Software Requirements Specification (SRS) for *natsu*

TophUwO

December 2023

Contents

Version History

Below is a chronological list of changes, alongside metadata. Version *1.0* marks the first release version of this document.

Note

Dates used in this table are of the format ***MM/DD/YYYY***.

Date	Version	Contributor(s)	Changes
12/07/2023	0.1	TophUwO	initial commit
12/07/2023	0.1.1	TophUwO	**ADD** content structure for *Introduction* section; **ADD** text for *Purpose...*, *...Conventions*, and *...References*
12/07/2023	0.1.2	TophUwO	**ADD** initial content for *...Rationale* sub-section
12/07/2023	0.1.3	TophUwO	**ADD** contents for product feature overview
12/07/2023	0.2	TophUwO	**ADD** contents for section *General Description*
12/08/2023	0.3	TophUwO	**ADD** rest of document structure; **ADD** contents for multiple sections
12/08/2023	0.5	TophUwO	**ADD** content for section *Core System Features* (i.e. the requirements overview); **ADD** content for all sections in section *General Description*; **ADD** table describing software quality requirements; **ADD** content for section *External Interface Requirements*
12/09/2023	0.5.2	TophUwO	**ADD** contents for detailed requirements for *Creation of Subjects...* sub-section inside the *Specific Requirement* section
12/12/2023	0.5.5	TophUwO	**ADD** specific requirements for *ReviewControl* widget and additional (optional) requirements for the application
12/12/2023	0.5.6	TophUwO	**FINISH** specific requirements for *Review of Items* section; **ADD** diagram illustrating *stroke fanning*
18/01/2023	0.5.7	TophUwO	**ADD** package diagram v1

1 Introduction

This chapter serves as a general introduction to the product, its features, and its working environment. The features and requirements mentioned here will be discussed and elaborated on in chapter 3 and 4.

1.1 Purpose of This Document

This document describes the software requirements specification (SRS) for a standalone desktop and notebook application dedicated to enhancing the user's *Hanzi/Kanji/Hanja*[1] learning experience and efficiency. The application is codenamed *natsu* (jp. for *Summer*) and will be referred to as such throughout the remainder of this document. This document was created in accordance with the *IEEE 830* standard, is part of the internal developer documentation, and serves as a benchmark of what the application is intended to do and be used for.

1.2 Document Conventions

In this document, technical and other domain-specific terms relevant to the implementation are in *italics* while table heads are rendered in **boldface**. See section *Glossary* for an explanation of important technical terms used in the context of the product's domain. Other nomenclature (also in *italics*) may be further explained by either footnotes or in the section *External References*. Terms in *italics* are also hyperlinks to other sections within this document.
This document manages its own version history (see section *Version History*). Each commit holds data such as *Date*, *Version*, *Contributor(s)*, and *Changes*. Changes begin with a verb in infinitive form, all-caps, in boldface, describing the change's category, e.g. **ADD**, **REMOVE**, **FIX**, etc.
Figures such as GUI mock-ups and OOA diagrams can be found in the Appendix A and B, respectively.

1.3 Background and Rationale

Languages are a very prevalent subject in today's globalized society. For the current and following generations of people, mastery of different languages appears mandatory in a variety of industries that thrive in an international environment. Particularly Eastern-Asian languages are popular in the Western world due to interest, requirements, or simply popular culture. What many of the most popular languages like Mandarin, Japanese, and to an extent (South-)Korean have in common is the use of logographic characters of ancient Chinese origin. While these characters are referred to as Kanji (漢字) in Japanese, regional names may differ.
This SRS describes an application that is supposed to substantially relieve some of the frustrations commonly encountered when learning Kanji. While there are high-quality solutions aimed at learning Kanji in context, no solution of this quality exists for writing practice. This is the gap natsu will fill. I believe that the application will be useful to those who want to learn Japanese hand-writing in a controlled and customizable environment.

1.4 Product Feature Overview

At its core, the application should be a software that manages flashcards by the means of an SRS (*spaced repetition system*). The user is able to create new or import existing ones. Progress on each item is saved persistently. The user reviews an item by writing it by hand into a widget on the screen. There should be multiple algorithms used to calculate the review interval. Styling of the application should be possible through settings dialogs.

1.5 Stakeholders and Audience

Since this is a personal project, the only stakeholder is the project manager, developer and maintainer, TophUwO. This document is also meant for future contributors of the software described within this document.

