ISYS2102

Software Engineering 2

DevFortress Game Development

Technical Report document

Lecturer: Kevin Jackson

Group: Work Insurance

Nguyen Thanh Luan – s3312335

Le Hoang Hai – s3298775

Vu Xuan Bach – s3298809

Nguyen Ba Dao – s3296796

Date: 17 December, 2012

# Abstract

This technical report outlines the design of DevFortress game and relevant issues encountered during development process. Different design patterns and Scrum approach used in this project will be discussed thoroughly. This also includes changes made to the specification and justification for the changes, additional constraints that are not listed in the specification and technology stack and tools chosen and justification.

Table of Contents

[Abstract 2](#_Toc344322872)

[Introduction 4](#_Toc344322873)

[Design approaches 4](#_Toc344322874)

[Singleton pattern 4](#_Toc344322875)

[Factory pattern 4](#_Toc344322876)

[Mediator pattern 5](#_Toc344322877)

[MVC pattern 6](#_Toc344322878)

[Strategy pattern 6](#_Toc344322879)

[Requirements 7](#_Toc344322880)

[Changes specifications and justifications 7](#_Toc344322881)

[Additional constraints 7](#_Toc344322882)

[Management 7](#_Toc344322883)

[Teamwork 7](#_Toc344322884)

[Methodology approaches 7](#_Toc344322885)

[Scrum 7](#_Toc344322886)

[Is R.U.P suitable? 7](#_Toc344322887)

[Technologies justification 8](#_Toc344322888)

[If we have more time 8](#_Toc344322889)

[References 8](#_Toc344322890)

[Appendices 9](#_Toc344322891)

[Appendix A: User Interface mockups and designs 9](#_Toc344322892)

# Introduction

DevFortress is a simulation game about studio management life cycle. In this game, player will take the director role of a software development studio. They are responsible for managing a team of developers, receive projects and buy project related items.

The purpose of this document is to provide a comprehensive architectural overview of the DevFortress game and clearly depict different aspects of the system. It is intended to convey the significant architectural decisions which our team has made during the development process. Various problems, solutions, decisions such as design, diagram, process approach and technology will also be covered.

# Design approaches

In this project, we have applied many design patterns including Singleton, Factory, Mediator, Strategy, and MVC.

## Singleton pattern

DevModel represents all the logical parts of the system (the game engine) and it is used by many controller stubs. As a result, we have two problems including: maintaining consistency throughout the game and providing a global access point. Therefore, Singleton pattern is applied in this case. Even though Singleton pattern is not suitable with multiple threads handling, by using MVC, the controller is the only one that can interact with model as it is put in one thread.

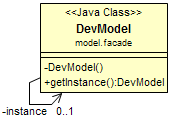


Figure 1. Singleton Pattern for DevModel

## Factory pattern

The Factory pattern is used to create Skill based on name of the skill. Each skill will have different implementations such as cost for level upgrade, function point, random effects to game. By implementing Factory, client will not need to know how the skills are implemented.

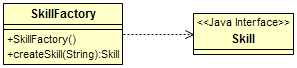


Figure 2. Skill Factory

The Factory pattern is also used in implementing GUI components such as Button. For example, in our game, we created a ButtonFactory class to control how user-interface-related button should look. As a result, this makes it easier to switch between looks and feels without breaking the code.

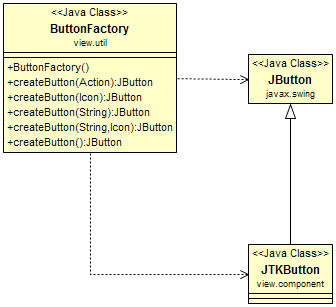


Figure 3. Button Factory

JTKButton is a customized button created for the game look and feel.

## Mediator pattern

Mediator pattern is mainly applied for view and controller. Mediator plays as a central control point encapsulating all interaction between view and controller components. It promotes loose coupling by keeping different classes from referring to each other explicitly, lets us vary their interaction independently. As a result, each component in our game is able to broadcast their own message and receive messages from others without requiring connections to those classes.

