<Project Name>

Pflichtenheft (SRS - Software Requirements Specification)

Version <1.0>

Revisionsverlauf

Datum	Version	Beschreibung	Author

Inhaltsverzeichnis

1. Einführung

1.1 System

[A brief description (one paragraph) of the system to be specified by its requirements: what it is used for, what it offers in general to the user, what the context of the system is like]

1.2 Zweck

[Specify the purpose of this **SRS**. The **SRS** should fully describe the external behavior of the application or subsystem identified. It also describes a domain model, nonfunctional requirements, design constraints and other factors necessary to provide a complete and comprehensive description of the requirements for the software. Furthermore, it defines an iterative development plan in the form of Timeboxes.]

1.3 Umfang

[A brief description of the scope of this **SRS** document; what Project(s) it is associated with, and what is not included in this document.]

1.4 Referenzen

[This subsection should provide a complete list of all documents referenced elsewhere in the **SRS** document. Each document should be identified by title, report number (if applicable), date, and publishing organization (if any). Specify the sources from which the references can be obtained. This information may be provided by reference to an appendix or to another document.]

1.5 Überblick

[This subsection should describe what the rest of the **SRS** document contains and explain how the document is organized. The SRS is adressed to a variety of readers, and in this section, it should be specified which part of the document is interesting for which type of user]

2. Stakeholder- und Benutzerbeschreibung

[To effectively provide products and services that meet your stakeholders' and users' real needs, it is necessary to identify and involve all of the stakeholders as part of the Requirements Modeling process. You must also identify the users of the system and ensure that the stakeholder community adequately represents them. This section provides a profile of the stakeholders and users involved in the project and the key problems that they perceive to be addressed by the proposed solution. It does not describe their specific requests or requirements as these are captured in a separate stakeholder requests artifact. Instead it provides the background and justification for why the requirements are needed.]

2.1 Stakeholder/Benutzer Zusammenfassung

[Present a summary list of all the identified stakeholders.]

Name	Rolle/Funktion	interessiert an
[Name the stakeholder type]	[Briefly describe the role she/he is playing in the development. and/or in the usage of the final product]	[Briefly describe the interests of this stakeholders relevant to system requirements]

2.2 Benutzerumgebung

[Detail the working environment of the target user. Here are some suggestions:
Number of people involved in completing the task? Is this changing?
How long is a task cycle? Amount of time spent in each activity? Is this changing?
Any unique environmental constraints: mobile, outdoors, in-flight, etc.?
Which systems platforms are in use today? Future platforms?
What other applications are in use? Does your application need to integrate with them?]

3. Produkt Überblick

[This section provides a high level view of the product capabilities, interfaces to other applications, and systems configurations]

3.1 Zusammenfassung Produktfähigkeiten/Eigenschaften

[Summarize the major benefits and features the product will provide. For example, a **SRS** document for a customer support system may use this part to address problem documentation, routing, and status reporting without mentioning the amount of detail each of these functions requires.

Organize the functions so the list is understandable to the customer or to anyone else reading the document for the first time. A simple table listing the key benefits and their supporting features might suffice. For example :]

Produktfähigkeit/-eigenschaft	**Stakeholder Nutzen/Gewinn**
xxx1	yyy1
xxx2	yyy2

3.2 Produkt Fähigkeiten/Eigenschaften

[List and briefly describe the product features. Features are the high-level capabilities of the system that are necessary to deliver benefits to the users. Each feature is an externally desired service that typically requires a series of inputs to achieve the desired result. For example, a feature of a problem tracking system might be the ability to provide trending reports. **This description refers to the use cases and usually describe them in the short form style.** Several lightweight Usecases may be described at a higher level as one feature.

Because the **SRS** document is reviewed by a wide variety of involved personnel, the level of detail should be general enough for everyone to understand. More details will be presented in chapter 5-6.