[1]logographic characters of Chinese origin; used to this day in Chinese languages, Japanese, (South-)Korean, etc.

1.6 Glossary

Below is a curated list of context-specific nomenclature used throughout this document, alongside brief explanations.

Term	Explanation
radical	graphical parts of a Kanji used for indexing
reading	a way of pronouncing a character; context-sensitive
stroke	smallest distinguishable grapheme in a Kanji/radical
Qt	cross-platform application framework
GPL	popular license for open-source development
(Kanji) variant	different variant of given Kanji such as *Hyōgai* variants
review item	same as *flashcards*

1.7 External References

Below is a non-exhaustive list of useful resources that are likely to be frequently consulted throughout the conceptualization and implementation of the software described by this document.

Note

The resources presented beneath may serve as a starting point but do in no way represent a complete and necessarily sufficient resource for the entirety of the subject. Additional research is expressly endorsed.

Resource	URL
IEEE 830 specification	`http://www.math.uaa.alaska.edu/~afkjm/cs401/IEEE830.pdf`
Qt 6 documentation	`https://doc.qt.io/qt-6/`
spaced repetition	`https://en.wikipedia.org/wiki/Spaced_repetition`
KanjiVG project	`https://kanjivg.tagaini.net/`
DOXYGEN	`https://www.doxygen.nl/`
ISO/IEC 25010:2011	`https://iso25000.com/index.php/en/iso-25000-standards/iso-25010`
Hyōgai kanji	`https://en.wikipedia.org/wiki/Hy%C5%8Dgai_kanji`
stroke fanning	`http://dragice.fr/strokefanning/`

2 General Description

2.1 Perspective

The software described by this document is a standalone application, supposed to be used by myself for enhancing my Japanese studies. A release is not planned as of now, but is thinkable if the application turns out worthwhile.

2.2 Product Features

At its core, the application should be a software that manages flashcards by the means of an SRS (*spaced repetition* system). These flashcards should contain information on the Kanji character, such as *radicals*, *readings*, *stroke information*, *common vocabulary*, etc. These flashcards can be created and modified by the user. Flashcards are part of a *deck* that can be imported/exported from either the proprietary natsu deck format, or other formats such as *CSV* or SQL-based databases. Progress on each item must be saved persistently in a database. Optionally, items can be grouped together in a *level order* which allows for certain subjects to be studied to retention until other items are processed. The application will be shipped with a large number of pre-installed Kanji characters, sourced from the excellent *KanjiVG* project. There should be a way to add new characters to the repository, either by pre-made data or manual creation by the means of drawing. In the review, the user is now required to write the subject whose meaning/hint is shown. The application relies on self-evaluation in order to determine progress. Furthermore, the application should be focused on customization via application and icon styles. It should be possible to set a global accent color that will be used for icons, focus indicators and miscellaneous items present in the GUI.

2.3 Application Users

The application's target group is avid language learners of Japanese. As such, they should be able to use a mouse and keyboard as well as be familiar with the usage of a touchscreen. Furthermore, they should be keen on working with common user interface elements such as buttons, dialogs, and sliders.

2.4 User Classes and Characteristics

There should be a way to create multiple profiles in the same application. Decks and subjects can be shared across profiles but progress cannot. There is no user hierarchy. Therefore, only one user class, *user*, exists.

2.5 Operating Environment

The application does not directly interact with hardware, but uses interfaces provided by the host platform. Similarly, there is no direct communication with any other software product. The application is to be used with a mouse and keyboard and various touch devices like Wacom touch tablets and touchscreens.

2.6 Developer Limitations

There are little limitations in place. The application should be written in C++ using the *Qt* (widgets) application development framework. Qt should be version 6.6 or later. Third-party frameworks can be used if they are not licensed under a strong copyleft license like *GPL*. Inside the Qt ecosystem, all APIs that are not solely available under a strong copyleft license can be used freely. Persistent storage for application settings should be JSON documents. The language used in the code-base must be *English (United States).*

2.7 User Documentation

Aside from the SRS provided by this document, there will be a *DOXYGEN*-based online developer documentation available. Furthermore, there must be a user documentation, outlining the application's features. The user documentation must be accessible from the application (via a hyperlink or an embedded manual). The format of the user documentation is HTML. A printable PDF edition is not planned.