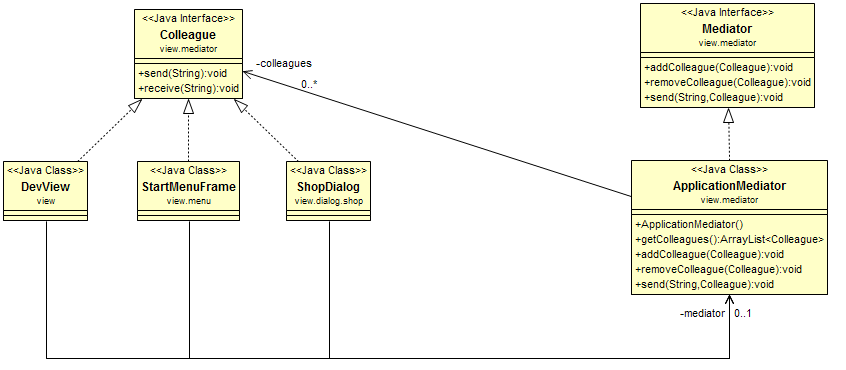


Figure 4. Mediator in action

## MVC pattern

The MVC pattern is used to separate the game engine with game UI, making the code clearer and easier to maintain. In this game the applied version of MVC is M-VC, which is described below:

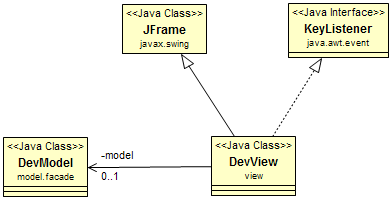


Figure 5. M-VC pattern

The M-VC is to have Model separated with the View-Controller. The game display will act as the View: renders out user interfaces, and also the Controller: handles events and calls model to process data and return results. This approach helps lessen the numbers of class (instead of having M-V-C) and easier to trace and debug: only one place to go when needed.

## Strategy pattern

The Strategy pattern is used to implement the difficulty level in the game. As the game has three levels (easy, normal, hard), we also have three classes to represent them. All three difficulty levels have to implement different algorithm: calculating randomly events occurred, starting money, etc.

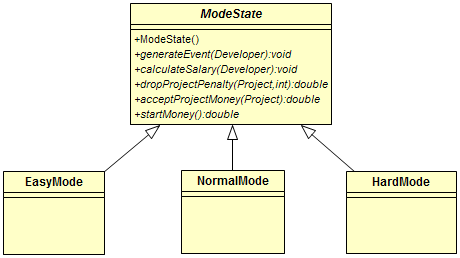


Figure 6. Difficulty levels with Strategy pattern

This is where Strategy pattern is needed: encapsulate what varies so that the difficulty levels are interchangeable. Thus the client do not need to know about the logic behind each level.

# Requirements

## Changes specifications and justifications

## Additional constraints

# Management

## Teamwork

## Methodology approaches

### Scrum

We believe Scrum methodology is suitable for the project. DevFortress is a quite complex game and it is necessary to deliver a shippable game in a short amount of time. As a result, Scrum is the correct process to accomplish this as we can iteratively develop our game, dividing the process into many Sprints. Through this way, after each Sprint, we are able to release an acceptable product. Another reason is the Scrum meetings, especially daily meetings, help our team members to have more opportunities to communicate, share experience and discuss the project thoroughly. This helps the team make essential changes and decisions to the game effectively. As a result, the flexibility of adding, removing functionalities of the game and changing Scrum rules is significant.

### Is R.U.P suitable?

Our team decide not to apply R.U.P in this project for many reasons:

Firstly, R.U.P requires many works in researching markets and analyzing requirements. This means a huge number of planning and business modeling and designing are required. Moreover, testing phase occurs at the end of project development, which causes difficulty in solving problems when bugs are found. Also, the requirement of the game is not fixed, any functionalities can be added or removed. As a consequence, changing a function requires huge amount of works in R.U.P. Finally, the time required to develop this game is limited. Therefore, RUP process is not suitable.

