Northeastern University

CSYE 7230 Software Engineering Fall 2024

Project 2 Phase 2

Posted: 10/24/2024
Due: 12/8/2024, 11:59pm
Questions? email me at a.alali@northeastern.edu

Swishy!

My Assistant Manager (Swishy!) is a Swiss-knife secure chat bot that assist me and keep me organized. It helps me be productive in my day-to-day activities. Swishy understands a specific language, called the Swish language. Swishy! is a vault, a note taker, and a task manager. A user of Swishy! would need to get authenticated to be able to use Swishy!

High-level System Features

1. Users can activate the app functionalities using the Swish language

2. App users would register using their full name, password, and email. Then be able to login anytime afterwards to be allowed to use the full app features. Customers can reset their user password or delete themselves/their users. The initial user is a root admin and its password is root. Root can list and delete any user. All passwords are encrypted once saved.

```
> This is Swishy! if you want to learn how to talk to me, type @LearnSwissh
> @register @user @name Abdulkareem Alali @key a.alali@northeastern.edu
@password abc123
> @login @user @name Abdulkareem Alali @key a.alali@northeastern.edu
@password abc123
>
abd.alali@gmail.com is logged in
> @reset @user @key a.alali@northeastern.edu @password abc123
> @delete @user @key a.alali@northeastern.edu
```

3. Users creates a key and password pairs, they get saved and retrieved

```
> This is Swishy! if you want to learn how to talk to me, type @LearnSwissh
```

```
> save @key abd.alali@gmail.com @password abc123
> Can you @get me the @password for my email @key abd.alali@gmail.com
>
password123
> can you @save the email @key abd.alali@gmail.com using @password abc1234567
> can you @get me the @password for this @key abd.alali@gmail.com
>
abc1234567
```

4. Users creates a note, saves and retrieves them, notes are dated and timed

```
> This is Swishy! if you want to learn how to talk to me, type @LearnSwissh
> would you please @get me all the @notes from @date 3/3/2024
>
9/20/2024 13:23:34 the book clean code architecture seems to a great book to read, the
book author of the book is the famous uncle Bob.
9/20/2024 14:45:34 This is a great article
agileinaflash.blogspot.com/2009/02/first.html, great find!
> @save @note a weekend article https://dannorth.net/introducing-bdd/,
important topic! Will be a fun read
```

5. Users creates a task, saves and retrieves them, tasks are dated and timed, and a reminder can be scheduled, which is an email sent on a specific date, with the task embedded in the body, and the subject is “Swishy! reminds”

```
> This is Swishy! if you want to learn how to talk to me, say @LearnSwissh
> @save this @task submit the time estimates for the portfolio reskin UI work by the
end of this week.  And @remind me about it on @datetime 9/27/2024 13:00:00
> @save this @task grade project 1 by the end of Tuesday @remind @datetime 9/27/2024
13:00:00
> @get me all the @tasks from @date 9/20/2024
>
9/20/2024 13:23:34 submit the time estimates for the portfolio reskin UI work by the
end of this week.
9/20/2024 14:45:34 grade project 1 by the end of Tuesday
```

6. If the user types `@LearnSwissh`, app will display a tutorial on how to speak and interact with the chat bot Swishy!

7. All data are saved into simple database management system, use SQLite.

GitHub

3

Use your GitHub account, and if you don't have one, make one, then:

1. Create a new repository and call it **Swishy**. Make it private, and add the team to it aalali, and your classmate GitHub account: cecillyaliu or snguyen1122
2. Create a branch, call it **swishy-phase3-feature**
3. Open a draft pull request, then commit your code as frequently as possible
4. Once finished, add us as reviewers (due date is at this point)
5. We will review, and add comments, and then we will approve
6. You merge into the main, delete the feature branch

You might find this reference and this reference helpful.

Construction & Transition Phases 4
Artifacts to Submit

From Elaboration to Construction to Transition:

Project 2 phase 1 covered the UP-inception phase, project 2 phase 2 covered the UP-elaboration phase, project 2 phase 3 to cover UP-Construction and UP-Transition phases. For this phase, proceed through the Construction and Transition phases and refine the artifacts. Larman's e/3 book as your main source for formatting, guidelines, context, concepts, and definitions **closely.**

A. Requirements

1. **Resubmit** the **Vision** (`vision.txt`) including Introduction, Positioning, Stakeholder Descriptions, Product Overview, and Summary of System Features. Checkout **III**.
2. **Use-Case Model**
   - **a. Resubmit** the **Fully-dressed Use Cases** from the inception phase (phase 1) following the naming format `usecase_01_USECASENAME.txt`, **III**.
   - **b. Resubmit** the **Fully-dressed Use Cases** from the elaboration phase (phase 2) following naming format `usecase_01_USECASENAME.txt`, **III**.
   - **c. Resubmit Use Case diagram** (`usecase_diagram_v1.png`), **III**.
   - **d. Resubmit System Sequence Diagrams** (SSDs) of the system events for each inception's fully dressed use case from **a**. Name your files as in the following, `ssd_USECASENAME_USECASENUMBER_v1.png`. **III**.
   - **e. Submit** the rest of the **System Sequence Diagrams** (SSDs) of system events for each Elaboration's fully dressed use case from **b**. Name your files as in the following `ssd_USECASENAME_USECASENUMBER_v1.png`. The events are mainly the calls (or messages) that your system (as a black box) would receive from the UI clients, the commanding top layer. Each event can be named as a function call, e.g. ***enterItem***`(itemID, quantity)`

