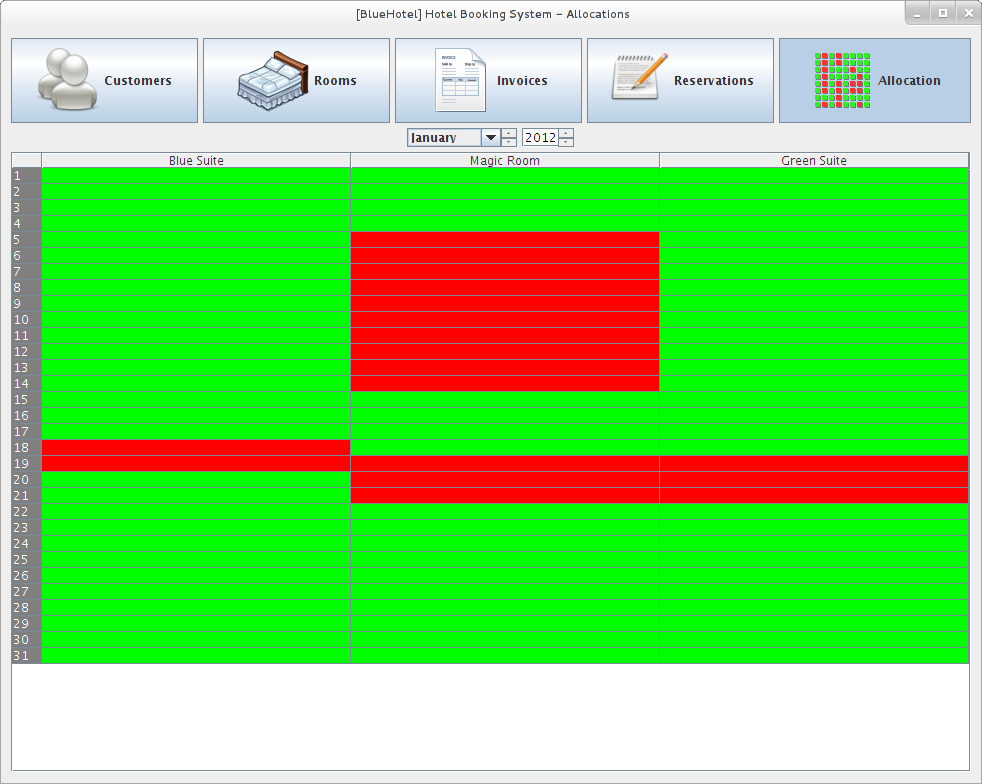
Wie im vorherigen Abschnitt bereits erwähnt ist die Rechnungslegung ein vitales Instrument zur Aufrechterhaltung des Hotelbetriebs aufgrund Geldeinnahmen durch Kundenzahlungen.

Nachdem man im Menüpunkt „Invoices“ (Rechnung erstellen) die Funktion „Print Invoice“ (Rechnung drucken) aktiviert hat, wird automatisch eine Rechnung in der Datenbank angelegt, die betroffenen Reservierungen als „bezahlt“ markiert und eine ausdruckbare Form des Rechnung als HTML-Datei im Dateisystem abgelegt. Dies hat einige Vorteile – unter anderem können die Rechnungen so leicht archiviert werden, und sind auch ohne die Benutzung des Programms verfügbar – was vorallem bei einer Vorhaltedauer von Rechnungen von 7 Jahren durchaus hilfreich ist.

Die Rechnungs-Informationen werden aus den Kunden- und Reservierungsdaten generiert, wobei hier auch Kundenrabatte (zB bei Stammgästen) Berücksichtigung finden.

Um den Ausdruck der Rechnung weiter zu beschleunigen, wird nach dem Speichern des HTML-Dokuments dieses sofort im Standard-Webbrowser des Systems geöffnet.

**Raumbelegung anzeigen**

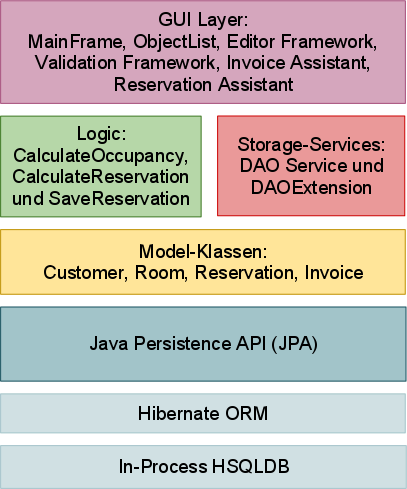
Abbildung: Anzeige der Raumbelegung für den Monat Jänner 2012

Für einen guten, visuellen Überblick über die momentane Auslastung der einzelnen Zimmer haben wir zusätzlich eine Belegungs-Liste der Zimmer implementiert. Diese zeigt optisch ansprechend die Reservierungen pro Zimmer und Tag an. So lässt sich auf einen Blick erkennen, wann noch Zimmer frei sind – dies ist vorallem bei Telefon-Anfragen („Wann haben Sie im März noch ein Zimmer frei?“) hilfreich.