# Technologies justification

# If we have more time

# References

# Appendices

## Appendix A: User Interface mockups and designs

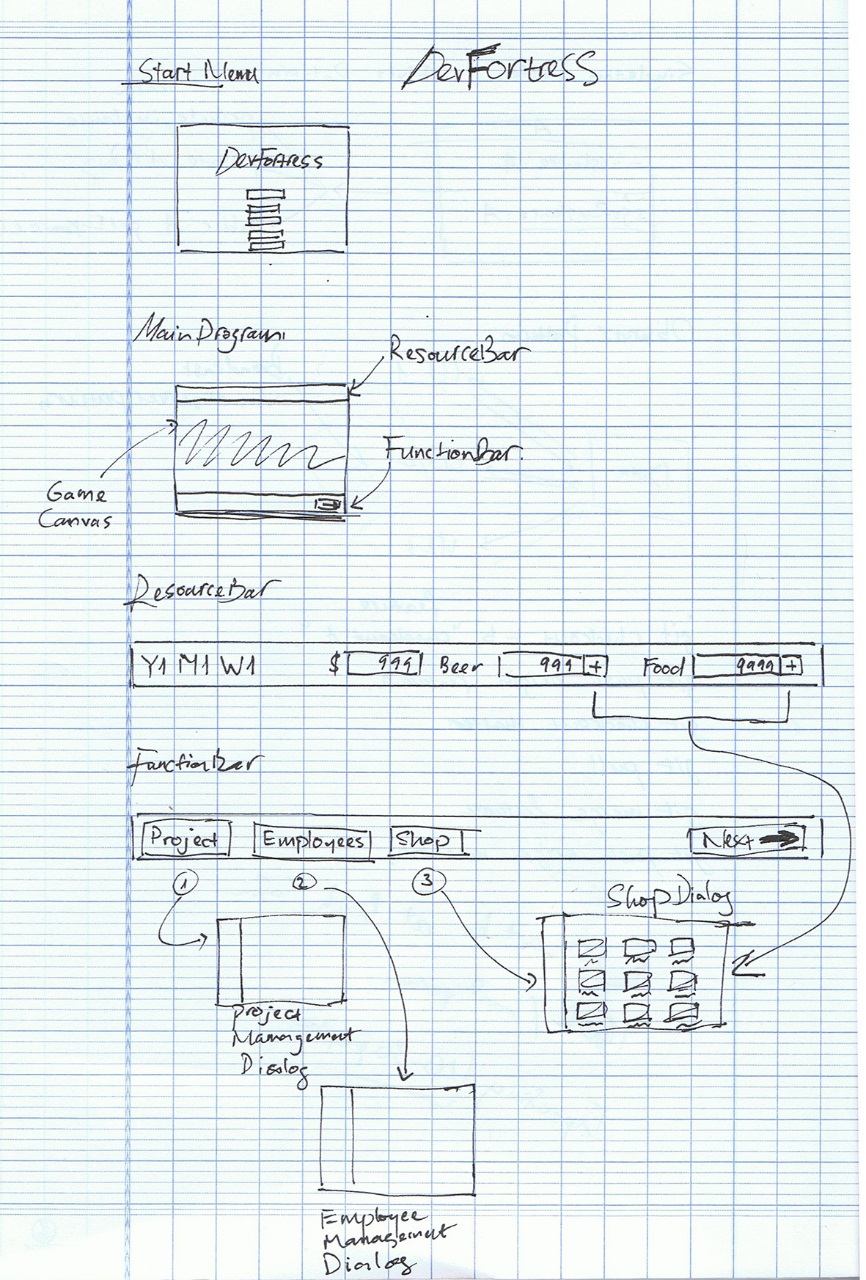


Figure 7. Overall design of the game

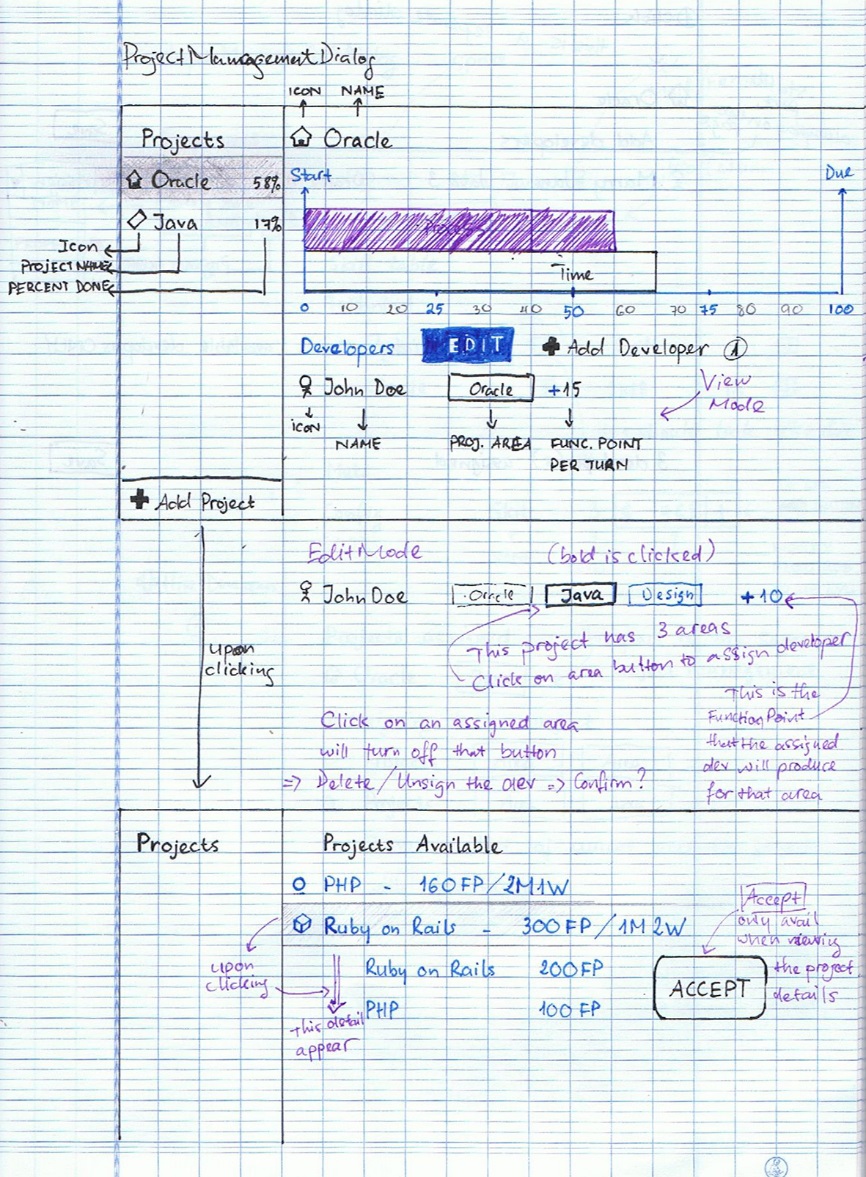