3. **Resubmit** the **Supplementary Specification** including URPS+ which are Usability, Reliability, Performance, Supportability, + Constraints (Implementation, Interface, Operations, Packaging, Legal) (`supplementary_specification.txt`). **III**.
4. **Resubmit** the **Glossary**/Data Dictionary (`glossary.txt`). **III**.
5. **Resubmit** the **Business Rules** (`business_rules.txt`). **III**.

B. Business Modeling

1. Domain Model

a. **Resubmit a refined** **Domain Model** based on all fully dressed use cases from **A.2. a** and **A.2. b** (`domain_model_v2.png`). **III**.

C. Design

1. Design Model

Build the Design Model based on the Domain Model from **B.1.** and the fully dressed use cases from **A.2. a** and **A.2. b**.

a. **Resubmit a refined** **Software Architecture** UML package diagram (`package_diagram_v2.png`). A layered logical architecture representing your packages or namespaces, the other layers too, although we are implementing only the domain layer. **III**.

b. **Resubmit** the **Sequence Diagram** for each system event from the events in each SSD in **A.2. d**, name files as the following `sd_USECASENAME_USECASENUMBER_SYSTEMEVENTNAME_v1.png` for example `sd_process_order_01_enterItem_v1.png`. In case of any updates to these Sequence diagrams, check out the rename note **III**.

c. **Submit** the rest of the **Sequence Diagrams** for each system event from the new events in each of the new SSDs in **A.2. d**, name files as the

following `sd_USECASENAME_USECASENUMBER_SYSTEMEVENTNAME_v1.png` for example `sd_process_order_01_ enterItem_v1.png`.

d. **Resubmit a refined** **Class Diagram** (`class_diagram_v2.png`) that represents your software domain layer.

D. Implementation

1. Implementation Model

Build the Implementation Model based on the Design Model from **C.1.** (Class diagram **C.1. d**, Sequence Diagrams **C.1. b** and **C.1. b**, and logical architecture **C.1. a**) and the fully dressed use cases from **A.2. a** and **A.2. b**, and the SSDs from **A.2. d** and **A.2. e**.

a. **Resubmit a refined** **Java Source Code Implementation** for your **swishy!** service domain layer. Addressed features are **1—7**.

b. **Resubmit a refined** **Testing Client**. Build a mockup **UI** layer client code to execute and test the domain layer. The source of the testing code can be inspired from the fully-dressed use cases and scenarios from **A.2. a** and **A.2. b**. Create multiple users:

- Swishy! **Customer** with a username **APE** (uppercase) and password **NIRVANA** (uppercase)
- Another **Customer** user with a username **ZEPPELIN** (uppercase) and password **BLACKDOG** (uppercase)

This client operates in two ways:

1. **User Scenarios**. The user can interact with the system from the command line. The testing-client would allow the user to interact with the system via asking questions, displaying menus, options, and display responses, interactions, and transactions.
2. **Input Scenarios**. A mockup input data from TXT files. A simulation to a user using the system by providing the input that the user would normally provide via command line. The input data files

represent testing scenarios inspired from your use case scenarios from **A.2. a** and **A.2. b**. Let the system pause at every transaction, so the user can observe the interactions. Place data input files under a `data` folder.

c. Addressed features are **1—7**.

Resubmit a refined Logging Service. Build a logging service that streams out all detailed transactions to TXT files (`log_DATETIME.txt`) and to the command line. Log files have a `DATETIME` which is a unique string of the system's time and date used within the name of every new log file every time a user starts a new interaction with the system. Let the system place these files under the `log` folder.

E. Swishy! Report

1. Submit a report (`swishy_report_YOURFULLNAME_NEUUSERNAME.pdf`) for the final version of your project artifacts. Report structure is as follows:

a. A **preface** page+ that introduces your system to the reader:

- Project Name
- Project Description
- Project Sponsor
- Project Author
- Project Submission Date
- Project Successes
- Project Unexpected Events
- Project Lessons Learned
- Project Performance

b. Number your pages, use auto numbering options

c. This report is mainly a combination of all your artifacts, TXT, Source C++, PNGs into one document.

d. Follow the **exact** structure as in this section, the **Artifacts to Submit** section, disciplines and artifacts under **A**, **B**, **C**, and **D**.

e. Styling is not much of an importance; basic styling would be sufficient.

f. Table of contents for the sections and subsections, this can be auto generated using an MS Word or similar software[1][2]

g. A table of figures, this can be auto generated using MS Word or similar software[3]

h. A table of tables, this can be auto generated using a MS Word or similar software[4]

i. You can create a references section at the end of the report to cite any external resources, if any were used

j. You can add any external materials, pictures, etc. to support your report, if needed.

