SE 3XA3: Module Guide
MacSidenotes

Team 4
Josh Mitchell mitchjp3
Matthew Shortt shorttmk

December 7, 2016

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
Nov 13th	Rev0	Complete Sections 1-7
Dec 7th	Rev 1.0	Revision 1 Modifications

1 Introduction

MacSidenotes is a Chrome Extension for taking notes on any web site. Your notes are linked with the URL and will appear upon your next visit. Works offline using Chrome's local storage.

MacSidenotes uses an MVC architecture. The model of the application would be the HTML, which is the skeleton for the content. The CSS of the extension would represent the view, as it adds the visual styling to the application. This perfectly embodies the separation of concerns as any changes to the CSS will not affect the functionality of the application. Finally, the controller would be the JS (JavaScript) file in tandem with the browser, in our case Google Chrome. Chrome is responsible for the combination of the HTML and CSS as well as rendering them to the screen, while the JS file recognizes input from the user and handles it accordingly.

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (Parnas et al., 1984). We advocate a decomposition based on the principle of information hiding (Parnas, 1972). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules layed out by Parnas et al. (1984), as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is used in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (Parnas et al., 1984). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.

- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 2 lists the anticipated and unlikely changes of the software requirements. Section 3 summarizes the module decomposition that was constructed according to the likely changes. Section 4 specifies the connections between the software requirements and the modules. Section 5 gives a detailed description of the modules. Section 6 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 7 describes the use relation between modules.

2 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 2.1, and unlikely changes are listed in Section 2.2.

2.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: The trigger of the save functionality.

AC2: Appearance of the UI (eg. buttons).

AC3: Location & format of saved notes.

AC4: Method of determining Master List visibility.

AC5: Method by which the Master List is updated.

AC6: Method by which the page URL is fetched.

AC7: Method by which the User is notified of successful Note saves/deletions.

2.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: Input/Output devices (Input: File and/or Keyboard, Output: File, Memory, and/or Screen).

UC2: There will always be a source of input data external to the software.

UC3: The software will be run within a Chrome Extension environment.

UC4: ~~The goal of the software: to save notes for the user for later viewing.~~

3 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 2. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Hardware-Hiding Module - handled by the Chrome Extension environment on which the software is running. Once Chrome has been installed, the hardware is hidden from the rest of the software.

M2: Save/Delete Note Module - includes functions: saveNote, updateNote

M3: Master List Visibility Module - includes functions: showList, clickCounter as appearance specification in the CSS file

M4: Update Master List Module - includes functions: updateMasterList, deleteMasterList, populateMasterList

M5: URL Fetch Module - includes function: getURL

M6: Save/Delete Trigger Module - event listeners located in the 'DOMContentLoaded' handle user interaction, including button clicks

M7: ~~Notify User Module~~

M8: UI Appearance Module - CSS and HTML files handle the appearance of the extension

M9: Note Typing Module - includes function: updateNote in tandem with the HTML and CSS files to display a textArea.

~~**M1** is handled by the Chrome Extension environment on which the software is running. Once Chrome has been installed, the hardware is hidden from the rest of the software.~~

Level 1	**Level 2**
Hardware-Hiding Module	Google Chrome API
Behaviour-Hiding Module	Save Trigger Module ~~Notify User Module~~ UI Appearance Module Note Typing Module
Software Decision Module	Update Master List Module Save Note Module URL Fetch Module Master List Visibility Module

Table 2: Module Hierarchy

4 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table 3.

5 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas et al. (1984). The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. Also indicate if the module will be implemented specifically for the software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented. Whether or not this module is implemented depends on the programming language selected.

5.1 Hardware Hiding Modules (M1)

Secrets: The data structure and algorithm used to implement the virtual hardware.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software. So, the system can use it to display outputs or to accept inputs.

Implemented By: OS, Chrome

5.2 Behaviour-Hiding Modules

5.2.1 Save Trigger Module (M6)

Secrets: What causes the Note to be saved (button press vs closing the sidebar).

Services: Allows the call of the Save Note module **M2** (specified below).

Implemented By: MacSidenotes

5.2.2 Notify User Module (M7)

Secrets: ~~The format of the user notification message.~~

Services: ~~Allows the user to be alerted when they have successfully saved or deleted a Note.~~

Implemented By: ~~MacSidenotes~~

5.2.3 UI Appearance Module (M8)

Secrets: The format and appearance of the UI elements.

Services: Allows the user to have an uncluttered and pleasant experience with the software.