Figure 8. Project Management designs

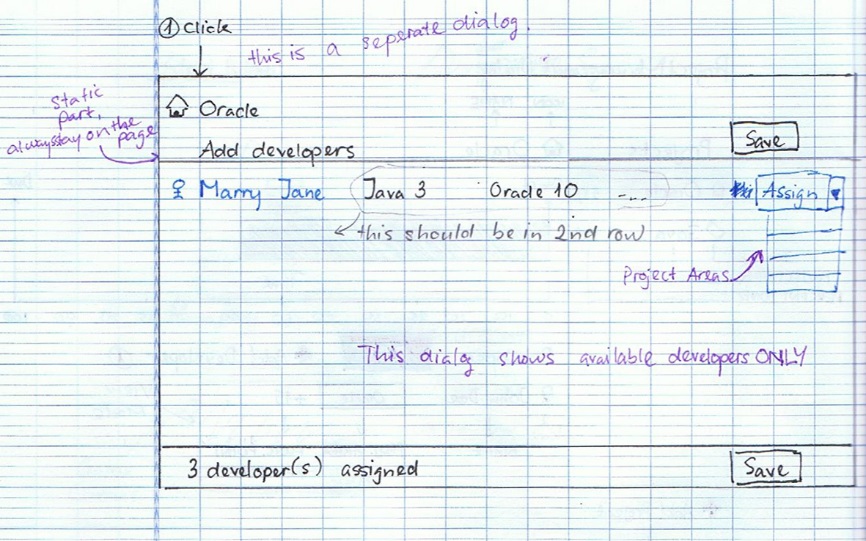


Figure 9. Assigning developers to a project