In der aktuellen Implementierung dient die Belegungs-Grafik rein zur Visualisierung der Belegungen. Für zukünftige Erweiterungen wurde das System so flexibel gehalten, dass es möglich sein wird, die Reservierungen per Doppelklick auf rote Bereiche zu öffnen.

Weiters wurde von uns ein nice-to-have Feature angedacht, bei dem man durch drag'n'drop im grünen Bereich der Ansicht eine neue Reservierung erstellen kann, bei der das Ankunfts- und Abfahrtsdatum (Zeilen) sowie das gewählte Zimmer (Spalte) bereits vorausgefüllt sind. Dies wurde aber in der vorliegenden Version noch nicht implementiert.

* 1. Architektur

  
Abbildung: Architektur-Übersicht des Projekts BlueHotel

Beim Design der Software haben wir besonderen Wert auf eine gelayerte Architektur gelegt – im Idealfall greift ein Layer nur auf die Services des eigenen Layers und des direkt darunter liegenden Layers zu. In unserem Projekt verwenden wir als Datenbank eine In-Process HSQLDB, die gleichzeitig mit dem Programm gestartet wird.

Als ORM-Layer kommt Hibernate zum Einsatz; das Ansprechen des ORM-Layers erfolgt durch die standardisierte Java Persistence API (JPA). Die Model-Klassen verwenden die JPA mittels Annotationen. Die Model-Klassen werden von den Storage-Services (DAO mittels Generics plus DAOExtension für spezialisierte Abfragen) in die Datenbank geschrieben und von dieser gelesen.

Weiters wurde vom Projektteam ein spezielles Logic-Modul entwickelt, das parallel zu den DAO-Services Funktionalitäten überhalb der Datenbank zur Verfügung stellen, die aber nicht von der GUI abhängig sind. So eine Lösung hat den Vorteil, dass in einem möglichen Nachfolgeprojekt, bzw. einer möglichen Erweiterung der UI-Layer ausgetauscht werden kann, ohne die Applikations-Logik neu zu schreiben.

Im Architekturdiagramm zur bessern Lesbarkeit nicht abgebildet sind einige Hilfsklassen und Relations-Tabellen, die für die korrekte Persistierung von Objekten und als Glue-Code zwischen den Modulen und Schichten eingesetzt werden.

* 1. Technologien

Nach ursprünglichen Überlegungen und Besprechungen im Team (beim ersten Meeting) haben wir – unter Berücksichtigung des technischen Backgrounds aller Teammitgleider – uns für Java-Technologie als kleinsten gemeinsamen Nenner entschieden.

Weiters haben wir uns mit HSQLDB und Swing auf uns bekannte Technologien zur Datenspeicherung, bzw. zum UI-Design entschieden, da das Einlernen in andere Technologien den Projektfortschritt nur unnötig verlangsamt.

Als Testing-Framework haben wir uns bei Unit-Tests auf JUnit4 geeinigt, die funktionalen Tests werden ausschließlich manuell durchgeführt.

Die Vorteile dieser Technologien sind ausschließlich:

* Erfahrung aller Projektmitglieder mit der Sprache Java
* Distributions-Möglichkeit als „self-contained“ JAR-Datei
* Tooling-Support mit Eclipse und WindowBuilder Pro
* Portabilität von Java SE (Windows, Mac OS X und Linux)

Die Nachteile der gewählten Technologien sind unter anderem:

* Wenig Flexibilität der Sprache Java im Vergleich zu dynamischen Programmiersprachen
* Komplizierte Formulierung des Datenbank-Modells mittels Annotations und XML-Dokumenten
* UI-Integration (optisch sowie im Verhalten) von Swing-UIs in die nativen Oberflächen von Windows, Mac OS X und Linux
  1. Fazit

Verfassen Sie als Gruppe ein Fazit über die gewählten Architekturen und Technologien sowie die Auswirkungen des Prozesses auf die Auswahl.

Anhang

Zusätzliche Dokumente und Informationen.

* 1. Tagebuch Teilnehmer 1
  2. Tagebuch Teilnehmer 2
  3. Tagebuch Teilnehmer n

1. Quelle: [www.scrumalliance.org](http://www.scrumalliance.org) [↑](#footnote-ref-1)