Throughout this section, each feature should be externally perceivable by users, operators or other external systems. These features should include a description of functionality and any relevant usability issues that must be addressed. The following guidelines apply:

- avoid design. Keep feature descriptions at a general yet precise level. Focus on capabilities needed and why, not how they should be implemented. A good style is that of UseCase descriptions in their short form

- several lightweight Usecases may be described at a higher level as one feature.]

3.2.1 <eine Fähigkeit/Eigenschaft>

[for example a usecase in short form style, or several usecases in synthesis]

3.2.2 <noch eine Fähigkeit/Eigenschaft>

[for example a usecase in short form style, or several usecases in synthesis]

3.3 Annahmen und Abhängigkeiten

[List each of the factors (mainly non functional features) that affects the features stated in 3.2. This is a synthesis of the non functional requirements stated in more detail in chapter 6. Here, assumptions are listed that, if changed, will alter the **SRS** document. For example, an assumption may state that a specific operating system will be available for the hardware designated for the software product. If the operating system is not available, the product features will need to change.]

4. Domänenmodell

[ein Domänenmodell ist die statische Darstellung von Klassen, die Konzepte aus dem Problemraum darstellen, und ihre statischen Beziehungen untereinander.
Die Klassen stellen *keine Softwareklassen* dar, sondern Konzepte aus dem Problemraum der Anwendung. Beziehungen des Domänenmodells beschreiben eine dauerhafte statische Beziehung zwischen Klassen. Die Beziehung soll selber ein Konzept ausdrücken]

4.1 Überblick

[das Überblicksklassendiagramm: Klassen ohne Attribute, mit Beziehungen]

4.2 Detailliertes Modell

[einzelne, detaillierte Klassendiagramme, möglichst nach Themenbereichen geordnet. Siehe die Strukturierung im Rose Template. In den Klassen sind auch Attribute visualisiert]

4.2.1 <eine Klasse>

[textliche Beschreibung einer Klasse: allgemeine Beschreibung ihrer Verantwortlichkeit, und Angabe ihrer Attribute, mit Beschreibung wenn die Bedeutung nicht selbstverständlich ist]

4.2.2 <eine andere Klasse>

[textliche Beschreibung einer Klasse: allgemeine Beschreibung ihrer Verantwortlichkeit, und Angabe ihrer Attribute, mit Beschreibung wenn die Bedeutung nicht selbstverständlich ist]

4.3 Einschränkungen (Constraints)

[falls auf manchen Klassen/Attributen/Beziehungen Einschränkungen bestehen. z.B. dass gewisse Beziehungen sich ausschliessen, oder gewisse Teile nicht relevant für das DB-Modell sind]

5 Dynamisches Modell

5.1 Detaillierte Benutzungsfälle (UseCases)

5.1.1 <ein detaillierter Benutzungsfall>

5.1.1.1 detaillierte Benutzungsfallbeschreibungen

5.1.1.2 Sequenz Diagramm

[optional für solche Benutzungsfälle, deren Schritte mit Kontrakten beschrieben werden]

5.1.1.3 Kontrakte

[für den UseCase als Ganzes, und für einzelne Arbeitsschritte darin. Ist optional, für Usecases bei denen wichtige, oder nicht selbstverständliche Voraussetzungen oder Nachbedingungen auf dem Domänenmodell zu definieren sind]

5.2 Objekt Lifecycles

[mit Zustandsdiagrammen (state charts), für Klassen deren Objekte komplexe Lebensläufe haben]

6. Nichtfunktionale Anforderungen

[nach den FURPS+ Kategorien, siehe Vorlesungsfolien Kapitel 2: Anforderungestypen. Nur das auswählen, was wir für den Roomanizer wirklich als Anforderung wissen]

6.1 Regeln

[Geschäftsregeln]

6.2 Usability

6.3 Zuverlässigkeit

6.4 Performanz

6.5 Unterstützbarkeit

6.6 Online Benutzerdokumentation und Help System

6.8 zugekaufte Komponenten

6.9 Schnittstellen

6.9.1 Benutzerschnittstellen

6.9.2 Software Schnittstellen

6.9.3 Kommunikationsschnittstellen

6.10 zusätzliche Lizenzierungen

6.11 Copyright und andere rechtliche Anforderungen

6.12 Anzuwendende Standards

7. Iterationenplan (Timeboxes)

[eine Beschreibung der 3 nächsten Realsierungsphasen (Iterationen) in Form von Timeboxen. Verlangt eine vorherige Berwertung von UseCases – das Ranking.

