SE 3XA3: Software Requirements
Spann

Team 5
Christopher Stokes — stokescd
Varun Hooda — hoodav

November 13, 2016

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
Nov 10, 2016	1.0	Initial Changes
Nov 12, 2016	1.1	Changes and Module Hierarchy

1 Introduction

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: User interface styling (CSS).

AC2: User interface layout (CSS/JavaScript).

AC3: User Session/Authentication

AC4: Addition or Deletion of UI Components

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: Application platform (web browser)

UC2: Web front end framework

UC3: Web server and server framework

UC4: Database (SQL and PostgreSQL)

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

M2: API Controllers

M3: Python Console

M4: Python Console Manager

M5: Python Runners

M6: Repository Model

M7: Domain Models

M8: UI Screens

Note: Since this system utilizes libraries and frameworks to abstract away the low level implementation details of browser interaction, operating system interaction and networking hardware interaction, there is(are) no Hardware-Hiding Module(s).

Level 1	Level 2	Level 3
Hardware-Hiding Module		
Behaviour-Hiding Module	API Controllers	Console Controller
	–	Fiddle Controller
	–	File Controller
	–	Project Controller
	–	User Controller
	Python Console	–
	Python Console Manager	–
	UI Screens	App
	–	Full Screen Frame
	–	Project Frame
	–	Selection Frame
	–	Account
	–	Console
	–	Dock Screen Output
	–	Screen Fiddle
	–	Dialog Demo
	–	Dialog Menu
	–	Editor
	–	Home
	–	Login
	–	Login State Manager
	–	Project Dialog
	–	Project Dialog State Manager
	–	Project Edit

Table continues on next page

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

5.1 Hardware Hiding Modules

Secrets: Networking functionality and CSS/HTML/JavaScript rendering and execution.

Services: Networking serves as the backbone of the API, servicing web requests. The web browser provides a platform for rendering website content and execution JavaScript code.

Implemented By: ASP.NET, Web Browsers

Level 1	Level 2	Level 3
Software Decision Module	Domain Models	Python File Data Transfer Object
	–	Python Project Data Transfer Object
	–	User Data Transfer Object
	–	Base Data Transfer Object
	–	Message Data Model
	–	Python File Data Model
	–	Python Project Data Model
	–	User Data Model
	Repository Model	–
	Python Runners	Iron Python Location
	–	Python File
	–	Python Project
	–	Python Project Manager
	–	Python Runner
	–	Python Server Manager
	–	Python Tools
	–	Iron Python Manager
	–	Notification Stream Writer

Table 2: Module Hierarchy

5.2 Behaviour-Hiding Module

5.2.1 API Controllers

Secrets: API request handling and routing.

Services: Accept and respond to API calls from the network. Provide an interface with the rest of the server.

Implemented By: –

5.2.2 Console Controller

Secrets: API request handling and routing for the console.

Services: Handles API requests related to the python code execution on the console.

Implemented By: Spann

5.2.3 Fiddle Controller

Secrets: API request handling and routing for the fiddle.

Services: Handles API requests related to the fiddle code execution.

Implemented By: Spann

5.2.4 File Controller

Secrets: API request handling and routing for files.

Services: Handles API requests related to source code files.

Implemented By: Spann

5.2.5 Project Controller

Secrets: API request handling and routing for projects.

Services: Handles API requests related to projects.

Implemented By: Spann

5.2.6 User Controller

Secrets: API request handling and routing for users and user data.

Services: Handles API request related to users, sessions and user data.

Implemented By: Spann

5.2.7 Python Console

Secrets: Handles code execution for the client (front end).

Services: Takes in a string of code and uses standard library functions to execute the code with a python interpreter and returns result.

Implemented By: Spann

5.2.8 Python Console Manager

Secrets: Manges python consoles.

Services: Provides a support for managing multiple consoles.

Implemented By: Spann

5.2.9 UI Screens

Secrets: UI layout and functionality.