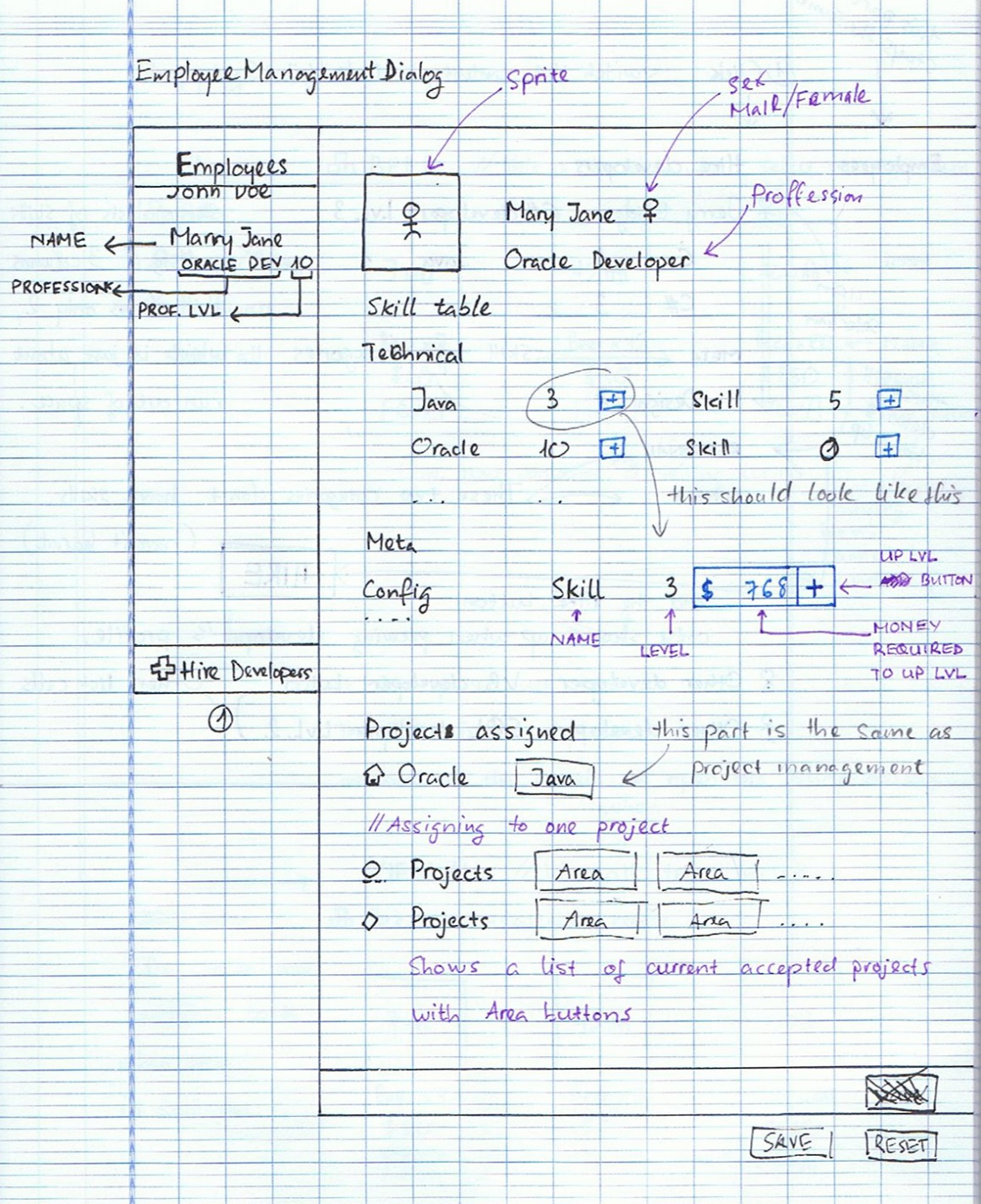


Figure 10. Managing employees mockup

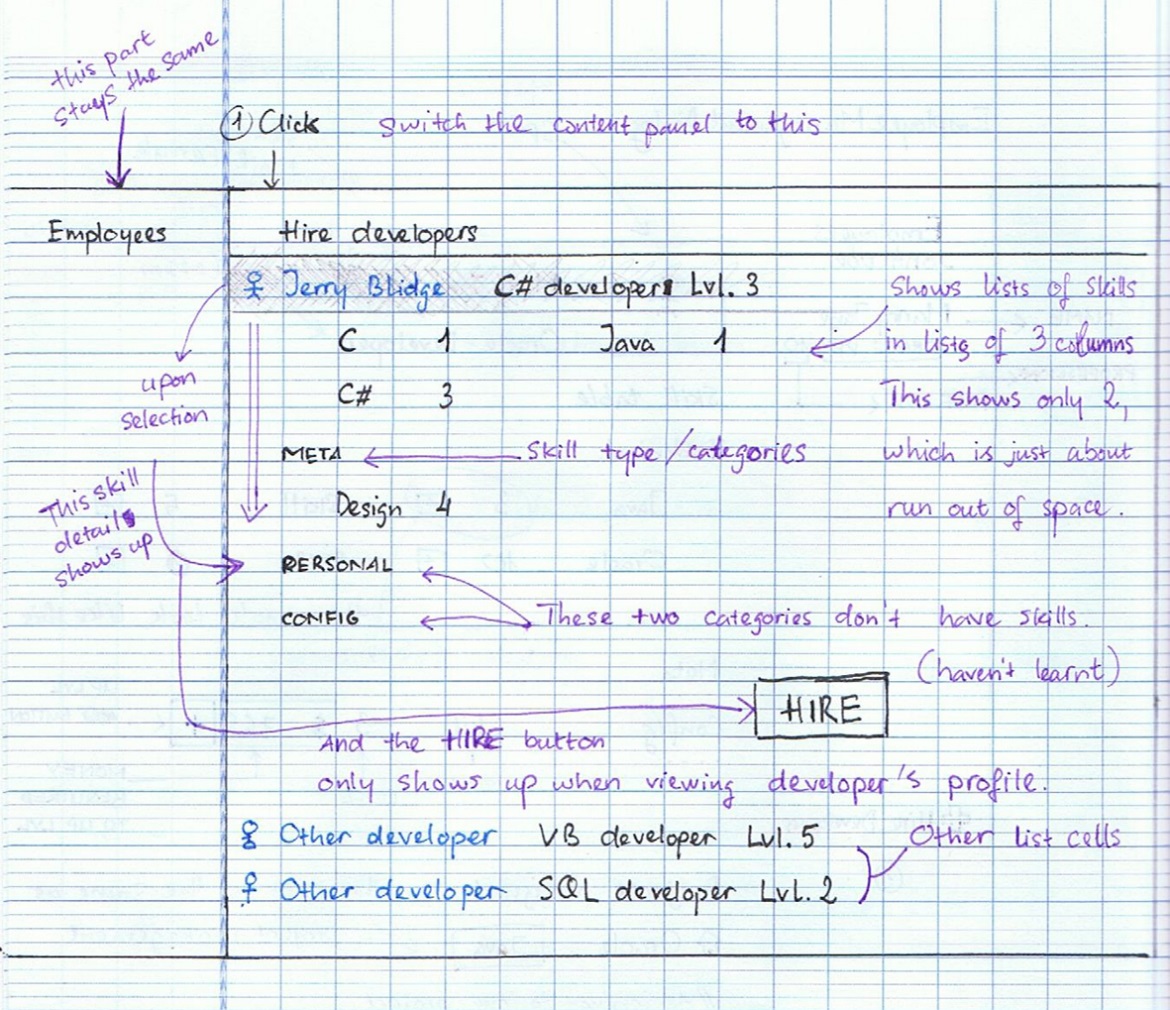