Implemented By: MacSidenotes

5.2.4 Note Typing Module (M9)

Secrets: The format and appearance of the text area the user types in.

Services: Allows the user to type a note into the extension.

Implemented By: MacSidenotes

5.3 Software Decision Modules

5.3.1 Update Master List Module (M4)

Secrets: How the Notes are fetched from storage and added to the Master List.

Services: Loads all previously saved Notes into the Master List.

Implemented By: MacSidenotes

5.3.2 Save Note Module (M2)

Secrets: The format and location of saved Notes.

Services: Allows the user to save notes for future viewing.

Implemented By: MacSidenotes

5.3.3 URL Fetch Module (M5)

Secrets: The technique by which the URL is fetched.

Services: Allows each Note to be saved alongside an accompanying URL.

Implemented By: MacSidenotes

5.3.4 Master List Visibility Module (M3)

Secrets: The technique by which the visibility of the Master List is determined.

Services: Allows the user to toggle the Master List on and off.

Implemented By: MacSidenotes

6 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	Modules
F.1	M1, M8
F.2	M8
F.3	M2, M6
F.4	M8, M3
F.5	M3
NF.1	M8
NF.2	M8
NF.3	M8
NF.4	M2, M4, M3
NF.5	M8
NF.6	M1, M8, M2, M4, M3
NF.7	M1
NF.8	M1
NF.9	M1
NF.10	M2
NF.11	M1
NF.12	M8
NF.13	M8, M2
NF.14	M8

Table 3: Trace Between Requirements and Modules

AC	Modules
AC1	M6
AC2	M8
AC3	M2
AC4	M3
AC5	M4
AC6	M5
AC7	~~M7~~ M6

Table 4: Trace Between Anticipated Changes and Modules

7 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas (1978) said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of

B. Figure 1 illustrates the unrevised use relation between the modules. Figure 2 illustrates the revised use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

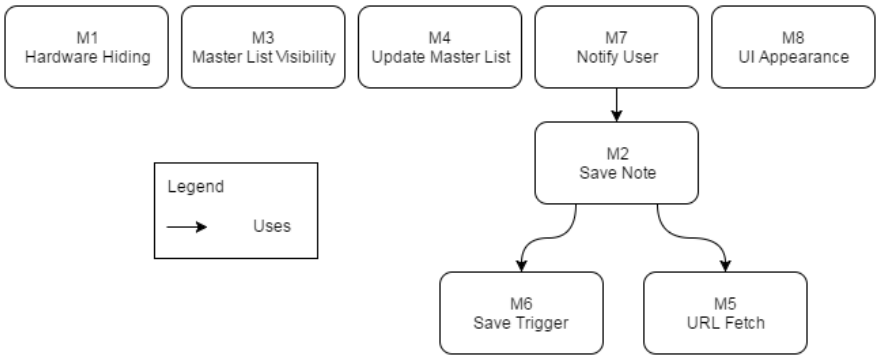

Figure 1: Unrevised use hierarchy among modules

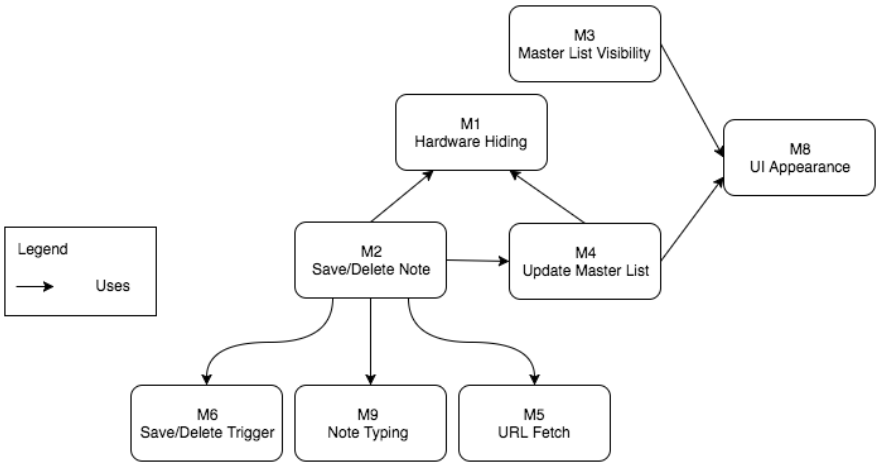

Figure 2: Revised use hierarchy among modules

References

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

David L. Parnas. Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press. ISBN none.

D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.