2.8 Assumptions

Qt is cross-platform. The application should be compilable and runnable on all major desktop operating systems. Furthermore, the application, when used with a mobile (touch-enabled) device, should only be runnable if the display size is greater than or equal to 7 inch.

2.9 Distribution

The application's main distribution format is compiled and packaged, ready to be installed. The application should also be available as a portable version. The application's source code is not to be distributed alongside the compiled binaries. A release of the application's full source code under a non-copyleft open-source license is thinkable. Details regarding this must be worked out in time.

2.10 Optional Features

Optionally, the review items can be extended to provide more information on the review item such as – but not limited to – *reading(s)* (sequence of ordered pairs where the first value denotes the type of reading, e.g. *go'on*, *kan'on*, etc. and the second value the reading as a UTF-8 encoded sequence of *hiragana* characters). During the review, these information may or may not be displayed alongside the item's solution. Another additional feature is the import of stroke data from SVGs when creating new review subject entries.

Optionally, the canvas' drawing tool will support tilt and pressure handling. This allows for prettier and more realistic stroking. Furthermore, the canvas may also support an eraser tool, functioning on a stroke basis (i.e. can delete individual strokes by selecting the tool and clicking on the stroke). The canvas may show outlines of the item's representation during the self-evaluation phase. This allows the user to see how a character is positioned inside the grid and where the strokes begin/end in direct comparison to the user's drawn strokes.

The review module may support *reverse review*, which will show the review item's representation inside the *ReviewItemView* widget and the meaning of that item has to be written into the canvas. Reverse review will use its own SRS state. The rest is similar to the normal review.

3 Core System Features

3.1 Creation of Subjects, Decks, and Review Items

The application should allow the creation of review subjects (i.e. Kanji entries) inside the subject repository. This should be done through a *SubjectEditor* component which provides classes and widgets for the creation of subjects. There should be a way to extract subject stroke path data from SVG files.
The application should feature a *DeckManager* component that allows for creation of *decks* and *review items*. Review items must be part of a deck. Review items are associated with a *subject representation*, typically a text. Optionally, review items can be grouped into *levels* which allow for more control of what review items come first.

3.2 Review of Items

The *ReviewManager* component handles the creation of and application of SRS algorithms to determine the review intervals. After the user has selected a deck to review, the review starts and shows the *ReviewView* component, where the user is asked to hand-write the review item currently in focus. When the user is done, the answer will be shown and the user has to evaluate their answer and confirm their evaluation with designated controls.
The *ReviewView* also features a *ReviewItemView* component that resembles a media-player like interface, allowing the user to view the stroke order of the current review item, allowing the user to skip through strokes, characters, etc.

3.3 Customization

To enhance user experience, the application should support visual customization. Besides setting a global accent color, most view components should be freely-dockable and resizable widgets, allowing the user to freely arrange the working area to their liking.
Additionally, there should be multiple configurable color schemes to adjust the general look of the application.

4 External Interface Requirements

4.1 User Interfaces Overview

Generally, the application is a graphical UI. It uses dialogs for user input and uses the mouse and the keyboard extensively (for shortcuts, etc.). Data is submitted through submit buttons present in the respective dialogs. Standard dialogs like the *File Open dialog* or the *Choose Color dialog* as well as *Message Boxes* will be native dialogs provided by either the host OS or the respective Qt implementation. The top of the main window contains an omnipresent toolbar, showing buttons, some with menus, allowing access to main application features. It allows the user to authenticate, access settings, quit the application, access help resources, launch the deck, card, and subject editors. The rest of the application's functionality should be exposed via (docked and undocked) widgets.

4.2 Hardware and Software Interfaces

The application does not implement any device drivers itself. Therefore, all interaction with hardware is done through host OS-provided interfaces. Similarly, apart from the classes provided by the Qt framework, the application does not interact with any third-party software directly. The application framework used for building the application is Qt 6.6 or newer.

