

**REPORT 2 – SOFTWARE PROJECT MANAGEMENT PLAN**

Contents

[2.1. Purpose 1](#_Toc42278834)

[2.2. Project organization 1](#_Toc42278835)

[2.2.1. Software process model 1](#_Toc42278836)

[2.2.2. Organization structure 2](#_Toc42278837)

[2.2.3. Roles and Responsibilities 3](#_Toc42278838)

[2.2.4. Project team member 4](#_Toc42278839)

[2.2.5. Tools and Techniques 4](#_Toc42278840)

[2.3. Project management plan 4](#_Toc42278841)

[2.3.1. Tasks schedule 4](#_Toc42278842)

[2.3.2. Meeting minutes 5](#_Toc42278843)

[2.3.3. Coding conventions 6](#_Toc42278844)

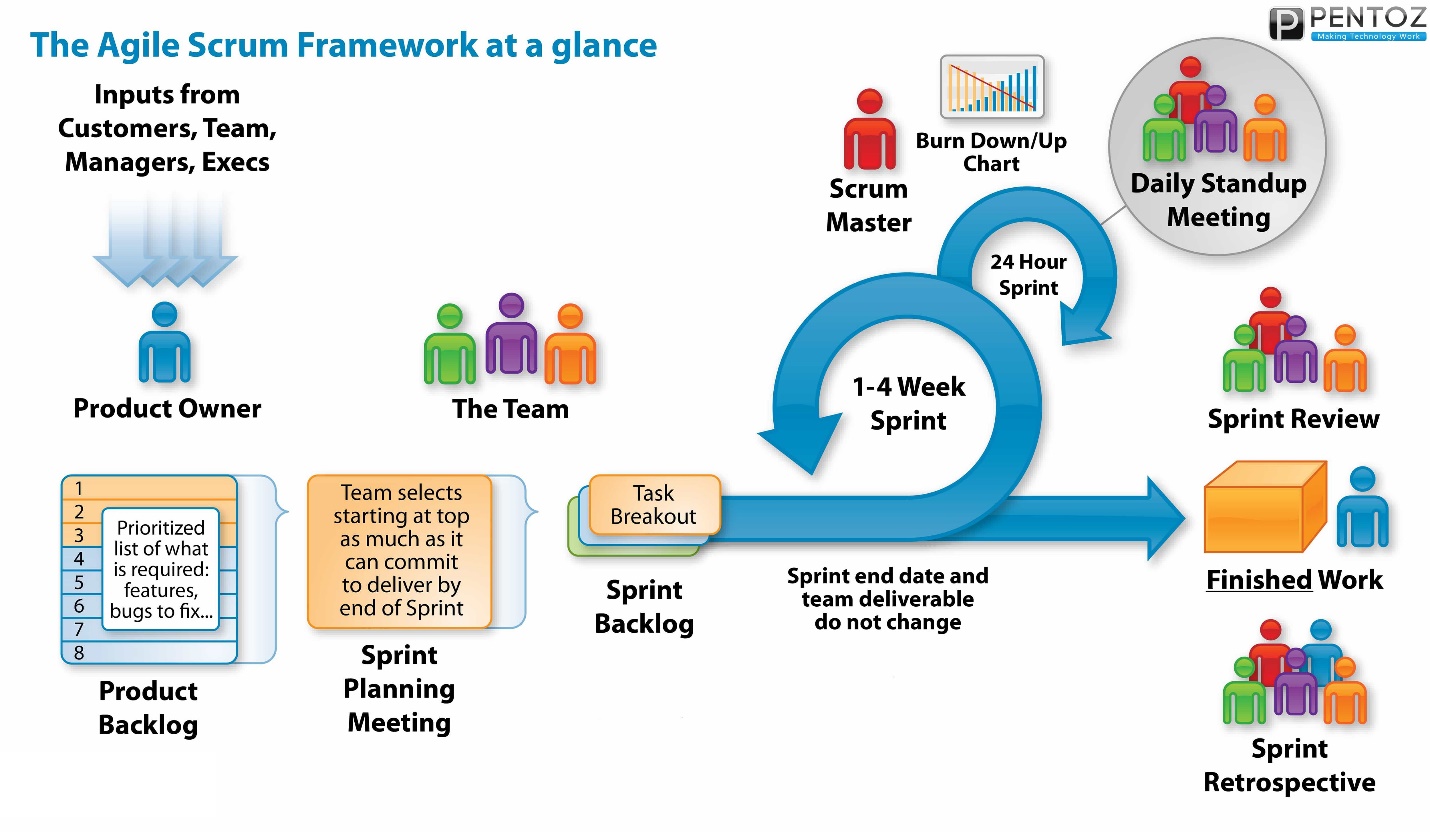
[2.3.4. Risks management 8](#_Toc42278845)