Ranking:

UseCases werden geordnet nach ihrer Bewertung. Die drei höchstbewerteten werden in den drei ersten Timeboxes realisiert. Die Berwertung geschieht nach folgenden Kriterien:

1. Risiko
    - Komplexität des UseCses
    - Anforderungen (inkl. an die Usability): haben/bekommen wir die richtigen.Anforderungen? wie stabil sind sie? haben wir Ansprechpartner?
    - Team KnowHow: der dazu notwendigen Technologie, und des Domänenverständnisses
    - Technologie: ist die einzusetzendeTechnologie gut definiert, stabil, erprobt, einfach?
    - politisch
2. Architekturrelevanz
   gibt die Realisierung des UseCases wichtige Einblicke in die SW-Architektur des Systems? Hilft er bei den wichtigsten Entscheidungen zur Architektur?
3. Benutzerrelevanz
   ist der Benutzer an einer frühen Realiserung dieses UseCases interessiert?
4. Lern- und Entwicklungsrelevanz
   ist der UseCase geeignet, um mangelndes teaminternes KnowHow bzgl. GUI-Entwurf, Klassenentwurf, Datenbankanbindung etc. aufzubauen?

Tip: Ranking zunächst nach 1-3 , dann 4 miteinbeziehen. Nur 1-3 in diesem Dokument erwähnen.

Da in unserem Zeitplan die Zeiten der drei ersten Timeboxes feststeht, muss für jede Timebox der zu realsierende UseCase (Main Success Scenario/Extensions) und die Architektur festgelegt werden in der er realsiert wird.

die Architektur betrifft Einschränkungen bezüglich:

- GUI
- Domänenmodell
- DB-Anbindung
- Reports, Ausdrucke, Networking, Security, HW-Plattform
- Fehlerbehandlung

und muss eventuell berücksichtigen, dass gewisse Systemteile, die der UseCase voraussetzt, bisher noch fehlen und in irgendeiner Weise zunächst bereitgestellt werden müssen, bis spätere Iterationen diese teile liefern werden.]

7.1 Überblick

[Reihenfolge der Timeboxen, welche UseCses, in welchen Zeiten, mit welchen Abhängigkeiten]

7.2 1. Timebox

[eine Iterationsbeschreibung, in der ein Systemteil realisiert wird, in einer fest vorgegebenen Zeit]

7.2.1 Benutzungsfall/fälle (UseCase(s))

[der zu realsierende UseCase, oder mehrere. Und der Umfang davon: Main Success Scenario, wenn das in früheren Timeboxes noch nicht realisert wurde, dann welche Extensions?]

7.2.2 Architektur

[siehe oben, Info zu 7. Immer aus der Perspektive von Einschränkungen]

7.2.3 Deliverables

[Code, Manuale, Spezifikationen, Testdaten/-ergebnisse]

7.2.4 Abhängigkeiten

[als Vorbedingungen: von Systemfähigkeiten, von UseCases anderer Timeboxes, Testdaten, Benutzerinput/-einbezug]

7.3 2. Timebox

7.3.1 Benutzungsfall/fälle (UseCase(s))

7.3.2 Architektur

7.3.3 Deliverables

7.3.4 Abhängigkeiten

7.4 3. Timebox

7.4.1 Benutzungsfall/fälle (UseCase(s))

7.4.2 Architektur

7.4.3 Deliverables

7.4.4 Abhängigkeiten

8 Glossar

[wichtige Begriffe des Domänenbereichs in kurzer Textform definiert]