Services: Provides a way of dynamically generating the UI and managing all the functionality of the UI.

Implemented By: –

5.3 Software Decision Module

5.3.1 Domain Models

Secrets: Models the data for different objects used throughout the application.

Services: Provides a way to model objects and hold data.

Implemented By: –

5.3.2 Python File Data Transfer Object

Secrets: Model python files.

Services: A way to model python files that are transferred to the front end.

Implemented By: Spann

5.3.3 Python Project Data Transfer Object

Secrets: Model python projects.

Services: A way to model python projects that are transferred to the front end.

Implemented By: Spann

5.3.4 User Data Transfer Object

Secrets: Model users.

Services: A way to model user data that will be transferred to the front end.

Implemented By: Spann

5.3.5 Base Data Transfer Object

Secrets: Base for other data transfer objects to model off.

Services: A way to provide a base class for extending other data transfer object classes.

Implemented By: Spann.

5.3.6 Message Data Model

Secrets: Model message data.

Services: Provides a way to model messages and message data that will be transferred to the front end.

Implemented By: Spann

5.3.7 Python File Data Model

Secrets: Model python file data.

Services: Provides a way to model python file data that will be transferred to the front end.

Implemented By: Spann

5.3.8 Python Project Data Model

Secrets: Model python project data.

Services: Provides a way to model python project data that will be transferred to the front end.

Implemented By: Spann

5.3.9 User Data Model

Secrets: Model user data.

Services: Provides a way to model user data that will be transferred to the front end.

Implemented By: Spann

5.3.10 Repository Model

Secrets: Model repository and repository data.

Services: Provides a way to model repository that will be used by the application during runtime to handle different manager.

Implemented By: Spann

5.3.11 Python Runners

Secrets: Python functions for various python execution tasks.

Services: Provides python functions to aid in managing python execution.

Implemented By: –

5.3.12 Iron Python Location

Secrets: Iron Python Runtime location.

Services: Provides the interface to the Iron Python Runtime.

Implemented By: Spann

5.3.13 Python File

Secrets: Python file object.

Services: Provides a data structure and methods for python files.

Implemented By: Spann

5.3.14 Python Project

Secrets: Python project object.

Services: Provides a data structure and methods for python projects.

Implemented By: Spann

5.3.15 Python Project Manager

Secrets: Python project manager object.

Services: Provides a data structure and methods for python project manager.

Implemented By: Spann

5.3.16 Python Runner

Secrets: Python runner for executing python source code and whole projects.

Services: Provides an interface for launching python code or projects.

Implemented By: Spann

5.3.17 Python Server Manager

Secrets: Python server manager object.

Services: Provides functionality for managing python execution.

Implemented By: Spann

5.3.18 Python Tools

Secrets: Manages various python tools.

Services: Provides various python related tools for use by other components.

Implemented By: Spann

5.3.19 Iron Python Manager

Secrets: Manages the Iron Python runtime.

Services: Provides an interface to manage the Iron Python Runtime.

Implemented By: Spann

5.3.20 Notification Stream Writer

Secrets: Write a notification to a stream.

Services: Provides notification writing functionality when communicating with the front end.

Implemented By: Spann

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**
Mode	M2, M8
Editing	M2, M3, M5, M6, M7, M8
Editing Support	M3, M8
Refactoring	M3, M8
File Handling	M2, M5, M6, M7, M8
Code Execution	M2, M3, M4, M5
Shell Interpreter	M2, M3, M4, M5, M8
Networking	M1, M2, M8
Accounts	M2, M6, M7, M8
Account Management	M2, M6, M7, M8
Look and Feel	M8
Usability	M8
Performance	M1, M4, M8
Power Usage	M1
Security	M2, M6, M7, M8
Health and Safety	M8

Table 3: Trace Between Requirements and Modules

AC	**Modules**
AC1	M8
AC2	M8
AC3	M2, M6, M7
AC4	M8

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 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: 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.