## 2.1. Purpose

This chapter describes functional and non-functional requirements. It gives the user an overview of the requirements to be completed in the project, the application cycle to complete those tasks, and how to complete the tasks in the project. Project members rely on this plan to carry out their duties.

## 2.2. Project organization

### 2.2.1. Software process model

Table 2.2.1.1 SCRUM process

We choose SCRUM process - one of the most widely-used Agile methodologies to develop our project because it is a method which allows us deliver each part of the whole project in a short time till all parts are deployed completely. SCRUM process concludes of many sprints, each sprint will have specific start date, end date and a product, prioritized tasks will be arranged in product backlog. When a sprint start team selects starting at top of product backlog as much as it can commit by end of sprint, a sprint will be closed on end date no matter that all tasks are completed or not. At the head of a sprint during a sprint, daily meeting will take place every day within about 15 minutes to review tasks done, problem occurred, discuss about new tasks. A sprint usually lasts in 1 – 4 weeks. Problem or expericence will be noted to improve next sprint.

## 2.2.2. Organization structure

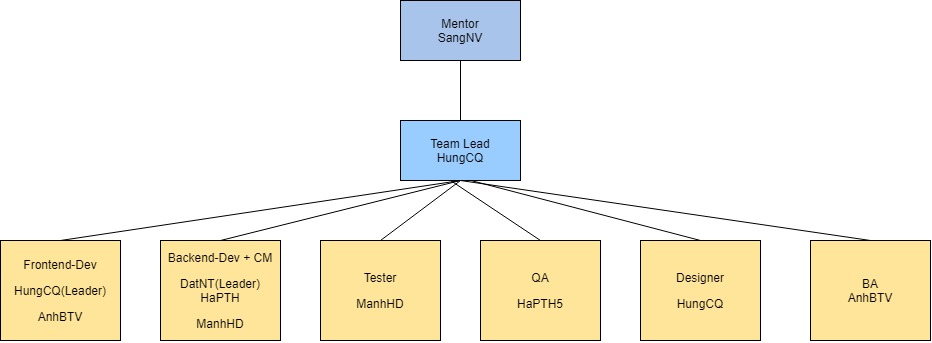


Image 2.2.2.1 Organization structure

## 2.2.3. Roles and Responsibilities

|  |  |
| --- | --- |
| Role | Responsibility |
| Leader | Get requirement specification, create scope plan, time plan, cost plan, resource plan, risk and issue plan, project management completed on schedule. Ensure the project is operating according to the plan. Communicate with stakeholder. Update plan and manage risk. |
| Backend-Dev | Implementing the use cases related to the backend part of the system. Fix backend bug. Support QA complete document. |
| Frontend-Dev | Implementing the use cases related to the frontend part of the system. Fix frontend bug. Update frontend source code if design is updated. Support QA complete document. |
| Tester | It is the person who checks whether the source code for the backend and the front end is correct as in the design. Check to see if the bug is properly fixed. |
| CM | Understanding and setting up the environment for the project, creating environment installation documents for the whole team. Learn about environment-related bugs and configuration during product creation. |
| QA | The person who ensures the project documents is complete, the processes are done fully and seriously. |
| Designer | Create UI design |
| BA | The person who analize bussiness |

## 2.2.4. Project team member

|  |  |
| --- | --- |
| Member | Role |
| Chế Quang Hưng | Leader, designer, frontend dev(lead) |
| Phạm Thị Hồng Hà | QA, backend dev |
| Nguyễn Thành Đạt | Backend dev(lead) |
| Bùi Trần Việt Anh | Frontend dev, BA |
| Hồ Đức Mạnh | Tester, backend |

