System design document for the

Speedtype project (SDD)

Contents

[1Introduction 3](#_Toc325060845)

[1.1 Design goals 3](#_Toc325060846)

[1.2 Definitions, acronyms and abbreviations 3](#_Toc325060847)

[2 System Design 3](#_Toc325060848)

[2.1 Overview 3](#_Toc325060849)

[2.1.1 Rules 3](#_Toc325060850)

[2.1.2 The model functionality 4](#_Toc325060851)

[2.1.3 Value objects 4](#_Toc325060852)

[2.1.4 Unique identifiers, global look-ups 4](#_Toc325060853)

[2.1.5 Spaces 4](#_Toc325060854)

[2.1.6 Event handling 4](#_Toc325060855)

[2.1.7 Internal representation of text 4](#_Toc325060856)

[2.2 Software decomposition 4](#_Toc325060857)

[2.2.1 General 5](#_Toc325060858)

[2.2.2 Decomposition into subsystems 5](#_Toc325060859)

[2.2.3 Layering 5](#_Toc325060860)

[2.2.4 Dependency analysis 5](#_Toc325060861)

[2.3 Concurrency issues 5](#_Toc325060862)

[2.4 Persistent data management 5](#_Toc325060863)

[2.5 Access control and security 5](#_Toc325060864)

[2.6 Boundary conditions 5](#_Toc325060865)

[3 References 5](#_Toc325060866)

Version: 1.0  
Date 2012-05-17  
Author: Daniel Larsson

This version overrides all previous versions.

# 1 Introduction

## Design goals

The goal is to make the design very loosely coupled in order for it to be possible to switch both GUI and model in possible future application additions or changes. The design must support the addition of new game modes and settings. The model must be possible to test completely isolated. For usability, see RAD.

## Definitions, acronyms and abbreviations

All definitions and terms regarding the core Speedtype game are as defined in the reference section.

* GUI, graphical user interface
* Java, platform independent programming language.
* JRE, the Java Run time Environment. Additional software needed to run a Java application.
* SDK, Software Development Kit. Needed for developing Android applications.
* Host, a phone where the game will run.
* Round, one complete game ending with game over or possibly canceled.
* Score, the score for the player during one round.
* Activity, a class that holds the GUI, comparable to a JFrame in Swing.
* Service, an application component representing an application’s desire to perform a long-running operation in the background, without interaction with the user. I.e. background music.

# 2 System Design

## 2.1 Overview

The application will use a classic MVC model.

### 2.1.1 Rules

There are very few rules to the game. The ones that do exist are very simple, and also vary between the different game modes. Therefore not much refactoring can be done. The main rule of all game modes is what happens when a player gives input in the form of a letter. Depending on which game mode is being played, completely different actions will be taken by the application.

### 2.1.2 The model functionality

The abstract class GameFactory enables the possibility to create several GameModes easily without heavy modification to the code. Words are stored in a SQLite database for the application to support future additions to the game logic. That way, the dictionary is dynamic and allows for the application to add new words to it. The solution also provides support for future additions of new languages.

### 2.1.3 Value objects

The model classes expose some functionality. The only object that is completely free from functionality is the class Word. Skriv mer häääääär

### 2.1.4 Unique identifiers, global look-ups

The database is the only globally unique identifier used. Vad look-up är har jag inte en jävla aning om, Jocke borde gå kommunikationskursen.

### 2.1.5 Event handling

### 2.1.6 Internal representation of text

### 2.2 Software decomposition

### 2.2.1 General

### 2.2.2 Decomposition into subsystems

### 2.2.3 Layering

### 2.2.4 Dependency analysis

## 2.3 Concurrency issues

## 2.4 Persistent data management

## 2.5 Access control and security

## 2.6 Boundary conditions

# 3 References