5 Specific Requirements

5.1 Functional Requirements

5.1.1 Creation of Subjects, Decks, and Review Items

Regarding the review subject editor, besides stroke data, the subject entry should also feature other properties such as *radicals* and *variants*. There should be a way to view a list of all review subjects, alongside buttons to *edit*, *delete*, and *duplicate*. Pressing the *edit* button launches the review subject editor. The review subject editor consists of a *drawing area*, a *properties editor*, and a *subject preview window*. The drawing area is a canvas that allows the user to draw a review subject's Kanji representation stroke by stroke, with usual image editing functions such as undo, redo, and clear. The current state of the canvas can be saved to an image. To draw a stroke, the user has to select the *draw* tool, and draw the stroke. A stroke has to be drawn in one go where the stroke is being recorded as long as the user is drawing. The properties editor shows a form that allows the user to edit the subject's attributes mentioned above. The subject preview window shows a preview of the created subject strokes, similarly to the *ReviewItemView* component explained beneath. By default, the application is shipped with subjects extracted from the KanjiVG project. These subjects should be editable as well.
In the deck manager, the user can create and manage decks, as well as the deck's review items. A deck has properties like *name*, *author*, *version*, *description*, and *comments*. A deck can hold a practically unlimited number of review items. Deleting a deck will cause the deck to be deleted only for the current user. Also, all review progress for that deck of the current user is deleted. Alternatively, the deck can be *reset* which causes all review progress to be deleted as well. Decks can also be duplicated which will, however, not duplicate the review progress. Review items can be grouped in so-called *levels* which allow the user to have only a portion of the deck be in the review until another portion is unlocked. A review item should have properties such as *representation* (UTF-8 encoded sequence, representing the subject's textual representation), *meaning(s)* (sequence of UTF-8 encoded strings), and *part(s) of speech* (bitfield). The deck manager should check whether the review item's representation can be displayed using the current review subject repository. If this is not possible, a warning message is shown but the review item will still be reviewable.

5.1.2 Review of Items

A review can be started from the deck overview screen or from a menu/dialog. The deck overview screen shows the deck's name, description, how many new cards there are and how many are available for review. Optionally, it shows the deck's current level. It can be structured in a table-like structure, or a card-like structure. Once a review/lesson is started, the main window will show three components: A *Canvas*, a *ReviewControl* widget, and a *ReviewItemView* widget.
The canvas largely functions like the canvas used in the review subject editor, just that it allows for more than once cell be shown at a time. It can optionally display a grid which can be dynamically sized. Sizing the grid will clear its contents. The user can use a pointing device to draw strokes on the canvas. The drawing tool is a pen that supports basic drawing. Tilt and pressure handling is an optional feature. The color of the drawing tool can be configured as well as the tool's line width. There should be a way to save the current canvas state to a disk file, in a variety of raster-graphics formats. Another function is a clear function. Another feature is an undo/redo buffer of a configurable size. The canvas' size is configurable as well. It should be settable in units of cells, where cells are in units of pixels.
The ReviewControl widget shows the *meaning(s)* of the current review item, with the first meaning in the list of meanings being shown first. The review control features buttons for controlling the number of meanings shown. When multiple meanings are shown, they are shown in the order they appear in the meanings list for that particular review item. A default number of meanings can be configured. If a review item has less meanings than the minimum number of shown meanings configured, all meanings of that review item will be shown initially. Apart from the buttons controlling the meanings, a button exists that allows the user to have the answer shown to them. When this button is pressed, this buttons disappears and four buttons appear in its place that allow the user to self-evaluate their answer. The buttons should be *again*, *hard*, *okay*, *easy*. Based on the evaluation, the new review interval will be calculated. After the evaluation button was pressed, the review will continue with the next review item. The font, font size, text color, and font style of the review view can be configured. Furthermore, alongside meanings, the view should also be able to show what part(s) of speech the review item is classified as, if any are specified.

Lastly, the ReviewItemView widget displays the current review item's representation during the self-evaluation phase. During the answer phase, this control is visible but completely intangible. Basically, it allows the user to display the item's representation in different ways. The main way is *animation*, where the item's strokes are drawn one-by-one by animating the stroke painting. The next display method, *sequential*, is mostly like animation, but it does not animate the stroke painting, so the item is shown stroke by stroke by sequentially adding them to the view. Both *animation* and *sequential* allow the user to control the current drawing state of the item using controls that resemble those of a media player. The following controls are mandatory: *Play/Pause* (as one button), *Prev/Next Stroke* (one button each), *Prev/Next Character* (one button each), *Begin/End* (one button each). Aside from these "media control buttons", the view should feature a slider that allows the user to skip through or jump to any stroke position in the representation. Holding down the slider handle should show the current character and stroke index (as a tool-tip). Furthermore, the "player" supports *looping* and a *grid* that can be either shown or hidden. Also, when *animation* or *sequential* are are the selected drawing mode, the view may also display the stroke number where the stroke begins. This can be turned on/off via a button. The grid should be freely-configurable but presets exist. Another way of displaying character strokes is *stroke fanning*, where strokes are shifted to the left but maintain their y-position, alongside a dot where the stroke begins. See section *Supplementary Resources* for an example. The ReviewItemView widget should be a dockable window that can be resized and optionally floated as a top-level window. The representation should always be as large as possible while not having to add scroll-bars and while maintaining aspect ratio.