Table 2.2.4.1 Project team member

## 2.2.5. Tools and Techniques

|  |  |
| --- | --- |
| Programming languages | HTML, JavaScript, css, java |
| Web development techniques | AJAX |
| Framework | Vue.js, Java spring |
| DBMS | mySQL |
| IDEs/Editors | Spring tool |
| UML tools | Astah |
| Version Control | Github |
| Deployment server | Google cloud platform |
| Project management tool | Proofhub |
| Development process | Agile SCRUM |
| Platform | Docker |
| API | Rest |

# 2.3. Project management plan

## 2.3.1. Tasks schedule

|  |  |
| --- | --- |
| **Objective** | **Success criteria** |
| Kick off meeting |  |
| Planning | Determine project scope, resource, cost, time, risk/issue |
| Analysis | * Create usecase diagram for module LOGIN/LOGOUT/HOME/REGISTER/POST/VIEW PROFILE/DASHBOARD FOR MEDIATION/ SHORT-TERM AUCTIONS/ LONG -TERM AUCTION/POST PRODUCT/ ADD MONEY TO WALLET * Create Software Requirement Specification * Perform Requirement validation * Perform Requirement Management |
| Design | * Design interface for modules above |
| Complete design with customer | * Response to design of all modules |
| Review design with customer | * Customer accepts design for all modules |
| Execute code frontend for each module | * Design for all module are implemented |
| Execute code backend for each module | * Function |
| Testing | * Perform unit testing * Perform integration testing * Perform system testing * Perform acception testing |
| Addition module | * Receive and validate identity card image |

## 2.3.2. Meeting minutes

|  |  |  |  |
| --- | --- | --- | --- |
| **Meeting/Project Name:** | OBOS |  |  |
| **Date of Meeting:** | 10/4/2020 | Time: (Type) | 1 hours (Face-to-face) |
| **Meeting Called by:** | HungCQ | Location: | Google meet |
| **Note Taker:** | HaPTH5 | Time Keeper: | HungCQ |
| **1. Meeting Objective:** | | | |
|  | | | |
|
| **2. Attendance** | | | |
| **Name** | **Roles** | **Email** | **Phone** |
| Chế Quang Hưng | Leader, designer | hungcqse05930@fpt.edu.vn | 0935435973 / 0854066333 |
| Phạm Thị Hồng Hà | QA, backend dev | phamhongha.innerpiece@gmail.com | 0963514484 |
| Nguyễn Thành Đạt | Backend dev | datntse06039@fpt.edu.vn | 0943379203 |
| Bùi Trần Việt Anh | Frontend dev | anhbtvse06022@fpt.edu.vn | 0936281998 |
| Hồ Đức Mạnh | Tester, backend | manhhdse05476@fpt.edu.vn | 0326188282 |
| **3. Done task:** | | | |
|  | | | |
|
| **4. New task** | | | |
|  | | | |
|
|
| **5. Risk & Difficulty** | | | |
|  | | | |

2.3.2.1 Meeting report template

## 2.3.3. Coding conventions

**Capitalization rules**

Pascal case

The first letters of each word are capitalized.

The remaining letters are lowercase.

For example: MyProvider, StringBuilder.

Camel case

Similar to Pascal case but the first letter of the first word lowercase.Example: myProvider, stringBuilder

**Name the package**

Packages are often named the same as naming folders on disk which means they will start with a large name to a smaller range.

Characters in package identifiers are lowercase letters and numbers.

Usually we will name the package with the following scope and order:

* Organization name, domain name
* Company name
* Name of project
* Module name (later sub-modules (if any))
* Module function name

For example:

* My domain name is stackjava.com
* The project name is Demo
* The project has two modules, demo1 and demo2
* In the demo1 module, there is a class called Demo.java

Then we will declare the package in Demo.java class is package stackjava.com.demo.demo1

**Name the class, interface**

* Classes should be nouns and interfaces should be verbs or adjectives.
* Avoid using names that are identical to the predefined data types (such as Number, String, Float ...)
* Name the name "camelCase", ie the first letter of each uppercase letter, the next letters are lowercase (for example: DemoJava.java, DemoHelloWorld.java ...)

**Name the method**

* Method names are usually verbs.
* Name it "camelCase" but the first letter is lowercase.