Figure 11. Hiring new developers

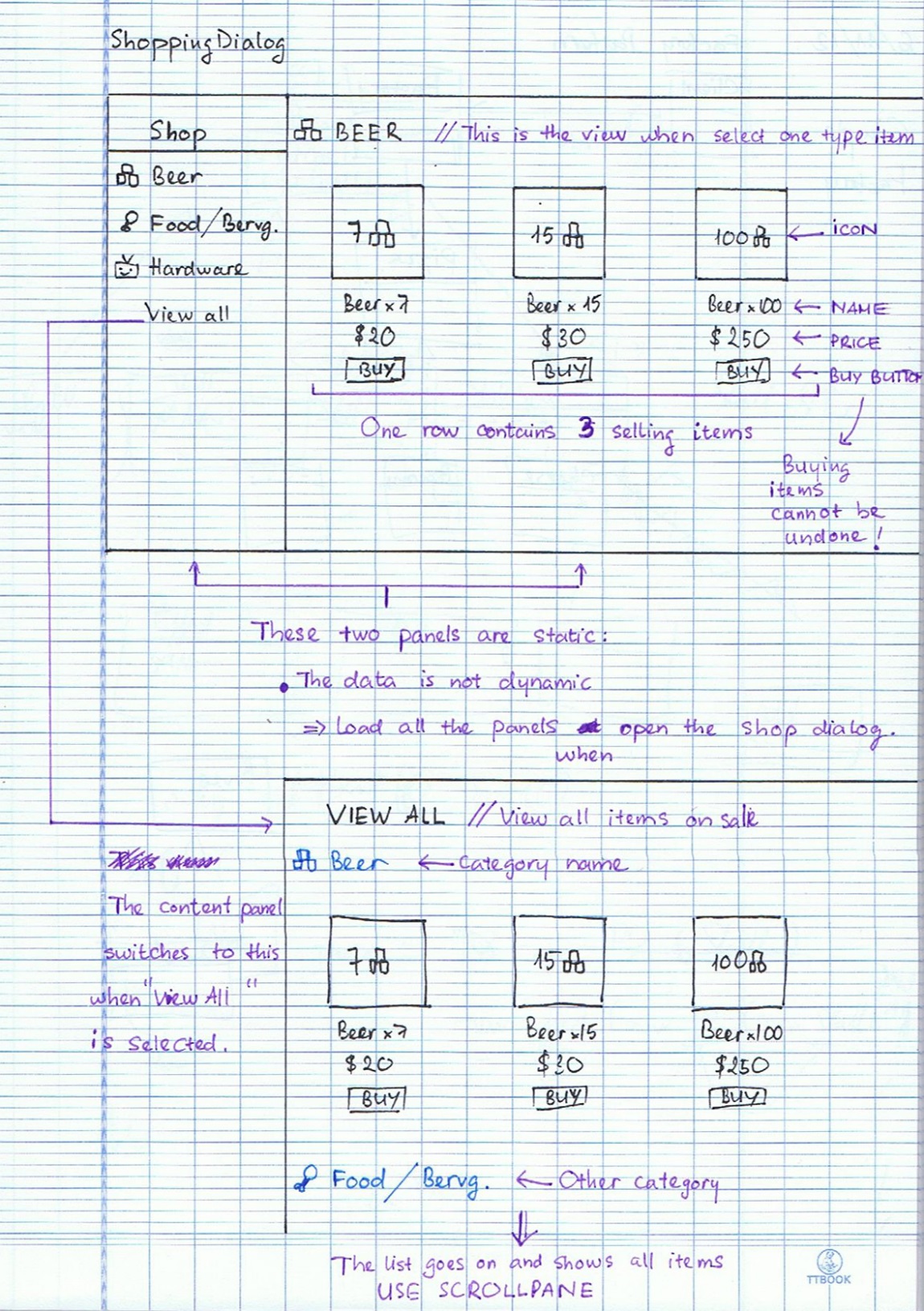


Figure 12. Shopping dialog design