5.1.3 Customization

Customization options should be custom application themes altering the appearance of the application. These themes can change the color scheme or the lower-level QSS-based stylesheet of the application.

5.2 Non-functional Requirements

5.2.1 Performance

There should be no noticeable lag when interacting with the application's user interface. If an action is expected to take longer than 0.5 seconds, a progress bar indicating progress should be shown. Very expensive operations should be offloaded to separate threads in order to not block the main thread. Where possible, GPU-accelerated rendering should be used. When using GPU-accelerated rendering, Qt's built-in *QOpenGLWidget* should be used where supported. Rendering using other technologies (i.e. *Direct2D*, *Vulkan*, etc.) are completely optional features.

5.2.2 Internationalization

The application's default language is *English (United States).* Additionally, the application must provide a way to change the GUI's language to *German (Germany).* Translations into languages other than English or German are optional features. There should be a way to install new languages. Language packs should be distributed as Qt **.qm* files generated from Qt's own in-ecosystem internationalization tool, *Qt Linguist.*

5.2.3 Software Quality Requirements

Below is a list of all *ISO/IEC 25010:2011* software quality requirements alongside an integer value, the level of importance, in the interval [1, 5] where *5* denotes very high importance and *1* denotes low importance relative to level *5*.

Requirement	Description	Priority
functional suitability	This characteristic represents the degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions.	**4**
performance efficiency	This characteristic represents the performance relative to the amount of resources used under stated conditions.	**2**

compatibility	Degree to which a product, system or component can exchange information with other products, systems or components, and/or perform its required functions while sharing the same hardware or software environment.	**1**
usability	Degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.	**5**
reliability	Degree to which a system, product or component performs specified functions under specified conditions for a specified period of time.	**2**
security	Degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization.	**2**
maintainability	This characteristic represents the degree of effectiveness and efficiency with which a product or system can be modified to improve it, correct it or adapt it to changes in environment, and in requirements.	**3**
portability	Degree of effectiveness and efficiency with which a system, product or component can be transferred from one hardware, software or other operational or usage environment to another.	**4**

A Supplementary Resources

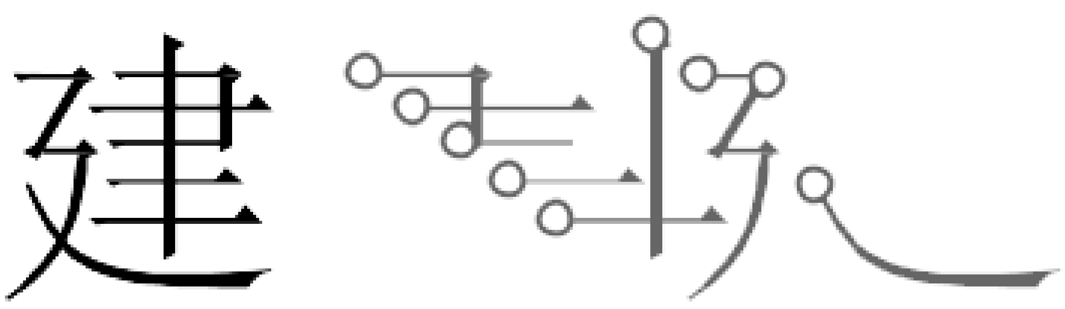

Figure 1: example result of *stroke fanning*, via `http://dragice.fr/strokefanning/`