For example: setAge, isTurnOn, getAge ...

**Name the variable**

* Name it "camelCase" but the first letter is lowercase.
* Use memorable words.
* Avoid using only 1-character identifiers.
* The identifier consists of 1 character usually used only as a temporary variable, for example i, j, k are used as temporary variables for numeric types; c, d, e are often used as temporary variables with type characters

For example: dateOfBirth, age ...

Name the constant

Constants are accompanied by static and final keywords when declared.

* Usually nouns.
* Constant names should use all capital letters and be separated by underscores.

For example: MIN\_HEIGHT, MAX\_WIDTH

**Static variable, enum**

All words are capitalized and separated by underscores (\_).

**Final variable**

For global final variables: naming is the same as the static variable. All words are capitalized and separated by underscores (\_) For local fianl variables: the variable names are the same as regular variables.

With 1 effective method

When a code appears in many places in the program, we combine it into a method: Save maintenance time, fix errors.When in a method with complex processing code, we should separate complex code that makes it into a separate method: Easy to monitor, debug. Declare the input parameters just enough, avoid the parameter declaration but do not use. Each method only executes one Size of a method: Many experiments have shown that a method with about 50 to 150 lines of code is reasonable. (Excerpt from Steve McConnell, Chapter 7.4 - Code Complete, Second Edition. 2004).

**Comment**

Do not write comments that just repeat code, comment redundantly. Some problems encountered when the comment is not good: Comments only describe as looping, not providing any additional information to the reader. Making the code longer. Readers spend more time reading. fussy; As simple as possible, when using multiple endline comments on consecutive lines of code, the comments must be aligned. Avoid writing code and then comment. Do not touch anywhere, just comment when you feel that your code is too complicated.

**Import libraries used**

Only import the necessary libraries used. Do not use import all.Example: use import java.util.List; instead of importing java.util. \*;

*Reference source standards*

[Google Java Style Guide](https://translate.google.com/translate?hl=vi&prev=_t&sl=vi&tl=en&u=https://google.github.io/styleguide/javaguide.html)

[Java\_standards\_v1.0\_ (NEAF)](https://translate.google.com/translate?hl=vi&prev=_t&sl=vi&tl=en&u=https://gpcoder.com/wp-content/uploads/2017/10/Java_standards_v1.0_NEAF.pdf)

[JavaCodingConvention\_Sun\_1997](https://translate.google.com/translate?hl=vi&prev=_t&sl=vi&tl=en&u=https://gpcoder.com/wp-content/uploads/2017/10/JavaCodingConvention_Sun_1997.pdf)

## 2.3.4. Risks management

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No** | **Risk** | **Mitigation** | **Contingency** | **Impact** | **Posibility** |
| R1 | Someone left the project while it was operating. | Regularly reconfirm the working schedule of project members. | * PM knows the specifics of the members' work to request additional resources if one of them leaves the project. PM can also be a temporary replacement for that member. * Hold daily meetings in order that all members know other’s work and can have ability to replace his work. | low | high |
| R2 | When launched, the application was not used by many patients as expected | Find customer at the initiating stage and keep inform with them. | Update the software according to feedback from the customer, who has tried it, launch the marketing campaign | Medium | High |
| R3 | Time to deliver products is delayed | Commit to release the product in phases on time. Over time or Over night promptly. Avoid affecting the execution time of different phases | Request extend delivery time | Medium | Medium |
| R4 | Conflicts occurred among project members | Observe, find out the obvious reason, resolve a small conflict before it becomes a major conflict in the project. | Replace member cause project conflict. | Low | Low |
| R5 | Wrong technology selection | Create document and show it to instructor | Usually recheck the tecknology used with instructor | Medium | Medium |
| R6 | Lose database | Create a backup version and update every 2 days, commit to githup |  | High | Low |
| R7 | COVID outbreak again |  | Keep meeting and discussing via online tools such as slack, facebook, google meet. | Medium | Medium |

2.3.4.1 Management risk

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Probability** | High | R1 | R2 |  |
| Medium |  | R3 R5 R7 |  |
| Low | R4 |  | R6 |
|  | | Low | Medium | High |
| **Impact** | | |

2.3.4.2 Impact of risk