BBM 384 – SOFTWARE ENGINEERING LABORATORY

SOCIAL INTEREST E-CLUB
Configuration and Change Management Report

COPY-PASTERS

HAZAL UZAR	**21727834**
HACI KERİM ARSLAN	**21726928**
İBRAHİM KOZ	**21786303**
BAŞAK ŞÜKRAN MELAHAT ÇONTU	**21827282**
FATMA NUR DEMİRBAŞ	**21727116**

Social Interest e-Club
Configuration and Change Management Report

1 Introduction

The configuration management focuses on managing the configurable items and the state of the system and it is the process for establishing and maintaining consistency of a product's performance, functional, and physical attributes with its requirements, design, and operational information throughout its life.

The change management focuses on managing the changes that affect the configurable items and the system. It is the process of analyzing the costs and benefits of proposed changes and tracking which components in the system have been changed.

Our system needs the change management to control the lifecycle of all changes, enabling beneficial changes to be made with minimal disruption. We also need configuration management to keep track of configurable items and the changes have been made on them.

The Configuration and Change Management Report contains the specifications for identifying, managing and documenting the changes that wanted be done or occurred unexpectedly. This report also explains how to do version control of the project to track the previous versions of the project components.

2 Purpose

To be able to fulfill the needs of configuration and change management, we adopt some methods to apply in our project. Each developer writes code individually in their own branches. After the other team member(s) approve the changes, the developer pushes his/her code to the main branch on GitHub. Since we plan each member's responsibility and the area that they work, we minimize the chance of getting merge conflicts. After all changes are verified, the new codes are merged to the main branch with explanations of changes and version number. The version number is an integer if there are big changes in the new version, else the number is a decimal number. The change control is achieved with good documentation and strong communication between the group members.

3 Configuration and Change Management Specifications

The configuration management activities of our software system are listed below, and it involves four closely related activities:

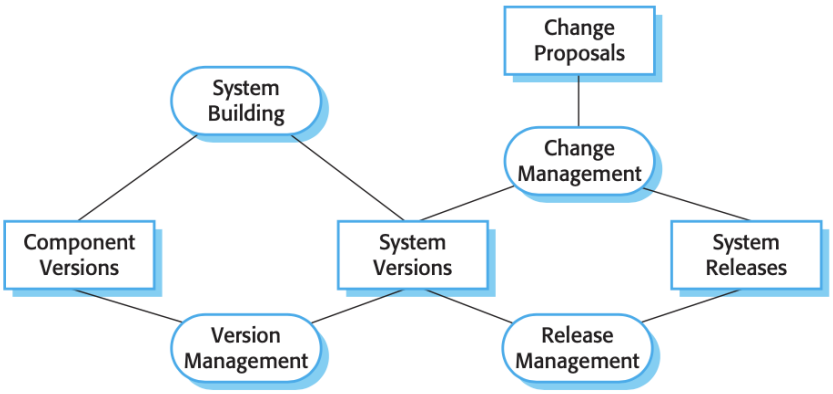
1. Version Control: We use distributed version control system (DVCS) that brings a local copy of the complete repository to every team member's computer, so we can commit, branch, and merge locally for this project. This system has the advantages of having reliable backup copies, fast merging and flexible branching, getting rapid feedback and fewer merge conflicts and the flexibility to work offline. Git is an example of DVCS, and it allows to do experiments without fearing that the changes that the developers make damage to the source code.

2. System Building: Each developer adds the necessary files for the system (such as code files, dependencies, external resources) to their own branch. They keep the folders systematic, so merging those files to create an executable system gets easier. In our system, front-end developers use Vue files to write HTML, CSS and JS codes and a folder to save external resources such as photos. Back-end developers use this path information to connect the back-end system with the front-end.

3.Change Management: Firstly, temporary changes are controlled to create a stable software development environment. Changes are committed to the repository. Then the demand is checked according to technical values, possible side effects and overall effect on the other configuration objects. Changes are managed and configuration items are made available throughout the software lifecycle. The changes are monitored through both GitHub and "configuration and change management document". All the changes are recorded to this document and compared with the older versions to optimize the change management.

4. Release Management: In our project, we used systems development life cycle to do release management. The SDLC helps us to plan, develop, maintain and replace software systems with a high degree of efficiency and quality**.** Our software release will be from the baseline and release objects files will be held in configuration control. All releases from development to integration and validation must provide change notes that contain following information:

1. What is in the release,
2. Who the release being provided to and when,
3. Any known problems in the release,
4. Any known fixes in the release,
5. Installation instructions



4 Key Considerations

We use GitHub for this project. It allows us to control development version with Git. The explicit version of our code is in our GitHub repository. Our group members clone it to their own workspace by using Git or just simply downloading the zip file. Before we add/change something in our code, we must make sure that our code doesn't break and works fine after changes thus we test it in our computer before modifying the project repository. This way changed version of our code will be available to other team members.

Change management is done by push commits, request for pull, review pull request and merge pull request. Each team member has his/her own branch to develop their own code in it and they make sure to keep their branches up to date with the main branch. If we want to modify the main branch, we follow these steps:

1. **Add commits:** We modify the files and make commits with these changes in our branches. We monitor our development by adding commits. Commits enable us to monitor other team members' work thus we can understand our teammates' code. We write explanatory commit messages to our commits. This way we can understand changes when another member modifies the code.
2. **Open a Pull Request:** After we commit changes to our branches and test them to make sure that they work, we create a pull request to modify the main branch too. We add explanatory comments to our pull requests to make specify what changes we have made.
3. **Discuss and review your code:** After pull request opening the pull request, team reviews the code. Pull request owner continue to push with respect to discussions about the changes.

4. **Deploy:** Before merging to main branch, we deploy from our own branch for testing. After pull request is reviewed and the code in this branch works fine, we can deploy changes to verify them in production. But if our branch doesn't work fine, we can cancel the changes by deploying from the main branch into the production.
5. **Merge:** We merge the code into the main branch after changes have been verified in production. Pull requests keep a record of changes and they are searchable thus we can see why and how a decision was made.

Our system is basically separated into two parts: Front-End and Back-end. Initially, we did not separate the system strictly. Everything was in the same repository, and developers worked on the same place. As the system progressed and growth, we decided to separate the parts of the system to different repositories. This arrangement allowed us (developers) to focus on what we are working at the time. Also, this prevented any undesirable/accidental effects that could be triggered by the developers.

There are different developers working on this system, and each one is working on her/his own branch on a different part of the system. The original version of the system is always reserved in the main branch (of that repository). However, all the branches had to be compatible with the main branch. Merging the branches into main branch was sometimes difficult because many merge conflicts would have been risen. So, we agreed on that the developers would merge the main branch into their branch frequently so that those conflicts would happen less. This convention increased the compatibility between the developers and allowed them to developed faster and easier.

P.S: The red parts of the document are updated on 26.05.2021