B GUI Prototypes

C OOA Diagrams

C.1 OOA package diagram

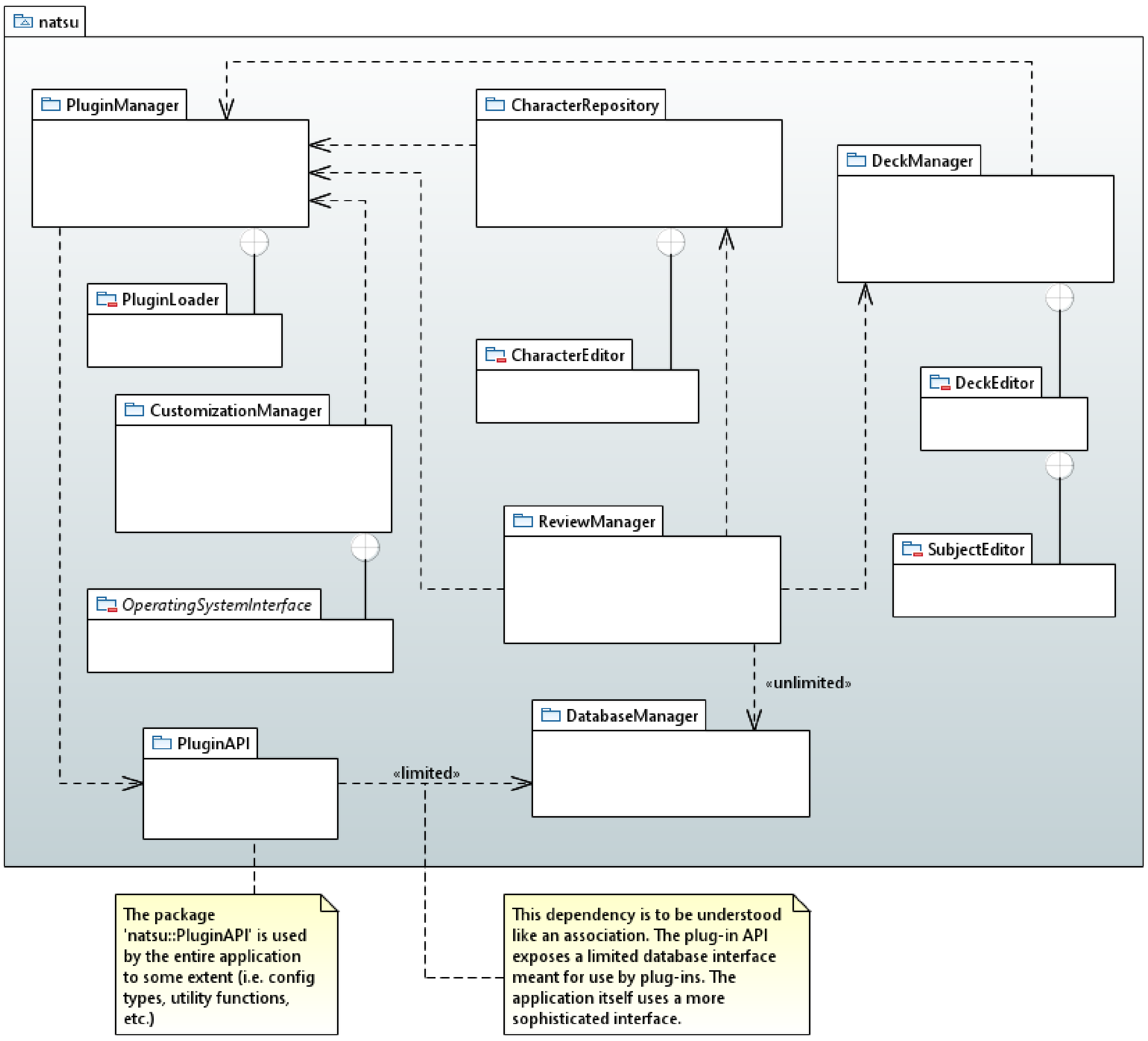
natsu
PluginManager
CharacterRepository
DeckManager
PluginLoader
CharacterEditor
DeckEditor
CustomizationManager
ReviewManager
SubjectEditor
OperatingSystemInterface
«unlimited»
DatabaseManager
PluginAPI
«limited»
The package 'natsu::PluginAPI' is used by the entire application to some extent (i.e. config types, utility functions, etc.)
This dependency is to be understood like an association. The plug-in API exposes a limited database interface meant for use by plug-ins. The application itself uses a more sophisticated interface.

Figure 2: package diagram for the top-level architecture of *natsu*

C.2 Use-Case Diagrams

C.3 OOA Class Diagrams