[1] Create a Table of Contents and Table of Figures - Microsoft Word
[2] Create a Table of Contents in Word, Table of Figures and Table of Tables
[3] Create a Table of Figures
[4] Adding a table of contents, table of figures, or table of tables

Notes

I. For any text (`TXT`) or source (`Java`) files, use a header (example below):

```
/*----------------------------------------------------------------

        System Name: Swishy!
        Artifact Name: xxx.java
        Create Date: Jan 30, 2122
        Author: Abdulkareem Alali, a.alali@northeastern.edu
        Version: 3.0

----------------------------------------------------------------*/
```

II. For PNG files, include the version number into the file name.

III. Rename/Update Note: You need to resubmit all the files that you have submitted for the inception phase. *If any changes have been made* to any of the phase 1 files, you need to **update** the file header's version number (e.g. `version 1.0` to `2.0`). For non-text files, `PNG`s, **rename** and include a new version number into the file name (e.g. `usecase_diagram_v1.png` to `usecase_diagram_v2.png`). *If a file has not been changed, then no updating to the headers or renaming is required.*

IV. Use `draw.io`, `PlantUML`, `Mermaid`, `Lucidcharts` or `Microsoft Visio` for UML diagraming or any UML tools you choose. Don't use hand drawing or something as simple as Microsoft Paint.

V. All diagrams must be submitted to a PNG type file. For documents, use TXT type files.

VI. Commit all your artifacts into your project repository, each should be placed in the right folder as in the directory tree structure, see **XII**.

VII. The instructor is a manager at Swishy! co, *he is not*, but he may give feedback, change requirement, add or remove features at any moment in time.

VIII. Code compiles and runs with jdk.java.net/22

IX. Write unit tests using junit.org/junit5, with 90%+ code coverage, and make sure I can understand your code by reading only your unit tests!

X. A `README.txt` file that explains how to compile, run your app, and how to run the tests

XI. Use LLMs to help you understand, finish tasks, and write better code faster, Codium in vscode is free.

XII.

Your directory tree structure **should** (exact names and structure) look like:


```
.
|-- projects
|   |-- project2
|   |   |-- phase1
|   |   |-- phase2
|   |   |-- phase3
|   |   |   |-- swishy
|   |   |   |   |-- Requirements
|   |   |   |   |   |-- vision.txt
|   |   |   |   |   |-- UseCaseModel
|   |   |   |   |   |   |-- usecase_diagram_*.png
|   |   |   |   |   |   |-- usecase_01_USECASENAME.txt
|   |   |   |   |   |   |-- usecase_02_USECASENAME.txt
|   |   |   |   |   |   |-- ...
|   |   |   |   |   |   |-- ssd_USECASENAME_01_v1.png
|   |   |   |   |   |   |-- ssd_USECASENAME_02_v1.png
|   |   |   |   |   |   |-- ...
|   |   |   |   |   |-- supplementary_specification.txt
|   |   |   |   |   |-- glossary.txt
|   |   |   |   |   |-- business_rules.txt
|   |   |   |   |-- BusinessModeling
|   |   |   |   |   |-- DomainModel
|   |   |   |   |   |   |-- domain_model_v1.png
|   |   |   |   |-- Design
|   |   |   |   |   |-- DesignModel
|   |   |   |   |   |   |-- package_diagram_v1.png
|   |   |   |   |   |   |-- class_diagram_v1.png
|   |   |   |   |   |   |-- sd_USECASENAME_01_SYSTEMEVENTNAME_v1.png
|   |   |   |   |   |   |-- sd_USECASENAME_01_SYSTEMEVENTNAME_v1.png
|   |   |   |   |   |   |-- ...
|   |   |   |   |-- Implementation
|   |   |   |   |   |-- ImplementationModel
|   |   |   |   |   |   |-- src
|   |   |   |   |   |   |   |-- *.java
|   |   |   |   |   |   |-- data
|   |   |   |   |   |   |-- log
|   |   |   |-- Report
|   |   |   |   |-- swishy_report_YOURFULLNAME_NEUUSERNAME.pdf
```

Grading

The grade will be broken down as follows:

Point	Mark
A	5/100
B	5/100
C	15/100
D	65/100
E	10/100

Use Cases (UC), Features (F), and Actors Table (Instructor's Vision/Expectation)[5] [6]

Primary Actors	Goal	UC#	F#	To Implement	Most Critical
App User	Manage User	**1**		**Yes**	*
	Manage Tasks	**2**		**Yes**	
	User Authentication		**1**	**Yes**	
	Manage Notes	**3**		**Yes**	
	Manage Vault	**4**		**Yes**	*
	Logging		**2**	**Yes**	
Root User	Manage Root	**5**		**Yes**	
	User Authentication		**3**	**Yes**	
	Manage Users	**6**		**Yes**	
	Logging		**4**	**Yes**	

[5] This can be enhanced, and it's open to evolve. There is **no** need to commit and follow this table, but feel free to do so. This is an **example** that breaks down **Swishy**! to use cases and features

[6] Non-primary actors have not been included